Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: **Master in Petroleum Engineering** **Specialisation in Drilling Engineering**	[✓ **Spring**] / Autumn semester, **2019** [✓ **Open**]/Confidential
Author: **Attila Nagy**	Nagy Attila (signature of author)
Programme coordinator: **Øystein Arild** Supervisor(s): **Jan Einar Gravdal**	
Title of master's thesis: **Availability and Quality of Drilling Data in the Volve Dataset**	
Credits: **30 ECTS**	
Keywords: **OpenLab, Volve dataset, Drilling simulator,** **Daily drilling report, Drilling data, Python**	Number of pages: **46** + supplemental material/other: **44** Stavanger, **14.06.2019**

Abstract

The importance of data sharing cannot be overexaggerated today, when the recent trend toward automation fundamentally reshapes the way of how operations are done in the oil industry. The emergence of real-time trend analysis softwares and digital twins led to the increase need of reliable data sources to build, test and verify the physical and data-driven models, which represent the core of these softwares. The OpenLab project at NORCE – by integrating physical and virtual drilling and well operations – is a prime example of these novel solutions. The purpose of this thesis is to provide actual field data for the OpenLab drilling simulator, so that students, researchers and industry experts can train and run simulations on virtual wellbores which are based on real well configurations.

The Volve dataset – disclosed by Equinor – provided the database for this work. Various data sources have been investigated in the drilling-related features, and both manual and automated data mining methods were developed in order to extract the required data. When the processed data were available, a web application has been developed and integrated into OpenLab. The end result enables the user to get an overview of the Volve wells with interactive visualization, and create configurations in the OpenLab drilling simulator based on actual well data. As a side-project, a browser-based application for well data management and visualization has also been created.

This thesis serves as an example of how publicly available field data combined with free educational softwares help new ideas to emerge. Hopefully the future holds similar initiatives both by the industry and by the research community. This way innovators from the academia could easily contribute to improved operational efficiency with creative and fresh ideas.

Table of Contents

List of Figures

List of Abbreviations

BHA – Bottom Hole Assembly

BLOB – Binary Large Object

BHP – Bottomhole Pressure

BPD – Barrels per Day

COO – Chief Operating Officer

CSS – Cascading Style Sheets

CSV – Comma-separated Values

CTO – Chief Technical Officer

DDR – Daily Drilling Reports

DLS – Dogleg Severity

EDM – Engineer's Data Model

E&P – Exploration and Production

FIT – Formation Integrity Test

GUI – Graphical User Interface

HPHT – High Pressure High Temperature

HSE&Q – Health, Safety, Environment and Quality

HTML – Hypertext Markup Language

IADC – International Association of Drilling Contractors

ID – Inner Diameter

IOR – Improved Oil Recovery

IRIS – International Research Institute of Stavanger

JPT – Journal of Petroleum Technology

JSON – JavaScript Object Notation

LCM – Loss Control Material

LOT – Leak-off Test

MATLAB – Matrix Laboratory

MD – Measured Depth

MPD – Managed Pressure Drilling

NCS – Norwegian Continental Shelf

NORCE – Norwegian Research Centre

NPD – Norwegian Petroleum Directorate

NPT – Non-productive Time

OD – Outer Diameter

PCA – POSC Caesar Association

PDF – Portable Document Format

PSA	Petroleum Safety Authority
PVT	Pressure, Volume, Temperature
PV	Plastic Viscosity
RDL	Reference Data Library
ROP	Rate of Penetration
RPM	Revolutions per Minute
TFA	Total Flow Area
UCS	Unconfined Compressive Strength
UK	United Kingdom
WITSML	Wellsite Information Transfer Standard Markup Language
XML	Extensible Markup Language
XSD	XML Schema Definition
YP	Yield Point

Acknowledgements

First and foremost, I give thanks to my supervisor, Jan Einar Gravdal for his guidence and the original idea to set off for this journey. To the whole OpenLab team at NORCE, especially to Robert Ewald and Andrew Holsaeter for helping me integrating the web application into OpenLab. To Dan Sui for the weekly meetings, which proved to be a perfect platform for new ideas to emerge. To Ankit Rohatgi, developer of WebPlotDigitizer. To all my professors, fellow students and industry experts who made me realize the importance of the digital transformation in our industry.

I need to thank the Norwegian state and society for providing me with free education, I am most appreciative.

The biggest thanks of all goes to my family, especially to my Mom. Without them I would have never been able to achieve this goal.

1 Introduction

1.1 Toward Drilling Automation

The recent trend toward automation is a key driver in the drilling industry. Semi- or fully autonomous drilling rigs can mitigate the risk when drilling in challenging conditions, such as narrow geopressure windows, deepwater or HPHT conditions. Physical models of the drilling process are an essential part of most automated drilling solutions. The use of thermodynamic, mechanical and hydraulic models allow these systems to estimate the current state of the entire wellbore, perform forward simulations and detect drilling incidences. (Cayeux, Daireaux, Dvergsnes, & Florence, 2014) Continuous modeling requires rig specifications, properties of the wellbore configuration and most importantly real-time operational parameters as input.

Softwares based on physical models can also be used as drilling simulators for technology development in digital drilling of oil and gas wells, enabling industry experts and researchers to test new methods, techniques and equipment (Exebenus, 2019). Moreover, this approach is extremely well suited for educational purposes to demonstrate drilling technology for students. Simulators also allow to compare historical and simulator-generated real-time datasets.

1.2 Neccessity of Access to Field Data

As digitalization gains momentum in the oil industry, the sharing of field data becomes of the utmost importance. It means experience transfer between companies, new source of input for research institutes and academia, as well as a positive change in the public perception of the industry, demonstrating transparency and willingness of cooperation.

Though in Norway a vast amount of petroleum related information has been publlicly available thanks to the Norwegian Petroleum Directorate, the sharing of the subsurface dataset of the Volve field by Equinor in June 2018 (Equinor, 2018) marked the beginning of a new era. Since then Lundin announced the collaboration with Aker BP regarding real-time production data sharing between the Edvard Grieg and Ivar Aasen platforms (Lundin, 2018), also providing all data from the Edvard Grieg field to the National IOR Centre of Norway for generating new research projects (UiS, 2019). It seems like this trend will not stop at the borders of Norway. The UK Oil and Gas Authority made available to the public some 130 terabytes of well, geophysical, field, and infrastructure data earlier this year (JPT, 2019). These events demonstrate how players from all levels realize the need of seeking new ways of cooperation in order to improve operating efficiency.

Data sharing is especially important for educational and research purposes. Generating new ideas would not be possible without close contact between the industry and academia. Often the very existence of research projects relies on the availability of actual field data, which is essential to verify physical models and run simulations. Moreover, the emergence of data-driven models leads to the increased need of consistent and reliable datasets to train these models with machine learning methods. As data-driven models also have their limitations, the future lies in combining physics- and data-based approaches (Noshi & Schubert, 2018).

Jannicke Nilsson, Equinor's COO summarized their intention with the Volve disclosure as follows: „We believe that the learning potential for students is huge when they can train on real data, and it will prepare them further for working on real cases in the future. ... We also share this data set to encourage higher productivity and innovation in the industry. We hope that it will not only help future energy innovators in their work, but also contribute to more efficient operation and possibly better interaction between players in our industry." (Equinor, 2018)

1.3 Introduction to OpenLab Drilling

A prime example of all the above is the OpenLab Drilling project at NORCE (formerly IRIS). The project has been developed in close collaboration with universities and the oil industry to provide a unique integration of physical and virtual well operations. OpenLab comprises three systems, of which the last two is under development:

- Web-based drilling simulator
- Drilling control room
- Full-scale on-site operational drilling rig

The core module of OpenLab is the web enabled drilling simulator, which is based on highly advanced well flow and drillstring mechanics models. The user interface offers a simple environment, where the user can create new well configurations or use existing templates, then run simulations either in the web browser or through MATLAB and Python clients.

The drilling control room provides control systems and 3D visualization of the drill floor to run downhole simulations. Last but not least, OpenLab will be connected to the physical rig Ullrigg, which is a full-scale drilling test site with access to seven wells. In this stage, surface and downhole measurements will be partially replaced by input from virtual wells using the OpenLab simulator (OpenLab Drilling, 2019).

1.4 Motivation

The purpose of this thesis has been to provide field data for running simulations in the OpenLab drilling simulator. To achive this goal, first the Volve dataset's drilling related features have been explored by analyzing data quality and availability The next step has been to extract well data and develop an integrated web application to display well configurations and export them to OpenLab. Therefore simulations on virtual wellbores resembling the ones drilled on the Volve field would be possible to conduct in OpenLab.

Well configuration in OpenLab refers to the elements and properties of the wellbore, rig, drilling equipment, drilling fluids and formation, all of which directly or indirectly influence the circulation system and drillstring mechanics. Therefore the scope of the thesis has been to collect these data and organize them in a suitable way, which makes the export to OpenLab possible.

While analyzing various data sources, the idea of another use case has emerged. It lead to the development of a browser-based application for daily drilling report data management and visualization, which provides a simple, interactive interface to scroll through the timeline of well operations and identify drilling problems.

2 Openlab Drilling Simulator

2.1 Configurations

The simulator can be accessed from the homepage of OpenLab (OpenLab Drilling, 2019). The web client has a user-friendly interface, but OpenLab also provides tutorials and manuals to begin with. The opening page offers the option of creating a new configuration when signing in for the first time. Several templates are available, and these can be modified later according to the purpose of the simulation. The well configuration page consists of six tabs:

- Hole section
- Wellpath
- Fluid
- Drillstring
- Geology
- Rig

Each tab consists of tables that take input and an interactive visualization which displays the input data and shows information on hover. Input is either manually typed or imported from CSV files. In the following the detailed content of each tab will be described.

The Hole Section tab takes input to create the wellbore structure, such as ID and OD of the riser, casings and liners, open hole interval and diameter together with wellhead elevation, casing setting depths and liner hanger depths. All depths are measured depth in this section. The visualization shows the wellbore structure in 2D.

The Wellpath tab requires trajectory data including MD, inclination and azimuth, while TVD and DLS are calculated based on these values. A 3D wellbore trajectory is displayed to give an interactive overview of the wellpath.

In the Fluid tab the user can select from predefined drilling fluids and modify parameters as needed. Fluid properties include fluid type, density, oil-water ratio, gel strength, Fann dial readings and base-oil PVT values. Pie charts show the mass fraction and volume fraction distribution, while a rheogram displays the flow behaviour of the fluid. Fluid characteristics can be specified for the main and the reserve fluid separately.

The Drillstring tab needs input to specify the tools that comprise the BHA, such as tool ID, OD, length and linear weight, additionaly TFA for bits. Moreover, drillpipe specifications such as body ID and OD, tool joint ID and OD, average joint length, tool joint length and total length are taken to define the total length of the drillstring. The visualization shows a representaion of the drillstring, displaying the details of each element by hover.

In the Geology tab there are three subsections, these are: Geopressures, Geothermal and Formation. They take input to create the pressure, temperature and formation strength profile of the wellbore. Input data are TVD, formation pressure, fracture pressure, air temperature, water depth, temperature gradient and UCS. Each well profile is displayed on a separate graph.

Lastly, the Rig tab's visualization displays the a schematic of the rig layout. Input data in this section includes specifications of mud pump and MPD pump (maximum flow rate acceleration), BOP and MPD choke (maximum change rate), top drive (maximum rotation acceleration) and drawwork (maximum top of string acceleration), as well as traveling block weight, mud loss proportion at the shakers and volume of the mud tanks.

2.2 Simulations

All inputs described above are essential to build the physical models which are the heart of the simulator. These include flow (Lorentzen & Fjelde, 2005) (Lorentzen, Nævdal, Karlsen, & Skaug, 2014), temperature (Corre, Eymard, & Gounet, 1984), cuttings transport (Cayeux, Mesagan, Tanripada, Zidan, & Fjelde, 2014) and torque & drag models (Yunfeng & al., 2004) (Kyllingstad, 1995). Simulations can be run using either predefined templates or configurations constructed by the user. After setting some parameters (initial bit depth and top of string position, mud volume and temperature, influx and loss options) a new simulation is created. Simulations can be run in real-time, fast-forward or sequence mode. Parameters that can be adjusted during a simulation are:

- Main pump and MPD pump flow rate
- MPD choke opening
- Top of string speed
- ROP
- Surface RPM
- BOP open/closed
- Mud density

When a simulation is running, the state of the drilling operation is displayed in various time- and depth-based graphs. The complete list of these plots is found in Appendix A. It is out of the scope of this thesis to introduce the simulator in detail, but to get a grasp of the opportunities provided by OpenLab, some use cases are listed here:

- Keeping constant BHP with back pressure MPD.
- Initiating a well control incident, circulating a kick.
- Simulating cuttings bed accumulation and erosion.

3 Volve Dataset

The Volve dataset can be accessed at Equinor's Data Portal (Volve Data Village, 2019) after a free registration. The data village consists of more then 40000 files, including geological and geophysical interpretations, static and dynamic reservoir models, production data, well logs, well technical data and real-time drilling data. This thesis is focusing on drilling data that is needed to create well configurations in OpenLab. Due to the enormous size of the dataset, the scope was narrowed to well technical data and well logs. The analysis of real-time drilling data should be part of a future research.

3.1 Volve Field

The field is located in the Norwegian sector of the North Sea in block 15/9, approximately 200 km west of Stavanger (Figure 1).

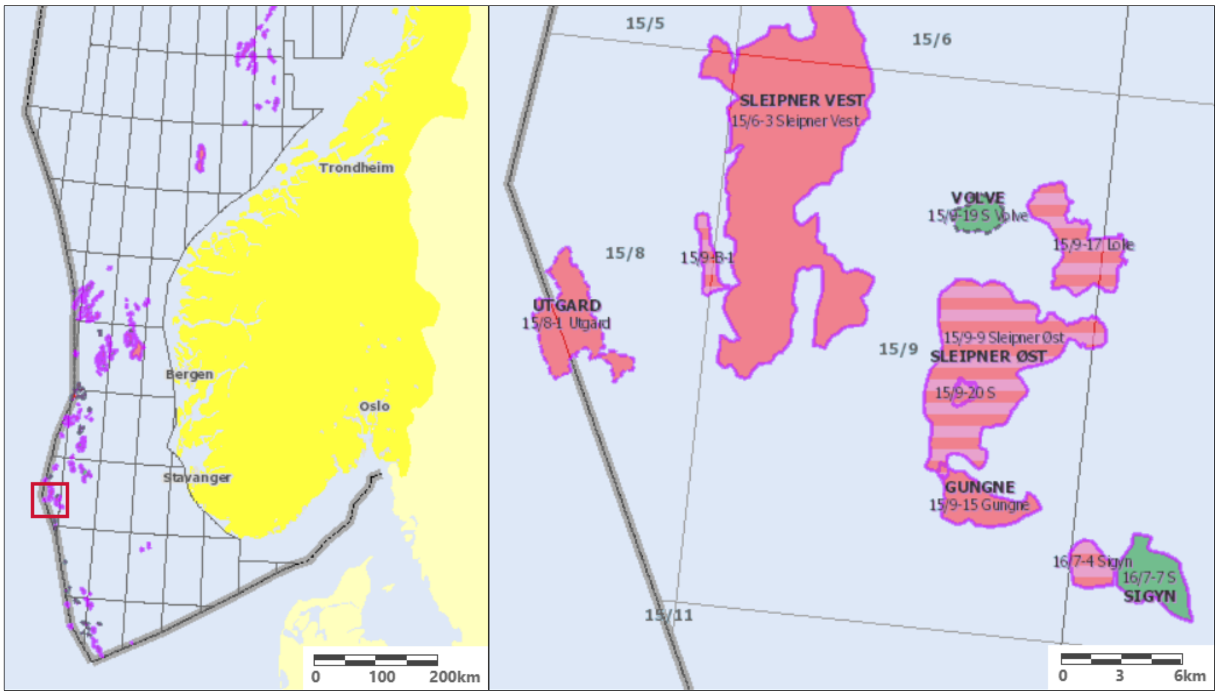


Figure 1: Location of the Volve field on the NCS (NPD, 2019)

The licensees were Equinor as operator, and ExxonMobil and Bayerngas as partners. The field was discovered in 1993, production started in 2008. Average water depth in the area is 80 meters, the Middle Jurassic sandstone reservoir rocks lie in a depth of 2700-3100 meters. Field development was based on production from the Mærsk Inspirer jack-up rig, pressure support was provided with water injection. A total of 25 wellbores were drilled: 5 exploration, 9 production, 8 observation and 3 injection wells. Oil was shipped with tankers, while gas was piped to the Sleipner A platform for final processing and export. At peak, Volve produced some

56000 BPD, delivering a total of 63 million barrels of oil until being shut down in 2016, resulting in a lifetime twice as long as initially planned. Recovery rate reached 54% (NPD, 2019).

3.2 Landmark EDM Database

Parts of this chapter are based on blog posts by Oliasoft CTO Marius Kjeldahl (Kjeldahl, 2019). The dataset named Well Technical Data has the following directory structure:

```
Well_technical_data
├── CasingSeat
├── CasingWear
├── Compass
├── Daily Drilling Report - HTML Version
├── Daily Drilling Report - XML Version
├── Daily Drilling report - PDF Version
├── EDM.XML
├── Site
├── Site_TemplateSlot
├── StressCheck
├── WellPlan
├── WellWellbore
├── Wellcat
├── EDT_EDM_read_me.txt
└── license.txt
```

The directories `CasingSeat`, `CasingWear`, `Compass`, `StressCheck`, `WellPlan` and `Wellcat` refer to Landmark products (Landmark is the name of software solutions for the E&P industry by Halliburton). Oddly enough these folders are either empty or contain files only for a fraction of the Volve wellbores. The `EDT_EDM_read_me.txt` file informs the user that the `Volve F.edm.xml` file in the `EDM.XML` folder holds all exported Landmark data. EDM database is an architecture that Landmark products can read and write to. XML is a popular language that encodes documents in a format that is both human- and machine-readable.

3.2.1 Database Structure

The `Volve F.edm.xml` file is a relational database, consisting of 318000 lines. Unfortunately half of the file is made up of BLOBs, these objects are useless unless they can be reverse engineered to their original format. Of course this was out of the scope of this thesis. The basic structure of the database is the following. Each line starts with a tag (blue), which has several attributes (red), which in turn have values (purple). The example below shows a line with the tag `CD_WELLBORE`.

```
<CD_WELLBORE
  well_id="6ekXXuRhFd"
  wellbore_id="FHcPbdxSIP"
  wellbore_type_id="Yerjn"
  bh_md="11368.4383201645"
  bh_tvd="10017.908058355397"
  is_deviated="Y"
  geo_offset_east_bh="1428265.842391806"
  geo_offset_north_bh="2.125394252713205E7"
  geo_latitude_bh="58.43840621636125"
  geo_longitude_bh="1.892469878637818"
  geo_offset_east_ko="1427330.08660846"
  geo_offset_north_ko="2.12551385169754E7"
  geo_latitude_ko="58.44163725137954"
  ko_md="298.556430445"
  ko_tvd="298.556430445"
  geo_longitude_ko="1.8874826218670009"
  well_legal_name="NO 15/9-F-12"
  wellbore_name="F-12"
  is_readonly="N"
  create_date="{ts '2005-11-02 13:22:53'}"
  create_user_id="pio"
  create_app_id="COMPASS"
  update_date="{ts '2017-09-30 12:49:27'}"
  update_user_id="frro(FRRO@STATOIL.NET)"
  update_app_id="COMPASS"
  default_fluid_id="E6O19" />
```

It is clear that this wellbore entry is connected to other entries through the `wellbore_id`, `wellbore_type_id` and `default_fluid_id` attributes. Attribute values contain information on depth, kick-off point and geographic location. This entry was created with Compass, which is the directional wellpath planning product of Landmark. Another example represents trajectory data. The `CD_DEFINITIVE_SURVEY_STATION` tag attributes contain values necessary for wellpath planning and anti-collision analysis.

```
<CD_DEFINITIVE_SURVEY_STATION
  well_id="2DfUHhBj3x"
  wellbore_id="3W9ZjV2yj0"
  def_survey_header_id="iM93r"
  definitive_survey_id="y5K7P"
  azimuth="44.712372334329274"
  offset_east="-525.0256422747742"
  offset_north="566.3351553136704"
  covariance_yy="1.0E-8"
  sequence_no="13"
  ellipse_vertical="0.0"
  covariance_yz="0.0"
  covariance_zz="0.0"
  covariance_xx="1.0E-8"
  data_entry_mode="0"
  covariance_xy="0.0"
  dogleg_severity="3.2004"
  inclination="60.79827671850135"
  covariance_xz="0.0"
  ellipse_east="0.0"
  ellipse_north="0.0"
  md="9367.125984214495"
  casing_radius="7.0"
  tvd="8606.708750482321"
  global_lateral_error="0.0" />
```

The above snippets serve to showcase the logic behind the architecture of this database. There are more than 140 different tags in the database. In addition to the above, information on drillstring, casing strings, fluids, temperature gradient and geopressure are available among others. Often several hundreds or thousands of lines with the same tags are present, containing information on the different wellbores. It is out of the scope of this thesis to present the whole database, but the interested reader can find the detailed list of tags together with their corresponding location in Appendix B.

3.2.2 Data Mining Challenges

An effort was made to write scripts in Python to automate data extraction from the Landmark data dump. A sample code snippet of these scripts is included in Appendix B. Several challenges arose during this process, the main ones were as follows:

- Missing wellbores: when collecting BHA, drilling fluid, formation pressure and fracture pressure related data, it became obvious that complete wellbore datasets were missing, that happened most often with the exploration and appraisal wellbores. One of the possible explanations for this is the presence of binary data in the database. Most likely these unreadable BLOBs have some valuable information, which cannot be accessed.
- Missing values: these were most pronounced in fluid entries, where the lack of important rheological properties made it impossible to collect all necessary information. Other common missing values were tubular IDs and ODs. Moreover, the temperature profiles of the wells were overly simplified, only one temperature gradient applied along the whole wellpath.
- Ambiguous data: prototype, planned and actual well data exist together in the database, and it often led to confusion whether an entry represents actual data, or it is just an experimental value used when the well planner was trying different prototypes for a specific scenario.

As the goal of this thesis was to construct well configurations, preferably using one consistent data source, the difficulties encountered during data extraction resulted in terminating further work with this database and continuing with other data sources.

3.3 Daily Drilling Reports

In the next stage the analysis of the Volve dataset focused on the daily drilling reports. These documents are also located in the dataset named Well Technical Data, and available in three different file formats, namely HTML, PDF and XML. As the content of these files are identical, further work was done on the DDRs in XML, as this format is the most suitable to parse files programmatically.

3.3.1 DDR Generation and WITSML Standards

DDRs are among the most important documents generated during drilling, completion, workover and intervention operations. They compile information from various independent data sources to give a comprehensive overview of well operations. The report content may vary from country to country, also depending on operators. In Norway the report structure and content are regulated by the Petroleum Safety Authority, operators are obliged to submit daily reports to both NPD and PSA. In 2008 the Norwegian industry adopted the so-called WITSML standards, an industry initiative to provide reference for data transmission from rig-site to offices onshore. It is also used to share information with partners and governmental agencies. WITSML is web-based and built on XML technology. By using these standards, all instrumentation and software can work together, and incompatible data structures are no longer a problem (Energistics, 2019).

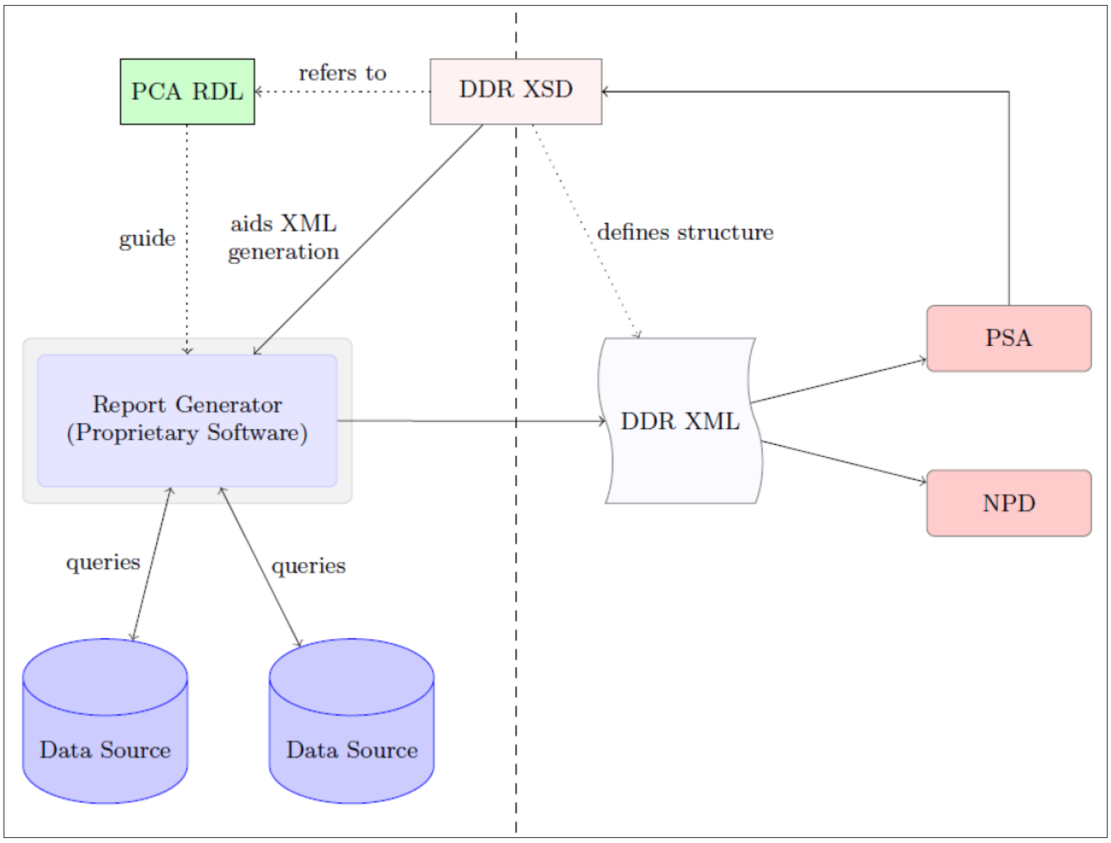


Figure 2: The process of DDR generation (Giese, Ornas, Overå, Svensson, & Waaler, 2012)

To better understand data quality and data availability issues when extracting data from DDRs, the process of DDR generation using WITSML standards shall be introduced (Figure 2). XML serves not only as a transmission format, but also as a documentation format through the associated XML Schema Definition (XSD) along with textual definitions from a Reference Data Library (RDL). The RDL contains definitions in natural language, moreover it defines the relationship between definitions in the form of superclasses, classes and subclasses (Giese, Ornas, Overå, Svensson, & Waaler, 2012). Basically, XSD defines the structure and form of the report, while RDL guides the user to select proper terms and definitions when reporting. More information about XSD is provided by PSA (PSA, 2019), while RDL can be accessed at the website of POSC Caesar Association (PCA, 2019).

