SCORE 2018

Travlendar by CPSoftware*

Anderson Morais Feitosa Júnior
Gabriel Lima Silva
Genilson Felismino de Almeida Júnior
Igor Santana Batista

Federal Institute of Alagoas (IFAL)
Center of Software Research (CPSoftware)
Maceió, Alagoas, Brazil

*Thanks to Prof. Dr. Flávio Medeiros for supporting us to develop the *Travlendar* project.

1 Introduction

In this section, we present the general information about the *Travlendar* project that we developed to participate in the SCORE contest 2018.

1.1 Project Overview

We implemented a solution for the *Travlendar* project supervised by Prof. Dr. Flávio Medeiros. Our solution provides a travel-time-aware calendar that automatically computes and accounts the travel time between appointments to minimize the chances of being late for appointments. It focuses on organizing the appointments of users and it alerts users when there is not enough time, considering different means of transportation, to get from one appointment location to another.

The team worked on the *Travlendar* project over six months. Throughout this time the team has worked hard and learned a lot about software engineering. In the following sections, we describe the processes, tools and techniques used, roles and responsibilities, our experiences and the resulting software.

1.2 Software Development Process

Software development processes describe the phases and activities that software development teams should perform when executing a software development project [20]. There are a number of software development processes, methods, and techniques in the software engineering literature, which consider different characteristics of specific software development projects, such as the *Rational Unified Process* [10, 9, 17], the *Scrum* framework [18, 5], *Agile* methods [13, 7], and others. To choose a suitable software development model for our team, we considered the following constraints of the team members:

- All the members participating in the team are students and do not work full-time for the project. The members are students of the same institution, the Federal Institute of Alagoas (IFAL), but they study in different campus. The travel time between campus takes around 50 minutes by car and almost one hour and a half by using public transportation;
- The team of students also have different timetables, as they participate in different courses, all related to computer science. Thus, they have different dates for exams, presentations, and activities.

Based on those constraints, it is not feasible to follow specific software models strictly, for example, there is no way of performing *Daily Scrum* [18]. Actually, it is not feasible to have regular meetings face-to-face. Thus, the constraints limit the usefulness of software processes, and require a large amount of work regarding resources planning.

Table 1: Tools used for the *Travlendar* project.

Tool	**Description**
Slack	Team communication
Skype	Retrospective and planning meetings
Visual Studio Code	IDE for *Ionic* development: *HTML*, *CSS*, and *JavaScript*
GitHub	Keep the history of the source code
Visual Studio Team Services	Create and keep the *Scrum* artifacts
Firebase	Database, storage, server-side functions, and support for social login

To execute the project, we decided to use practices of *Scrum* and *Agile* development, and applied tools that allowed us to develop the project virtually without face-to-face meetings. We also used *Software Prototyping* [19, 2, 4] techniques to define user interfaces incrementally, which were important to discuss about the project issues, and to get a better understanding about the functionalities.

As the first step, we used the documentation of the *Travlendar* project available at the *Score Website* to write User Stories [16] with the goal of staying focused on users and their needs. In our project the user stories represent the requirements of the system, and we used brainstormings and prototyping to elicit the software requirements. We used the concept of *Sprint* [18] in the project with a time box of one week. It was important to improve the team communication because we were developing the project in a totally distributed environment. At the end of every *sprint*, we performed the *sprint review* and *sprint retrospective*.

Next, we present the tools used in the *Travlendar* project and how we used each tool to run the project in a totally distributed environment.

1.3 Tools

To develop the *Travlendar* project in a distributed environment, it was necessary to adopt tools to make the development feasible. In Table 1, we present the tools that we used in the *Travlendar* project. The main communication of the team members was performed by using *Slack*.[1] We used it massively to discuss different things related to the project, such as to synchronize tasks, take questions, and discuss the functionalities.

Once a week, we planned a regular meeting on *Skype* to do the review and retrospective of the last *sprint*, and to perform planning for the new *sprint*. Our regular meeting normally took around two hours and it was the only time where all members were active at the same time.

[1] `https://slack.com/`

For development, we used *Visual Studio Code* for developing the frontend by using *Ionic*,[2] a multi-platform environment that integrates *JavaScript*, *CSS*, and *HTML*. In our backend, we also used *Visual Studio Code* to write *JavaScript* code to access *Firebase*,[3] which provides an infrastructure with database, storage, server-side functions, and support for social login. To keep the history of the source code, we used *GitHub*.

