SOFTWARE ENGINEERING 2: TRAVLENDAR+

IMPLEMENTATION AND TESTING DELIVERABLE

Agostini Andrea, Ciampiconi Lorenzo, Es-skidri Rachid

JANUARY 8, 2018

CONTENTS

TEAM

Andrea Agostini

Lorenzo Ciampiconi

Rachid Es-skidri

REPOSITORIES:

The complete code is available on GitHub, the android app source code at:

https://github.com/agostini95/travlendarAndroidClient

The server side at:

https://github.com/esskidri/AgostiniCiampiconiEs-skidri

The RASD and the DD are available at the same address inside the Delivery Folder page.

INTRODUCTION AND SCOPE

The purpose of this document is to provide an overview of the implementation and testing deliverable of the project. First we'll point out all the requirements and functionalities we implemented, as well as those we didn't implement, and try to explain the reasons behind these choices. Then we'll describe the adopted development frameworks, the structure of the source code and some information on how we performed the testing activity. Finally we will list all the necessary installation instructions.

PROJECT DESCRIPTION

ADOPTED DEVELOPMENT FRAMEWORKS:

For this section refer to the DD.

STRUCTURE OF THE SOURCE CODE:

The source code is divided into two parts, the server part and client part.

On the server side, the code is structured according to a classic web framework, in our case "Spring", in which there is a "handler" that is in fact the point of access to the server by the client. Client creates http GET Request to the server, which returns a standard Json format information as an answer, this solution is very scalable,it allows the server not to care about the type of client connected to it, so a future development (web app, IOS etc. ..) will be very simple due to the totally independence of the implemented server. The client has been developed following the latest google and material guidelines.

Server:

The server source code is divided in three main packages: **Model, Controller, Logic.**

The **Model** package include all the domain entities, the mapping with the DB by JPA (Hibernate), the CRUD repositories by spring ("**dao**" sub package) and classes that refer to the model adopted by the client(clientModel sub-pack).

The **Logic** package includes all the developed algorithms (transport solution calculation, events connection) described in the Design Document and the handling of Google API data, most important classes and interfaces are:

- **Calculator Core:** this interface is useful for the strategy pattern as described in the **Design Document** for the selection of meaning by the policy of the user;

- **EventConnector:** this class recognize the connection between event, recognize when to go home and handle event overlapping, the algorithm is described in the **Design Document (the first one)**.

- **MainLogic:** this is the class that has method statics to be called by the controller classes (Request Handler);

- **TransportSolutionCalculator:** this is probably the most important class and is in charge of the calculation of transport solution between events, the algorithm is described in the **Design Document (the second one)**.

There's also a sub-pacakge **"modelinterface"** that contains the interface used to mask the domain entities.

The **Controller** package contains the **DataManager** module that is connected with all the Dao interfaces that are properly autowired inside the class. The most important class is **RequestHandler** that contains all the http request mapping and some logic to call properly the **MainLogic** static method.

Client:

/main/res/layout:
This package contains all the xml files concerning the graphics component of the activity.

/main/res/menu:
This package contains all the menus used inside the application, the most relevant class is the *activity_navigation_drawer_drawer.xml* that describe the main menu shows in the navigation drawer.

/main/java/com.example.ago.travlendarAndroidClient/cardView:
This package contains two classes: cardModel and cardAdapter. The first is an abstraction of the concept of "card" in android and the second is the adapter to bind the view and the cards.

/main/java/com.example.ago.travlendarAndroidClient/fragment:
This package contains the fragments relevant for a correct representation of the listviews (*ListFragment, ItemAdapter*) and for building the transport solution view (*VerticalStepperAdapterDemoFragment*).

/main/java/com.example.ago.travlendarAndroidClient/model:
This package contains the representation of the domain in the client side. The objects are instantiated subsequently a deserialization process made in the serverStub.

/main/java/com.example.ago.travlendarAndroidClient/serverStub:
This package contains the classes that perform the communication with the server side. It composed by two classes:*HttpGetRequest* and *ServerConnection*.
The aim of the first is to provide a generic function to produce Get Request in background used exclusively by the second class that compose specific requests and ask the HttpGetRequest to produce them.

IMPLEMENTED REQUIREMENTS AND FUNCTIONALITIES

[G1] Allow a visitor to become a registered user:

- ✓ Account Manager. Developed and tested
- ✓ Mobile App component. Developed and tested

All module has been developed and everything is working fine.

[G2] Allow user to login to application.

- ✓ Account Manager. Developed and tested
- ✓ Mobile App component. Developed and tested

All module has been developed and everything is working fine.

[G3] Allow user to create a new event in the calendar.

- ✓ Data Manager. Developed and tested
- ✓ Mobile App component. Developed

All module has been developed and everything is working fine.

[G4] Allow user to modify an existing event of his/her calendar.

- ✓ Data Manager. Developed
- ✓ Mobile App component. To be developed GUI

The functionality is not granted at this state of developing, but is at a good state of developing.

[G5] Allow user to delete an existing event of his/her calendar.

- ✓ Data Manager. Developed
- ✓ Mobile App component.

All module has been developed and everything is working fine.

[G6] Allow user to consult the transport solutions between events in the calendar proposed by the system.