3.3.2 DDR Structure

The document below shows a daily drilling report of the 15/9-19-A well from the Volve field. At the time of the drilling of this well the WITSML format had not been in use, the original report must have been converted according to the new standards. It serves a good example to examine the structure of the report. Of course the information recorded in this file is only a fraction of the available options that is usually part of a report. Let us first examine the architecture. The main constituents are the elements, which consist of tags (blue), attributes (red) and values (black). Elements have different names depending on their status in the hierarchy, such as root (here `drillReports`), child (`drillReport`), subchild (`statusInfo`) and so on. This tree-like structure makes it easy for humans to read, without sacrificing machine-readability. The report starts with a declaration, then comes the root `drillReports`. The root attributes refer to the reporting schema, XSD has been discussed in Chapter 3.3.1. Information that can be of interest to this thesis is located within the child `drillReport`. After some general well data, the element `statusInfo` holds details about the recent status of the wellbore, together with a summary of past and forecasted activities.

```
<xml version="1.0" encoding="ISO-8859-1">
<witsml:drillReports
xsi:schemaLocation="http://www.witsml.org/schemas/1series
http://www.npd.no/schema//DDRS/1series/WITSML_drillReport_profiled_schema_p
hase2.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:witsml="http://www.witsml.org/schemas/1series"
version="1.4.0.0">
    <witsml:documentInfo>
        <witsml:documentName>Drill Report 1.2.2</witsml:documentName>
        <witsml:owner>IIP</witsml:owner>
    </witsml:documentInfo>
    <witsml:drillReport>
        <witsml:nameWell>NO 15/9-19 A</witsml:nameWell>
        <witsml:nameWellbore>NO 15/9-19 A</witsml:nameWellbore>
        <witsml:name>name</witsml:name>
        <witsml:dTimStart>1997-07-24T00:00:00+02:00</witsml:dTimStart>
        <witsml:dTimEnd>1997-07-25T00:00:00+02:00</witsml:dTimEnd>
        <witsml:versionKind>normal</witsml:versionKind>
        <witsml:createDate>2018-05-03T13:53:19+02:00</witsml:createDate>
        <witsml:wellAlias>
            <witsml:name>15/9-19 A</witsml:name>
            <witsml:namingSystem>NPD code</witsml:namingSystem>
        </witsml:wellAlias>
        <witsml:wellboreAlias>
```

```
            <witsml:name>15/9-19 A</witsml:name>
            <witsml:namingSystem>NPD code</witsml:namingSystem>
        </witsml:wellboreAlias>
        <witsml:wellboreAlias>
            <witsml:name>3145</witsml:name>
            <witsml:namingSystem>NPD number</witsml:namingSystem>
        </witsml:wellboreAlias>
        <witsml:wellboreInfo>
            <witsml:dTimSpud>1997-07-25T00:00:00+02:00</witsml:dTimSpud>
            <witsml:dateDrillComplete>1997-08-30</witsml:dateDrillComplete>
            <witsml:operator>Statoil</witsml:operator>
            <witsml:rigAlias>
                <witsml:name>BYFORD DOLPHIN</witsml:name>
                <witsml:namingSystem>NPD Name</witsml:namingSystem>
            </witsml:rigAlias>
            <witsml:rigAlias>
                <witsml:name>341331</witsml:name>
                <witsml:namingSystem>NPD code</witsml:namingSystem>
            </witsml:rigAlias>
        </witsml:wellboreInfo>
        <witsml:statusInfo>
            <witsml:reportNo>1</witsml:reportNo>
            <witsml:dTim>1997-07-25T00:00:00+02:00</witsml:dTim>
            <witsml:md uom="m">2213</witsml:md>
            <witsml:mdPlugTop uom="m">2203</witsml:mdPlugTop>
            <witsml:mdKickoff uom="m">2178</witsml:mdKickoff>
            <witsml:mdCsgLast uom="m">4643</witsml:mdCsgLast>
            <witsml:mdPlanned uom="m">-999.99</witsml:mdPlanned>
            <witsml:distDrill uom="m">-999.99</witsml:distDrill>
            <witsml:elevKelly uom="m">25</witsml:elevKelly>
            <witsml:wellheadElevation
            uom="m">106.5</witsml:wellheadElevation>
            <witsml:waterDepth uom="m">84</witsml:waterDepth>
            <witsml:sum24Hr>MU BAKER WINDOWMASTER WHIPSTOCK &amp; ASSOCIATED
            BHA &amp; TIH.</witsml:sum24Hr>
            <witsml:forecast24Hr>TIH &amp; SET BAKER WHIPSTOCK &amp; MILL WINDOW IN
            9 5/8" CASING.</witsml:forecast24Hr>
            <witsml:ropCurrent uom="m/h">-999.99</witsml:ropCurrent>
            <witsml:tightWell>0</witsml:tightWell>
            <witsml:hpht>0</witsml:hpht>
            <witsml:fixedRig>false</witsml:fixedRig>
        </witsml:statusInfo>
    </witsml:drillReport>
</witsml:drillReports>
```

The `statusInfo` element is just one example of the entries that can hold relevant operational data. Below follow the subchilds in the Volve DDRs which are to record information related to different well operations. The next chapter will elaborate on these entries separately.

`activity` – operational activities info
`bitRecord` – bit runs info
`casing_liner_tubing` – tubular info
`coreInfo` – core sampling info
`equipFailureInfo` – equipment failure info
`fluid` – properties of drilling and completion fluid
`gasReadingInfo` – gas reading and gas properties

`lithShowInfo` – rock class info
`logInfo` – well logging info
`perfInfo` – perforation info
`porePressure` – pore pressure evaluation
`statusInfo` – wellbore status info
`stratInfo` – geological formation info
`surveyStation` – directional survey info
`wellTestInfo` – well test info

3.3.3 DDR Content

It is important to mention that the collection below comprises the complete list of recordings. Separate DDRs only have entries that are relevant to the current operations for that given day.

The `activity` entry contains information on operational activities including date, start time, end time, MD, activity code, state of the activity and comments. Activity code refers to the operation phase, such as drilling, completion, formation evaluation, etc. The complete list of activity codes can be found in Appendix C. State of the activity differentiates between productive and non-productive time. Categories used here are success for productive time, mud loss & circulation loss, equipment failure, injury, operation failed and stuck equipment for NPT. Comments give a short description of the activity.

The `bitRecord` element consists of data on bit runs including IADC dull grading, total MD drilled, total hours drilled, average ROP, number of nozzles, nozzle diameter and a short comment on the bit condition.

The `casing_liner_tubing` section has information on tubular goods being ran in the hole including casing, liner and tubing. Properties listed are tubular type, material grade, connection type, ID, OD, linear weight, length, top MD, bottom MD and a short description of the purpose of the run.

The `coreInfo` element has information on core sampling including sampling length, core length, recovery rate, core barrel type, barrel length and description of the core.

The `equipFailure` entry holds equipment failure data including date and time of the failure, current MD where the failure happened, name of the equipment, description of the failure, time of the repair and missed productive time.

The `fluid` section consists of information on the properties of drilling or completion fluid used during operations including fluid class, fluid type, density, funnel viscosity, PV, YP, sampling location and fluid loss test results such as filtrate volume and filter cake thickness.

The `gasReadingInfo` entry contains information on gas readings including type of the reading, MD, TVD, total gas concentration, concentration of methane, ethane, propane, butane and pentane. Type of the reading can be connection gas, drilling gas peak, flow check gas and trip gas.

The `lithShowInfo` element comprises information on lithology including MD of top and bottom of the rock unit, description of the lithology and possible oil shows.

The `logInfo` section consists of well logging data including MD of the logging interval, description of the logging tool and name of the service company.

The `perfInfo` entry has information on perforation activities including MD of top and bottom of the perforated interval and time of firing the perforation guns.

The `porePressure` section contains information on formation pressure evaluation including MD where the evaluation is valid to, pore pressure expressed as equivalent mud weight and type of the evaluation. Evaluation type is either estimated or measured.

The `statusInfo` element consists of some general well data as well as information on the recent status of the wellbore including date, time of spudding, number of the report, water depth, Kelly bushing elevation, wellhead elevation, current MD and TVD of the wellbore, distance drilled, current average ROP, planned MD, MD of top of the hole section, MD and TVD of the last casing shoe, MD of the kick-off point (if applicable), MD of the plug (if applicable), diameter of the hole, diameter of the pilot hole (if applicable), average bottom hole pressure and temperature, summary of activities in the last 24 hours, forecasted activities in the next 24 hours. Additional information consists of formation strength or fracture pressure as recorded during a FIT or LOT, also MD and TVD of the FIT or LOT. Moreover, HPHT and tight well conditions are stated in this section, in addition to the type of the drilling rig.

The `stratInfo` entry has information on geological formations including MD and TVD of top of the formation and formation name

The `surveyStation` section comprises directional survey data including MD, TVD, inclination and azimuth.

The `wellTestInfo` entry consists of information on well tests including density, flow rate and total volume of oil, water and gas. Additional information consists of test type, MD of the flowing formation, choke orifice size, shut-in pressure, flowing pressure, gas-oil ratio, water-oil ratio. Moreover chloride, carbon dioxide and hydrogen sulfide concentrations are recorded.

3.3.4 Data Extraction and Missing Data

The presented XML file structure offers a perfect opportunity to automate data mining from the DDRs. Codes in Python have been written to automate the process, making it possible to parse all 1759 DDRs and extract data which is required for further use in a matter of a few seconds. As both the structure and the content are consistent thanks to the applied WITSML standards, the scripts developed for the Volve dataset DDRs should be transferable to other sources with only minor or no changes needed, given that the same standards are used. An example of these scripts is in Appendix C.

Though DDRs offer a wide variety of well data, it became obvious that not all information which is essential to construct well configurations in OpenLab is available. DDRs do not have any info on bottom hole assemblies, also the drilling fluid data are insufficient, oil water ratio as well as important rheological properties such as 10 sec and 10 min gel strength and Fann readings are missing. The recorded formation pressure, fracture pressure and temperature values are also deficient to create proper pressure and temperature profiles of the wellbores. To fulfill the purpose of this thesis, yet another data source should be explored, and these are going to be the End of Well Reports.

But to say that is was of no avail to examine the Landmark database and the DDRs is wrong. The lessons learned and experience gained during this work led to an idea to develop an application for well data management and visualization, which is based on automated data extraction from the DDRs. The detailed process of the development of this app and the description of how it works together with a possible use case is the topic of Chapter 5.

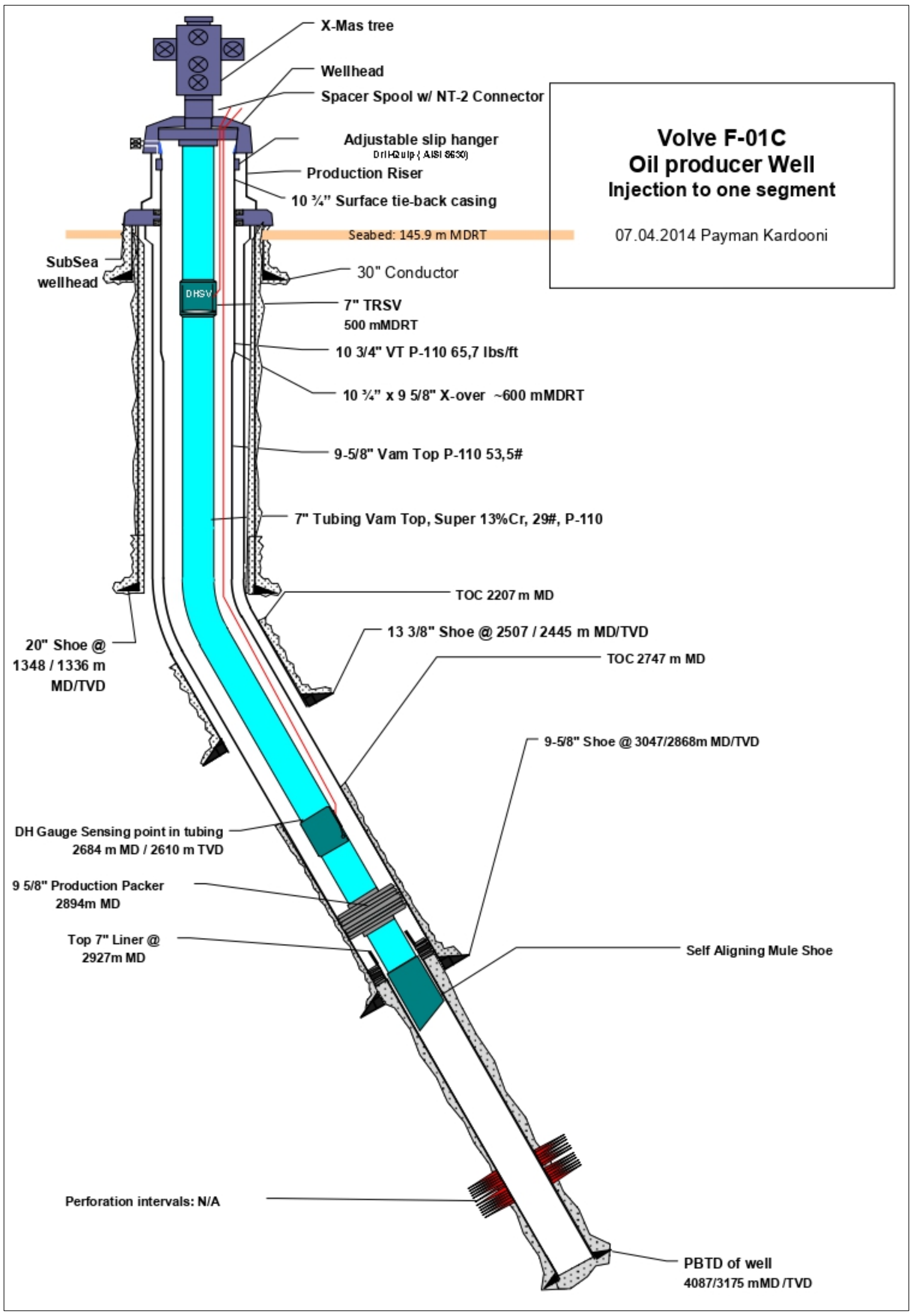
3.4 Final Well Reports

Final Well Reports, also known as End of Well Reports give a summary of general well data, HSE&Q experience listing with future recommendations, evaluation of incidences and drilling performance, cost and time distribution of well operations, activity highlights and formation evaluation. Appendices consist of directional surveys, detailed listing of operations, wellbore schematics, time/depth curve, pressure and temperature profiles, bit record, BHA list, drilling fluid program and cementing program. Basically, FWRs provide all the information that is needed to reach the goal of the thesis. The only disadvantage is that these reports require manual processing and data mining. As more than 20 years passed between the drilling of the first and the last well in the Volve field, the structure of the FWRs changed throughout the years. It means that the location and the layout of the different sections are inconsistent.

FWRs are located in the Diverse Reports directory of the Well Logs. The next chapter offers examples about how the required well data have been collected and which parts of FWRs provide relevant information. The 15/9-F1 C well serves to showcase data extraction methods, but the process is valid for the other wellbores as well.

3.4.1 Data Collection

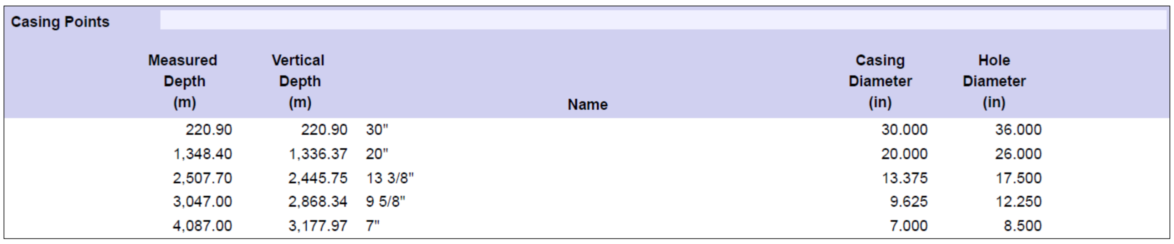
In this chapter the process of manual data extraction from FWRs is explained in detail. The order reflects the way of how well configurations are generated in the OpenLab drilling simulator.



X-Mas tree
Wellhead
Spacer Spool w/ NT-2 Connector
Adjustable slip hanger
DrilQuip (AISI 8630)
Production Riser
10 ¾" Surface tie-back casing
Seabed: 145.9 m MDRT
Volve F-01C
Oil producer Well
Injection to one segment
07.04.2014 Payman Kardooni
SubSea wellhead
30" Conductor
DHSV
7" TRSV
500 mMDRT
10 3/4" VT P-110 65,7 lbs/ft
10 ¾" x 9 5/8" X-over ~600 mMDRT
9-5/8" Vam Top P-110 53,5#
7" Tubing Vam Top, Super 13%Cr, 29#, P-110
TOC 2207 m MD
20" Shoe @ 1348 / 1336 m MD/TVD
13 3/8" Shoe @ 2507 / 2445 m MD/TVD
TOC 2747 m MD
9-5/8" Shoe @ 3047/2868m MD/TVD
DH Gauge Sensing point in tubing 2684 m MD / 2610 m TVD
9 5/8" Production Packer 2894m MD
Top 7" Liner @ 2927m MD
Self Aligning Mule Shoe
Perforation intervals: N/A
PBTD of well 4087/3175 mMD /TVD

Figure 3: 15/9-F1 C well schematic (Volve Data Village, 2019)

The hole section tab needs input to create the wellbore structure. Figure 3 shows a wellbore schematic displaying the detailed structure of the well. This data was double-checked using the casing points section of the directional surveys, confirming that the values are valid. Slice of a directional survey report containing information on casings is shown in Figure 4. No information on riser ID and OD was found, so standard drilling riser diameters (ID: 19 in, OD: 21 in) were used. Casing IDs were extracted from the DDRs `casing_liner_tubing` section.

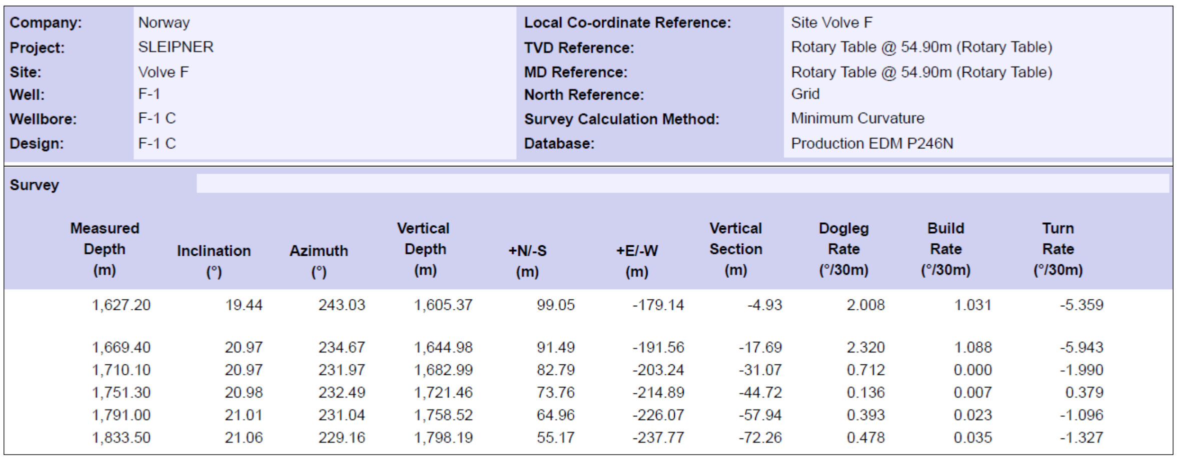


Casing Points

Measured Depth (m)	Vertical Depth (m)	Name	Casing Diameter (in)	Hole Diameter (in)
220.90	220.90	30"	30.000	36.000
1,348.40	1,336.37	20"	20.000	26.000
2,507.70	2,445.75	13 3/8"	13.375	17.500
3,047.00	2,868.34	9 5/8"	9.625	12.250
4,087.00	3,177.97	7"	7.000	8.500

Figure 4: Part of a directional survey showing casing points (Volve Data Village, 2019)

The wellpath section requires input to construct the well trajectory. Directional survey is part of the FWR and holds survey data as illustrated in Figure 5. Only entries such as MD, inclination and azimuth were extracted, as other values are computed internally by the OpenLab drilling simulator.



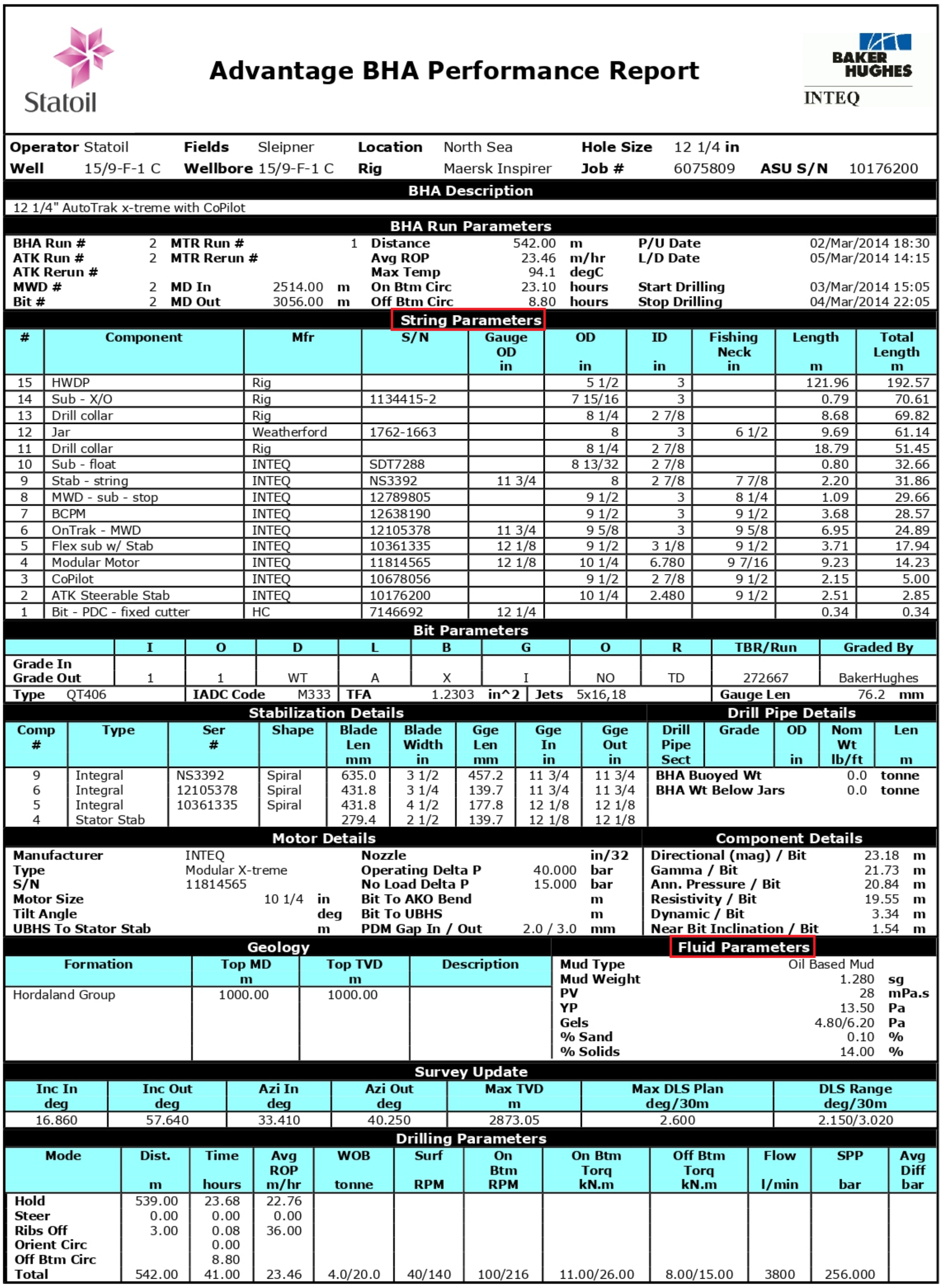
Company:	Norway	**Local Co-ordinate Reference:**	Site Volve F
Project:	SLEIPNER	**TVD Reference:**	Rotary Table @ 54.90m (Rotary Table)
Site:	Volve F	**MD Reference:**	Rotary Table @ 54.90m (Rotary Table)
Well:	F-1	**North Reference:**	Grid
Wellbore:	F-1 C	**Survey Calculation Method:**	Minimum Curvature
Design:	F-1 C	**Database:**	Production EDM P246N

Survey

Measured Depth (m)	Inclination (°)	Azimuth (°)	Vertical Depth (m)	+N/-S (m)	+E/-W (m)	Vertical Section (m)	Dogleg Rate (°/30m)	Build Rate (°/30m)	Turn Rate (°/30m)
1,627.20	19.44	243.03	1,605.37	99.05	-179.14	-4.93	2.008	1.031	-5.359
1,669.40	20.97	234.67	1,644.98	91.49	-191.56	-17.69	2.320	1.088	-5.943
1,710.10	20.97	231.97	1,682.99	82.79	-203.24	-31.07	0.712	0.000	-1.990
1,751.30	20.98	232.49	1,721.46	73.76	-214.89	-44.72	0.136	0.007	0.379
1,791.00	21.01	231.04	1,758.52	64.96	-226.07	-57.94	0.393	0.023	-1.096
1,833.50	21.06	229.16	1,798.19	55.17	-237.77	-72.26	0.478	0.035	-1.327

Figure 5: Part of a directional survey with columns such as MD, inclination and azimuth (Volve Data Village, 2019)

Data that is needed to populate the fluid tab are available in the Fluid Parameters section of the BHA Performance Report, which is part of the directional survey report (Figure 6). Oil-water ratio is not shown here, but it is also recorded in the same document. OpenLab also considers base-oil PVT as input to model the density change of the mud. This information was not available in the dataset, so the standard values given in the simulator were left unchanged in this case.