To create and keep the *Scrum* artifacts, such as the product backlog, the sprint backlog, and the task board, we used the *Microsoft Visual Studio Team Services* online.[4] In Figure 1, we present a screen of *visual studio* showing the *sprint backlog*. On the left side, we can see the list of *sprints*. On the right side, we can see the *scrum board* with *tasks* assigned to different team members. In general, all tools were required to run the project in a distributed environment.

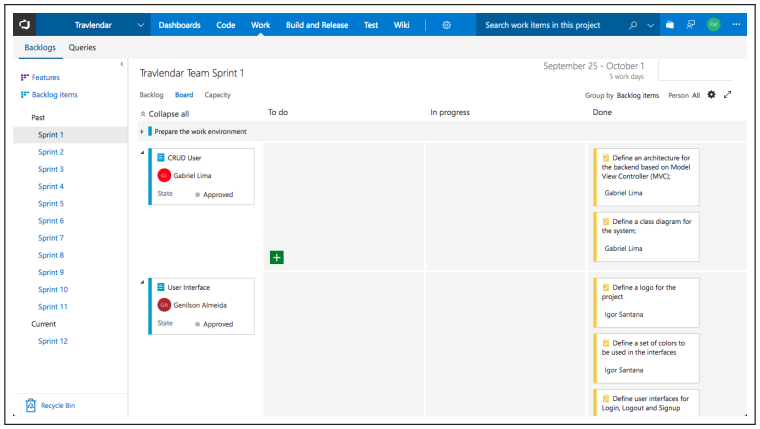


Figure 1: Microsoft visual studio with the *sprint backlog* and the *scrum board*.

1.4 Roles and Responsibilities

During the project, we tried to equally distribute the tasks between the team members. By performing our regular meeting to do the *sprint planning*, we attributed a set of tasks for every team member. The team members normally were assign to tasks based on their background experience to maximize the quality of the artifacts produced and to reduce time. In Table 2, we present the roles of the team members. Besides the roles assigned to each member, all members had the responsibility of discussing the problems of every member to find a solution for the problems quicker.

[2] `https://ionicframework.com/`
[3] `https://firebase.google.com/`
[4] `https://www.visualstudio.com/`

2 Overview of the Requirements

In this section, we present an overview of the requirements of the *Travlendar* project. A requirement is a service, function or feature that a user needs, which has to be available in the system [20]. Software requirements are classified into two categories:

1. Functional requirements, which express a function or feature, and define what is required in the system;

2. Non-functional Requirements, which define how well, or to what level a solution needs to behave.

To elicit the software requirements of the *Travlendar* project, we performed brainstormings guided by Prof. Dr. Flávio Medeiros, which advised our team as the project client, to get a general understanding about the project. As discussed in the Section 1.2, we also used prototyping to improve understanding and to elicit the software requirements, by creating the initial interfaces of the *Travlendar* system to guide our discussions in *Slack* and to get a common overview of the system shared by the whole team.

To describe the requirements of the project, we use *user stories* [12], which are popular in *agile software development* [3], as they focus on the viewpoint of a role who will use or be impacted by the solution, help to clarify the true reason for the requirement, and help to define high level requirements without necessarily going into low level detail too early [1]. Furthermore, for each user story, we defined *tasks* [11], which are pieces of activities that are required to get the user stories done. We present the *user stories* and their *tasks* next.

Table 2: Roles and responsibilities in the *Travlendar* project.

Name	**Role**
Anderson Feitosa	*Scrum* master, and developer
Gabriel Lima Silva	Software architect, and developer
Genilson Almeida	Developer
Igor Santana Batista	Designer, and developer

2.1 The User Entity

As the majority of software systems, the *Travlendar* project needs to keep control of users, by providing functionalities for registration, login, and logout.

User Story 1

As a user, I want to login and logout the system so that my schedule and personal information keep private.

Tasks

1.1: *Propose an interface for user signup, login, and logout based on the generic layout*
1.2: *Develop the functionalities to add, retrieve, update, and delete users*
1.3: *Implement the login and logout functionalities*
1.4: *Include social login with Facebook, Twitter, and Google Plus*
1.5: *Create test cases to verify the functionalities related to the user entity*

2.2 The Appointment Entity

Appointment is the main entity of the project. The end users will use the system to control what they need to do, when appointments will happen, and where they will take place. We planned to integrate our *Travlendar* project with *Google Calendar*, as it is one of the most popular calendars.

