Travlendar+

Implementation and Test Document

Calzavara Filippo, Filaferro Giovanni, Benedetto Maria Nespoli

Link to the zip attachments:

`https://github.com/fila95/CalzavaraFilaferroNespoli/tree/master/DeliveryFolder`

Link to the source code:

`https://github.com/fila95/CalzavaraFilaferroNespoli/tree/master/Implementation`

Version 1.0.0

Contents

1 Introduction

1.1 Scope of the document

This document is the Implementation and Test Document (ITD) for Travlendar+, the system assigned to be developed. The document describes the characteristics of the developed system, the requirements that meets for its first release, possible future advancement, the adopted development frameworks, the structure of the code, tests and installation instruction

1.2 Definitions, Acronyms, Abbreviations

1.2.1 Definitions

Definition	Explanation
Appointment	A period of time in which something take place at a certain time
Travlendar+	The name of the platform to develop
Event	A synonym for appointment
System	A synonym of Travlendar+
Up Coming Event	A particular event that is expected to occur soon
Up Next Event	A synonym of Up Coming
User	A potential utilizer of this project
Ride	A service performed by a ride-sharing company and Cabs

1.2.2 Acronyms

Acronym	Explanation
API	Application programming interface
RASD	Requirement Analysis and Specification Document
OS	Operative System
APN	Apple Push Notification

1.2.3 Abbreviations

Abbreviation	Explanation
App	A synonym of Travlendar+
[Gn]	N-goal
[Rn]	N-functional requirement

1.3 Revision history

- 1.0.0 - Initial Version (01/12/2017)

1.4 Reference Documents

- RASD version 1.3.0

- Design Document 1.2.1

2 General description of the functionalities

In this section, the developing team states those goals/features described on the RASD that are implemented within the first release of the project and those that will (possibly) be implemented in the following updates. The first set of functionalities are those considered essential for engaging new users and therefore creating a costumer base.

2.1 First release of Travlendar+

As first release of the application, the developing team ensures to meet the following requirements:

- [G0][R1] Allow people to use the app without a login function
- [G1][R2][R3][R4][R5] Allow people to view the daily schedule with coming up events at the top
- [G2][R6][R7][R8] Allow people to view previous events
- [G3] Allow people to view the details of each event
  - [G3.1][R9][R11] Allow people to get travel information for a particular event
    - [G3.1.1] Allow people to get the ETA to the appointment
- [G4] Allow the users to create an event
  - [G4.1][R15] Allow the user to define a name/description of the appointment
  - [G4.2][R16] Allow the user to define the location
  - [G4.3][R17] Allow users to set a flexible timing
    - [G4.3.1] Allow the user to select a time window in which the event will occur
    - [G4.3.2] Allow the user to select how long the event will last
  - [G4.4] Allow the user to set a starting time and ending time on non-flexible events
  - [G4.6][R19] Allow the user to assign the event to a specific calendar
  - [G4.7][R20] Allow the user to define the mean of transportations that he/she wants to use for the event
- [G5][R22] Allow people to see daily events on a map
  - [G5.1][R21] Allow the user to have information about the status of the journey to make in order to reach the appointment's location
  - [G5.2] Allow the user to see previous daily events on a map

- [G6][R25] Allow the user to set preferences
  - [G6.2][R23] Allow the user to set the walkable distance limit
  - [G6.3][R23] Allow the user to set the maximum biking range
  - [G6.4] Allow users to set the owned bike and/or bike sharing providers.
  - [G6.5][R24] Allow users to set a time range in which they can take public transits
- [G7][R27][R28] Allow the users to create calendars
  - [G7.1] Allow users to color calendar
  - [G7.2] Allow users to change calendar's color and name

2.2 Second Release of Travlendar+

The developing team has chosen to not proceed with the implementation of the below functionalities mainly because of their presence is intended to be complementary to the application and therefore not accountable for an initial startup of the project.

- [G3.2][R14] Allow people to explore a specific route
- [G3.2.1][R10] Allow people to purchase the transit ticket. Chosen not to be developed because involves primitives of the Operation System
- [G3.2.2][R12] Allow people to open the sharing services/ride provider's app and call the vehicle. Chosen not to be developed because requires contacting the companies and ask for API use clearance
- [G4.5][R18] Allow the user to repeat the event regularly throughout the week
- [G6.1] Allow the user to set the eco-friendly mode. Chosen not to be developed in order to prioritize better quality of the scheduling
- [G6.6][R25] Allow users to set if they own a car and/or their favorite car sharing providers
- [G6.7][R26] Allow users to set the preferred ride sharing services
- [G8][R31] The system must take into account the weather forecast which choosing the suggested mean of transportation. Chosen not to be developed since it could be considered as an additional feature
- [G8][R29][R30][R31] Notify the user that it's time to leave for the appointment

A second release of the application will eventually include the above mentioned requirements.

Furthermore we believe that an incremental update of the app with additional functionalities overtime would define a specific strategy of cost amortization and it would bring constant maintenance which increases the final perceived value of the product.

3 Adopted development frameworks

In this section the developing team describes more in depth the frameworks used, adopted programming languages and eventual middleware, dividing the client implementation with the server implementation.