Statoil

Advantage BHA Performance Report

BAKER HUGHES

INTEQ

Operator Statoil		**Fields**	Sleipner	**Location**	North Sea	**Hole Size**	12 1/4 **in**		
Well	15/9-F-1 C	**Wellbore**	15/9-F-1 C	**Rig**	Maersk Inspirer	**Job #**	6075809	**ASU S/N**	10176200

BHA Description

12 1/4" AutoTrak x-treme with CoPilot

BHA Run Parameters

BHA Run #	2	**MTR Run #**	1	**Distance**	542.00	**m**	**P/U Date**	02/Mar/2014 18:30
ATK Run #	2	**MTR Rerun #**		**Avg ROP**	23.46	**m/hr**	**L/D Date**	05/Mar/2014 14:15
ATK Rerun #				**Max Temp**	94.1	**degC**		
MWD #	2	**MD In**	2514.00 **m**	**On Btm Circ**	23.10	**hours**	**Start Drilling**	03/Mar/2014 15:05
Bit #	2	**MD Out**	3056.00 **m**	**Off Btm Circ**	8.80	**hours**	**Stop Drilling**	04/Mar/2014 22:05

String Parameters

#	Component	Mfr	S/N	Gauge OD in	OD in	ID in	Fishing Neck in	Length m	Total Length m
15	HWDP	Rig			5 1/2	3		121.96	192.57
14	Sub - X/O	Rig	1134415-2		7 15/16	3		0.79	70.61
13	Drill collar	Rig			8 1/4	2 7/8		8.68	69.82
12	Jar	Weatherford	1762-1663		8	3	6 1/2	9.69	61.14
11	Drill collar	Rig			8 1/4	2 7/8		18.79	51.45
10	Sub - float	INTEQ	SDT7288		8 13/32	2 7/8		0.80	32.66
9	Stab - string	INTEQ	NS3392	11 3/4	8	2 7/8	7 7/8	2.20	31.86
8	MWD - sub - stop	INTEQ	12789805		9 1/2	3	8 1/4	1.09	29.66
7	BCPM	INTEQ	12638190		9 1/2	3	9 1/2	3.68	28.57
6	OnTrak - MWD	INTEQ	12105378	11 3/4	9 5/8	3	9 5/8	6.95	24.89
5	Flex sub w/ Stab	INTEQ	10361335	12 1/8	9 1/2	3 1/8	9 1/2	3.71	17.94
4	Modular Motor	INTEQ	11814565	12 1/8	10 1/4	6.780	9 7/16	9.23	14.23
3	CoPilot	INTEQ	10678056		9 1/2	2 7/8	9 1/2	2.15	5.00
2	ATK Steerable Stab	INTEQ	10176200		10 1/4	2.480	9 1/2	2.51	2.85
1	Bit - PDC - fixed cutter	HC	7146692	12 1/4				0.34	0.34

Bit Parameters

	I	O	D	L	B	G	O	R	TBR/Run	Graded By
Grade In										
Grade Out	1	1	WT	A	X	I	NO	TD	272667	BakerHughes

Type	QT406	**IADC Code**	M333	**TFA**	1.2303 **in^2**	**Jets**	5x16,18	**Gauge Len**	76.2 **mm**

Stabilization Details

Comp #	Type	Ser #	Shape	Blade Len mm	Blade Width in	Gge Len mm	Gge In in	Gge Out in
9	Integral	NS3392	Spiral	635.0	3 1/2	457.2	11 3/4	11 3/4
6	Integral	12105378	Spiral	431.8	3 1/4	139.7	11 3/4	11 3/4
5	Integral	10361335	Spiral	431.8	4 1/2	177.8	12 1/8	12 1/8
4	Stator Stab			279.4	2 1/2	139.7	12 1/8	12 1/8

Drill Pipe Details

Drill Pipe Sect	Grade	OD in	Nom Wt lb/ft	Len m
BHA Buoyed Wt			0.0	tonne
BHA Wt Below Jars			0.0	tonne

Motor Details

Manufacturer	INTEQ		**Nozzle**		**in/32**
Type	Modular X-treme		**Operating Delta P**	40.000	**bar**
S/N	11814565		**No Load Delta P**	15.000	**bar**
Motor Size	10 1/4	**in**	**Bit To AKO Bend**		**m**
Tilt Angle		**deg**	**Bit To UBHS**		**m**
UBHS To Stator Stab		**m**	**PDM Gap In / Out**	2.0 / 3.0	**mm**

Component Details

Directional (mag) / Bit	23.18	**m**
Gamma / Bit	21.73	**m**
Ann. Pressure / Bit	20.84	**m**
Resistivity / Bit	19.55	**m**
Dynamic / Bit	3.34	**m**
Near Bit Inclination / Bit	1.54	**m**

Geology

Formation	Top MD m	Top TVD m	Description
Hordaland Group	1000.00	1000.00	

Fluid Parameters

Mud Type	Oil Based Mud	
Mud Weight	1.280	**sg**
PV	28	**mPa.s**
YP	13.50	**Pa**
Gels	4.80/6.20	**Pa**
% Sand	0.10	**%**
% Solids	14.00	**%**

Survey Update

Inc In deg	Inc Out deg	Azi In deg	Azi Out deg	Max TVD m	Max DLS Plan deg/30m	DLS Range deg/30m
16.860	57.640	33.410	40.250	2873.05	2.600	2.150/3.020

Drilling Parameters

Mode	Dist. m	Time hours	Avg ROP m/hr	WOB tonne	Surf RPM	On Btm RPM	On Btm Torq kN.m	Off Btm Torq kN.m	Flow l/min	SPP bar	Avg Diff bar
Hold	539.00	23.68	22.76								
Steer	0.00	0.00	0.00								
Ribs Off	3.00	0.08	36.00								
Orient Circ		0.00									
Off Btm Circ		8.80									
Total	542.00	41.00	23.46	4.0/20.0	40/140	100/216	11.00/26.00	8.00/15.00	3800	256.000	

Figure 6: Part of the BHA Performance Report as extracted from the 15/9-F1 C well's directional survey report (Volve Data Village, 2019)

A challenge was encountered, namely that Fann dial readings are not listed here, only PV and YP values. This problem was solved by calculating 300 RPM and 600 RPM dial readings using equation (1) and (2), then determining the missing values with curve-fitting.

$$PV = \theta 600 - \theta 300 \tag{1}$$

$$YP = \theta 300 - PV \tag{2}$$

Where

PV: plastic viscosity [cP]
YP: yield point [lb/100ft^2]
$\theta 600$: Fann dial reading at 600 RPM [-]
$\theta 300$: Fann dial reading at 300 RPM [-]

OpenLab takes Fann dial readings at 3, 6, 30, 60, 100, 200, 300 and 600 RPM as input. In order to calculate these, the Herschel-Bulkley fluid model, as shown in Equation (3), was implemented.

$$\tau = \tau_0 + K \cdot \gamma^n \tag{3}$$

Where

τ: shear stress [Pa]
τ_0: yield stress [Pa]
K: consistency index [Pa s]
γ: shear rate [1/s]
n: flow index [-]

Rearranging Equation (3) yields $log(\tau - \tau_0) = logK + n \cdot log\gamma$. This problem can be reduced to a simple curve fitting in the form of $y = a + b \cdot x$. Here τ refers to Fann dial reading, γ to RPM and τ_0 to YP. After determining K and n using YP, $\theta 300$ and $\theta 600$ values, Fann dial readings at any arbitrary RPM can be calculated.

The BHA Performance Report String Parameters section (Figure 6) holds data which is needed to construct the drillstring in OpenLab. ID, OD and length values were directly transferable, while component names needed to be changed to their OpenLab equivalent. Information on the linear weight of the different components, as well as drillpipe tool joint diameters were not available here, so it was looked up in technical sheets provided by manufacturers online.

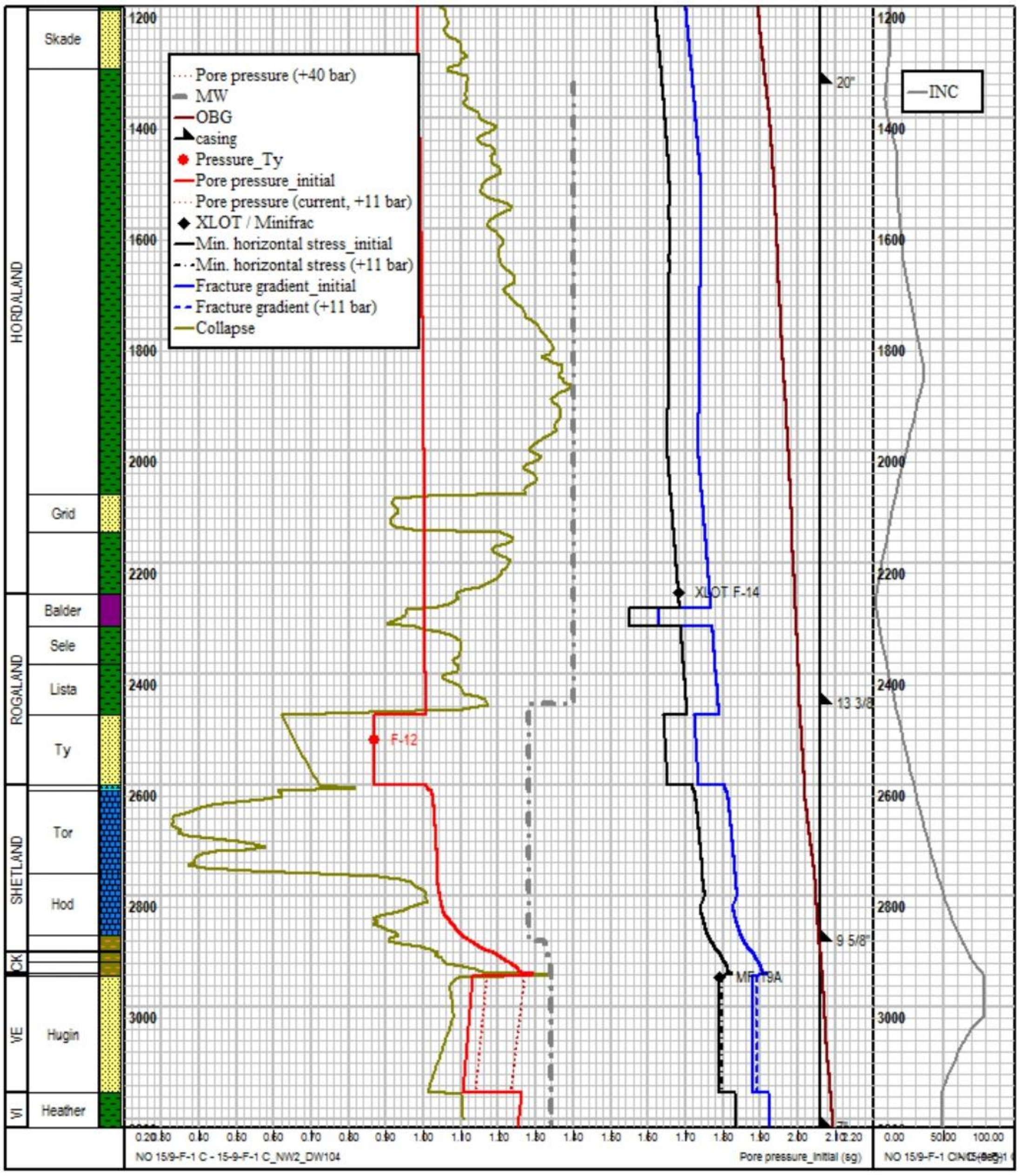


Figure 7: Pore pressure and wellbore stability prognosis for the 15/9-F1 C wellbore (Volve Data Village, 2019)

In order to provide the simulator with geological data, information on formation pressure, fracture pressure and geothermal gradient is required as input. It can be achieved by extracting data points from wellbore stability (Figure 7) and geothermal plots (Figure 8). These graphs are also part of the FWRs, but the datasets which are needed to reconstruct them are not available. Therefore a free software that was developed for this purpose, the WebPlotDigitizer was used, which enables semi-automated data extraction from various types of graphs. OpenLab also takes UCS as an optional input, but due to the lack of any compressive strength characteristics in the reports, this information is not available.

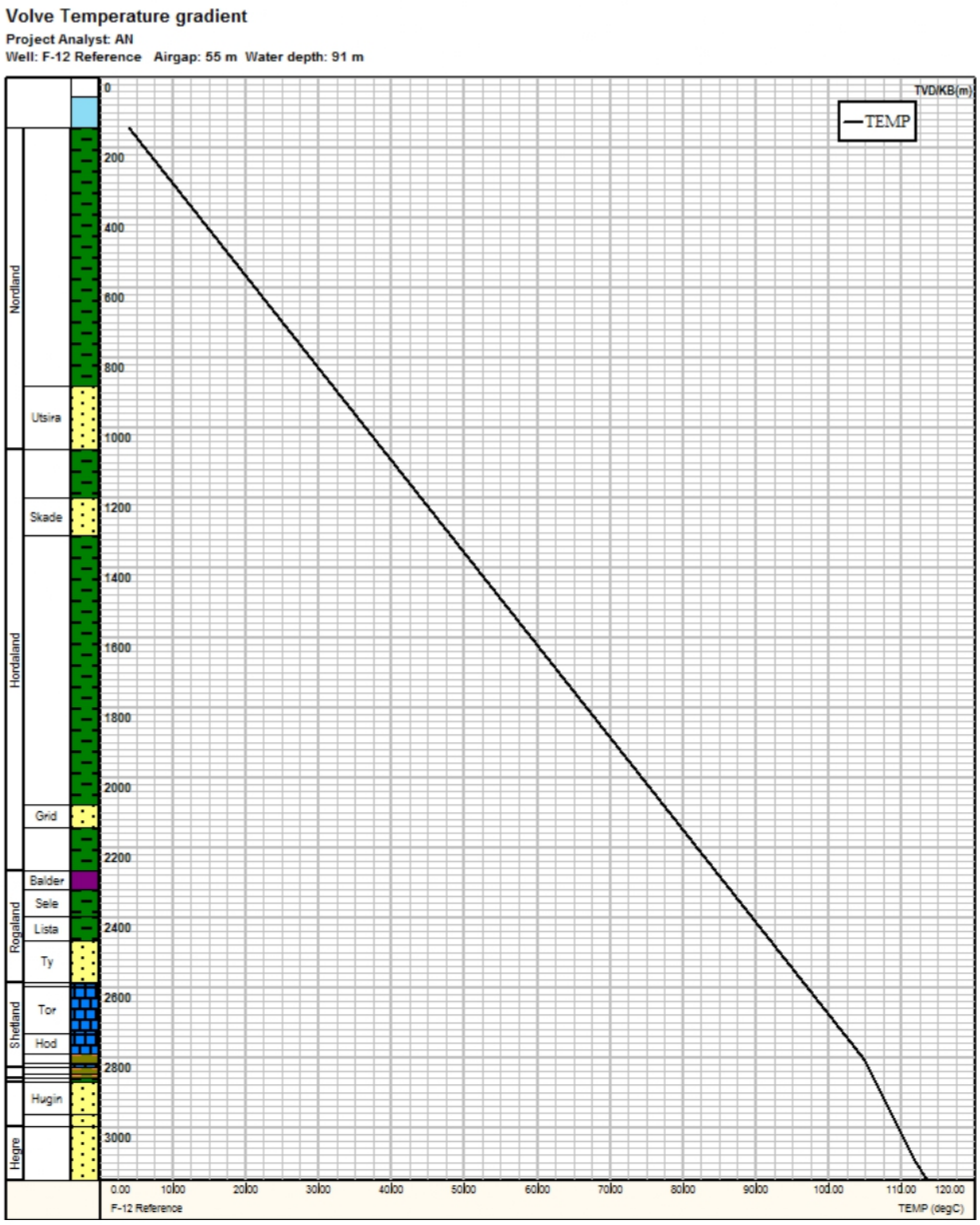


Figure 8: Geothermal gradient vs depth curve for the 15/9-F1 C wellbore (Volve Data Village, 2019)

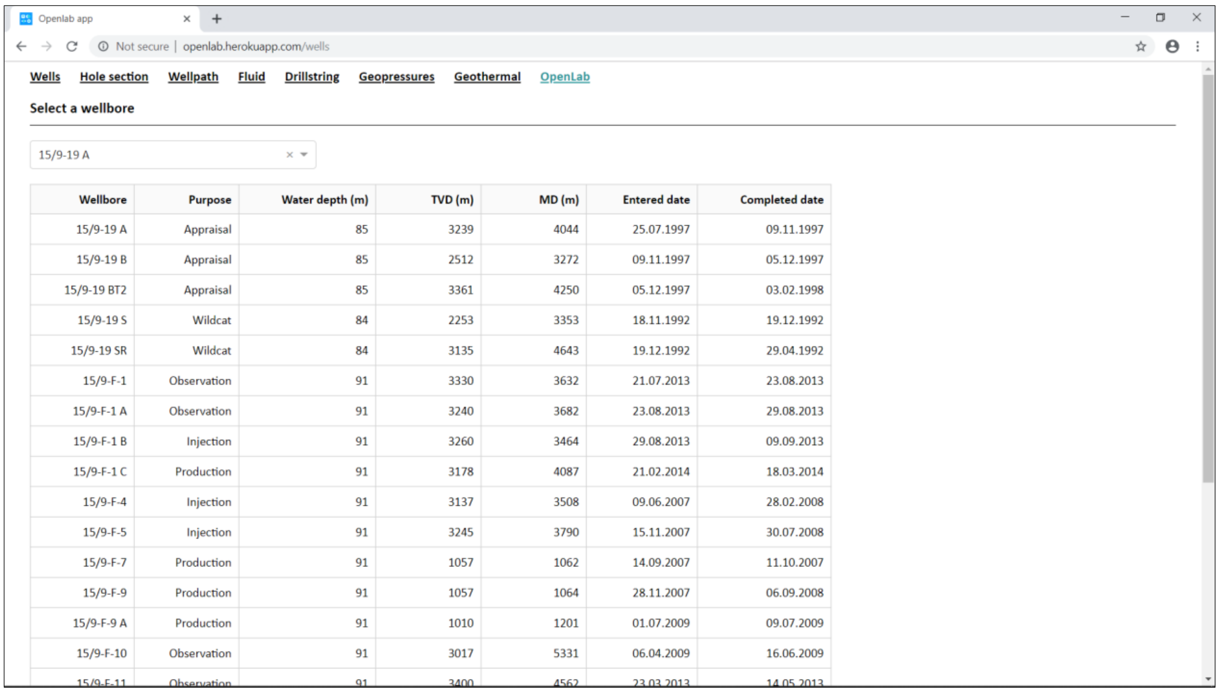
Lastly, the issue of missing data was also encountered when collecting data from the FWRs, though it was hardly as explicit as in case of the Landmark database and DDRs. It affected only fluid and drillstring information. To overcome this problem, information from offset Volve wellbores has been used to make up for the missing values.

4 Volve-OpenLab App

When all the required data were collected, checked, organized and saved as CSV files, it was time to design and develop the application that would serve as a user interface to display well data and communicate with the OpenLab drilling simulator. The idea was to follow the same structure and layout as it is in OpenLab, in order to make it easy for the user to get an overview of the different well configurations. The first intention was to develop a desktop application using the Python TkInter library. As the simulator itself is a web-enabled app, after some initial steps it was realized that creating a web application fits best for the purpose of the thesis. Therefore further work was done with the Dash framework.

Dash is a Python framework, which offers a simple solution to develop interactive web applications without JavaScript. Every aesthetic element in Dash is fully customizable thanks to CSS. Another huge advantage of Dash is its lightweight nature, minimal lines of code is required for a fully-functional GUI. Another Python libraly, Plotly was used to create interactive graphs resembling those in OpenLab as closely as possible. Finally, the OpenLab Python client made it possible to integrate the app with the drilling simulator and achieve the final goal of the thesis, by providing an integrated web application that contains well information of the Volve wells. The most recent version of the app is deployed to https://openlab.herokuapp.com. In the future it might be a part of the OpenLab simulator, in this case it will be accessable directly through the interface of the simulator. The Python code is attached in Appendix D.

The structure of the app is the following. On the top of the page is a navigation bar, where the user can go to the corresponding pages, the order follows the logic of the well configurations in OpenLab. There is an extra page called Wells (Figure 9), where a table shows general information, such as wellbore name, purpose of the well, water depth, TVD, MD, entered and completed data of the wellbore. Based on this information the user can select a wellbore from the dropdown, then examine the details of that certain wellbore by navigating to the Hole section, Wellpath, Fluid, Drillstring, Geopressures and Geothermal pages. The OpenLab page serves as the connection point to the OpenLab simulator. After selecting a wellbore and a hole section, the user can send that specific well configuration to his or her OpenLab account and it is ready to run simulations on. The idea was to provide configurations separately for each section of all the wellbores. It proved to be difficult to carry out, as OpenLab has a requirement regarding well architecture, it says that the minimum number of casing strings are two. It means configurations that could be used to simulate the drilling of the conductor and the surface section cannot be constructed in OpenLab. In the following the app content will be presented through an example of the 15/9-19 A well. This was an appraisal well, sidetracked from the 15/9-19 SR exploration well.



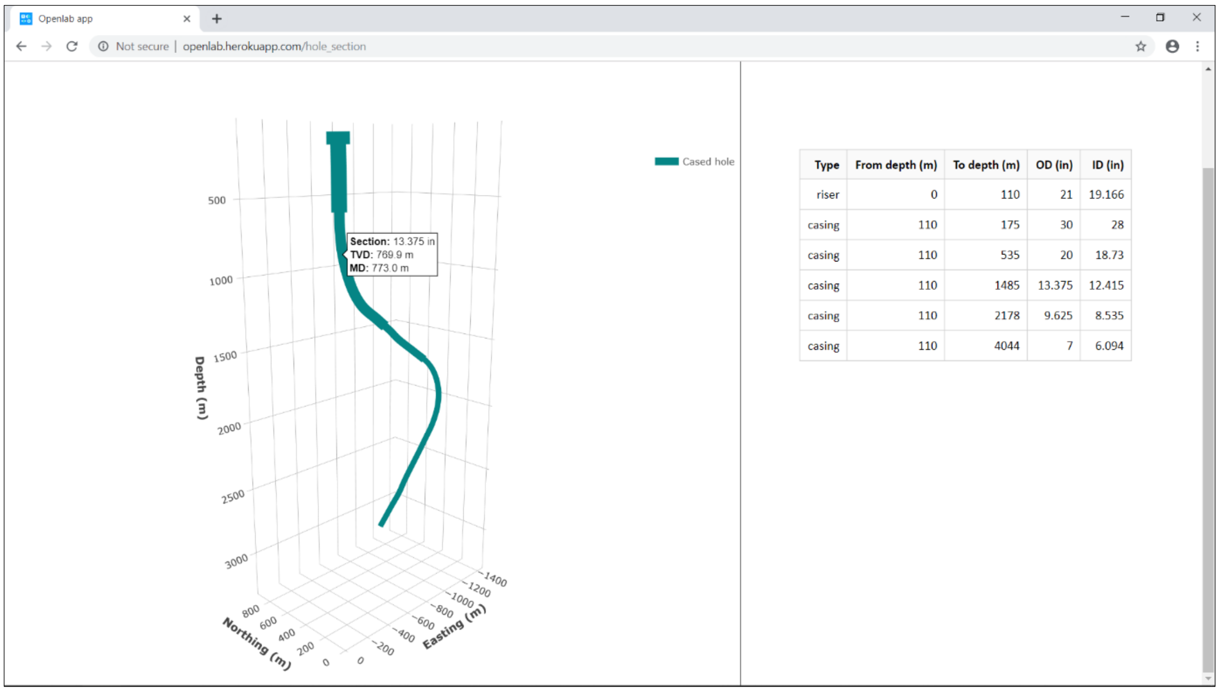
Select a wellbore

15/9-19 A

Wellbore	Purpose	Water depth (m)	TVD (m)	MD (m)	Entered date	Completed date
15/9-19 A	Appraisal	85	3239	4044	25.07.1997	09.11.1997
15/9-19 B	Appraisal	85	2512	3272	09.11.1997	05.12.1997
15/9-19 BT2	Appraisal	85	3361	4250	05.12.1997	03.02.1998
15/9-19 S	Wildcat	84	2253	3353	18.11.1992	19.12.1992
15/9-19 SR	Wildcat	84	3135	4643	19.12.1992	29.04.1992
15/9-F-1	Observation	91	3330	3632	21.07.2013	23.08.2013
15/9-F-1 A	Observation	91	3240	3682	23.08.2013	29.08.2013
15/9-F-1 B	Injection	91	3260	3464	29.08.2013	09.09.2013
15/9-F-1 C	Production	91	3178	4087	21.02.2014	18.03.2014
15/9-F-4	Injection	91	3137	3508	09.06.2007	28.02.2008
15/9-F-5	Injection	91	3245	3790	15.11.2007	30.07.2008
15/9-F-7	Production	91	1057	1062	14.09.2007	11.10.2007
15/9-F-9	Production	91	1057	1064	28.11.2007	06.09.2008
15/9-F-9 A	Production	91	1010	1201	01.07.2009	09.07.2009
15/9-F-10	Observation	91	3017	5331	06.04.2009	16.06.2009
15/9-F-11	Observation	91	3400	4562	23.03.2013	14.05.2013

Figure 9: Wells page shows general well data and enables the selection of wellbores

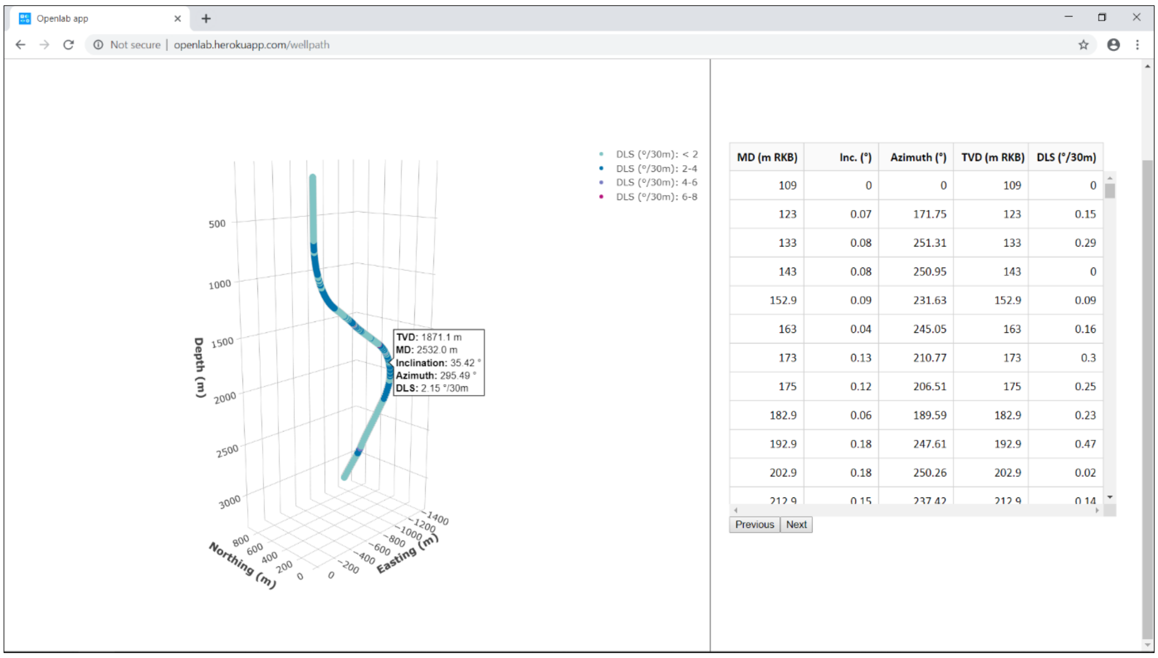
The Hole section page (Figure 10) shows the wellbore structure on a 3D interactive graph, which displays information about hole or casing diameter, MD and TVD on hover. Casing points and diameters are listed in the table on the right. The different sections are not true to scale, but it makes it easy to differentiate between casings with distinct diameters. Coloring also aids the user, cased hole is marked with bluish-green, while open hole with brown.