The key difference from *Travlendar* and other calendars is that our solution is concerned with the time to travel between appointment locations. So, it is important to alert users when there is not enough time to go from one location to another. Based on these points, we defined the following *user stories*.

User Story 2

As a user, I want to add, retrieve, update, and delete appointments and their locations so that I will not forget.

Tasks

2.1: *Propose an interface to add, retrieve, update and delete appointments based on the generic layout*
2.2: *Develop the functionalities to add, delete, update, and retrieve appointments*
2.3: *Create test cases to verify the functionalities of the appointment entity*

User Story 3

As a user, I want to import my Google Calendar events so that I will have an integrated environment.

Tasks

3.1: *Propose an interface to list events*
3.2: *Develop the functionality to import events*
3.3: *Create test cases to verify the functionality to import events*

User Story 4

As a user, I want to see warnings regarding the time to travel from my current location to the appointment location so that I will not be late.

Tasks

4.1: *Propose an interface to show warnings based on the generic layout*
4.2: *Develop the functionality to check the travel time between locations considering different means of transportation*
4.3: *Create test cases to verify the functionality*

2.3 Geolocation

Geolocation is a technique for identifying the geographical location of a person or device by means of digital information. It is getting popular in different kinds of applications and it is useful for the *Travlendar* project, as it takes the appointment locations into account.

User Story 5

As a user, I want to see my meeting locations on a map so that I will see my route along the day.

Tasks

5.1: *Propose an interface to show meeting locations on Maps*
5.2: *Develop the functionality by using the retrieving appointments functionality*

2.4 User Preferences

Many users like to customize the software systems according to their preferences. This way, as described in the project details at the *Score website*, users might save theirs preferences. The first kind of preferences is related to the means of transportation that users want to use.

User Story 6

As a user, I want to set my preferable (or non-preferred) means of transportation so that the system can consider that when making suggestions.

Tasks

6.1: *Propose an interface that users can activate and deactivate means of transportation*
6.2: *Allow users to set priorities*
6.3: *Develop the functionality to check the travel time between locations considering the preferences of users*
6.4: *Create test cases to verify the functionality*

User Story 7

As a user, I want to set the maximum walking and biking distances so that the system considers that when making suggestions.

Tasks

7.1: *Extend the preferences pages to include information about maximum distances for walking and biking*
7.2: *Develop the functionality to check the travel time between locations considering the preferences of users regarding the maximum walking and biking distances*
7.3: *Create test cases to verify the functionality*

User Story 8

As a user, I want to set the maximum time of the day to get public transportations so that the system considers that when making suggestions.

Tasks

8.1: *Extend the preferences pages to include information about time of day to use public transportation*
8.2: *Develop the functionality to check the travel time between locations considering the preferences of users regarding time of day to use public transportation*
8.3: *Create test cases to verify the functionality*

User Story 9

As a user, I want to specify my lunch periods and durations so that the system arranges them for me based on my appointments.

Tasks

9.1: *Extend the preferences pages to include information about lunch time and duration*
9.2: *Develop the functionality to arrange the lunch time for the user daily*
9.3: *Create test cases to verify the functionality*

2.5 Portability and Performance

In modern days, it is important to make our solutions available in a wide range of platforms. Thus, we made *Travlendar* to be available as *iOS*, *Android*, and *Browser* aplications. Furthermore, users require fast applications that they do not need to staying waiting for the responses. In this sense, we defined the following non-functional requirements.

User Story 10

As a user, I want to access the system in different platforms so that I can have access to it along the whole day.

Tasks

10.1: *Make the application available for iOS devices*
10.2: *Make the application available for Android devices*
10.3: *Make the application available for Web browsers*

User Story 11

As a user, I do not want to wait more the 5 seconds to get a response from the application so that I have a better user experience.

Tasks

11.1: *Develop test cases to ensure that all functionalities do not take longer than 5 seconds*

3 Architectural Design and Components

In this section, we present the architectural design and the compsonents.