Above all, on top of these two system, the developing team as used some external tools to check and balance the developing process: *Travis CI* to test and deploy the code, *Codecov* to check the test coverage client side and server side, and finally *Gitential* to check the coding hours needed to develop this project (without listing documentation time) and the code volume.

3.1 Client side

The client application is designed to be utilized on *iOS* smartphones (iPhones). The code was build using Apple's proprietary IDE *XCode* hence tested with *XCTest*, a library developed by the same company that allows to create and run test units. The app was written using *Swift* as programming language.

CocoaPods was used to manage the dependency in order to have improve scalability performance as well as elegantly code.

Moreover, *MapKit* was used to display maps and satellite imagery directly on the app's interface. This framework was also used to provide points of interest on the map.

Finally, the client side database is based on *Realm*, a reactive lightweight DBMS to that allows the app to work seamlessly in weak signal and offline scenarios. It was chosen to be the main local DBMS thanks to its setup simplicity and highly compatibility with iOS systems.

Generally, the use of the above mentioned development frameworks allowed to have a better implementation of the code written and highly performance in term of speed and battery life. As drawback we recognize the highly specified platform used to developed the application, hence low portability, which means that if a future development includes an expansion towards other smartphone operational system the app would have to be reengineered.

3.2 Server side

The server side application is written in *Node.JS*, a server environment wrapped over *JavaScript V8 Runtime*. This solution was chosen because of the its ability to provide asynchronous event driven I/O APIs, the simplicity of set up the

lightweight that it implies and its ability achieve great horizontal scalability though multiple instances. Node.JS is also an open source platform.

Above all, *npm* was used as package manager for Node.JS. It allows to manage dependencies and packages within the project boundaries.

Express was used as web framework. This library provides a robust set of features for web and mobile applications, as well as the ability of creating robust API quickly and easily.

Also, because of its use simplicity, the *body-parser* library was used to parse incoming request bodies in a middleware before handling the request

Moreover, as described on the Design Document, the developing team has chosen to use PostgreSQL as online database. An *ORM* framework called *node-orm2* was utilized to create an object relational mapping of the data that allowed to threat them with object-oriented alike programming within the software. The module *node-apn* was used to communicate with Apple Push Notification.

Finally, for the testing part was used *node-supertest*, a node's framework that provides a high-level abstraction for testing HTTP requests, the *crypto* library was used to generate secure random access token and *node-uuidv4* was used to create testing user tokens. For debug purposes a *logger* middleware was used to track API requests.

The main advantage the frameworks and the middleware adopted is represented by the maintainability and reliability of such Open Source modules. However, this implies losing part of code's control.

4 Structure of the source code

As described in the Design Document, Travlendar+ is composed of a two tier architecture with distributed data and logic. The developed code follows different structure depending on tier analyzed

4.1 Client side

The developed code is available at the path */Travlendar*. The subfolder *UI Components* contains the swift code of the tools that regards the UI, such as the date picker calendar view. The subfolder *Background Fetch & Server* contains the codes used by background processes such as the Network operation and position handling. *CalendarViewController & Relative* and *SettingsViewController & Relative* contains foreground operations for calendars and settings.

4.2 Server side

On path */Implementation/Server* the code of the NodeJS server is available. The folder contains a file: *index.js* all startup functions and connection to the database. Another file called *model.js* contains the model of the database (ORM). The file *auth.js* contains the middleware that authenticate the API request only if access_token is valid, and will eventually truncate the connection

with a 401 or 403 error code if access_token is invalid/missing. Also *sendNotification.js* contains the code to connect at APN.

The folder *routes* contains all the routing endpoints of the API calls, executed though Express. Finally, the folder *test* contains the code of the testing.

5 Testing

The reader is invited to read Section 6.3 of the Design Document, which contains information about procedures and frameworks used to test the code written. Every test unit can be run by opening a command line session at the folder *Implementation/Test* and typing *npm test*.

5.1 Test Cases

The most significant test cases follow here below. When a test is passed, the output occurred is considered to be the output expected.

5.1.1 Case A001

Description	Should create a new user and a new access_token if a valid user_token is provided
Component tested	User Services –> Account Manager Module (Authentication)
Call	POST */api/v1/login*
Input given	access_token
Output expected	200, *String* user_token
Result	PASSED (41ms)

5.1.2 Case C001

Description	Should create a calendar if a valid access token is provided
Component tested	Event Services –> Calendar Module
Call	PUT */api/v1/calendars/*
Input given	access_token, name, color
Output expected	201, *Calendar* calendar
Result	PASSED (28ms)

5.1.3 Case C002

Description	Should return a list of calendars if a valid access token is provided
Component tested	Event Services –> Calendar Module
Call	GET */api/v1/calendars/*
Input given	access_token
Output expected	200, *List<Calendar>* calendars
Result	PASSED (27ms)

5.1.4 Case C002

Description	Should update a calendar if a valid access token is provided
Component tested	Event Services –> Calendar Module
Call	PATCH */api/v1/calendars/:id*
Input given	access_token, name, color
Output expected	200, *Calendar* calendar
Result	PASSED (35ms)