Type	From depth (m)	To depth (m)	OD (in)	ID (in)
riser	0	110	21	19.166
casing	110	175	30	28
casing	110	535	20	18.73
casing	110	1485	13.375	12.415
casing	110	2178	9.625	8.535
casing	110	4044	7	6.094

Figure 10: Hole section page with 3D representation of the wellbore structure

The Wellpath page (Figure 11) is similar to the previous one, except that the graph here shows the wellbore trajectory and coloring reflects to the dogleg severity at the separate survey points. Hover info consists of TVD, MD, inclination, azimuth and DLS, these values are also shown in the table on the right side of the page. The 3D plot can be rotated and zoomed-in to give a better view of the wellpath.

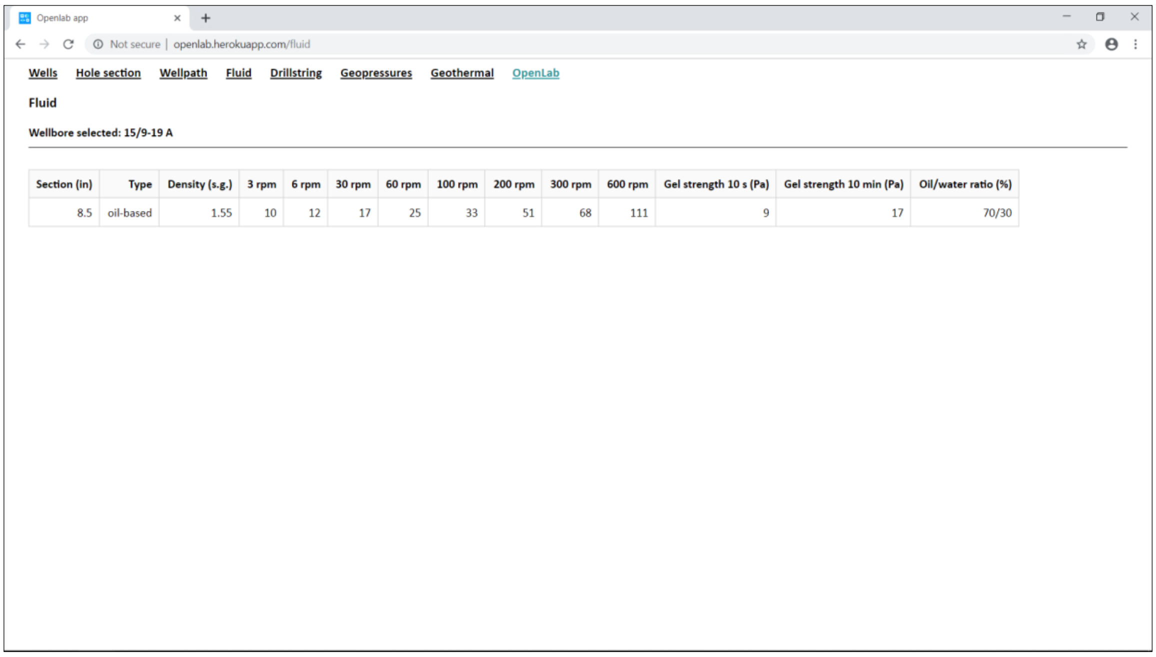


TVD: 1871.1 m
MD: 2532.0 m
Inclination: 35.42 °
Azimuth: 295.49 °
DLS: 2.15 °/30m

MD (m RKB)	Inc. (°)	Azimuth (°)	TVD (m RKB)	DLS (°/30m)
109	0	0	109	0
123	0.07	171.75	123	0.15
133	0.08	251.31	133	0.29
143	0.08	250.95	143	0
152.9	0.09	231.63	152.9	0.09
163	0.04	245.05	163	0.16
173	0.13	210.77	173	0.3
175	0.12	206.51	175	0.25
182.9	0.06	189.59	182.9	0.23
192.9	0.18	247.61	192.9	0.47
202.9	0.18	250.26	202.9	0.02
212.9	0.15	237.42	212.9	0.14

Figure 11: Wellpath page, the survey points are colored according to dogleg severity

The table on the Fluid page (Figure 12) shows information separately for each wellbore section.



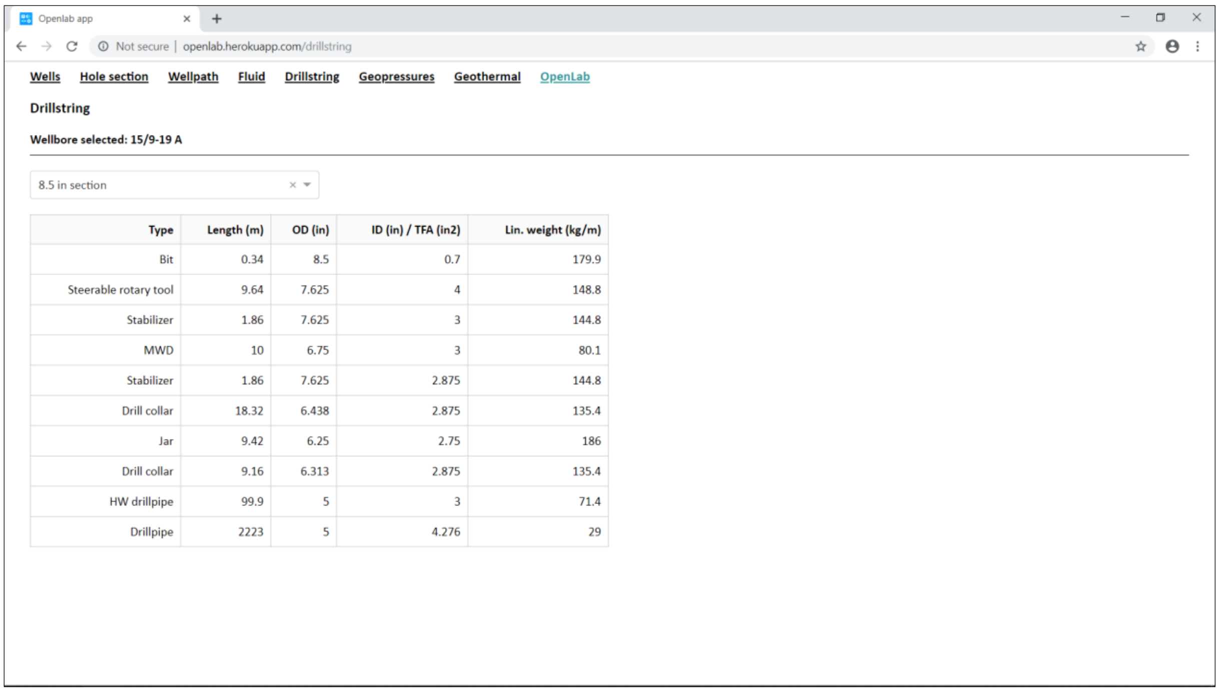
Fluid

Wellbore selected: 15/9-19 A

Section (in)	Type	Density (s.g.)	3 rpm	6 rpm	30 rpm	60 rpm	100 rpm	200 rpm	300 rpm	600 rpm	Gel strength 10 s (Pa)	Gel strength 10 min (Pa)	Oil/water ratio (%)
8.5	oil-based	1.55	10	12	17	25	33	51	68	111	9	17	70/30

Figure 12: Fluid page displaying density and rheological properties

As the 15/9-19 A well was a sidetrack, there is only one row, this holds data on the drilling fluid, which was used during drilling the 8 $^1/_2$ inches section. In most cases there were only slight adjustments in the rheology and the density of the fluid within one section. The properties shown in the app are the ones that were the most representative for the given section. OpenLab provides the option of specifying a reserve fluid with different characteristics. To simplify the case, this reserve fluid is assumed to have the same properties, except having a slightly higher density (+0.05 SG). It was already mentioned that the two uppermost section needs to be present for all configurations, so fluids that apply for those sections are not shown, even if the well is not a sidetrack.



Drillstring

Wellbore selected: 15/9-19 A

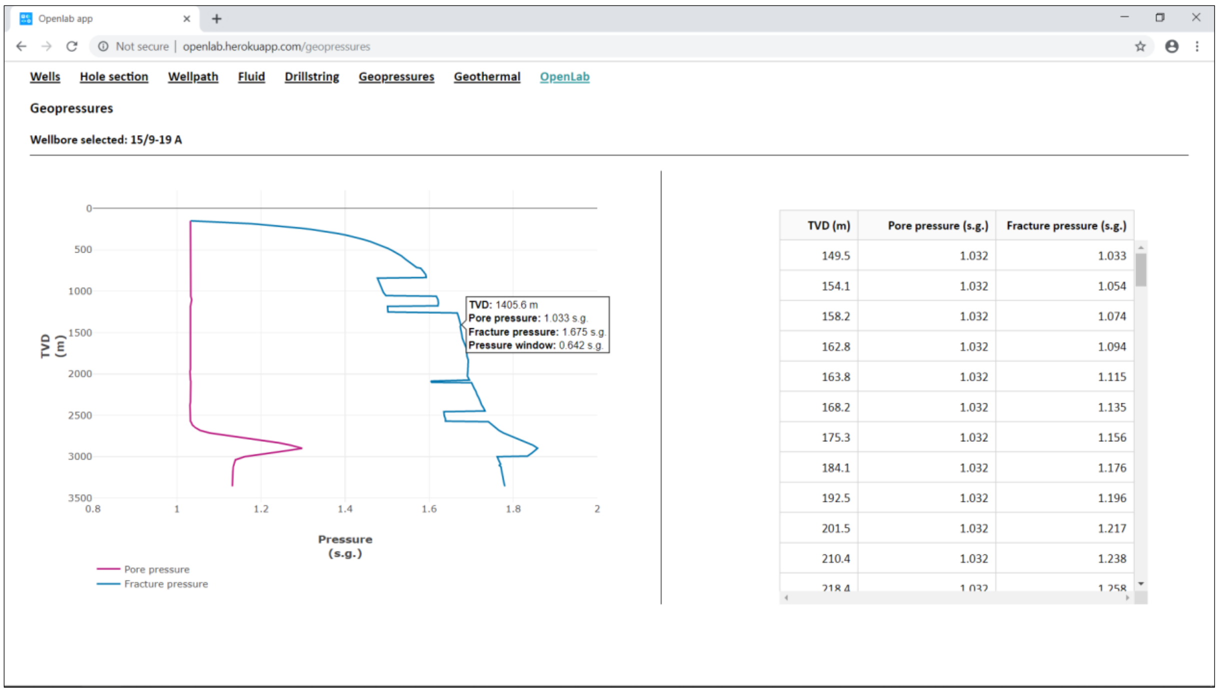
8.5 in section

Type	Length (m)	OD (in)	ID (in) / TFA (in2)	Lin. weight (kg/m)
Bit	0.34	8.5	0.7	179.9
Steerable rotary tool	9.64	7.625	4	148.8
Stabilizer	1.86	7.625	3	144.8
MWD	10	6.75	3	80.1
Stabilizer	1.86	7.625	2.875	144.8
Drill collar	18.32	6.438	2.875	135.4
Jar	9.42	6.25	2.75	186
Drill collar	9.16	6.313	2.875	135.4
HW drillpipe	99.9	5	3	71.4
Drillpipe	2223	5	4.276	29

Figure 13: Drillstring page, drillstring assembly is shown after selection from the dropdown

The Drillstring page table (Figure 13) has a similar layout as in OpenLab. Drilling BHAs are available for every hole section separately, but the same ’two-casing-strings-need-to-be-present’ rule applies here too. If more than one drilling BHA was used in a certain section, the one with the longest distance drilled was chosen. The user can select from the available options by using a dropdown.

The Geopressures page’s graph shows the drilling window, formation pressure is marked with purple and fracture pressure with bluish-green. The plot displays information on hover, such as TVD, pore pressure, fracture pressure and pressure window expressed in standard gravity. Pressure values and the corresponding depths are displayed in the right-side table (Figure 14).


Openlab app

Not secure | openlab.herokuapp.com/geopressures

Wells Hole section Wellpath Fluid Drillstring Geopressures Geothermal OpenLab

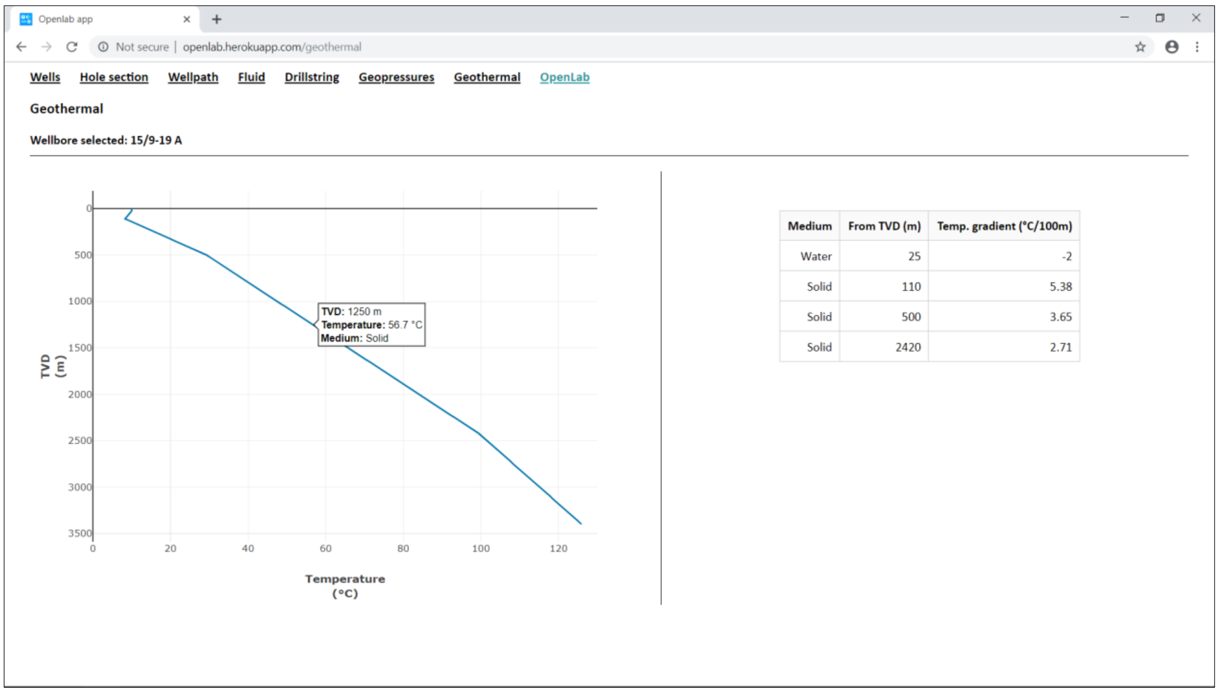
Geopressures

Wellbore selected: 15/9-19 A

TVD (m)	Pore pressure (s.g.)	Fracture pressure (s.g.)
149.5	1.032	1.033
154.1	1.032	1.054
158.2	1.032	1.074
162.8	1.032	1.094
163.8	1.032	1.115
168.2	1.032	1.135
175.3	1.032	1.156
184.1	1.032	1.176
192.5	1.032	1.196
201.5	1.032	1.217
210.4	1.032	1.238
218.4	1.032	1.258

Figure 14: Geopressures page with the drilling window shown on the left

The Geothermal page (Figure 15) consists of a plot showing the temperature profile of the well, and a table holding depth-specified geothermal gradients. Zero TVD refers to rotary kelly bushing elevation, an arbitrary value of 10 °C was chosen for this point, while a standard value of -2 °C is used as the geothermal gradient between sea level and seabed.


Openlab app

Not secure | openlab.herokuapp.com/geothermal

Wells Hole section Wellpath Fluid Drillstring Geopressures Geothermal OpenLab

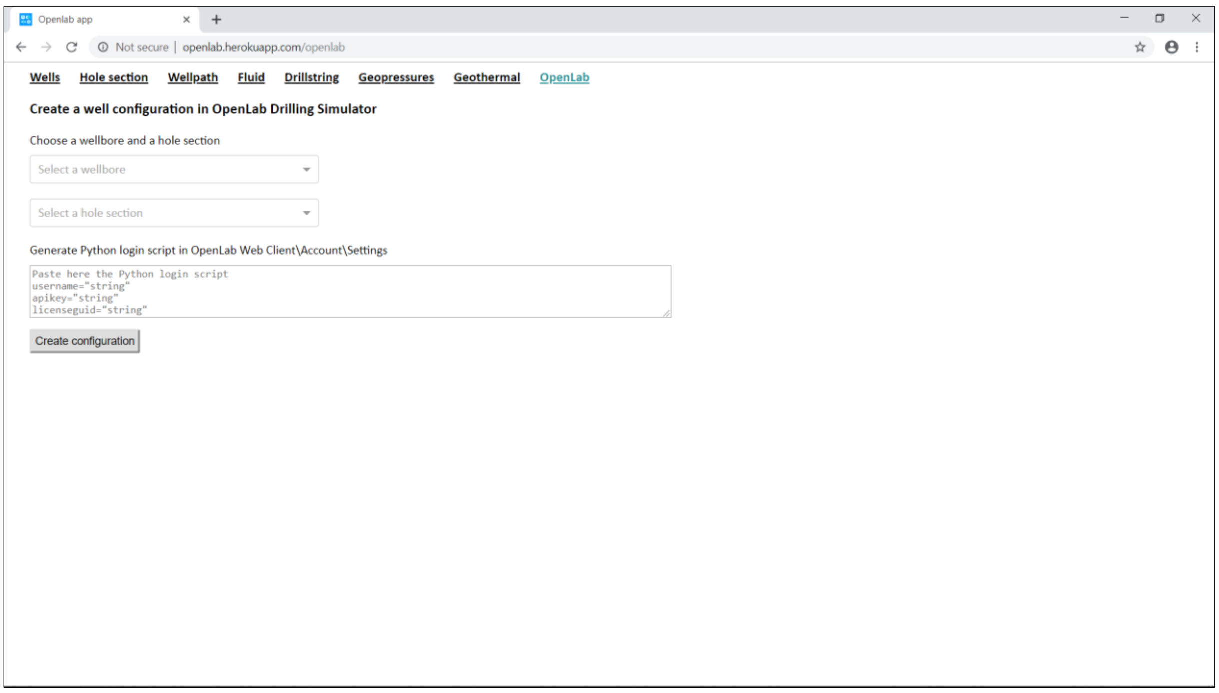
Geothermal

Wellbore selected: 15/9-19 A

Medium	From TVD (m)	Temp. gradient (°C/100m)
Water	25	-2
Solid	110	5.38
Solid	500	3.65
Solid	2420	2.71

Figure 15: Temperature profile and geothermal gradients on the Geothermal page

It is the OpenLab page (Figure 16) that integrates the application into the OpenLab drilling simulator. By running the OpenLab Python client in the background, the user is able to create a configuration directly in the simulator. The process is the following: after the user selected a well, chose a hole section and copy&pasted the required Python login script, a new configuration is created in OpenLab by clicking the Create configuration button. The Python login script is generated in the OpenLab drilling simulator by clicking on the account icon, then choosing generate Python login script in the Settings menu. After pressing the Create configuration button, the web browser's tab displays the text Updating..., then in a few seconds the new configuration is available in the user's OpenLab account.



Create a well configuration in OpenLab Drilling Simulator

Choose a wellbore and a hole section

Generate Python login script in OpenLab Web Client\Account\Settings

```
Paste here the Python login script
username="string"
apikey="string"
licenseguid="string"
```

Figure 16: OpenLab page integrates the app into the OpenLab drilling simulator

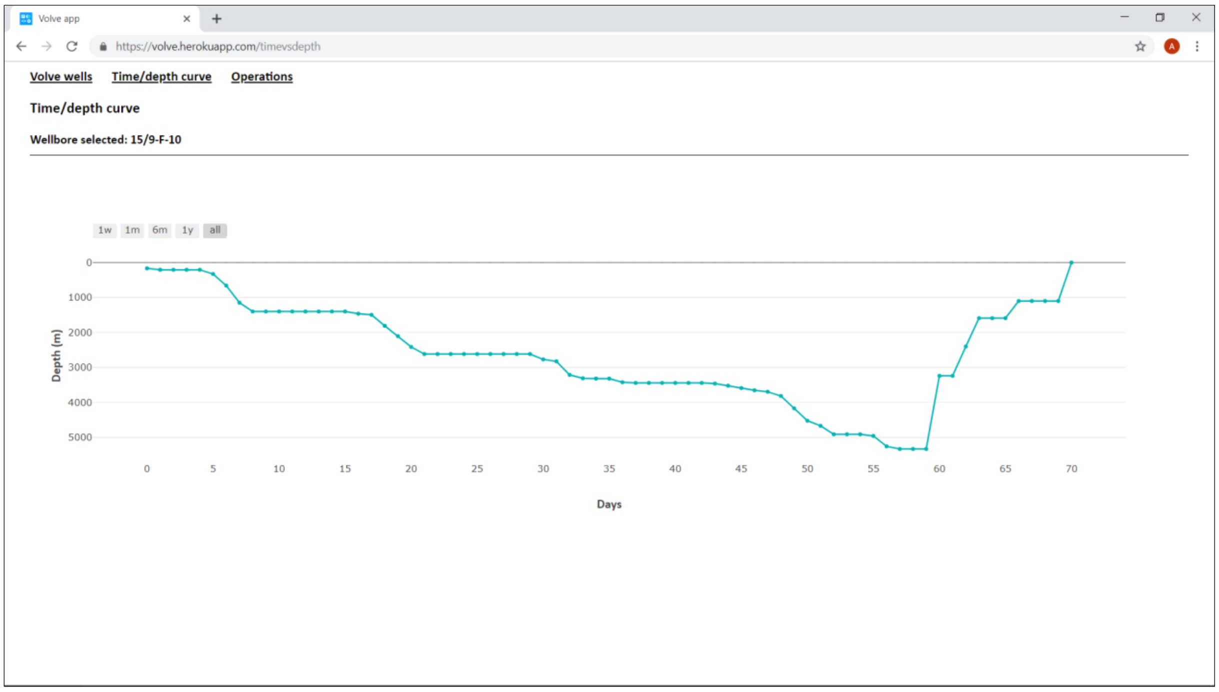
Configurations are named according to this format: wellbore name followed by the hole section size, for instance 15_9_19_A_section_8_5, where 15_9_19_A is the wellbore name and section_8_5 refers to the hole section size in inches. Hole size in a configuration name means that the drilling of that specific hole section can be simulated. In the above example the configuration which is created in the simulator represents the following case: the 9 $^{5}/_{8}$ inches casing is run and cemented, the casing shoe is drilled out, 5 meters are drilled ahead in the new formation and drilling BHA with the 8 $^{1}/_{2}$ inches bit is run in the hole. The same logic applies for the other configurations as well.

Rig technical parameters are not specified for the configurations. Default values provided by the OpenLab drilling simulator are used instead.

5 DDR Data Management and Visualization App

5.1 Introduction

The idea of this application has emerged while experimenting with data extraction from the DDRs of the Volve dataset. As mentioned earlier, the standardized structure and content of these reports made them easily suitable for automatic data extraction. Combined with interactive visualization, the end result is a powerful tool, which offers a simple environment for data management for all 25 wellbores that was drilled in the Volve field. The same Python libraries were used here as in case of the previously described Volve-OpenLab application. The data source is the XML DDRs in the Volve dataset, the program directly extracts the required information and visualizes them by the means of interactive plots. The Python code to construct the application is included in Appendix E.