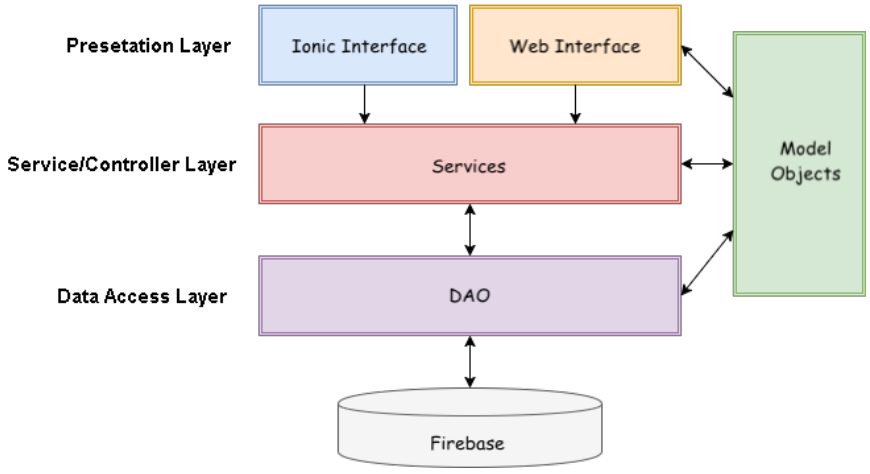
3.1 Overview

The architecture is divided by the concepts of *frontend* and *backend*. The terms refer to the separation of concerns between the presentation layer (*frontend*), and the data access layer (*backend*) of a piece of software. In particular, the *Travlendar* project provides a *backend*, developed by using *JavaScript*, and *Firebase*, which is used by three different *frontends* developed by using *HTML*, *CSS*, and *JavaScript*: (1) a web-based interface; (2) an *Android* application; and (3) an *iOS* application.

We defined the architecture based on our main design priorities: portability and performance. To improve performance, we made changes in our architecture, by substituting our backend written in *Java* by a new one written in *JavaScript*, based on *Firebase*, which is capable of handling requests asynchronously, helping to improve a better performance.

In Figure 2, we present the architecture of the *Travlendar* project. The architecture is based on the pattern Model View Controller (MVC). The data access and service layers, and the database are part of the *backend* of the project, while the presentation layer is part of the *frontend*. We represent the *iOS* and *Android* applications with the *Ionic Interface*.

Figure 2: The architecture based on the pattern Model View Controller (MVC).

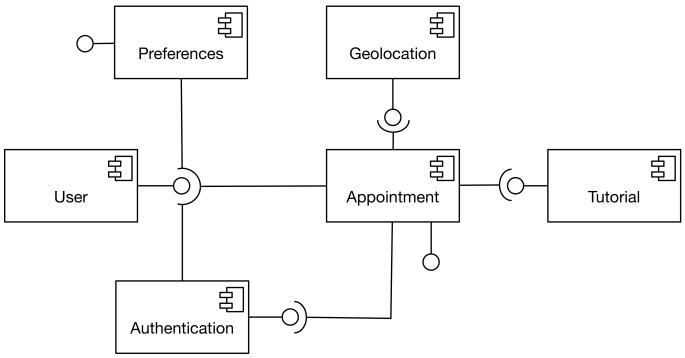


3.2 Components

In this section, we present the details of every component in the architecture of the system. In Figure 3, we present the components diagram, showing how the components communicate with each other. The appointment component is the main module, it uses the user and authentication components to check whether users are logged and to obtain information about the users. Futhermore, the appointment component also uses the geolocation component to get information about the location and travel time between appointment locations.

We design the architecture of the system in a modularized way by using software components, which contain well-established responsibilities. In this sense, changes in the requirements will affect one or a few components of the system, minimizing the costs of software maintainability. We present details of the components next.

Figure 3: Component diagram of the project.



3.2.1 Tutorial Component

This component presents to the user the main functionalities of the *Travlendar* system. For instance, In Figure 4, we present the first page of the tutorial component. After clicking the buttom, the user sees the main functionalities of the system. After present the tutorial pages, this components calls the authentication component, and the user is redirected to the login page that we present in the next section.