5.1.5 Case C003

Description	Should delete the calendar if a valid access token is provided
Component tested	Event Services –> Calendar Module
API Call	DELETE */api/v1/calendars/:id*
Input given	access_token
Output expected	204
Result	PASSED (35ms)

5.1.6 Case E001

Description	Should create an event with all the details if a valid access token is provided
Component tested	Event Services –> Event Manager Module
Call	PUT */api/v1/calendars/:id/events*
Input given	access_token, title, start_time, end_time, lat, lng, address*, duration*, repetitions*, transports*
Output expected	201, *Event event*
Result	PASSED (46ms)

* Could be null

5.1.7 Case E002

Description	Should return a list of events if a valid access token is provided with to and from parameters
Component tested	Event Services –> Event Manager Module
Call	GET */api/v1/calendars/:id/events*
Input given	access_token, from, to
Output expected	200, *List<Event> events*
Result	PASSED (54ms)

5.1.8 Case E003

Description	Should return a list of events if a valid access token is provided without to and from parameters
Component tested	Event Services –> Event Manager Module
Call	GET */api/v1/calendars/:id/events*
Input given	access_token
Output expected	200, *List<Event> events*
Result	PASSED (59ms)

5.1.9 Case E004

Description	Should modify the event if a valid access token is provided
Component tested	Event Services –> Event Manager Module
Call	PATCH */api/v1/calendars/:id/events/:id*
Input given	access_token, title, start_time, end_time, lat, lng, address*, duration*, repetitions*, transports*
Output expected	200, *Event event*
Result	PASSED (63ms)

* Could be null

5.1.10 Case E005

Description	Should delete the event if a valid access token is provided
Component tested	Event Services –> Event Manager Module
Call	DELETE */api/v1/calendars/:id/events/:id*
Input given	access_token
Output expected	204
Result	PASSED (64ms)

5.1.11 Case S001

Description	Should return the user's settings
Component tested	User Services –> Setting Module
Call	GET */api/v1/settings*
Input given	access_token
Output expected	200, Setting setting
Result	PASSED (27ms)

5.1.12 Case S002

Description	Should edit user's settings
Component tested	User Services –> Setting Module
Call	PATCH */api/v1/settings*
Input given	access_token, eco_mode*, max_walking_distance*, max_biking_distance*, start_public_transportation*, end_public_transportation*, enjoy_enabled*, car2go_enabled*, uber_enabled*, mobike_enabled*
Output expected	200, Setting setting
Result	PASSED (26ms)

* Could be null

6 Installation instructions

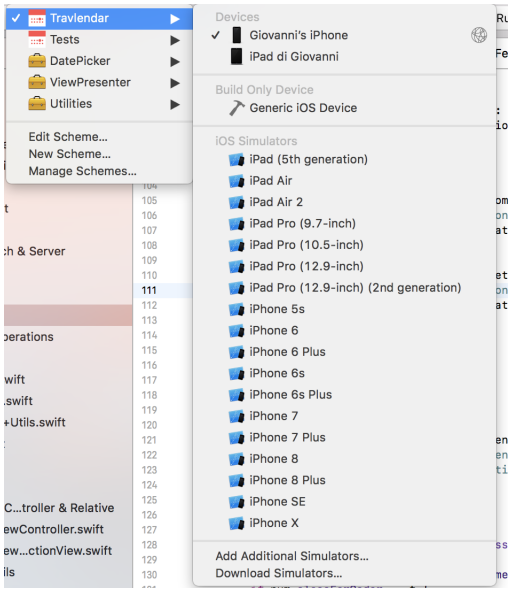
6.1 App installation

Please note that the application is compatible only with iOS platform's smartphones (iPhone) and it can only be compiled on OSX machines.

6.1.1 Compiling instructions

In case the reader would like to compile the app:

1. Download and install *XCode**: `https://developer.apple.com/xcode/ide/`
2. Download and install *Homebrew**: `https://brew.sh/`
3. Download and install *Cocoapods**: `https://cocoapods.org/`
4. Download the *App* package from the zip(s) attached to this file
5. Open the command line to */App/Travlendar/* and type *pod install* in order to install the dependencies.
6. Import the project "Travlendar+"
7. Product >> Run to compile >> Choose the platform (iPhone 8)



* If not already installed.

6.2 Server installation

1. Donwload and install an instance of PostgreSQL: `https://www.postgresql.org/`
2. Download and install *Node.JS* from the official NodeJS website: `https://nodejs.org/en/download/`
3. Download the *Server* package from the zip(s) attached to this file
4. Open the command line at the Server's folder
5. Type *npm install* to automatically install dependencies
6. Type *npm start* to start the Server Application

Please verify that an instance of PostgreSQL is running on port localhost 5432.

7 Effort spent

Time spent on this document: 10h

- Filippo Calzavara: 6.00h
- Giovanni Filaferro: 2.00h
- Benedetto Maria Nespoli: 2.00h

Time spent on the code: 110h

- Filippo Calzavara: 30.00h
- Giovanni Filaferro: 40.00h
- Benedetto Maria Nespoli: 40.00h