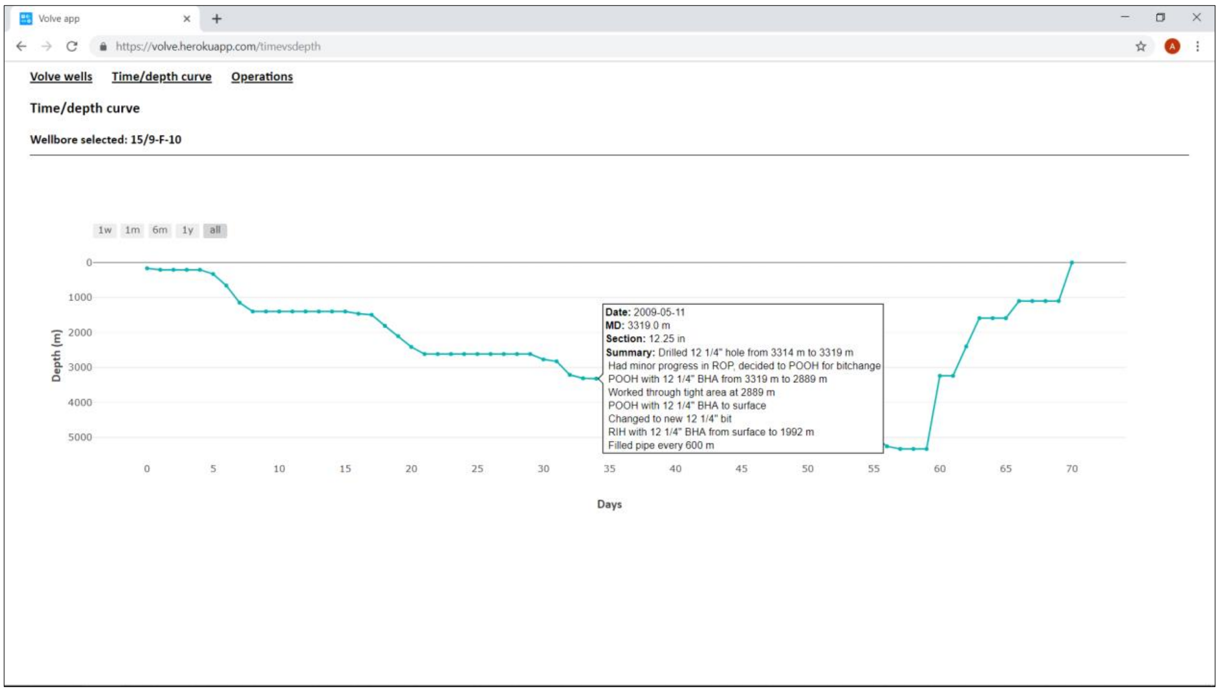


Time/depth curve

Wellbore selected: 15/9-F-10

Figure 17: Time/depth curve module in the Volve-DDR app

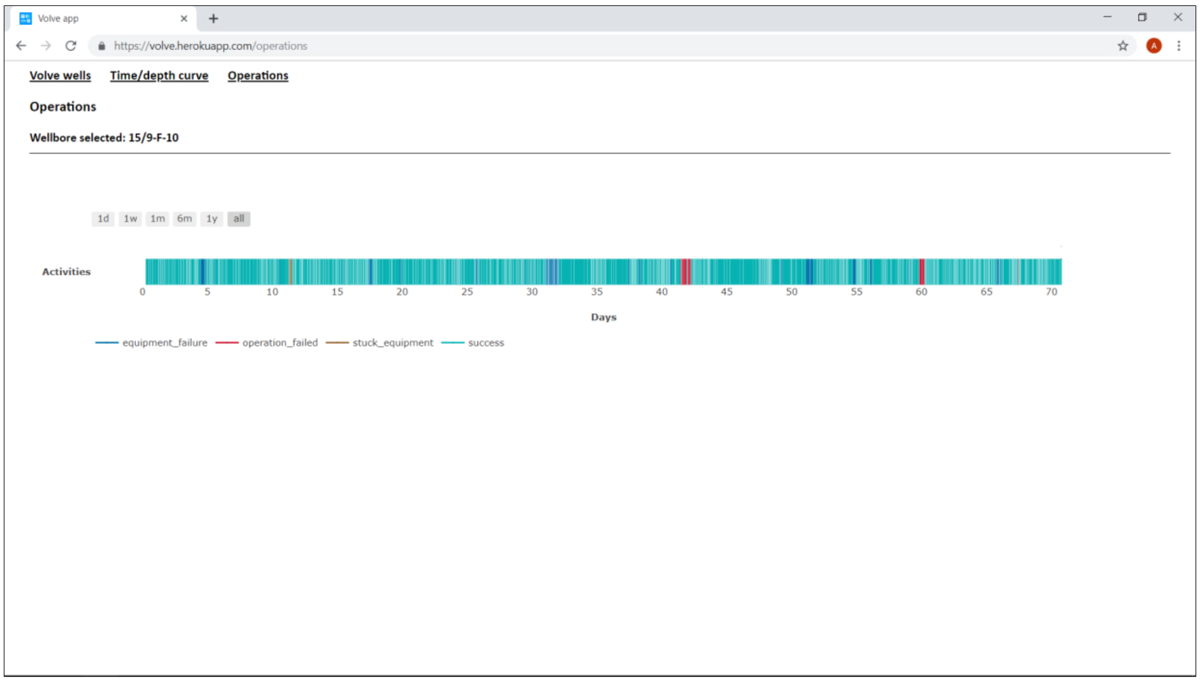
The opening screen of the app is identical to the Wells page in the Volve-OpenLab app (Figure 9), it displays general information about the Volve wells and enables the user to select a wellbore using a dropdown. The current version of the application consists of two modules: an interactive time versus depth curve and an interactive operation timeline. Time/depth curves are standard features in every well report and offer a convenient way to visualize the drilling progress. The vertical axis represents the depth while the horizontal axis shows the drilling days. The problem with these static plots is that they show only a limited amount of information due to the restricted space that is available. By turning them interactive the only limiting factor is data availability. This difference is shown in Figure 17 and Figure 18. The time/depth curve for the 15/9-F10 well is shown here. It was an observation well and plugged back immediately after the planned depth was reached. This process can be examined in Figure 17, but the real advantage of the app is showing extra information on hover (Figure 18), such as date, measured depth, section drilled and 24 hours summary of the operational activities. This information is extracted from the `statusInfo` elements of the DDRs. The user can zoom-in by drag-and-drop, zoom-out by double click and narrow the displayed time interval by using the range selectors (1 week, 1 month, 6 months, 1 year, show all) on the upper left side of the graph. When zoomed-in, the vertical axis serves as a range slider.



Date: 2009-05-11
MD: 3319.0 m
Section: 12.25 in
Summary: Drilled 12 1/4" hole from 3314 m to 3319 m
Had minor progress in ROP, decided to POOH for bitchange
POOH with 12 1/4" BHA from 3319 m to 2889 m
Worked through tight area at 2889 m
POOH with 12 1/4" BHA to surface
Changed to new 12 1/4" bit
RIH with 12 1/4" BHA from surface to 1992 m
Filled pipe every 600 m

Figure 18: Time/depth curve displays information on hover separately for each day

The operation timeline module comprises an interactive timeline that shows the detailed breakdown of the operational activities (Figure 19). The data that is needed to construct this plot is extracted from the `activity` elements of the DDRs. The activities are colored according to the state of the activity, which differentiates between productive and non-productive time. Categories used here are success for productive time, mud loss & circulation loss, equipment failure, injury, operation failed and stuck equipment for NPT. Hover info consists of start and end time, duration, measured depth, operation phase, state of the activity and a comment that gives a short summary of the activity. The interactive features (zooming, sliding) are similar to the ones already presented for the time/depth curve.

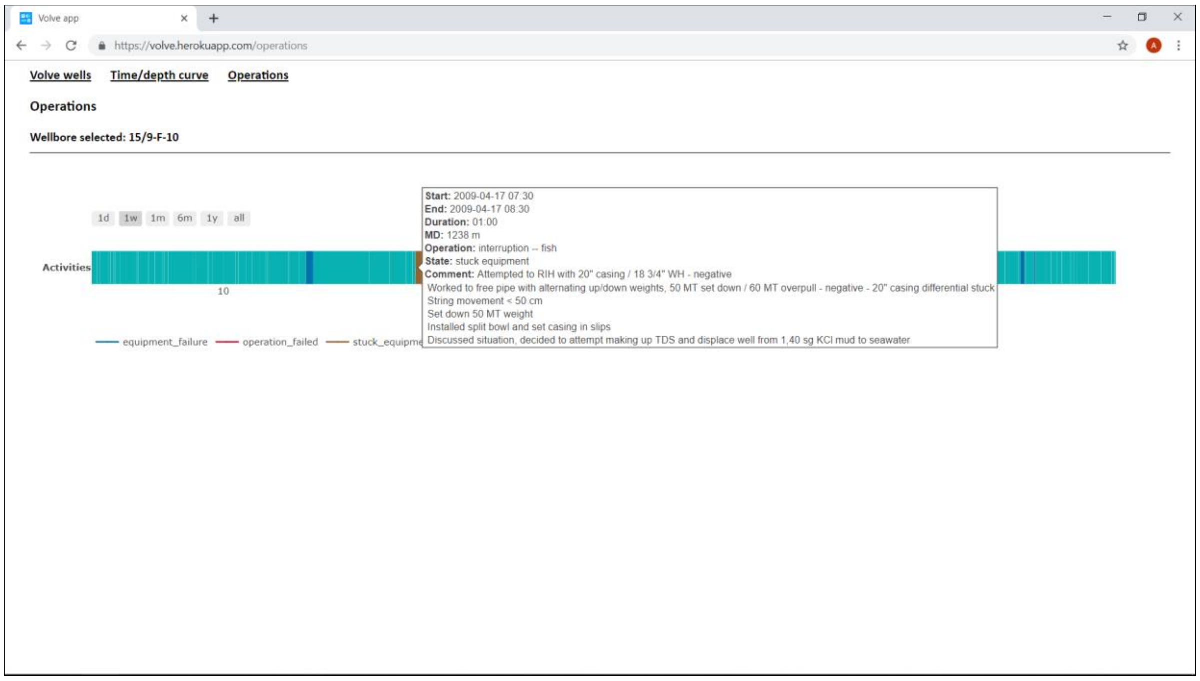


Operations

Wellbore selected: 15/9-F-10

Figure 19: Operation timeline displays the breakdown of the operational activities

By using the range selector and range slider, it becomes simple to get an overview of the operations and identify various drilling- and equipment-related problems (Figure 20). This feature could facilitate the investigation of drilling anomalies by identifying best practices to solve these problems and overcome the challenges arose. A use case will be presented in the next chapter.



Operations

Wellbore selected: 15/9-F-10

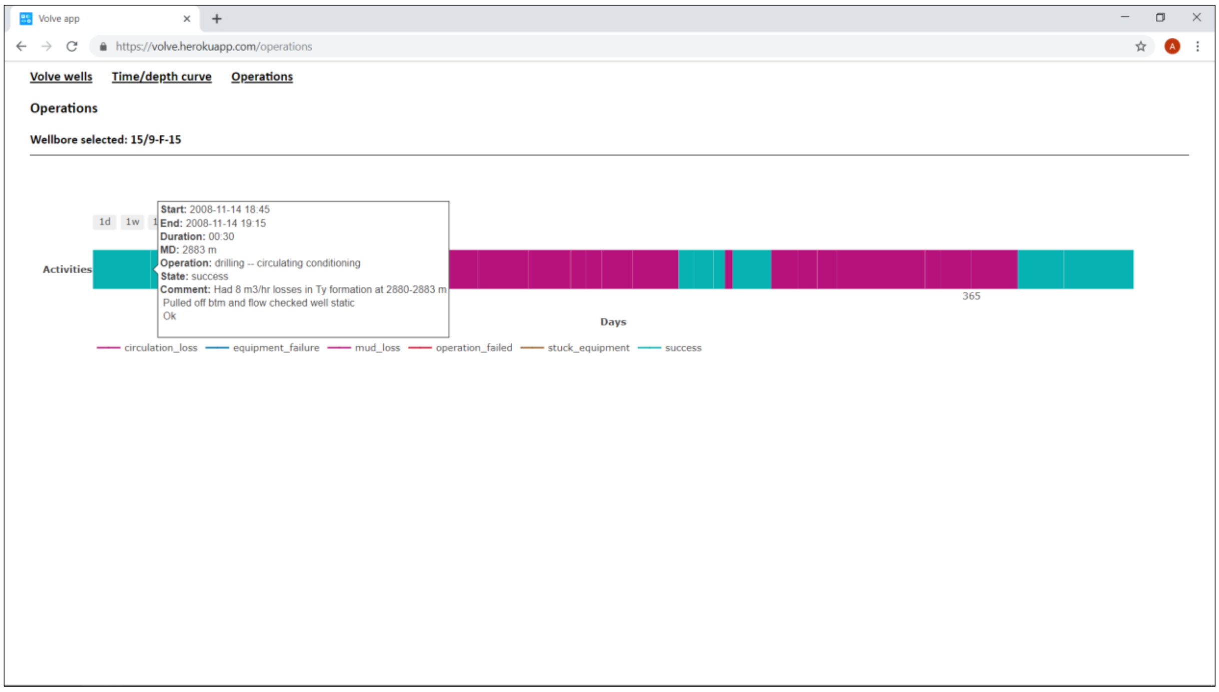
Start: 2009-04-17 07:30
End: 2009-04-17 08:30
Duration: 01:00
MD: 1238 m
Operation: interruption -- fish
State: stuck equipment
Comment: Attempted to RIH with 20" casing / 18 3/4" WH - negative
Worked to free pipe with alternating up/down weights, 50 MT set down / 60 MT overpull - negative - 20" casing differential stuck
String movement < 50 cm
Set down 50 MT weight
Installed split bowl and set casing in slips
Discussed situation, decided to attempt making up TDS and displace well from 1,40 sg KCl mud to seawater

Figure 20: Range selector is set to one week, hover info shows information on a stuck pipe incident

There are some issues that needs to be handled during data extraction from the DDRs. In the current version of the app, missing data is dealt with adding a zero to the array in case of integers (e.g. MD) and a dash (-) character in case of strings (e.g. 24 hours summary). It may lead to outlying data points in the time/depth curve. Moreover, 24 hours summaries and comments are extracted as strings, splitted at dot (.) characters and line breaks are inserted. If the dot character does not refer to a full stop, but stands for a decimal separator or a period in an acronym, the resulted text in the hover box may be a little confusing, but readability is maintained. Fortunately comma (,) is the default decimal separator in the majority of the drilling reports in the Volve dataset.

5.2 Circulation Loss Use Case

Heavy fluid losses were encountered while drilling the 15/9-F15 well (Figure 21). It happened when penetrating through the Ty formation between 2880 and 2915 meters. It is indicated with purple color on the operation timeline. By scrolling through this time interval and examining the comments in the hover box, the remedial actions and their effect on the loss rate can be easily checked. The mud loss started at 18:45 14.11.2008 and the drilling crew regained control by 03:00 17.11.2008, so it caused more than two days of NPT, shedding light on the importance of appropriately identifying best practices to overcome such challenges.



Operations

Wellbore selected: 15/9-F-15

Start: 2008-11-14 18:45
End: 2008-11-14 19:15
Duration: 00:30
MD: 2883 m
Operation: drilling -- circulating conditioning
State: success
Comment: Had 8 m3/hr losses in Ty formation at 2880-2883 m
Pulled off btm and flow checked well static
Ok

Figure 21: Fluid loss incident marked with purple on the timeline. The drilling of this well was suspended several times, and the days axis shows the total number of days that passed since spudding the well

The main steps to prevent, mitigate and control this fluid loss incident are listed below (all quotes are as per the drilling reports, only spelling mistakes and grammar errors are corrected):

- Drilled 8 $^1/_2$" hole from 2591 m to 2883 m. Parameters: Flow 2200 lpm / SPP 241-246 bar / String RPM 80 / Bit RPM 209 / TQ 12-15 kNm / ECD 1.46-1.47 EMW / ROP 20-45 m/hr Mud weight 1.35 SG. Started adding LCM according to plan 30 m above Ty formation.
- Had 8 m3/hr losses in Ty formation at 2880-2883 m. Pulled off bottom and flow checked well static. Ok.
- Staged up pumps to establish loss free rate. Meanwhile continued adding LCM chemicals to active fluid and prepared 150 kg/m^3 LCM pill in pit according contingency plan. Monitored losses while pumping and reciprocating pipe, Flow 440 lpm / SPP 25 bar / String RPM 10 / TQ 9-10 kNm. Average loss rate 1000 ltrs/hr.
- Lined up and pumped 5 m^3 LCM pill at 440 lpm. Spotted pill on TD at 2883 m. Pulled back one stand to 2843 m while pumping the LCM pill out of the BHA.
- Let LCM pill settle/soak. Monitored well on trip tank and rotated string with 5 rpm. No losses observed.
- RIH with 8 $^1/_2$" BHA to TD at 2883 m. Circulated bottoms up at 440 lpm while monitoring for losses. Increased flow rate in steps of 200 lpm to 1200 lpm. Had losses 1-1.5 m^3/hr. Increased flow rate to 1400 lpm and had losses 6 m^3/hr.
- Reduced flow rate to 600 lpm and monitored losses of 1 m^3/hr while preparing second LCM pill with 200 kg/m^3. Reciprocated string.
- RIH with 8 $^1/_2$" BHA to TD at 2883 m. Lined up and pumped 5 m^3 LCM pill at 600 lpm.
- Continued pumping 5 m^3 LCM pill at 600 lpm at TD. Pulled off bottom and flow checked well. Static.
- POOH with 8 $^1/_2$" BHA from 2842 m to 2735 m (above top of pill). Circulated, Flow 600 lpm / SPP 34 bar / String RPM 5 / TQ 8 kNm.
- Pumped 4 m^3 200 kg/m^3 LCM pill, Flow 600 lpm / SPP 34 bar / String RPM 5 / TQ 8 kNm. Spotted pill on top of previous pill to cover all of Ty formation.
- POOH with 8 $^1/_2$" BHA from 2735 m to 2670 m (above top of Ty formation). Staged up pumps to 1330 lpm and had 1 m^3/hr losses that gradually was reduced down to zero.
- Circulated and reciprocated pipe from 2670 m to 2641 m , Flow 1000 lpm / SPP 66 bar / String RPM 5 / TQ 5-7 kNm. Prepared for reducing mud weight down to 1.30 SG. Total loss since starting to loose mud on this section was approx 19 m^3.
- Bled into active to reduce mud weight from 1.35 SG to 1.30 SG. Meanwhile rotated and reciprocated pipe from 2670 m to 2641 m , Flow 1000 lpm / SPP 66 bar / String RPM 5 / TQ 5-7 kNm.

- Staged up flow rate in 200 lpm increments to 1800 lpm above top LCM pill and monitored for losses. Had 1 m^3/hr loss at 1600 lpm and 0.5 m^3/hr loss at 1800 lpm.
- Lined up to trip tank and RIH from 2670 m to TD at 2883 m. Broke circulation and staged up flow rate to 600 lpm whilst displacing out LCM pill to ship. Had 0.5 m^3/hr losses.
- Staged up flow rate in 200 lpm increments to 1000 lpm while continuing to displace out LCM pill. Had 0.5 m^3/hr losses at 1000 lpm.
- Staged up flow rate in 200 lpm increments from 1000 lpm to 1800 lpm. Meanwhile rotated and reciprocated pipe from 2883 m to 2855 m. Had 0.5-1.0 m^3/hr losses at all flow rates. Up weight 152 MT / Down weight 127 MT / Free rotation weight 138 MT (10 rpm).
- Made connection. Staged up pump rate slowly to 1800 lpm. Started drilling 8 $^1/_2$" hole at 2883 m. Parameters: Flow 1800 lpm / SPP 166 bar / String RPM 80 / Bit RPM 218 / TQ 10-12 kNm / ROP 10 m/hr. MUD weight 1.30 SG.
- Drilled 8 $^1/_2$" hole from 2883 m to 2893 m. Parameters: Flow 1800 lpm / SPP 167 bar / String RPM 80 / Bit RPM 218 / TQ 10-16 kNm / 1-2 MT WOB / ECD 1.38 EMW / ROP 10 m/hr. MUD weight 1.30 SG.
- Stopped drilling due to 6 m^3 losses/10 min. Discovered loss to be at shakers while changing screens. Changing screens was not communicated to drill floor.
- Staged up pump rate slowly to 1800 lpm to start drilling ahead. Drilled 8 $^1/_2$" hole from 2893 m to 2915. Parameters: Flow 1800 lpm / SPP 164 bar / String RPM 80 / Bit RPM 218 / TQ 11-13 kNm / 2-5 MT WOB / ECD 1.38 EMW / ROP 10 m/hr. MUD weight 1.30 SG.
- Pick-up off bottom and reduced flow rate due to heavy losses of 36 m^3/hr. Circulated at 450 lpm and staged up to find loss free rate at Flow 550 lpm / SPP 23 bar / String RPM 5 / TQ 5-6 kNm.
- Pumped 5 m^3 300 kg/m^3 LCM pill at 550 lpm and spotted on bottom/TD.
- POOH with 8 $^1/_2$" BHA from 2915 m to 2775 m (above top of LCM pill).
- Staged up pumps in 200 lpm increments to 1800 lpm above top LCM pill at 2775 m. Had 0.7 m^3/hr losses at 1800 lpm.
- Lined up to trip tank. Flow checked well. Static. RIH from 2775 m to TD at 2915 m.
- Broke circulation at 2914 m and staged up pumps in 200 lpm increments to 1000 lpm while rotating pipe.
- Continued staging up pumps in 200 lpm increments from 1000 lpm to 1800 lpm while rotating and reciprocating pipe slowly. Had 0.5 m^3/hr losses at 1800 lpm.

Despite adding LCM and spotting several LCM pills along the problematic Ty formation, the loss of mud could not be stopped, but eventually the loss rate could be minimized at an acceptable rate of 0.5 m^3/hour. This made it possible to drill further and reach the planned depth without encountering additional drilling related problems. The information provided above could be used for offset wells, either during the well planning phase or for operations when a similar incident occurs. In order to use this experience as an operational guideline during drilling, the neighboring wells should be analyzed too. The result would be a so-called best practice guideline for fluid loss incidences. It would contain the recommended flow rate, type and volume of LCM material added, volume of LCM pill spotted, required soaking time separately for each formation that carries the risk of mud loss. Similar guidelines could be compiled for stuck pipe and well control incidences.

6 Conclusions and Future Work

6.1 Conclusions

The overall objective of this thesis has been to develop an integrated web application for the OpenLab drilling simulator, in order to enable the user to simulate the drilling of the Volve wellbores as close to the real scenarios as possible. A potential use case is the benchmarking of well flow and torque & drag models against field data. The application also provides a useful online tool to get an overview of the Volve wells with interactive visualization, so the opportunities given are not limited to OpenLab. The Volve dataset's drilling related features have been used to achieve this goal. Multiple data sources have been investigated and several data mining challenges have arisen through this process. The Final Well Reports proved to be the best source to collect all the necessary data that are essential to create configurations in OpenLab. After the collected data have been analyzed and processed, the new web application has been designed and developed in Python. By utilizing the functionality of the OpenLab Python client, the integration of the newly developed app into the OpenLab drilling simulator has been achieved without any problem. The resulted app is deployed to https://openlab.herokuapp.com and hopefully will be added to the official OpenLab features in the foreseeable future.

The development of another application is started as a side-project. This app is based on automated data extraction from daily drilling reports, and offers a powerful tool for data management and visualization with hugh potential for further development. As the data mining scripts are developed to be compatible with the widely used WITSML standards, the usage of this app is not limited to the Volve daily drilling reports. Currently the app consists of two modules: an interactive time versus depth curve and an interactive operation timeline. The app can be accessed at https://volve.herokuapp.com.

Moreover, the Landmark database in the Volve dataset has been thoroughly explored, its structure and content together with possible data extraction methods have been described. It can facilitate the use of this data source for future research.

To conclude, this thesis have tried to prove the advantage of industrial data sharing combined with free softwares for educational purposes. Referring to the quote from Equinor's COO Jannicke Nilsson, the Volve dataset offers students the opportunity to set off for a digital journey which promises a steep learning curve, ultimately benefiting not only the academia, but also the industry as a whole.

6.2 Future Work

The application developed for the OpenLab drilling simulator comprises only static properties, such as wellbore structure, wellpath, drillstring assembly, geopressures and geothermal gradients. These variables are assumed being independent of time. Even fluid properties, for instance density and rheology parameters are taken constant for a given hole section. In order to enhance the quality of the simulations, real-time operational parameters, such as flow rate, rate of penetration and surface RPM should be considered. To achieve this goal, a web application specifically designed to visualize and display real-time drilling data should be developed. The Volve dataset's drilling related real-time features could be the main data source, but the development should focus on making the app compatible with other third-party databases. Ultimately the app would enable the import of real-time data series into OpenLab, facilitating simulations in sequence mode.

Regarding the daily drilling report application for data management and visualization, the future development should focus on adding new modules to the existing ones. As it was discussed earlier in Chapter 3.3.3, daily drilling reports represent a consistent data source that provides information about all the important operational activities and parameters. Therefore the app can be further developed by extending its modular architecture, the opportunitues are limited only by the available data. Possible extensions could be:

- Fluid module: provides information about the applied drilling and completion fluid properties on an interactive timeline.
- Equipment failure module: shows when an equpment has failed together with the caused non-productive time.
- Bit run module: bit records can be tracked by displaying IADC classification code, IADC dull grading, total distance drilled, etc.

In addition to the above, the data extraction scripts should be improved in order to better cope with challenges due to missing data. Natural language processing could be used to deal with spelling mistakes and grammar errors in input text, such as activity comments and 24 hours summary. It was suggested in Chapter 5.2 that so-called best-practice-guidelines could be created. Probably it is not a straightforward process and would require the involvement of machine leraning/statistical learning methods. It is definitely a challenging, but also very promising topic for a future thesis work.

References

Cayeux, E., Daireaux, B., Dvergsnes, E. W., & Florence, F. (2014). Toward Drilling Automation: On the Necessity of Using Sensors That Relate to Physical Models. *SPE Drilling & Completion*, 236-255.