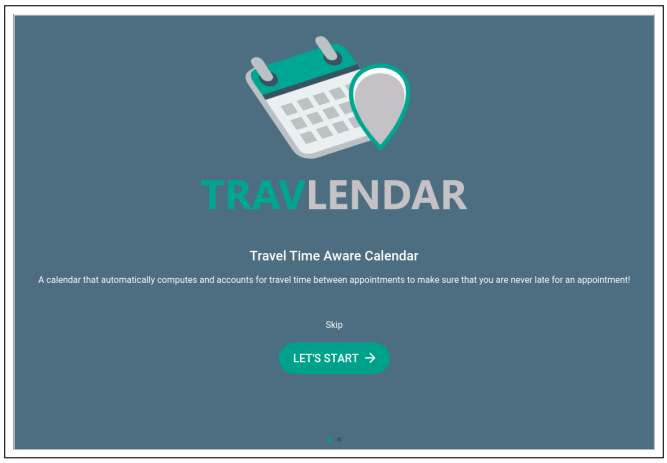


Figure 4: The first page of the tutorial component.

3.2.2 Authentication Component

The authentication component also implements social login, which provides to users to possibility to login the system by using their *Facebook*, *Google Plus*, and *Twitter* accounts, as we present in Figure 5.

All components of the system provide responsive web pages that fit well in different screen sizes. For instance, in Figure 5, we present the user login pages by running the system on an *iPad* device. In Figure 6, we present the social login page running on *Chrome Web browser*. When the user clicks on the social login button, the system shows the login options that we can see on the bottom.

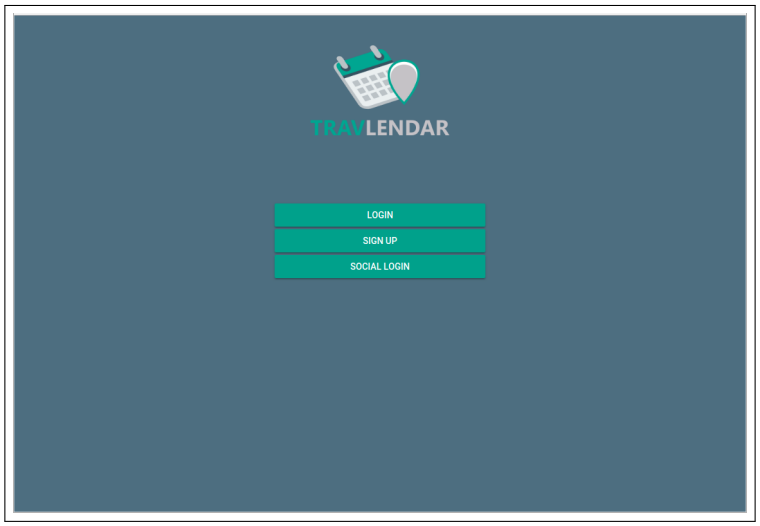


Figure 5: Login page of the *Travlendar* project running on an *iPad* device.

3.2.3 User Component

The user component is responsible for providing the functionalities to add, retrieve, delete, and update users. The users can register themselves by clicking on the sing up button at the login page presented in Figure 5. After registration the user may modify their personal information, i.e., the updating functionality. The system administrators use the delete and retrieve functionalities.

3.2.4 Appointment Component

The appointment component is responsible for providing the main functionalities of the system. That is, the functionalities to add, retrieve, delete, and update user appointments. In Figure 7, we present the main page of the system, which shows the appointment registered in the database. On the right side, we can see the plus button to add appointments, and by clicking on each appointment we can access the update and delete functionalities. Furthermore, notice that there are green and red arrows on the right side, which tell the users when there is not enough time to go from one appointment location to another. For instance, there is a red arrow in the first appointment, showing the user that it is not possible to arrive on that appointment on time based the user current location.