- ✓ Data Manager Developed and tested
- ✓ Route Manager Developed and tested
- ✓ Mobile App component. Developed

All module has been developed and everything is working fine.

[G7] Allow user to re-define dynamically his instant position to re-plan the transport solution.

- • Data Manager. Partially Developed
- ○ Mobile App component. Not Developed

Not developed at all.

[G8] Allow user to set free time (break) during the day schedule.

- ✓ Data Manager. Develop but not tested.
- ✓ Mobile App component. Develop but not tested.

All module has been developed, testing not finished.

[G9] Notifying events overlapping.

- ○ Data Manager.
- • Mobile App component. partially developed
- ○ Notifications Manager not developed

Not developed at all.

[G10] Notifying the time-unreachable event.

- ✓ Data Manager.
- ✓ Mobile App component.
- ○ Notifications Manager not developed

All module has been developed and everything is working fine. To be extended with push notification. In this version an user, after press the "show transport solution", will be notified with a message if there aren't transport solution to reach the event in time.

[G11] Allow user to configure transport preferences and external services to be used

- ✓ Data Manager.
- • Route Manager Missing interface with sharing API.
- ✓ -Mobile App component. Developed and tested

All module has been developed and everything is working fine for private means and policy, public means such as Sharing services are not interfaced with our system at this level.

[G12] Allow user to set an event as ending event.

- ✓ Data Manager. Developed and tested
- ✓ Mobile App component. Developed and tested

All module has been developed and everything is working fine.

[G13] Allow user to buy a ticket or to reserve a mean of transport of a suggested solution.

- • Data Manager. Partially developed.
- • Mobile App component. Partially developed
- • Sharing Services Interactions Manager Partially Developed

In this version we redirect the user to the web site of the relative transport agency.

At this state of the app release we develop the basic functionalities and reached goals that are crucial for the user experience (**G1, G2, G3, G5, G6**, **G8**) and some thing that can make our app more user friendly (**G10, G11, G12**).
G4 can be substituted at this state by combination of **G5** and **G3, G7** is an advanced goal but the server side is structured in a way that the developing of this functionality must be easy and many thing are already developed, the client for this goal have been not developed.
G9 is not developed in the client and in the server (notification module miss at all) but we don't think this corrupt the user experience because our server logic handle the eventual overlapping of events.
G13 : we redirect to the point closest to the online ticket shop based on the information received from the API.

TEST PERFORMED

We starting testing as we started developing every module.
The first module that has been developed is Data Manager with the integration between the SQL Database and JPA Spring Framework. The test that we have executed on this module aim to verify ACID properties so we tested:

- Adding of new entities in the database
- Deletion of existing entities with cascade options (for example the deletion of a user must delete every event, free time etc… or the deletion of an event must delete transport solution and so on.
- Updating of existing entities

The tests above described have all been checked and the module work as desired.
We tested the HTTP server verifying that the http request reaches as desired the server with a stable connection.

For server side logic we started doing some Unit test with JUNIT for simple Logic operations but some unexpected behaviors of Google Directions API in some particular use cases changes our direction of Test Driven development from automated to non-automated black box testing because often we changed very basic logic functions also on their expected results.

The tests on this module have different level of testing:

- Black Box testing for POJO mapping of google response: we tested that our functions map in every case (also when the request fail) Google JSON response and this was crucial for the developing of upper level functions.
- Black box testing for the Google response as well to understand the behavior of Google API for the information that are not included in Google Documentation such as:
    - Comparing results between the Google API answer and the usage of Google MAPS
    - Testing limit cases such as the change of a mean, the change of a BUS LINE in a response, the change of METRO LINE in a response.
    - The changing of returning parameters: Google doesn't return every time all the expected parameters (for example departing and arrival time) so we changed dynamically our google interpretation and tested limit cases.
- In parallel with the Google API interpretation (developing and test) we developed the module that uses the google response and tested if it returns valid results (compared with a valid user experience):

- Tested easy movement that must returns a solutions with every combination of mean of transport preferences.
- Tested the Ecologist policies and if it uses the preferred and most Ecologic (as indicated in the DB) mean available checking the results with google maps.
- Tested that a solution is coherent during a day: if you leave home without a car you cannot use this car until you come back to home!
- Tested particular cases:
- Transport between different cities
- Transport overseas (not supported by Google API)
- Transport in zones where there's no public transport coverage by Google
    - Overlapping between events
    - Cases in which two events totally collide
    - In case the user insert no end-event or just one (there's no day recognized)

For all of these test before we have a valid GUI and a valid connection between server and client we done test calling http request by browser and checking data consistency on the DB. The test accuracy has been improved after a stable GUI has been develop with consistent data representation, basic button to call desired function and the test has been extended to the connection between client request and server behavior with the above described case. All the above test have checked.

HOW TO INSTALL THE APP

To install the android app, download it from :
https://github.com/esskidri/AgostiniCiampiconiEs-skidri/blob/master/DeliveryFolder/

the file name is: "app-release (2).apk"

and install it into the device (the android version must be >= 7.0), don't forget to set in you device settings the possibility to "install app from unknown source".

Please be informed that we decide to deploy the server side application in our homemade server located in the Marche Region due to the educational trait of the project. By the way it could be more fragile and slower than a professional one.
Please contact us for every problem.