Cayeux, E., Mesagan, T., Tanripada, S., Zidan, M., & Fjelde, K. K. (2014). Real-Time Evaluation of Hole Cleaning Condidtions with a Transient Cuttings-Transport Model. *Drilling and Completion Journal*.

Corre, B., Eymard, R., & Gounet, A. (1984). Numerical computation of Temperature Distribution in a wellbore while drilling. *SPE Annual Technical Conference and Exhibition.* Houston: SPE.

Energistics. (2019, May 15). Retrieved from Energistics website: https://www.energistics.org/portfolio/witsml-data-standards/

Equinor. (2018, June 14). Retrieved from Equinor website: https://www.equinor.com/en/news/14jun2018-disclosing-volve-data.html

Exebenus. (2019, February 8). Retrieved from Exebenus website: http://www.exebenus.com/2019/02/08/exebenus-pulse-successfully-recognizes-non-drilling-operation-procedures-using-the-norce-openlab-drilling-simulator/

Giese, M., Ornas, J. I., Overå, L., Svensson, I., & Waaler, A. (2012). Using Semantic Technology to Auto-generate Reports: Case Study of Daily Drilling Reports. *SPE Intelligent Energy International.* Utrecht.

JPT. (2019, March 25). Retrieved from Society of Petroleum Engineers website: https://www.spe.org/en/jpt/jpt-article-detail/?art=5282&

Kjeldahl, M. (2019, April 14). Retrieved from Oliasoft Community website: https://wpforum.oliasoft.com/log-in-equinor-statoils-volve-dataset-well-technical-data-part-1/

Kyllingstad, A. (1995). Buckling of Tubular Strings in Curved Wells. *Journal of Petroleum Science and Engineering*, 209-218.

Lorentzen, R. J., & Fjelde, K. K. (2005). Use of slopelimiter techniques in traditional numerical methods for multi-phase flow in pipelines and wells. *International Journal for Numerical Methods in Fluids*, 723-745.

Lorentzen, R. J., Nævdal, G., Karlsen, H. A., & Skaug, H. J. (2014). Estimation of Production Rates With Transient Well-Flow Modeling and the Auxiliary Particle Filter. *SPE Journal*, 172-180.

Lundin. (2018, September 28). Retrieved from Lundin Norway website: https://www.lundin-norway.no/2018/09/28/real-time-production-data-to-be-shared-between-edvard-grieg-and-ivar-aasen/?lang=en

Noshi, C. I., & Schubert, J. J. (2018). The Role of Machine Learning in Drilling Operations; A Review. *SPE Eastern Regional Meeting.* Pittsburgh: Society of Petroleum Engineers.

NPD. (2019, May 13). *NPD FactPages*. Retrieved from NPD website: http://factpages.npd.no/factpages/

OpenLab Drilling. (2019, May 12). Retrieved from OpenLab Drilling website: https://openlab.app/product/

PCA. (2019, May 15). Retrieved from POSC Caesar Association website: http://data.posccaesar.org/rdl/

PSA. (2019, May 15). Retrieved from Petroleum Safety Authority website: https://www.ptil.no/en/contact-us/reporting-to-the-psa/drilling-reports-ddrs/

UiS. (2019, March 25). Retrieved from University of Stavanger website: https://www.uis.no/research/national-ior-centre-of-norway/shares-all-data-from-one-field-article132284-13152.html

Volve Data Village. (2019, May 22). Retrieved from Equinor Data Portal: https://data.equinor.com/dataset/Volve

Yunfeng, Y., & al. (2004). An Advanced Coiled Tubing Simulator for Calculations of Mechanical and Flow Effects; Model Advancements and Full-Scale Verification Experiments. *SPE Coiled Tubing Conference and Exhibition.* Houston: SPE.

Appendices

Appendix A

Lists of available graphs in OpenLab simulations:

Time-based Graphs	Depth-based Graphs
Back pressure	Cuttings bed
Bit depth and total depth	Cuttings transport
Bit pressure	Density
BOP pressure	Gas volume
Bottom hole pressure	ECD
Choke opening	Mud velocity
Flow rate	Temperature
Gas flow rate	Tension
Hook load	Torque
In slips	
Mud temperature	
Pit densities	
Pit temperatures	
Pit volumes	
Reservoir flow	
ROP	
Stand pipe pressure	
String and hook position	
String and hook speed	
Surface RPM	
Surface torque	
WOB	

Appendix B

List of tags in the Landmark EDM database, the numbers refer to their corresponding locations as line numbers in the database.

```
TEMPERATURE DERATION SCHEDULE 6 - 25
TEMPERATURE DERATION POINT 26 - 59
LITHOLOGY CLASS 60 - 62
CD MATERIAL 63 - 83
CD GRADE 84 - 104
CD GRADE SECTION TYPE 105 - 145
CD CLASS 146
CD REAL TIME CONFIG 147 - 22646
CD CUSTOM BASE FLUID 22647 - 23616
CD GEO SYSTEM 23617
CD GEO ZONE 23618
GEO DATUM 23619
GEO ELLIPSOID 23620
CD SURVEY TOOL 23624 - 23653
DP TOOL TERM 23654 - 24489
CD WELLBORE TYPE 24490 - 24519
DP PROJECT TARGET 24527 - 24593
DP PROJECT TARGET POINT 24594 - 24693
CD WELL ALL 24696 - 24716
CD WELL 24717 - 24737
CD DATUM 24738 - 24861
CD WELLBORE 24862 - 24915
CD ASSEMBLY 24916 - 28817
CD ASSEMBLY COMP 28818 - 39467
CD BHA COMP BIT 39468 - 40099
CD BHA COMP DP HW 40100 - 42576
CD BHA COMP JAR 42577 - 43393
CD BHA COMP MOTOR 43394 - 44142
CD BHA COMP MWD 44143 - 45769
CD BHA COMP NOZZLE 45770 - 47327
CD BHA COMP STAB 47328 - 48645
CD WEQP PACKER 48646 - 48651
TU COMP TEMPERATURE DERATION POINT 48652 - 50052
CD CASE 50053 - 52672
CD CASE TEMP GRADIENT 52673 - 54604
CD NBK LEAK OFF TEST 54605 - 55046
CD NBK NOZZLES 55047 - 55048
CD NBK PUMP OUTPUT 55049
TU CASE ASSEMBLY PARAMETER 55050 - 98540
TU CUSTOM LOAD PROFILE 98541 - 101604
TU LOAD HEADER 101605 - 101641
TU EPP PARAMETERS 101642 - 101749
TU LOAD PARAMETERS 101750 - 101929
TU LOAD PROFILE 101930 - 167604
WP BHA PARAMS 167605 - 168552
WP CASE ANNOTATIONS 168553
WP CASE BOOSTER PUMP 168554 - 169507
WP CASE CEMENT FOAM SCHD 169508 - 170455
WP CASE CEMENT JOB DATA 170456 - 171408
WP CASE CEMENT JOB DATA ARR 171409 - 172392
WP CASE CEMENT FOAM DATA ARR 172393 - 172409
WP CASE CEMENT JOB DATA PRESSURE 172410 - 173357
WP CASE CENTRALIZER SCHD 173358 - 174310
WP CASE CENTRALIZER SCHD INT 174311 - 174315
WP CASE CIRC SYSTEM 174316 - 175268
WP CASE CSA BOUNDARY COND 175269 - 175270
WP CASE CSA PARAMS 175271 - 176218
WP CASE PUMP 176219 - 176807
WP CASE PUMP SLOW 176808
WP CASE RANGE 176809 - 176811
WP CASE SOIL PROP 176812
WP CASE SURFACE 176813 - 177765
WP CASE WBSIMULATOR ANAL 177766 - 178713
WP DYNAMIC PROPERTY HEADER 178714 - 178934
WP HYD OPTIONS 178935 - 179887
WP HYD PARAMS 179888 - 180840
WP KILL SHEET GEN 180841 - 181202
WP SRG PARAMS 181203 - 181218
WP SRG RECIPROCATION 181219 - 182171
WP SRG SURGE 182172 - 182188
WP SRG SWAB SURGE 182189 - 183141
WP TDA ACTLOAD 183142 - 183369
WP TDA ANNULUS FLUID GRAD 183370 - 183400
WP TDA DISCRETE PARAMS 183401 - 183417
WP TDA DRAGCHART 183418 - 201571
WP TDA DRAGCHART COF 201572 - 203239
WP TDA DRAGCHART MAN COF 203240 - 203638
WP TDA DRAGCHART SENS 203639 - 205430
WP TDA FRD 205431 - 205996
WP TDA FRD FORCE 205997 - 206745
WP TDA FRD ALL 206746 - 207311
WP TDA FRD ALL FORCE 207312 - 208060
WP TDA MANUAL COF 208061 - 208086
WP TDA OPTIONS 208087 - 209039
WP TDA PARAMS 209040 - 209992
WP TDA RISERLESS OUT ANN FLUID 209993 - 209995
WP TDA STRING RISERLESS FLUID GRAD 209996 - 209997
WP TDA STRING FLUID GRAD 209998 - 210056
WP UBD PARAMS 210057 - 210073
WP WCN PARAMS 210074 - 211026
CD DEFINITIVE SURVEY HEADER 211027 - 211136
CD DEFINITIVE SURVEY STATION 211137 - 226245
CD SURVEY PROGRAM 226246 - 226577
CD VERTICAL SECTION 226578 - 226685
DP ANTICOL 226686 - 228151
WP CASE TORT INT 228152 - 228157
CD FLUID 228159 - 228475
WP FLUID TEMP 228476 - 228634
WP FLUID TEMP FANN DATA 228635 - 229525
CD FORMATION INFLUX GROUP 229526 - 229533
CD FRAC GRADIENT GROUP 229534 - 229668
CD FRAC GRADIENT 229669 - 266925
CD HOLE SECTION GROUP 266926 - 267887
CD HOLE SECTION 267888 - 270628
CD PORE PRESSURE GROUP 270629 - 270763
CD PORE PRESSURE 270764 - 309847
CD SCENARIO 309848 - 309957
TU CASE PARAMETER 309958 - 310308
TU CASE USER PARAMETER 310309 - 310329
TU DLS OVERRIDE GROUP 310330 - 310367
TU DLS OVERRIDE 310368 - 310385
TU MMS APD 310386 - 310424
TU MMS APD DETAIL 310425 - 310568
TU ZONE PRESSURE GROUP 310569 - 310606
CD SURVEY HEADER 310607 - 310800
CD SURVEY STATION 310801 - 315753
CD TEMPERATURE GRAD GROUP 315754 - 315888
CD TEMPERATURE GRAD 315889 - 315918
CD WB REAL TIME CONFIG 315919 - 316143
CD WELLBORE FORMATION 316144 - 317411
DP MAGNETIC 317412 - 317465
CD PROJECT TARGET SITE LINK 317472 - 317538
CD PROJECT TARGET WB LINK 317539 - 317629
CD PROJECT TARGET WELL LINK 317630 - 317726
CD PROJECT TARGET SCENARIO LINK 317727 - 317893
CD SCENARIO FORMATION LINK 317894 - 318043
```

Python script to extract trajectory data from the Landmark EDM database. The same algorithm can be used to process information from other tags.

```
# import libraries
import xml.etree.ElementTree as et
import pandas as pd

# parse xml file
volve_tree = et.parse('Volve F.edm.xml')
volve_root = volve_tree.getroot()

# make a list of wellbore ids
wellbore_id = []

for child in volve_root:
    if child.tag == 'CD_DEFINITIVE_SURVEY_HEADER':
        if child.attrib['phase'] == 'ACTUAL':
            wellbore_id.append(child.attrib['wellbore_id'])

# make a list of corresponding wellbore names
wellbore_name = []

for child in volve_root:
    if child.tag == 'CD_WELLBORE':
        if child.attrib['wellbore_id'] in wellbore_id:
            wellbore_name.append(child.attrib['well_legal_name'].replace('/','-'))

# make a dictionary holding wellbore ids as keys and wellbore names as values
id_name_dict = dict(zip(wellbore_id, wellbore_name))

# create a function to extract trajectory data from the database
def get_wellpath(dct):

    survey_header = []

    # consider only actual wellbores
    for child in volve_root:
            if child.tag == 'CD_DEFINITIVE_SURVEY_HEADER':
                if child.attrib['phase'] == 'ACTUAL':
                    survey_header.append(child.attrib['def_survey_header_id'])

    # for loop to collect data for every wellbore in the dictionary that was
created earlier
    for key in dct:

        # append wellpath data to corresponding lists
        for item in survey_header:

            azimuth = []
            inclination = []
            md = []
            tvd = []
            easting = []
            northing = []

            for child in volve_root:
                if child.tag == 'CD_DEFINITIVE_SURVEY_STATION':
                    if child.attrib['wellbore_id'] == key:
                        if child.attrib['def_survey_header_id'] == item:
                            azimuth.append(float(child.attrib['azimuth']))
                            inclination.append(float(child.attrib['inclination']))
                            md.append(float(child.attrib['md']) * 0.3048)
                            tvd.append(float(child.attrib['tvd']) * 0.3048)
                            easting.append(float(child.attrib['offset_east']) *
0.3048)
                            northing.append(float(child.attrib['offset_north']) *
0.3048)

            # create a dataframe and save the dataframe as csv file
            if md:
                wellpath = pd.DataFrame(list(zip(md, azimuth, inclination)),
                                        columns = ['MD (m)','Azimuth (°)', 'Inc.
(°)'])
                wellpath.sort_values('MD (m)', inplace=True)
                wellpath = wellpath.reset_index(drop = True)
                wellpath = wellpath.round({'MD (m)': 0, 'Azimuth (°)': 2, 'Inc
(°)':2})
                wellpath.to_csv(f'{dct[key]}_wellpath.csv', sep = ';', index =
False)

# run function
get_wellpath(id_name_dict)
```

Appendix C

Complete list of activity codes in the Volve DDRs.

```
completion -- bop/wellhead equipment
completion -- circulating conditioning
completion -- completion string
completion -- other
completion -- perforate
completion -- sand control
completion -- stimulate
completion -- test scsssv
completion -- wire line
drilling -- bop activities
drilling -- bop/wellhead equipment
drilling -- casing
drilling -- circulating conditioning
drilling -- drill
drilling -- hole open
drilling -- other
drilling -- pressure detection
drilling -- ream
drilling -- survey
drilling -- trip
drilling -- wait
formation evaluation -- circulating conditioning
formation evaluation -- circulation samples
formation evaluation -- core
formation evaluation -- drill stem test
formation evaluation -- log
formation evaluation -- other
formation evaluation -- rft/fit
formation evaluation -- rig up/down
formation evaluation -- trip
formation evaluation -- wait
interruption -- fish
interruption -- lost circulation
interruption -- maintain
interruption -- other
interruption -- repair
interruption -- rig up/down
interruption -- sidetrack
interruption -- wait
interruption -- waiting on weather
interruption -- well control
moving -- anchor
moving -- position
moving -- skid
moving -- transit
plug abandon -- cement plug
plug abandon -- circulating conditioning
plug abandon -- cut
plug abandon -- equipment recovery
plug abandon -- mechanical plug
plug abandon -- mill
plug abandon -- other
plug abandon -- perforate
plug abandon -- squeeze
plug abandon -- trip
plug abandon -- wait
workover -- bop/wellhead equipment
workover -- completion string
workover -- other
workover -- perforate
workover -- rig up/down
workover -- test scsssv
workover -- wait
workover -- wire line
```

A sample script for automated data extraction from the Volve DDRs. This code collects data about the operational activities and returns them as a dataframe. By changing the strings before `elem.tag` and `subelem.tag` in the for loop, data from other entries can be extracted as well.

```python
# import libraries
import os
import xml.etree.ElementTree as et
import pandas as pd
from datetime import datetime

# create a function for automated data extraction
def get_operations(well):

    # location of the DDRs
    report_list = os.listdir('Reports')

    # define lists and dataframe structure
    start = []
    end = []
    md = []
    operation = []
    comment = []
    duration = []
    state = []

    df = pd.DataFrame(
        list(zip(
            start,
            end,
            md,
            duration,
            operation,
            comment,
            state
        )),
        columns = [
            'Start',
            'End',
            'MD (m)',
            'Duration',
            'Operation',
            'Comment',
            'State'
    ])

    os.chdir('Reports')

    # for loop to parse all the DDRs
    for file in report_list:
        if file[:-15] == well:
            report_tree = et.parse(file)
            report_root = report_tree.getroot()

            start = []
            end = []
            md = []
            operation = []
            comment = []
            duration = []
            state = []

            # append relevant values to the corresponding lists
            for child in report_root:
                for elem in child:
                    if elem.tag ==
'{http://www.witsml.org/schemas/1series}activity':
                        for subelem in elem:
                            if 'dTimStart' in subelem.tag:
                                start.append(datetime.strptime(subelem.text[:16],
'%Y-%m-%dT%H:%M'))

                                start_temp = datetime.strptime(subelem.text[:16],
'%Y-%m-%dT%H:%M')
                            elif 'dTimEnd' in subelem.tag:
                                end.append(datetime.strptime(subelem.text[:16],
'%Y-%m-%dT%H:%M'))

                                end_temp = datetime.strptime(subelem.text[:16],
'%Y-%m-%dT%H:%M')
                            elif 'md' in subelem.tag:
                                md.append(subelem.text)
                            elif 'proprietaryCode' in subelem.tag:
                                operation.append(subelem.text)
                            elif 'comments' in subelem.tag:
                                comment.append(subelem.text)
                            elif 'stateDetailActivity' in subelem.tag:
                                state.append(subelem.text)
                            else:
                                pass
                        duration.append((end_temp-start_temp))

            df = df.append(
                pd.DataFrame(
                    list(zip(
                        start,
                        end,
                        md,
                        duration,
                        operation,
                        comment,
                        state
                    )),
                    columns = [
                        'Start',
                        'End',
                        'MD (m)',
                        'Duration',
                        'Operation',
                        'Comment',
                        'State'
                    ]),
                ignore_index = True,
                sort = False
            )

    df.sort_values(['Start'], inplace=True)
    os.chdir('../')
    # return the dataframe
    return df
```

Appendix D

Python code for the Volve-OpenLab app. The same scripts together with documentation, requirements and data files are pushed to https://gitlab.com/a-nagy/volve-openlab-app. Here the Data folder holds CSV files with data that is used by the app to populate the graphs and tables, while the Configurations folder contains JSON files holding data needed to create configurations in OpenLab. The structure and content of these JSON files follow strict rules set by the OpenLab drilling simulator. As the web application itself is a multi-page app, the scripts are splitted into separate Python files to keep the editing and maintenance of the code simpler. The same file structure is presented here.

app.py

```
import dash

#external stylesheet for css styling
external_stylesheets = ['https://cdn.jsdelivr.net/gh/attilanagy1986/Dash-
css@master/undo.css']

#initiate Dash app
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
#title of the app
app.title = 'Openlab app'
#initiate server
server = app.server
#if set to False, Dash will raise an exception due to the multipage
structure of the app
app.config.suppress_callback_exceptions = True
```

index.py

```
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
```

```python
from app import app
import wells
import hole_section
import wellpath
import fluid
import drillstring
import geopressures
import geothermal
import open_lab

#wellbores for dropdown selection
wells_dict = {
            '15_9_19_A': '15/9-19 A',
            '15_9_19_B': '15/9-19 B',
            '15_9_19_BT2': '15/9-19 BT2',
            '15_9_19_S': '15/9-19 S',
            '15_9_19_SR': '15/9-19 SR',
            '15_9_F_1': '15/9-F-1',
            '15_9_F_1_A': '15/9-F-1 A',
            '15_9_F_1_B': '15/9-F-1 B',
            '15_9_F_1_C': '15/9-F-1 C',
            '15_9_F_4': '15/9-F-4',
            '15_9_F_5': '15/9-F-5',
            '15_9_F_7': '15/9-F-7',
            '15_9_F_9': '15/9-F-9',
            '15_9_F_9_A': '15/9-F-9 A',
            '15_9_F_10': '15/9-F-10',
            '15_9_F_11': '15/9-F-11',
            '15_9_F_11_A': '15/9-F11 A',
            '15_9_F_11_B': '15/9-F-11 B',
            '15_9_F_12': '15/9-F-12',
            '15_9_F_14': '15/9-F-14',
            '15_9_F_15': '15/9-F-15',
            '15_9_F_15_A': '15/9-F-15 A',
            '15_9_F_15_B': '15/9-F-15 B',
            '15_9_F_15_C': '15/9-F-15 C',
            '15_9_F_15_D': '15/9-F-15 D'
            }

#define app layout
app.layout = html.Div([
    dcc.Location(id='url'),
    dcc.Link(
            'Wells',
            href='/wells',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold'
                }),
    dcc.Link(
            'Hole section',
            href='/hole_section',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    dcc.Link(
            'Wellpath',
            href='/wellpath',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    dcc.Link(
            'Fluid',
            href='/fluid',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    dcc.Link(
            'Drillstring',
            href='/drillstring',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    dcc.Link(
            'Geopressures',
            href='/geopressures',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    dcc.Link(
            'Geothermal',
            href='/geothermal',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    dcc.Link(
            'OpenLab',
            href='/openlab',
            style={
                'color': 'rgb(43,151,155)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    html.Div([
        html.H3(['Select a wellbore'], style={'paddingBottom': '10px',
'font-weight': 'bold', 'border-bottom': '1px solid black'}),
        dcc.Dropdown(
            options=[{'label':value, 'value':key} for key, value in
wells_dict.items()],
            value='15_9_19_A',
            placeholder='Select a wellbore',
            id='wells-dropdown',
            style={'width':'50%'}
        ),
        html.Br(),
    ], id='external-page-wells', style={'paddingLeft':'25px'}),
    html.Div(id='page-content')
], style={'font-family': 'Calibri', 'paddingLeft':'25px',
'paddingRight':'25px'})

#callback to update page content
@app.callback(
    Output('page-content', 'children'),
    [Input('url', 'pathname')]
)
def populate_content(url):
    if url == '/wells':
        return wells.page_layout
    elif url == '/hole_section':
        return hole_section.page_layout
    elif url == '/wellpath':
        return wellpath.page_layout
    elif url == '/fluid':
        return fluid.page_layout
    elif url == '/drillstring':
        return drillstring.page_layout
    elif url == '/geopressures':
        return geopressures.page_layout
    elif url == '/geothermal':
        return geothermal.page_layout
    elif url == '/openlab':
        return open_lab.page_layout

#callback for mulltipage persistence of the wellbore dropdown
@app.callback(
    Output('external-page-wells', 'style'),
    [Input('url', 'pathname')]
)
def hide_external(url):
    if url == '/wells':
        return {'display': 'block'}
    else:
        return {'display': 'none'}

#callback to display the selected wellbore name
@app.callback(
    Output('dropdown-output', 'children'),
    [Input('wells-dropdown', 'value')]
)
def display_dropdown_contents(val):
    if val:
        return f'Wellbore selected: {wells_dict[val]}'

#run the app
if __name__ == '__main__':
    app.run_server()
```

wells.py

```
import dash_table
import dash_html_components as html
import pandas as pd

#read data from csv
df_wells = pd.read_csv('Data/volve_wells.csv', sep=';')

#define wells page layout and content
page_layout = html.Div([
    dash_table.DataTable(
        id='wells-table',
        columns=[{"name": i, "id": i} for i in df_wells.columns],
        data=df_wells.to_dict("rows"),
        style_header={'font-family': 'Calibri', 'font-weight': 'bold'},
        style_cell={'padding':'10px', 'font-family': 'Calibri', 'font-
size': '16px', 'width': '100px'}
    )], style={'width':'70%'})
```