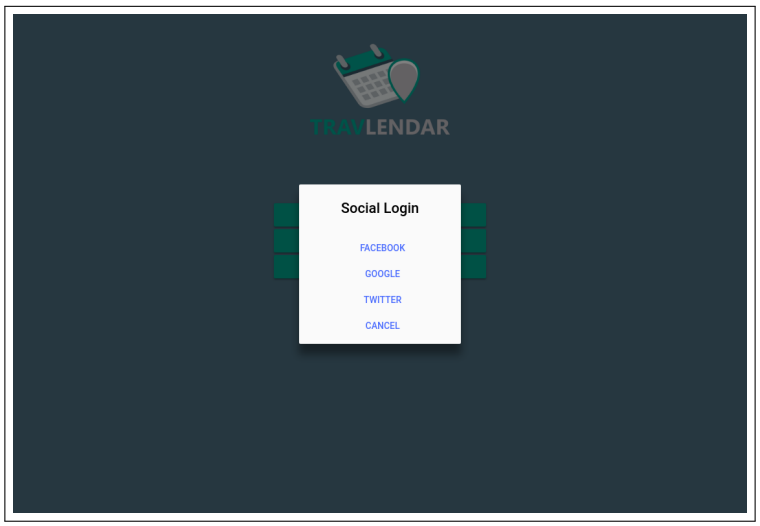


Figure 6: Social login page of the *Travlendar* project running on *Chrome*.

3.2.5 Geolocation Component

The geolocation component is responsible for providing geolocation information for the users. It is a crucial component of the system, as it provides the routes between appointment locations and travel-time information. We developed it by using the *Google Maps API*. In Figure 8, we present the route maps page, which shows the locations of the user appointments for a specific date. We use letters A, B, .., E to represent the chronological order of appointment locations.

3.2.6 Preferences Component

The user preferences component is responsible for keeping the user preferences with regards to: (1) lunch time and duration; (2) maximum time to get public transportations; (3) number of days to list appointments in the main page, for instance, users can see only the appointments of the day, or also the appointment for the next two days; (4) maximum distances for walking and biking; and (5) preferable means of transportation. In Figure 9 (a), we present the preferences screen running on an *iPhone* device. In Figure 9 (b) and (c), we present the page to integrate *Google Calendar* events, and the route map page running on an *iPhone* respectively.

3.2.7 Model Objects

For every entity of the project, we defined a *JavaScript* class for representation. In particular, *Travlendar* requires the following model entities: appointment, location, user, user preferences, and mean of transportation. In Figure 10, we

Figure 7: Main page of the system with the appointments of a user.

present how the entities relates to each other. As we can see, we defined four relationships:

1. One to many between location and appointment;
2. One to one between user and user preferences;
3. One to many between user and appointment;
4. One to many between user preferences and mean of transportation.

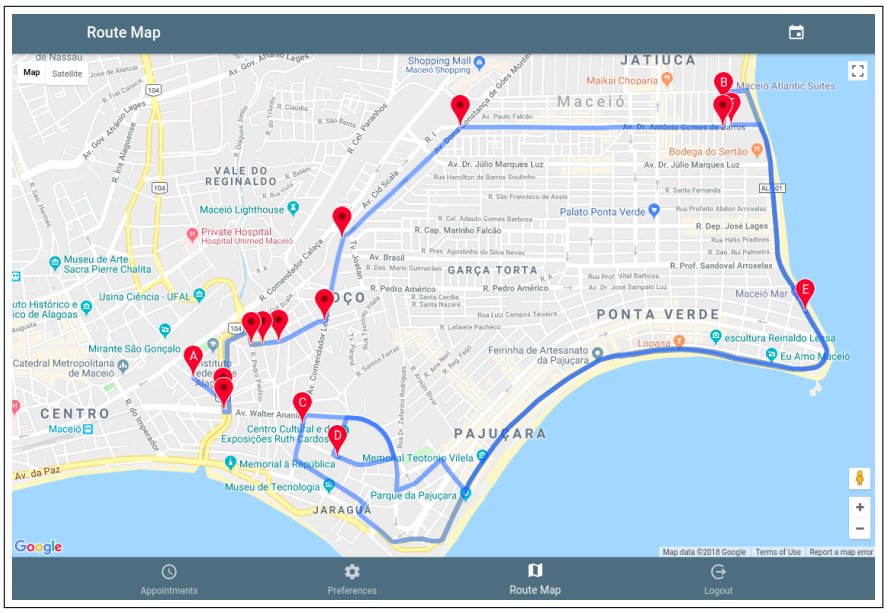


Figure 8: Maps page with the locations of appointments for a specific date.

4 Testing

In this section, we present the test strategy used in the *Travlendar* project.

