

Project: Software Engineering

Course DLMCSPSE

**Portfolio Project**

SENOUCI BRIKSI Djelloul

Registration number: 9225022

dd.mm.yyyy

Fribourg, Switzerland

# Table of contents

[Table of contents 2](#_Toc169251362)

[Image directory 3](#_Toc169251363)

[Glossary 3](#_Toc169251364)

[Introduction 4](#_Toc169251365)

[Requirements 5](#_Toc169251366)

[Design 6](#_Toc169251367)

[Realisation 7](#_Toc169251368)

[Testing 8](#_Toc169251369)

[Conclusion - Lessons Learned 9](#_Toc169251370)

[References 10](#_Toc169251371)

[Tmp 14](#_Toc169251372)

[Literature 15](#_Toc169251373)

# Image directory

[Figure 1 Vote Probabilities 14](#_Toc169251374)

# Glossary

|  |  |
| --- | --- |
| CAB |  |
| Complex Action |  |
| mobile SQLite database |  |
| PIS | Passenger Information System |
| Route |  |
| Train Number |  |

# Introduction

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

# Requirements

This section contains the functional and non-functional requirements of our project. The requirements below are ordered following a top-down view. This order does \*not\* define the priority of the requirement. It is intended to implement all requirements listed in this section within the realization of this project.

## Functional Requirements

The CAB is web-based application that allows browsing complex action trees.

The CAB loads all complex actions contained in a PIS mobile SQLite database.

Complex actions are linked to train numbers (routes). The CAB shall show the corresponding list of train numbers contained in the mobile SQLite database.

Train numbers (routes) are usually valid on certain dates or range of dates. The CAB shall give a possibility to choose a date to filter with.

Furthermore, the CAB shall give the possibility to filter the train number (route) to search into.

The CAB shall enable browsing a complex action tree by collapsing and expanding the tree nodes (complex actions).

How this tree of complex actions is visually presented will be defined during the implementation phase.

The CAB shall give the possibility to search complex actions by their attributes.

The following complex actions attributes (search criteria) are defined:

* 1111
* 2222
* 3333
* 4444
* 5555

The complete list of these attributes will be finalized during the implementation phase.

The CAB shall also give a way to check the plausibility of a given complex action tree.

The following plausibility checks are defined:

* 1111
* 2222
* 3333
* 4444
* 5555

The complete list of plausibility checks will be finalized during the implementation phase.

The following diagram shows the main features of the Complex Action Browser:

Une image contenant texte, capture d’écran, cercle, croquis

Description générée automatiquement

Figure 1 – CAB: Main Features

## Non-Functional Requirements

The CAB shall be operable on Chrome, Firefox and Edge browsers.

The CAB shall be able to load mobile SQLite databases up to 1 GB.

When collapsing or expanding a complex action tree, the CAB shall render with a maximum of 300ms response time.

If the mobile SQLite database cannot be read (eg. corrupt or wrong/unknown structure), the CAB shall show a corresponding error message.

The CAB shall present the features (browsing a complex action tree, searching possibilities, etc) in an intuitive and meaningful way. 🡪 how to test ?

The CAB shall be easily updated, when e.g. the structure of the mobile SQLite database changes.

# Design

## Scope and Context

The Complex Action Browser (orange box in Figure 2) is the scope of our project.

Une image contenant texte, capture d’écran, Police, logo

Description générée automatiquement

Figure 2 – CAB: Context Diagram

It is mainly accessed via a standard browser. It loads data from a mobile SQLite database.

The CAB does not provide nor use any other interface.

## Components

The CAB will be implemented as a backend and a frontend application.

The backend application is written in python, and uses the ‘django’ framework (django) to access the mobile SQLite database.

We have chosen python and django because we have some experience with programming in python. Furthermore, django already supports SQLite databases. Moreover, we found many examples in the net concerning the integration of django into google cloud. Indeed, it is required that our application must be hosted in a cloud.

The frontend application will be implemented in a classical js/html/css solution. We will use the ‘vue js’ framework (vue js) for the general web user interface, and the special library ‘d3 js’ (d3 js) to visualize and browse trees of elements.

We have chosen vue js because this library is already in use in our company. So, we can get support from experienced colleagues in this matter. The choice of the d3 js library was motivated by the fact that this library is great help when trying to visualize complex graphs (such as trees) in javascript.

Une image contenant texte, capture d’écran, diagramme, conception

Description générée automatiquement

Figure 2 – CAB: Component Diagram

The backend component will access the mobile SQLite database (sub-component ‘DB Reader’) to retrieve the train numbers (routes) and their corresponding complex actions. It converts and prepares these data in a json-structure (sub-componentn ‘Data Handler’), that is delivered to the frontend application.

The frontend component parses these json-data and presents them to the user to be viewed. The main feature here is the visualization of complex action trees (sub-component ‘Tree Viewer’). Moreover, the frontend application gives the possibility to search into train numbers and complex actions (sub-component ‘Searcher’), and also to check plausibility of complex actions (sub-component ‘Checker’).

# Realization and Project Plan

## Software Development Methodology

We will realize our project following Scrum agile methodology. There are many reasons for that:

* The main feature of our product is how to best visualize and browse complex action trees. The exact way on how to visualize such trees is not defined as the beginning of the project. We have to make several tests and iterations, to find the best way to achieve it. Thus, an agile approach is the best choice to do it.
* Our customers are software developers (of our company). We will be in close contact with them everyday to better match their needs. We need to quickly react to their needs and wishes (as long as within the scope of the requirements and allocated time). The agile approach will help us keep this flexibility.
* The customers (developers) will be able to use the very first deliveries of the product, which shall mainly contain the browsing of complex action trees (see plan below). This means that the product will be used even if not yet finalized. Developers will always get the latest – thus not yet final – version of the product, to play with. The agile approach is ideal for this.

We will be playing all the following roles during the realization of the product:

* Scrum Product Owner
* Scrum Master
* Developer
* Tester (though some developer colleagues will also test delivered versions regularly, but not as formally as we will do).

As we are the only person in the team, no daily Scrums (standups) will be conducted.

## Tools

Requirements

Plan

Coding

Testing

## Plan

Scrum Events: Sprint, Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective.

Scrum Artifacts: Product Backlog (Product Goal), Sprint Backlog (Sprint Goal), Increment (Definition of Done).

# Testing

# Conclusion - Lessons Learned

# References

# References

* Django Framework  
  <https://www.djangoproject.com/>
* Vue js  
  <https://vuejs.org/>
* D3 js  
  <https://d3js.org/>