hole_section.py

```
import dash_table
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import pandas as pd

from app import app

#function to create hole section plot
def section_plot(df):
    x=df['N/S (m)'].max()-df['N/S (m)'].min()
    y=df['E/W (m)'].max()-df['E/W (m)'].min()
    z=df['TVD (m RKB)'].max()-df['TVD (m RKB)'].min()
    trace1 = go.Scatter3d(
        x=df[df['Section (in)']==30]['N/S (m)'],
        y=df[df['Section (in)']==30]['E/W (m)'],
        z=df[df['Section (in)']==30]['TVD (m RKB)'],
        mode='lines',
        name='Cased hole',
        line=dict(
            color='rgb(5,133,133)',
            width=2*30
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==30].index],
        hoverinfo='text',
        showlegend=True
    )

    trace2 = go.Scatter3d(
        x=df[df['Section (in)']==20]['N/S (m)'],
        y=df[df['Section (in)']==20]['E/W (m)'],
        z=df[df['Section (in)']==20]['TVD (m RKB)'],
        mode='lines',
        name='Cased hole',
        line=dict(
            color='rgb(5,133,133)',
            width=2*20
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==20].index],
        hoverinfo='text',
        showlegend=False
    )

    trace3 = go.Scatter3d(
        x=df[df['Section (in)']==14]['N/S (m)'],
        y=df[df['Section (in)']==14]['E/W (m)'],
        z=df[df['Section (in)']==14]['TVD (m RKB)'],
        mode='lines',
        name='Cased hole',
        line=dict(
            color='rgb(5,133,133)',
            width=2*14
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==14].index],
        hoverinfo='text',
        showlegend=False
    )

    trace4 = go.Scatter3d(
        x=df[df['Section (in)']==13.375]['N/S (m)'],
        y=df[df['Section (in)']==13.375]['E/W (m)'],
        z=df[df['Section (in)']==13.375]['TVD (m RKB)'],
        mode='lines',
        name='Cased hole',
        line=dict(
            color='rgb(5,133,133)',
            width=2*13.375
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==13.375].index],
        hoverinfo='text',
        showlegend=False
    )

    trace5 = go.Scatter3d(
        x=df[df['Section (in)']==12.25]['N/S (m)'],
        y=df[df['Section (in)']==12.25]['E/W (m)'],
        z=df[df['Section (in)']==12.25]['TVD (m RKB)'],
        mode='lines',
        name='Open hole',
        line=dict(
            color='rgb(198,137,75)',
            width=2*12.25
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==12.25].index],
        hoverinfo='text',
        showlegend=True
    )

    trace6 = go.Scatter3d(
        x=df[df['Section (in)']==9.625]['N/S (m)'],
        y=df[df['Section (in)']==9.625]['E/W (m)'],
        z=df[df['Section (in)']==9.625]['TVD (m RKB)'],
        mode='lines',
        name='Cased hole',
        line=dict(
            color='rgb(5,133,133)',
            width=2*9.625
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==9.625].index],
        hoverinfo='text',
        showlegend=False
    )

    trace7 = go.Scatter3d(
        x=df[df['Section (in)']==8.5]['N/S (m)'],
        y=df[df['Section (in)']==8.5]['E/W (m)'],
        z=df[df['Section (in)']==8.5]['TVD (m RKB)'],
        mode='lines',
        name='Open hole',
        line=dict(
            color='rgb(198,137,75)',
            width=2*8.5
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==8.5].index],
        hoverinfo='text',
        showlegend=True
    )

    trace8 = go.Scatter3d(
        x=df[df['Section (in)']==7]['N/S (m)'],
        y=df[df['Section (in)']==7]['E/W (m)'],
        z=df[df['Section (in)']==7]['TVD (m RKB)'],
        mode='lines',
        name='Cased hole',
        line=dict(
            color='rgb(5,133,133)',
            width=2*7
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==7].index],
        hoverinfo='text',
        showlegend=False
    )

    trace9 = go.Scatter3d(
        x=df[df['Section (in)']==6.625]['N/S (m)'],
        y=df[df['Section (in)']==6.625]['E/W (m)'],
        z=df[df['Section (in)']==6.625]['TVD (m RKB)'],
        mode='lines',
        name='Cased hole',
        line=dict(
            color='rgb(5,133,133)',
            width=2*6.625
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==6.625].index],
        hoverinfo='text',
        showlegend=False
    )

    trace10 = go.Scatter3d(
        x=df[df['Section (in)']==6]['N/S (m)'],
        y=df[df['Section (in)']==6]['E/W (m)'],
        z=df[df['Section (in)']==6]['TVD (m RKB)'],
        mode='lines',
        name='Open hole',
        line=dict(
            color='rgb(198,137,75)',
            width=2*6
        ),
        text = [
                "<b>Section:</b> {} in<br>"
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                .format(
                        df['Section (in)'].loc[i],
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        )
                for i in df[df['Section (in)']==6].index],
        hoverinfo='text',
        showlegend=True
    )

    data =
[trace1,trace2,trace3,trace4,trace5,trace6,trace7,trace8,trace9,trace10]

    layout = dict(
        width=900,
        height=800,
        margin=dict(t=0,b=0, pad=0),
        autosize=False,
        legend=dict(x=1,y=0.85),
        hoverlabel=dict(
                        bgcolor='rgb(255,255,255)',
                        bordercolor='rgb(0,0,0)'
                        ),
        hovermode='closest',
        scene=dict(
            xaxis=dict(
                title='<b>Northing (m)</b>',
                gridcolor='rgb(169,169,169)',
                zerolinecolor='rgb(169,169,169)',
                showbackground=True,
                backgroundcolor='rgb(255,255,255)',
                showspikes=False
            ),
            yaxis=dict(
                title='<b>Easting (m)</b>',
                gridcolor='rgb(169,169,169)',
                zerolinecolor='rgb(169,169,169)',
                showbackground=True,
                backgroundcolor='rgb(255,255,255)',
                showspikes=False
            ),
            zaxis=dict(
                title='<b>Depth (m)</b>',
                gridcolor='rgb(169,169,169)',
                zerolinecolor='rgb(169,169,169)',
                showbackground=True,
                backgroundcolor='rgb(255,255,255)',
                showspikes=False,
                autorange='reversed'
            ),
            camera=dict(
                up=dict(
                    x=0,
                    y=0,
                    z=1
                ),
                eye=dict(
                    x=-1.25,
                    y=1.25,
                    z=0.5,
                )
            ),
        aspectmode = 'manual',
        aspectratio=dict(x=x/z, y=y/z, z=1)
        )
    )

    fig = dict(data=data, layout=layout)
    return fig

#default wellbore
df_survey = pd.read_csv(f'Data/Hole_section/Plot/Volve_15_9_19_A.csv',
sep=';', float_precision='round_trip')
df_hsection = pd.read_csv(f'Data/Hole_section/Table/Volve_15_9_19_A.csv',
sep=';', float_precision='round_trip')

#define hole section page layout and content
page_layout = html.Div([
    html.H3(['Hole section']),
    html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'border-
bottom': '1px solid black', 'font-weight': 'bold'}),
    html.Br(),
    html.Div(
        id='hole-section',
        children=[
            html.Div(dcc.Graph(
                id='hsection-graph',
                figure=section_plot(df_survey),
                config=dict(displayModeBar=False)
                            ), style={'display': 'inline-block', 'float':
'left', 'border-right': '1px solid black'}),
            html.Div(children=[
                html.Br(),
                dash_table.DataTable(
                    id='hsection-table',
                    columns=[
                        {"name": 'Type', "id": 'Type'},
                        {"name": 'From depth (m)', "id": 'From depth (m)'},
                        {"name": 'To depth (m)', "id": 'To depth (m)'},
                        {"name": 'OD (in)', "id": 'OD (in)'},
                        {"name": 'ID (in)', "id": 'ID (in)'}
```

```
                            ],
                        data=df_hsection.to_dict("rows"),
                        style_header={'font-family': 'Calibri', 'font-weight':
'bold'},
                        style_cell={'padding':'10px', 'font-family': 'Calibri',
'font-size': '16px'}
                                            )], style={'display': 'inline-
block', 'paddingTop': '100px', 'paddingLeft': '75px'})
                ]
            )
])

#callback to update table according to wellbore selection
@app.callback(
    Output('hsection-table', 'data'),
    [Input('wells-dropdown', 'value')]
            )
def display_hsection_table(val):
    df_hsection = pd.read_csv(f'Data/Hole_section/Table/Volve_{val}.csv',
sep=';', float_precision='round_trip')
    data=df_hsection.to_dict("rows")
    return data

#callback to update plot according to wellbore selection
@app.callback(
    Output('hsection-graph', 'figure'),
    [Input('wells-dropdown', 'value')]
            )
def display_hsection_graph(val):
    df_survey = pd.read_csv(f'Data/Hole_section/Plot/Volve_{val}.csv',
sep=';', float_precision='round_trip')
    return section_plot(df_survey)
```

wellpath.py

```
import dash_table
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import pandas as pd

from app import app

#function to create wellpath plot
def wellpath_plot(df):
    x=df['N/S (m)'].max()-df['N/S (m)'].min()
    y=df['E/W (m)'].max()-df['E/W (m)'].min()
    z=df['TVD (m RKB)'].max()-df['TVD (m RKB)'].min()
    trace0 = go.Scatter3d(
        x=df['N/S (m)'],
        y=df['E/W (m)'],
        z=df['TVD (m RKB)'],
        mode='lines',
        line=dict(
            color='rgb(211,211,211)',
            width=20
        ),
        hoverinfo='none',
        showlegend=False
    )

    trace1 = go.Scatter3d(
        x=df[df['DLS (deg/30m)']<2]['N/S (m)'],
        y=df[df['DLS (deg/30m)']<2]['E/W (m)'],
        z=df[df['DLS (deg/30m)']<2]['TVD (m RKB)'],
        mode='markers',
        marker=dict(
            color='rgb(128,196,196)',
            size=5
        ),
        name='DLS (°/30m): < 2',
        text = [
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                "<b>Inclination:</b> {} °<br>"
                "<b>Azimuth:</b> {} °<br>"
                "<b>DLS:</b> {} °/30m"
                .format(
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        df['Inc (deg)'].loc[i],
                        df['Azim (deg)'].loc[i],
                        df['DLS (deg/30m)'].loc[i]
                        )
                for i in df[df['DLS (deg/30m)']<2].index],
        hoverinfo='text'
    )

    trace2 = go.Scatter3d(
        x=df[(df['DLS (deg/30m)']>=2)&(df['DLS (deg/30m)']<4)]['N/S (m)'],
        y=df[(df['DLS (deg/30m)']>=2)&(df['DLS (deg/30m)']<4)]['E/W (m)'],
        z=df[(df['DLS (deg/30m)']>=2)&(df['DLS (deg/30m)']<4)]['TVD (m
RKB)'],
        mode='markers',
        marker=dict(
            color='rgb(0,115,172)',
            size=5
        ),
        name='DLS (°/30m): 2-4',
        text = [
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                "<b>Inclination:</b> {} °<br>"
                "<b>Azimuth:</b> {} °<br>"
                "<b>DLS:</b> {} °/30m"
                .format(
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        df['Inc (deg)'].loc[i],
                        df['Azim (deg)'].loc[i],
                        df['DLS (deg/30m)'].loc[i]
                        )
                for i in df[(df['DLS (deg/30m)']>=2)&(df['DLS
(deg/30m)']<4)].index],
        hoverinfo='text'
    )

    trace3 = go.Scatter3d(
        x=df[(df['DLS (deg/30m)']>=4)&(df['DLS (deg/30m)']<6)]['N/S (m)'],
        y=df[(df['DLS (deg/30m)']>=4)&(df['DLS (deg/30m)']<6)]['E/W (m)'],
        z=df[(df['DLS (deg/30m)']>=4)&(df['DLS (deg/30m)']<6)]['TVD (m
RKB)'],
        mode='markers',
        marker=dict(
            color='rgb(120,123,194)',
            size=5
        ),
        name='DLS (°/30m): 4-6',
        text = [
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                "<b>Inclination:</b> {} °<br>"
                "<b>Azimuth:</b> {} °<br>"
                "<b>DLS:</b> {} °/30m"
                .format(
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        df['Inc (deg)'].loc[i],
                        df['Azim (deg)'].loc[i],
                        df['DLS (deg/30m)'].loc[i]
                        )
                for i in df[(df['DLS (deg/30m)']>=4)&(df['DLS
(deg/30m)']<6)].index],
        hoverinfo='text'
    )

    trace4 = go.Scatter3d(
        x=df[(df['DLS (deg/30m)']>=6)&(df['DLS (deg/30m)']<8)]['N/S (m)'],
        y=df[(df['DLS (deg/30m)']>=6)&(df['DLS (deg/30m)']<8)]['E/W (m)'],
        z=df[(df['DLS (deg/30m)']>=6)&(df['DLS (deg/30m)']<8)]['TVD (m
RKB)'],
        mode='markers',
        marker=dict(
            color='rgb(183,18,124)',
            size=5
        ),
        name='DLS (°/30m): 6-8',
        text = [
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                "<b>Inclination:</b> {} °<br>"
                "<b>Azimuth:</b> {} °<br>"
                "<b>DLS:</b> {} °/30m"
                .format(
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        df['Inc (deg)'].loc[i],
                        df['Azim (deg)'].loc[i],
                        df['DLS (deg/30m)'].loc[i]
                        )
                for i in df[(df['DLS (deg/30m)']>=6)&(df['DLS
(deg/30m)']<8)].index],
        hoverinfo='text'
    )

    trace5 = go.Scatter3d(
        x=df[df['DLS (deg/30m)']>8]['N/S (m)'],
        y=df[df['DLS (deg/30m)']>8]['E/W (m)'],
        z=df[df['DLS (deg/30m)']>8]['TVD (m RKB)'],
        mode='markers',
        marker=dict(
            color='rgb(204,19,51)',
            size=5
        ),
        name='DLS (°/30m): > 8',
        text = [
                "<b>TVD:</b> {} m<br>"
                "<b>MD:</b> {} m<br>"
                "<b>Inclination:</b> {} °<br>"
                "<b>Azimuth:</b> {} °<br>"
                "<b>DLS:</b> {} °/30m"
                .format(
                        df['TVD (m RKB)'].loc[i],
                        df['MD (m RKB)'].loc[i],
                        df['Inc (deg)'].loc[i],
                        df['Azim (deg)'].loc[i],
                        df['DLS (deg/30m)'].loc[i]
                        )
                for i in df[df['DLS (deg/30m)']>8].index],
        hoverinfo='text'
    )

    data = [trace0,trace1,trace2,trace3,trace4,trace5]

    layout = dict(
        width=900,
        height=800,
        margin=dict(t=0,b=0, pad=0),
        autosize=False,
        legend=dict(x=1,y=0.85),
        hoverlabel=dict(
                        bgcolor='rgb(255,255,255)',
                        bordercolor='rgb(0,0,0)'
                        ),
        hovermode='closest',
        scene=dict(
            xaxis=dict(
                title='<b>Northing (m)</b>',
                gridcolor='rgb(169,169,169)',
                zerolinecolor='rgb(169,169,169)',
                showbackground=True,
                backgroundcolor='rgb(255,255,255)',
                showspikes=False
            ),
            yaxis=dict(
                title='<b>Easting (m)</b>',
                gridcolor='rgb(169,169,169)',
                zerolinecolor='rgb(169,169,169)',
                showbackground=True,
                backgroundcolor='rgb(255,255,255)',
                showspikes=False
            ),
            zaxis=dict(
                title='<b>Depth (m)</b>',
                gridcolor='rgb(169,169,169)',
                zerolinecolor='rgb(169,169,169)',
                showbackground=True,
                backgroundcolor='rgb(255,255,255)',
                showspikes=False,
                autorange='reversed'
            ),
            camera=dict(
                up=dict(
                    x=0,
                    y=0,
                    z=1
                ),
                eye=dict(
                    x=-1.25,
                    y=1.25,
                    z=0.5,
                )
            ),
            aspectratio = dict(x=x/z, y=y/z, z=1),
            aspectmode = 'manual'
        )
    )
    fig = dict(data=data, layout=layout)
    return fig

#default wellbore
df_survey = pd.read_csv(f'Data/Wellpath/Volve_15_9_19_A.csv', sep=';',
float_precision='round_trip')

#define wellpath page layout and content
page_layout = html.Div([
    html.H3(['Wellpath']),
    html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'border-
bottom': '1px solid black', 'font-weight': 'bold'}),
    html.Br(),
    html.Div(
        id='wellpath',
        children=[
            html.Div(dcc.Graph(
                id='wellpath-graph',
                figure=wellpath_plot(df_survey),
                config=dict(displayModeBar=False)
                        ), style={'display': 'inline-block', 'float':
'left', 'paddingRight': '10px', 'border-right': '1px solid black'}),
            html.Div(children=[
                html.Br(),
                dash_table.DataTable(
                    id='wellpath-table',
                    columns=[
                        {"name": 'MD (m RKB)', "id": 'MD (m RKB)'},
                        {"name": 'Inc. (°)', "id": 'Inc (deg)'},
                        {"name": 'Azimuth (°)', "id": 'Azim (deg)'},
                        {"name": 'TVD (m RKB)', "id": 'TVD (m RKB)'},
                        {"name": 'DLS (°/30m)', "id": 'DLS (deg/30m)'}
                            ],
                    n_fixed_rows=1,
                    data=df_survey[['MD (m RKB)', 'Inc (deg)', 'Azim
(deg)', 'TVD (m RKB)', 'DLS (deg/30m)']].to_dict("rows"),
                    style_header={'font-family': 'Calibri', 'font-weight':
'bold'},
                    style_cell={'padding':'10px', 'font-family': 'Calibri',
'font-size': '16px', 'width': '100px'}
                                        )], style={'display': 'inline-
block', 'paddingTop': '100px', 'paddingLeft': '25px'})
                ]
            )
])
```

```
#callback to update table according to wellbore selection
@app.callback(
    Output('wellpath-table', 'data'),
    [Input('wells-dropdown', 'value')]
            )
def display_wellpath_table(val):
    df_survey = pd.read_csv(f'Data/Wellpath/Volve_{val}.csv', sep=';',
float_precision='round_trip')
    data=df_survey[['MD (m RKB)', 'Inc (deg)', 'Azim (deg)', 'TVD (m RKB)',
'DLS (deg/30m)']].to_dict("rows")
    return data

#callback to update plot according to wellbore selection
@app.callback(
    Output('wellpath-graph', 'figure'),
    [Input('wells-dropdown', 'value')]
            )
def display_wellpath_graph(val):
    df_survey = pd.read_csv(f'Data/Wellpath/Volve_{val}.csv', sep=';',
float_precision='round_trip')
    return wellpath_plot(df_survey)
```

fluid.py

```
import dash_table
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import pandas as pd

from app import app

#default wellbore
df_fluid = pd.read_csv('Data/Fluid/Volve_15_9_19_A.csv', sep=';',
float_precision='round_trip')

#define fluid page layout and content
page_layout = html.Div([
    html.H3(['Fluid']),
    html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'border-
bottom': '1px solid black', 'font-weight': 'bold'}),
    html.Br(),
    html.Div([
            dash_table.DataTable(
                id='fluid-table',
                columns=[{"name": i, "id": i} for i in df_fluid.columns],
                data=df_fluid.to_dict("rows"),
                style_header={'font-family': 'Calibri', 'font-weight':
'bold'},
                style_cell={'padding':'10px', 'font-family': 'Calibri',
'font-size': '16px'}
    )], style={'paddingTop': '10px', 'width':'50%'})
])


#update table according to wellbore selection
@app.callback(
    Output('fluid-table', 'data'),
    [Input('wells-dropdown', 'value')]
            )
def display_fluid_table(val):
    df_fluid = pd.read_csv(f'Data/Fluid/Volve_{val}.csv', sep=';',
float_precision='round_trip')
    data=df_fluid.to_dict("rows")
    return data
```

drillstring.py

```
import dash_table
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import pandas as pd

from app import app

#default wellbore section
df_drillstring = pd.read_csv('Data/Drillstring/Volve_15_9_19_A_8_5.csv',
sep=';', float_precision='round_trip')

#define drillstring page layout and content
page_layout = html.Div([
    html.H3(['Drillstring']),
    html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'font-
weight': 'bold', 'border-bottom': '1px solid black'}),
    html.Br(),
    dcc.Dropdown(
        options=[{'label':'8.5 in section', 'value':'8_5'}],
        placeholder = 'Select a hole section',
        id='drillstring-dropdown',
        style={'width':'50%'}
    ),
    html.Div(),
    html.Br(),
    html.Div(id='table-container', children=[
        dash_table.DataTable(
            id='drillstring-table',
            columns=[{'name': i, 'id': i} for i in df_drillstring.columns],
            data=df_drillstring.to_dict("rows"),
            style_header={'font-family': 'Calibri', 'font-weight': 'bold'},
            style_cell={'padding':'10px', 'font-family': 'Calibri', 'font-
size': '16px'}
    )], style={'width':'50%', 'paddingTop':'25px'})
])

#update dropdown according to wellbore selection
@app.callback(
    Output('drillstring-dropdown', 'options'),
    [Input('wells-dropdown', 'value')]
            )
def change_drillstring_dropdown(val):
    drillstring_dict = {
        '15_9_19_A': [8.5],
        '15_9_19_B': [8.5],
        '15_9_19_BT2': [6, 8.5],
        '15_9_19_S': [12.25, 17.5],
        '15_9_19_SR': [8.5, 12.25],
        '15_9_F_1': [8.5, 17.5],
        '15_9_F_1_A': [8.5],
        '15_9_F_1_B': [8.5, 12.25],
        '15_9_F_1_C': [8.5, 12.25, 17.5],
        '15_9_F_4': [8.5, 12.25],
        '15_9_F_5': [8.5, 12.25],
        '15_9_F_7': [12.25],
        '15_9_F_9': [12.25],
        '15_9_F_9_A': [8.5, 12.25],
        '15_9_F_10': [8.5, 12.25, 17.5],
        '15_9_F_11': [8.5, 17.5],
        '15_9_F_11_A': [8.5],
        '15_9_F_11_B': [8.5, 12.25],
        '15_9_F_12': [8.5, 12.25, 17.5],
        '15_9_F_14': [8.5, 12.25, 17.5],
        '15_9_F_15': [8.5, 12.25],
        '15_9_F_15_A': [8.5, 17.5],
        '15_9_F_15_B': [8.5],
        '15_9_F_15_C': [8.5, 12.25],
        '15_9_F_15_D': [8.5, 12.25, 17.5]
    }
    if val:
        drillstring_options = drillstring_dict[val]
        drillstring_options = list(reversed(drillstring_options))
        options=[{'label':str(drillstring_options[i])+' in section',
'value':str(drillstring_options[i]).replace('.', '_')} for i in
range(len(drillstring_options))]
        return options

# update table according to wellbore and hole section selection
@app.callback(
    Output('drillstring-table', 'data'),
    [Input('wells-dropdown', 'value'), Input('drillstring-dropdown',
'value')]
)
def display_drillstring_table(val1, val2):
    df_drillstring =
pd.read_csv(f'Data/Drillstring/Volve_{val1}_{val2}.csv', sep=';',
float_precision='round_trip')
    data=df_drillstring.to_dict("rows")
    return data

#hide table until selection is made
@app.callback(
            Output('table-container', 'style'),
            [Input('drillstring-dropdown', 'value')]
            )
def hide_table(input):
    if input:
        return {'width':'50%'}
    else:
        return {'display':'none'}
```

geopressures.py

```
import dash_table
import dash_core_components as dcc
```

```python
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import pandas as pd

from app import app

#function to create geopressures plot
def pressure_plot(df):
    trace0 = go.Scatter(
        x=df['Pore pressure (s.g.)'],
        y=[0 for num in df.index],
        mode='markers',
        marker = dict(
            color = 'rgb(0,0,0)',
            size = 0.01,
                    ),
        name='dummy_data',
        hoverinfo='none',
        showlegend=False,
    )

    trace1 = go.Scatter(
        x=df['Pore pressure (s.g.)'],
        y=df['TVD (m)'],
        mode='lines',
        line = dict(
            color = ('rgb(183,18,124)'),
            width = 2
                    ),
        name='Pore pressure',
        text=[
            "<b>TVD:</b> {} m<br>"
            "<b>Pore pressure:</b> {} s.g.<br>"
            "<b>Fracture pressure:</b> {} s.g.<br>"
            "<b>Pressure window:</b> {} s.g.<br>"
            .format(
                    df['TVD (m)'].loc[num],
                    df['Pore pressure (s.g.)'].loc[num],
                    df['Fracture pressure (s.g.)'].loc[num],
                    round(df['Fracture pressure (s.g.)'].loc[num]-df['Pore
pressure (s.g.)'].loc[num], 3)
                    )
            for num in df.index
            ],
        hoverinfo="text",
        hoverlabel=dict(
                        bgcolor='rgb(255,255,255)',
                        bordercolor='rgb(0,0,0)'
                        ),
        showlegend=True,
    )

    trace2 = go.Scatter(
        x=df['Fracture pressure (s.g.)'],
        y=df['TVD (m)'],
        mode='lines',
        line = dict(
            color = ('rgb(0,115,172)'),
            width = 2
                    ),
        name='Fracture pressure',
        text=[
            "<b>TVD:</b> {} m<br>"
            "<b>Pore pressure:</b> {} s.g.<br>"
            "<b>Fracture pressure:</b> {} s.g.<br>"
            "<b>Pressure window:</b> {} s.g.<br>"
            .format(
                    df['TVD (m)'].loc[num],
                    df['Pore pressure (s.g.)'].loc[num],
                    df['Fracture pressure (s.g.)'].loc[num],
                    round(df['Fracture pressure (s.g.)'].loc[num]-df['Pore
pressure (s.g.)'].loc[num], 3)
                    )
            for num in df.index
            ],
        hoverinfo="text",
        hoverlabel=dict(
                        bgcolor='rgb(255,255,255)',
                        bordercolor='rgb(0,0,0)'
                        ),
        showlegend=True,
    )
    data = [trace0,trace1,trace2]
    layout = go.Layout(
                    title=None,
                    height = 550,
                    width = 800,
                    margin=dict(t=25, pad=0),
                    autosize = False,
                    xaxis=dict(
                        title='<b>Pressure<br>(s.g.)</b>',
                        range=[0.8, 2.0]
                            ),
                    yaxis=dict(
                        title='<b>TVD<br>(m)</b>',
                        autorange='reversed'),
                    legend=dict(x=0, y=-0.3),
                    hovermode='closest'
                        )
    fig = go.Figure(data=data,layout=layout)
    return fig

#default wellbore
df_geopressures = pd.read_csv(f'Data/Geopressures/Volve_15_9_19_A.csv',
sep=';', float_precision='round_trip')

#define geopressures page layout and content
page_layout = html.Div([
    html.H3(['Geopressures']),
    html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'border-
bottom': '1px solid black', 'font-weight': 'bold'}),
    html.Br(),
    html.Div(
        id='geopressures',
        children=[
            html.Div(children=[
                html.Div(dcc.Graph(
                    id='geopressures-graph',
                    figure=pressure_plot(df_geopressures),
                    config=dict(displayModeBar=False)
```

```
                            ), style={'display': 'inline-block', 'float':
'left', 'border-right': '1px solid black'}),
                html.Div(children=[
                    dash_table.DataTable(
                        id='geopressures-table',
                        columns=[
                            {"name": 'TVD (m)', "id": 'TVD (m)'},
                            {"name": 'Pore pressure (s.g.)', "id": 'Pore
pressure (s.g.)'},
                            {"name": 'Fracture pressure (s.g.)', "id":
'Fracture pressure (s.g.)'}
                                ],
                        n_fixed_rows=1,
                        data=df_geopressures.to_dict("rows"),
                        style_header={'font-family': 'Calibri', 'font-
weight': 'bold'},
                        style_cell={'padding':'10px', 'font-family':
'Calibri', 'font-size': '16px'},
                        style_cell_conditional=[
                            {'if': {'column_id': 'TVD (m)'},
                             'width': '100px'},
                            {'if': {'column_id': 'Pore pressure (s.g.)'},
                             'width': '175px'},
                            {'if': {'column_id': 'Fracture pressure
(s.g.)'},
                             'width': '175px'},
])], style={'display': 'inline-block', 'paddingTop': '50px', 'paddingLeft':
'150px'})
])])])

#callback to update table according to wellbore selection
@app.callback(
    Output('geopressures-table', 'data'),
    [Input('wells-dropdown', 'value')]
            )
def display_geopressures_table(val):
    df_geopressures = pd.read_csv(f'Data/Geopressures/Volve_{val}.csv',
sep=';', float_precision='round_trip')
    data=df_geopressures.to_dict("rows")
    return data

#callback to update plot according to wellbore selection
@app.callback(
    Output('geopressures-graph', 'figure'),
    [Input('wells-dropdown', 'value')]
            )
def display_geopressures_graph(val):
    df_geopressures = pd.read_csv(f'Data/Geopressures/Volve_{val}.csv',
sep=';', float_precision='round_trip')
    return pressure_plot(df_geopressures)
```