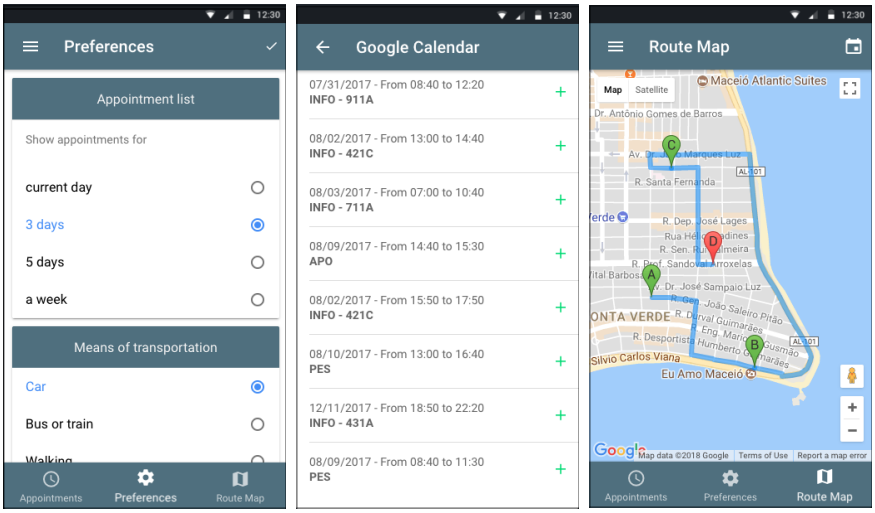
4.1 Test Approach

Our development includes test cases for testing the functionalities of the system. Thus, we developed test cases for every individual component: user, preferences, authentication, tutorial, geolocation, and appointment. We use a unit test strategy, which is a software testing method by which individual units of the source code are tested to determine whether they behave correctly according to the requirements of the system [8, 15, 6, 14].

We also invited people to use the system and give feedback. The most important feedback was regarding performance and visual components. To address performance, we changed our backend and now we are using Firebase, a platform that provides a real-time database capable of resolving queries in an asynchronous way, giving us a better performance. Furthermore, we changed several screens to address the comments about the visual interface.

4.2 Testing Tools and Environment

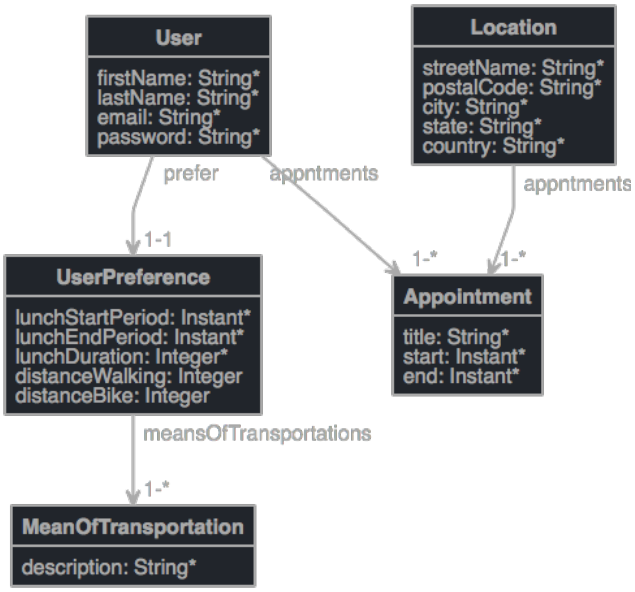
To test the *Travlendar* system, we created tests cases in *JavaScript* for the functionalities of the *backend* and *frontend*. We run the test cases in three different environments focusing on the *Google Chrome Web browser*.



(a) (b) (c)

Figure 9: *Travlendar* project running on a smartphone device with the following screens: (a) User Preferences, (b) *Google Calendar*, and (c) Route Maps.

Figure 10: The *Travlendar* Entity Relationship (ER) model.



4.3 Test Cases

As discussed, we developed test cases for every component of the system. To illustrate the test cases, we present a test case associated with the *User Story 1* of the user and authentication components.

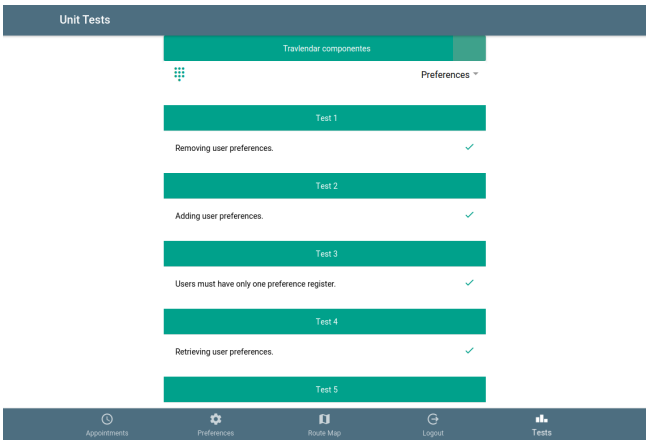
```
export class TestsPage {

 runTestUserStory1 () {
  // Registering a user
  userComponent.createUser('user@domain.com', 'x')
  .then(success => {
     this.tests.push({
       description: 'Adding_user.',
       result: true
     });
   })
   .catch(error => {
     this.tests.push({
       description: 'Adding_user.',
       result: false
     });
   });

   // Using user information to login
   authComponent.login('user@domain.com', 'x')
   .then((user) => {
     this.tests.push({
       description: 'User_login.',
       result: true
     });
   })
   .catch(error => {
     this.tests.push({
       description: 'User_login.',
       result: false
     });
   });
 }
}
```

We created a page that runs the test cases, showing the status of the system during runtime. In Figure 11, we present this screen by running the test cases of the preferences components in which all tests cases have run successfully.

Figure 11: The *Travlendar* test page.



4.4 Lessons Learned

During the project, every team member has learned several lessons as individual and as a team. The most important lesson learned was the importance of communication. As we run the project in a distributed environment, it was extremely necessary to adopt tools to improve the team communication and to share the source code and artifacts.

The second lesson was related to planning, everything seems to take longer than you have planned. As all team members did not work full-time for the project, it is difficult to make estimations. The team members participate in different courses and have different timetables, so it was always difficult to keep the project up and running during exams.

Overall, we believe that the execution of the project was very successfully despite of all difficulties that we faced during the project. We know that even after many tests, there will always be bugs, but we are fixing the known errors, and looking for what is left to improve the quality of the system.

References

[1] Agile Business Consortium. Requirements and user stories. `https://www.agilebusiness.org/content/requirements-and-user-stories`.

[2] Arnowitz, J., Arent, M., and Berger, N. *Effective Prototyping for Software Makers*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[3] Cockburn, A. *Agile Software Development*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[4] Davis, A. M. Operational prototyping: A new development approach. *IEEE Software 9*, 5 (Sept. 1992), 70–78.

[5] Green, M. D. *Scrum: Novice to Ninja Methods for Agile, Powerful Development*, 1st ed. Sitepoint, 2016.

[6] Hamill, P. *Unit Test Frameworks*, first ed. O'Reilly, 2004.

[7] Highsmith, J. *Agile Software Development Ecosystems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[8] Kolawa, A., and Huizinga, D. *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press, 2007.

[9] Kroll, P., and Kruchten, P. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[10] Kruchten, P. *The Rational Unified Process: An Introduction*, 3 ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[11] Liskin, O., Schneider, K., Fagerholm, F., and Münch, J. Understanding the role of requirements artifacts in kanban. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering* (New York, NY, USA, 2014), CHASE 2014, ACM, pp. 56–63.

[12] Mahnič, V., and Hovelja, T. On using planning poker for estimating user stories. *J. Syst. Softw. 85*, 9 (Sept. 2012), 2086–2095.

[13] Martin, R. C. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003.

[14] Massol, V., and Husted, T. *JUnit in Action*. Manning Publications Co., Greenwich, CT, USA, 2003.

[15] Osherove, R. *The Art of Unit Testing: With Examples in .Net*, 1st ed. Manning Publications Co., Greenwich, CT, USA, 2009.

[16] Patton, J., and Economy, P. *User Story Mapping: Discover the Whole Story, Build the Right Product*, 1st ed. O'Reilly Media, Inc., 2014.

[17] Schwaber, K. *The Enterprise and Scrum*, first ed. Microsoft Press, Redmond, WA, USA, 2007.

[18] Schwaber, K., and Beedle, M. *Agile Software Development with Scrum*, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[19] Smith, M. F. *Software Prototyping: Adoption, Practice and Management*. McGraw-Hill, Inc., New York, NY, USA, 1991.

[20] Sommerville, I. *Software Engineering*, 9th ed. Addison-Wesley Publishing Company, USA, 2010.