geothermal.py

```
import dash_table
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import pandas as pd
```

```
from app import app

#function to create temperature plot
def temp_plot(df):
    trace1 = go.Scatter(
        x=df['Temperature (degC)'],
        y=df['TVD (m)'],
        mode='lines',
        line = dict(
            color = ('rgb(0,115,172)'),
            width = 2
                ),
        name='Temperature gradient',
        text=[
            "<b>TVD:</b> {} m<br>"
            "<b>Temperature:</b> {} °C<br>"
            "<b>Medium:</b> {}<br>"
            .format(
                    df['TVD (m)'].loc[num],
                    df['Temperature (degC)'].loc[num],
                    df['Medium'].loc[num]
                    )
            for num in df.index
            ],
        hoverinfo="text",
        hoverlabel=dict(
                        bgcolor='rgb(255,255,255)',
                        bordercolor='rgb(0,0,0)'
                        ),
        showlegend=False
    )
    data = [trace1]
    layout = go.Layout(
                    title=None,
                    height = 550,
                    width = 800,
                    margin=dict(t=25, pad=0),
                    autosize = False,
                    xaxis=dict(
                        title='<b>Temperature<br>(°C)</b>',
                        range=[0, 130]
                            ),
                    yaxis=dict(
                        title='<b>TVD<br>(m)</b>',
                        autorange='reversed'),
                    hovermode='closest'
                        )

    fig = go.Figure(data=data,layout=layout)
    return fig

#default wellbore
df_geothermal = pd.read_csv(f'Data/Geothermal/Plot/Volve_15_9_19_A.csv',
sep=';', float_precision='round_trip')
df_geothermal_table =
pd.read_csv(f'Data/Geothermal/Table/Volve_15_9_19_A.csv', sep=';',
float_precision='round_trip')

#define geothermal page layout and content
page_layout = html.Div([
    html.H3(['Geothermal']),
    html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'border-
bottom': '1px solid black', 'font-weight': 'bold'}),
    html.Br(),
    html.Div(
        id='geothermal',
        children=[
            html.Div(children=[
                html.Div(dcc.Graph(
                    id='geothermal-graph',
                    figure=temp_plot(df_geothermal),
                    config=dict(displayModeBar=False)
                            ), style={'display': 'inline-block', 'float':
'left', 'border-right': '1px solid black'}),
                html.Div(dash_table.DataTable(
                    id='geothermal-table',
                    columns=[
                        {"name": 'Medium', "id": 'Medium'},
                        {"name": 'From TVD (m)', "id": 'From TVD (m)'},
                        {"name": 'Temp. gradient (°C/100m)', "id": 'Temp.
gradient (°C/100m)'}
                            ],
                    data=df_geothermal_table.to_dict("rows"),
                    style_header={'font-family': 'Calibri', 'font-weight':
'bold'},
                    style_cell={'padding':'10px', 'font-family': 'Calibri',
'font-size': '16px'}
                                        ), style={'display': 'inline-
block', 'paddingTop': '50px', 'paddingLeft': '150px'})])
                ]
            )
])

#callback to update table according to wellbore selection
@app.callback(
    Output('geothermal-table', 'data'),
    [Input('wells-dropdown', 'value')]
            )
def display_geothermal_table(val):
    df_geothermal_table =
pd.read_csv(f'Data/Geothermal/Table/Volve_{val}.csv', sep=';',
float_precision='round_trip')
    data=df_geothermal_table.to_dict("rows")
    return data

#callback to update plot according to wellbore selection
@app.callback(
    Output('geothermal-graph', 'figure'),
    [Input('wells-dropdown', 'value')]
            )
def display_geothermal_graph(val):
    df_geothermal = pd.read_csv(f'Data/Geothermal/Plot/Volve_{val}.csv',
sep=';', float_precision='round_trip')
    return temp_plot(df_geothermal)
```

open_lab.py

```
import dash_core_components as dcc
import dash_html_components as html
```

```
from dash.dependencies import Input, Output, State
import pandas as pd
import json
import openlab

from app import app

#wellbores for dropdown selection
well_dict = {
            '15_9_19_A': '15/9-19 A',
            '15_9_19_B': '15/9-19 B',
            '15_9_19_BT2': '15/9-19 BT2',
            '15_9_19_S': '15/9-19 S',
            '15_9_19_SR': '15/9-19 SR',
            '15_9_F_1': '15/9-F-1',
            '15_9_F_1_A': '15/9-F-1 A',
            '15_9_F_1_B': '15/9-F-1 B',
            '15_9_F_1_C': '15/9-F-1 C',
            '15_9_F_4': '15/9-F-4',
            '15_9_F_5': '15/9-F-5',
            '15_9_F_7': '15/9-F-7',
            '15_9_F_9': '15/9-F-9',
            '15_9_F_9_A': '15/9-F-9 A',
            '15_9_F_10': '15/9-F-10',
            '15_9_F_11': '15/9-F-11',
            '15_9_F_11_A': '15/9-F11 A',
            '15_9_F_11_B': '15/9-F-11 B',
            '15_9_F_12': '15/9-F-12',
            '15_9_F_14': '15/9-F-14',
            '15_9_F_15': '15/9-F-15',
            '15_9_F_15_A': '15/9-F-15 A',
            '15_9_F_15_B': '15/9-F-15 B',
            '15_9_F_15_C': '15/9-F-15 C',
            '15_9_F_15_D': '15/9-F-15 D'
            }

#define openlab page layout
page_layout = html.Div([
    html.H3(['Create a well configuration in OpenLab Drilling Simulator']),
    html.Div(['Choose a wellbore and a hole section'],
style={'paddingBottom':'10px'}),
    dcc.Dropdown(
        options=[{'label':value, 'value':key} for key, value in
well_dict.items()],
        placeholder = 'Select a wellbore',
        id='wellbore-dropdown',
        style={'width':'50%'}
    ),
    html.Br(),
    dcc.Dropdown(
        placeholder = 'Select a hole section',
        id='section-dropdown',
        style={'width':'50%'}),
    html.Br(),
    html.Div(["Generate Python login script in OpenLab Web
Client\Account\Settings"], style={'paddingBottom':'10px'}),
    dcc.Textarea(
        id='text-input',
        placeholder='Paste here the Python login
script\nusername="string"\napikey="string"\nlicenseguid="string"',
        rows=4,
        style={'width': '55%'}),
    html.Br(),
    html.Div(
        html.Button(
            children='Create configuration',
            id='create-button',
            n_clicks=0,
            style={
                'font-size': '14px',
                'border-radius':'2px',
                'padding':'5px',
                'cursor': 'pointer'}
    ), style={'paddingTop':'10px'}),
    html.Div(id='text-output')
])

#update hole section dropdown according to wellbore selection
@app.callback(
    Output('section-dropdown', 'options'),
    [Input('wellbore-dropdown', 'value')]
            )
def change_section_dropdown(val):
    section_dict = {
        '15_9_19_A': [8.5],
        '15_9_19_B': [8.5],
        '15_9_19_BT2': [6, 8.5],
        '15_9_19_S': [12.25, 17.5],
        '15_9_19_SR': [8.5, 12.25],
        '15_9_F_1': [8.5, 17.5],
        '15_9_F_1_A': [8.5],
        '15_9_F_1_B': [8.5, 12.25],
        '15_9_F_1_C': [8.5, 12.25, 17.5],
        '15_9_F_4': [8.5, 12.25],
        '15_9_F_5': [8.5, 12.25],
        '15_9_F_7': [12.25],
        '15_9_F_9': [12.25],
        '15_9_F_9_A': [8.5, 12.25],
        '15_9_F_10': [8.5, 12.25, 17.5],
        '15_9_F_11': [8.5, 17.5],
        '15_9_F_11_A': [8.5],
        '15_9_F_11_B': [8.5, 12.25],
        '15_9_F_12': [8.5, 12.25, 17.5],
        '15_9_F_14': [8.5, 12.25, 17.5],
        '15_9_F_15': [8.5, 12.25],
        '15_9_F_15_A': [8.5, 17.5],
        '15_9_F_15_B': [8.5],
        '15_9_F_15_C': [8.5, 12.25],
        '15_9_F_15_D': [8.5, 12.25, 17.5]
    }
    if val:
        section_options = section_dict[val]
        section_options = list(reversed(section_options))
        options=[{'label':str(section_options[i])+' in section',
'value':str(section_options[i]).replace('.', '_')} for i in
range(len(section_options))]
        return options

#create configuration in OpenLab
@app.callback(
    Output('text-output', 'children'),
    [Input('create-button', 'n_clicks')],
    [
    State('text-input', 'value'),
    State('wellbore-dropdown', 'value'),
    State('section-dropdown', 'value')]
)
def create_config(n_clicks, val, wellbore, section):
    if (val and wellbore and section):
        input_text = val
        input_text.strip()
        text = input_text.split()
        username = text[0][text[0].find('=')+2:-1]
        apikey = text[1][text[1].find('=')+2:-1]
        licenseguid = text[2][text[2].find('=')+2:-1]

        session =
openlab.http_client(username=username,apikey=apikey,licenseguid=licenseguid
)
        with open(f'Configurations/{wellbore}_section_{section}.json', 'r')
as f:
            data = json.load(f)

        new_config =
session.create_configuration(f'{wellbore}_section_{section}', data['Data'])
```

Appendix E

Python code to create the Volve-DDR app. The same scripts together with documentation, requirements and data files are pushed to https://gitlab.com/a-nagy/volve-ddr-app. Here the Data folder holds a CSV file that contains general well data to populate the opening screen, while the Reports folder contains the DDRs in XML format. As the web application itself is a multi-page app, the scripts are splitted into separate Python files to keep the editing and maintenance of the code simpler. The same file structure is presented here. The data_extraction.py file contains the scripts for automated data extraction from the DDRs.

data_extraction.py

```
import os
import xml.etree.ElementTree as et
import pandas as pd
from datetime import datetime

#function to extract data from the daily drilling reports for the
time/depth curve
def get_timevsdepth(well):

    report_list = os.listdir('Reports')

    time = []
    md = []
    section = []
    summary = []

    df = pd.DataFrame(
        list(zip(
            time,
            md,
            section,
            summary
        )),
        columns = [
            'Time',
            'MD (m)',
            'Section (in)',
            'Summary'
    ])

    os.chdir('Reports')

    for file in report_list:
        if file[:-15] == well:
            report_tree = et.parse(file)
            report_root = report_tree.getroot()

            time = []
            md = []
            section = []
            summary = []
            time_temp = True
            md_temp = True
            section_temp = True
            summary_temp = True

            for child in report_root:
                for elem in child:
                    if elem.tag ==
'{http://www.witsml.org/schemas/1series}statusInfo':
                        for subelem in elem:
                            if subelem.tag ==
'{http://www.witsml.org/schemas/1series}dTim':

time.append(datetime.strptime(subelem.text[:16], '%Y-%m-%dT%H:%M'))
                                time_temp = False
                            elif subelem.tag ==
'{http://www.witsml.org/schemas/1series}md':
                                md.append(float(subelem.text) if
float(subelem.text)>0 else 0)
                                md_temp = False
                            elif subelem.tag ==
'{http://www.witsml.org/schemas/1series}diaHole':
                                section.append(float(subelem.text))
                                section_temp = False
                            elif subelem.tag ==
'{http://www.witsml.org/schemas/1series}sum24Hr':
                                summary.append(subelem.text)
                                summary_temp = False
                            else:
                                pass

            if time_temp:
                time.append('None')
            if md_temp:
                md.append(0)
            if section_temp:
                section.append('-')
            if summary_temp:
                summary.append('-')
            df = df.append(
                pd.DataFrame(
                    list(zip(
                        time,
                        md,
                        section,
                        summary
                    )),
                    columns = [
                        'Time',
                        'MD (m)',
                        'Section (in)',
                        'Summary'
                    ]),
                ignore_index = True,
                sort = False
            )

    df.sort_values(['Time'], inplace=True)
    os.chdir('../')
    return df

#function to extract data from the daily drilling reports for the
operations timeline
def get_operations(well):

    report_list = os.listdir('Reports')

    start = []
    end = []
    md = []
    operation = []
    comment = []
    duration = []
    state = []

    df = pd.DataFrame(
        list(zip(
            start,
            end,
            md,
            duration,
            operation,
            comment,
            state
        )),
        columns = [
            'Start',
            'End',
            'MD (m)',
            'Duration',
            'Operation',
            'Comment',
            'State'
    ])

    os.chdir('Reports')

    for file in report_list:
        if file[:-15] == well:
            report_tree = et.parse(file)
            report_root = report_tree.getroot()

            start = []
            end = []
            md = []
            operation = []
            comment = []
            duration = []
            state = []

            for child in report_root:
                for elem in child:
                    if elem.tag ==
'{http://www.witsml.org/schemas/1series}activity':
                        for subelem in elem:
                            if 'dTimStart' in subelem.tag:

start.append(datetime.strptime(subelem.text[:16], '%Y-%m-%dT%H:%M'))
                                start_temp =
datetime.strptime(subelem.text[:16], '%Y-%m-%dT%H:%M')
                            elif 'dTimEnd' in subelem.tag:

end.append(datetime.strptime(subelem.text[:16], '%Y-%m-%dT%H:%M'))
                                end_temp =
datetime.strptime(subelem.text[:16], '%Y-%m-%dT%H:%M')
                            elif 'md' in subelem.tag:
                                md.append(subelem.text)
                            elif 'proprietaryCode' in subelem.tag:
                                operation.append(subelem.text)
                            elif 'comments' in subelem.tag:
                                comment.append(subelem.text)
                            elif 'stateDetailActivity' in subelem.tag:
                                state.append(subelem.text)
                            else:
                                pass
                        duration.append((end_temp-start_temp))

            df = df.append(
                pd.DataFrame(
                    list(zip(
                        start,
                        end,
                        md,
                        duration,
                        operation,
                        comment,
                        state
                    )),
                    columns = [
                        'Start',
                        'End',
                        'MD (m)',
                        'Duration',
                        'Operation',
                        'Comment',
                        'State'
                    ]),
                ignore_index = True,
                sort = False
            )
```

```
    df.sort_values(['Start'], inplace=True)
    os.chdir('../')
    return df
```

app.py

```
import dash

#external stylesheet for css styling
external_stylesheets = ['https://cdn.jsdelivr.net/gh/attilanagy1986/Dash-
css@master/undo.css']

#initiate Dash app
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
#title of the app
app.title = 'Volve app'
#initiate server
server = app.server
#if set to False, Dash will raise an exception due to the multipage
structure of the app
app.config.suppress_callback_exceptions = True
```

index.py

```
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
#import app and the separate app pages
from app import app
import volve_wells
import timevsdepth
import operations

#wellbore names for dropdown selection
wells_dict = {
            '15_9_19_A': '15/9-19 A',
            '15_9_19_B': '15/9-19 B',
            '15_9_19_BT2': '15/9-19 BT2',
            '15_9_19_S': '15/9-19 S',
            '15_9_19_ST2': '15/9-19 ST2',
            '15_9_F_1': '15/9-F-1',
            '15_9_F_1_A': '15/9-F-1 A',
            '15_9_F_1_B': '15/9-F-1 B',
            '15_9_F_1_C': '15/9-F-1 C',
            '15_9_F_4': '15/9-F-4',
            '15_9_F_5': '15/9-F-5',
            '15_9_F_7': '15/9-F-7',
            '15_9_F_9': '15/9-F-9',
            '15_9_F_9_A': '15/9-F-9 A',
            '15_9_F_10': '15/9-F-10',
            '15_9_F_11': '15/9-F-11',
            '15_9_F_11_A': '15/9-F11 A',
            '15_9_F_11_B': '15/9-F-11 B',
            '15_9_F_11_T2': '15/9-F-11 T2',
            '15_9_F_12': '15/9-F-12',
            '15_9_F_14': '15/9-F-14',
            '15_9_F_15': '15/9-F-15',
            '15_9_F_15_A': '15/9-F-15 A',
            '15_9_F_15_B': '15/9-F-15 B',
            '15_9_F_15_C': '15/9-F-15 C',
            '15_9_F_15_D': '15/9-F-15 D'
            }

#define the app layout
app.layout = html.Div([
    dcc.Location(id='url'),
    dcc.Link(
            'Volve wells',
            href='/volve_wells',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold'
                }),
    dcc.Link(
            'Time/depth curve',
            href='/timevsdepth',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    dcc.Link(
            'Operations',
            href='/operations',
            style={
                'color': 'rgb(0,0,0)',
                'font-size': '17px',
                'font-weight': 'bold',
                'paddingLeft':'25px'
                }),
    html.Div([
        html.H3(['Select a wellbore'], style={'paddingBottom': '10px',
'font-weight': 'bold', 'border-bottom': '1px solid black'}),
        dcc.Dropdown(
            options=[{'label':value, 'value':key} for key, value in
wells_dict.items()],
            value='15_9_19_A',
            placeholder='Select a wellbore',
            id='wells-dropdown',
            style={'width':'50%'}
        ),
        html.Br(),
    ], id='external-page-wells', style={'paddingLeft':'25px'}),
    html.Div(id='page-content')
], style={'font-family': 'Calibri', 'paddingLeft':'25px',
'paddingRight':'25px'})

#callback to update page content
@app.callback(
    Output('page-content', 'children'),
    [Input('url', 'pathname')]
)
def populate_content(url):
    if url == '/volve_wells':
        return volve_wells.page_layout
    elif url == '/timevsdepth':
        return timevsdepth.page_layout
    elif url == '/operations':
        return operations.page_layout

#callback for mulltipage persistence of the wellbore dropdown
@app.callback(
    Output('external-page-wells', 'style'),
    [Input('url', 'pathname')]
)
def hide_external(url):
    if url == '/volve_wells':
        return {'display': 'block'}
    else:
        return {'display': 'none'}

#callback to display the selected wellbore name
@app.callback(
    Output('dropdown-output', 'children'),
    [Input('wells-dropdown', 'value')]
)
def display_dropdown_contents(val):
    if val:
        return f'Wellbore selected: {wells_dict[val]}'

#run the app
if __name__ == '__main__':
    app.run_server()
```

volve_wells.py

```
import dash_table
import dash_html_components as html
import pandas as pd

#read data from csv
df_wells = pd.read_csv('Data/volve_wells.csv', sep=';')

#define volve wells page layout and content
page_layout = html.Div([
    dash_table.DataTable(
        id='wells-table',
        columns=[{"name": i, "id": i} for i in df_wells.columns],
        data=df_wells.to_dict("rows"),
        style_header={'font-family': 'Calibri', 'font-weight': 'bold'},
        style_cell={'padding':'10px', 'font-family': 'Calibri', 'font-
size': '16px', 'width': '100px'}
)], style={'width':
```

timevsdepth.py

```
import dash_table
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import pandas as pd
```

```python
import numpy as np

from app import app
import data_extraction

#function to create the time/depth plot
def timevsdepth_plot(df):

    trace0 = go.Scatter(
        x=df['Time'],
        y=[0 for num in df.index],
        mode='markers',
        marker = dict(
            color = 'rgb(0,0,0)',
            size = 0.1,
            ),
        name='dummy_data',
        hoverinfo='none',
        showlegend=False,
    )

    trace1 = go.Scatter(
        x=df['Time'],
        y=df['MD (m)'],
        mode='lines+markers',
        line = dict(
            color = ('rgb(7,178,178)'),
            width = 2
            ),
        marker = dict(
            color = 'rgb(7,178,178)',
            size = 5,
            ),
        name='MD',
        text=[
            "<b>Date:</b> {}<br>"
            "<b>MD:</b> {} m<br>"
            "<b>Section:</b> {} in<br>"
            "<b>Summary:</b> {}"
            .format(
                str(df['Time'].loc[num])[0:10],
                df['MD (m)'].loc[num],
                df['Section (in)'].loc[num],
                df['Summary'].loc[num].replace('.','<br>')
            )
            for num in df.index
        ],
        hoverinfo="text",
        hoverlabel=dict(
            bgcolor='rgb(255,255,255)',
            bordercolor='rgb(0,0,0)'
        ),
        showlegend=False,
    )

    data = [trace0,trace1]

    layout = go.Layout(
        title=None,
        xaxis=dict(
            title=dict(
                text='<b>Days</b>',
                font=dict(family='Calibri', size=16)
            ),
            tickvals=pd.date_range(
                str(df['Time'].iloc[0])[0:10],
                str(df['Time'].iloc[-1])[0:10],
                freq='5d'
            ),
            rangeselector=dict(
                x=0,
                y=1.05,
                buttons=list([
                    dict(count=7,
                        label='1w',
                        step='day',
                        stepmode='backward'),
                    dict(count=1,
                        label='1m',
                        step='month',
                        stepmode='backward'),
                    dict(count=6,
                        label='6m',
                        step='month',
                        stepmode='backward'),
                    dict(count=1,
                        label='1y',
                        step='year',
                        stepmode='backward'),
                    dict(step='all')
                ])
            ),
            showgrid=False
        ),
        yaxis=dict(
            title=dict(
                text='<b>Depth (m)</b>',
                font=dict(family='Calibri', size=16)
            ),
            autorange='reversed'),
        hovermode='closest'
    )

    fig = go.Figure(data=data,layout=layout)

fig['layout']['xaxis'].update(ticktext=5*np.array(list(range(len(fig['layou
t']['xaxis']['tickvals'])))))
    return fig

#default wellbore
df_timevsdepth = data_extraction.get_timevsdepth('15_9_19_A')

#define time/depth curve page layout
page_layout = html.Div([
    html.H3(['Time/depth curve']),
    html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'border-
bottom': '1px solid black', 'font-weight': 'bold'}),
    html.Br(),
    html.Div(dcc.Graph(
        id='timevsdepth-plot',
        figure=timevsdepth_plot(df_timevsdepth),
        config=dict(displayModeBar=False)
    ), style={'display': 'block'}
)])

#callback to change the plot according to wellbore selection
@app.callback(
    Output('timevsdepth-plot', 'figure'),
    [Input('wells-dropdown', 'value')]
            )
def display_timevsdepth_plot(value):
    df_timevsdepth = data_extraction.get_timevsdepth(value)
    return timevsdepth_plot(df_timevsdepth)
```

operations.py

```
import dash_table
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.figure_factory as ff
import pandas as pd
import numpy as np

from app import app
import data_extraction

#function to create the operations timeline plot
def operations_plot(df):

    data = [
        dict(
            Task='<b>Activities</b>',
            Start=df['Start'].loc[num],
            Finish=df['End'].loc[num],
            Resource=df['State'].loc[num].replace(' ','_')
            )
        for num in df.index
        ]

    colors = dict(
        equipment_failure='rgb(0,115,172)',
        injury='rgb(255,210,0)',
        mud_loss='rgb(183,18,124)',
        circulation_loss='rgb(183,18,124)',
        stuck_equipment='rgb(149,99,47)',
        success='rgb(7,178,178)',
        operation_failed='rgb(204,19,51)'
    )

    fig = ff.create_gantt(
                    data,
                    title=None,
                    colors=colors,
                    show_colorbar=True,
                    index_col='Resource',
                    bar_width=1,
                    group_tasks=True
                    )

    text = [
        "<b>Start:</b> {}<br>"
        "<b>End:</b> {}<br>"
        "<b>Duration:</b> {}<br>"
        "<b>MD:</b> {} m<br>"
        "<b>Operation:</b> {}<br>"
        "<b>State:</b> {}<br>"
        "<b>Comment:</b> {}"
        .format(
                str(df['Start'].loc[i])[:16],
                str(df['End'].loc[i])[:16],
                str(df['Duration'].loc[i])[7:12],
                df['MD (m)'].loc[i],
                df['Operation'].loc[i],
                df['State'].loc[i],
                df['Comment'].loc[i].replace('.','<br>')
                )
        for i in df.index
        ]

    for i in range(len(text)):
        fig['data'][i].update(text=text[i], hoverinfo='text')

    for i in range(len(fig['layout']['shapes'])):
        fig['layout']['shapes'][i].update(line={'width':0.1,
'color':'rgb(255,255,255)'})

    fig['layout']['xaxis']['rangeselector'].update(
        x=0,
        y=1.5,
        buttons=[
            dict(count=1,
                label='1d',
                step='day',
                stepmode='backward'),
            dict(count=7,
                label='1w',
                step='day',
                stepmode='backward'),
            dict(count=1,
                 label='1m',
                 step='month',
                 stepmode='backward'),
            dict(count=6,
                 label='6m',
                 step='month',
                 stepmode='backward'),
            dict(count=1,
                label='1y',
                step='year',
                stepmode='backward'),
            dict(step='all')
    ])

    fig['layout'].update(
                    autosize=False,
                    width=1400,
                    height=240,
                    margin=dict(r=0),
                    legend=dict(
                        orientation='h',
                        x=0,
                                    y=-1.2
                                ),
                                hovermode='closest')

fig['layout']['xaxis'].update(title=dict(text='<b>Days</b>',
font=dict(size=12)),tickvals=pd.date_range(str(df['Start'].iloc[0])[0:10],
str(df['End'].iloc[-1])[0:10], freq='5d'))

fig['layout']['xaxis'].update(ticktext=5*np.array(list(range(len(fig['layou
t']['xaxis']['tickvals'])))))

return fig

#default wellbore
df_operations = data_extraction.get_operations('15_9_19_A')

#define operations page layout
page_layout = html.Div([
html.H3(['Operations']),
html.Div(id='dropdown-output', style={'paddingBottom': '10px', 'border-
bottom': '1px solid black', 'font-weight': 'bold'}),
html.Br(),
html.Div(dcc.Graph(
    id='operations-plot',
    figure=operations_plot(df_operations),
    config=dict(displayModeBar=False)
), style={'display': 'block'}
)])

#callback to change the plot according to wellbore selection
@app.callback(
Output('operations-plot', 'figure'),
[Input('wells-dropdown', 'value')]
)
def display_operations_plot(value):
df_operations = data_extraction.get_operations(value)
return operations_plot(df_operations)
```