Politecnico di Milano

POLITECNICO
MILANO 1863

Software Engineering 2 Course

Travlendar +

Design Document

di
Gianluigi Oliva, Marco Mussi e Lukasz Moskwa

Github: https://github.com/Gigioliva/OlivaMussiMoskwa
Heroku: http://travlendarmom.herokuapp.com/

January 7, 2018

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 Definitions, Acronyms, Abbreviations . . . 4
- 1.4 Revision History . . . 6
- 1.5 Reference Documents . . . 6
- 1.6 Document Structure . . . 6

2 Architectural Design **8**
- 2.1 Overview: Highlevel components and their interaction . . . 8
- 2.2 Component View . . . 11
- 2.3 Deployment View . . . 17
- 2.4 Runtime View . . . 18
- 2.5 Component Interfaces . . . 29
- 2.6 Selected Architectural styles and patterns . . . 32
- 2.7 Other design decisions . . . 33

3 Algorithm Design **34**

4 User Interface Design **37**

5 Requirements Traceability **39**

6 Implementation,Integration and Test Plan **42**

7 Effort Spent **45**

8 References **46**

1 Introduction

1.1 Purpose

The purpose of this document is to provide technical details about the information contained in RASD of Travelandar+ Application and lead the developers that viewing this document can develop the application in the correct way.

The output of this document is an architectural description that shows all the critical feature of the problem taken into account. In particular in this document will be treated:

- The high architectural level;
- The design patterns;
- The main component and the interface that the application provides;
- The runtime behavior;
- The data structure used for the developing;

The relation among the different models is represented by using UML diagram and other useful kind of diagram that show the structure of the system (Entity – Relation diagram).

1.2 Scope

The scope of this application (Travlendar +) is provide a tool for the target users to schedule in an effective way their time optimizing the travels.

Travlendar+ allows to create a calendar which fits the meetings and other kind of commitments. The key feature of this application is to determine if the meeting location is reachable in the scheduled time and then provide a fast way to get to the destination, otherwise it notify the user that it is not possibile by any mean to fullfill the request. Furthermore the application also arranges the schedule in a flexible way, taking into account weather condition and user's preferences as well, for some kind of events (like the lunch) and offers the possibility to slightly change its time.

With Travlendar+ is possible as well to buy public transport's tickets on the fly for the journey. If the user requires often the same path, he is suggested by the application

to buy the best offered subscription.

The architecture must be designed with the intent of being maintainable and extensible. For this purpose, we will implement some known design patterns which will facilitate the development of the system.

1.3 Definitions, Acronyms, Abbreviations

Definitions

- **Platform**: system/application as a whole.
- **User**: An end user who is currently registered to the Travlendar+ application and has credentials to access.
- **Guest**: Person not registered yet and with limited access to features.
- **Event**: A scheduled meeting or other kind of appointment a user has to attend.
- **Journey**: The path chosen by the application as the one with all the fullfilled requirements.
- **Bad Weather**: A weather that prevents the user from choosing some path options. The listed bad weathers are snow, rain, storm and others.
- **Framework**: Reusable set of libraries or classes for a software system.
- **Cross-Platform**: software able to run on different platforms with same code.
- **Port**: in the internet protocol suite, it is an endpoint of communication in operating system
- **Web-Socket**:is a computer communications protocol, providing full-duplex communication channels over a single TCP connection
- **REST**: is a way of providing interoperability between computer systems on the Internet.
- **RESTful**: a system using REST
- **Client Server Architecture**: is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.
- **Client Tier**: is the tier/layer involving the client.
- **Fat Client**: is a client computer in client–server architecture or networks that typically provides rich functionality independent of the central server.

- **Layer**:In a logical multilayered architecture for an information system with an object-oriented design, the main layers are Presentation layer, Application layer and Data access layer.

Acronyms

- **RASD**: Requirements Analysis and Specification Document
- **DB**: Database
- **DBMS**: Database Management System
- **OS**: Operating System
- **HTML**: HyperText Markup Language
- **CSS**: Cascading Style Sheets
- **JS**: JavaScript
- **JSON**: JavaScript Object Notation
- **API**: Application Programming Interface
- **IDE**: Integrated Development Environment
- **RAM**: Random Access Memory
- **HTTP**: HyperText Transfer Protocol
- **HTTPS**: HyperText Transfer Protocol Secure
- **TCP**: Transmission Control Protocol
- **ACID**: Atomicity Consistency Isolation and Durability
- **DD**: Design Document
- **MVC**: Model View Component
- **UX**: User Experience
- **URL**: Uniform Resource Locator
- **OLTP**: On Line Transaction Processing

Abbreviations

- **Gn**: n-th goal
- **Rn**: n-th functional requirement
- **Dn**: n-th domain
- **Mn**: n-th mockup
- **WebApp**: WebApplication

1.4 Revision History

Version, date and summary

Version	Date	Summary
1.0.0	January 7, 2018	First release of this document

1.5 Reference Documents

We used the following documents:

1. The orginal Travlendar application:
   http://score-contest.org/2018/projects/travlendar.php
2. The revised document of the assignment:
   https://goo.gl/9m1ojy
3. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
4. IEEE Std 1016tm-2009 Standard for Information Tecnology-System Design-Software Design Descriptions.

1.6 Document Structure

- **Introduction**: This section provides a general introduction and overview of the Design Document
- **Architectural Design**: Overall description of the main system component and relationship between them. This part is divided in various subsection in which are show the main component view, deployment view, runtime view, component interface and design patterns.

- **Algorithm Design**: Overall description of the high level details about the most critical part of the algorithms that must be implemented for the system.
- **User Interface Design**: Overview of the way the user can interact with the system and the response of the application to the user input.
- **Requirements Traceability**: Overview of how the design part descripted in this document satisfy the functional requirement and the constraints explain the RASD.
- **Implementation, Integration and Test Plan**: Identification of the order we will use to realize the subcomponents of the system and how we will integrate and test them together.
- **Effort Spent**: Time and resource effort during the development of the application.
- **References**: References and software used during the process of creation of the system.

2 Architectural Design

2.1 Overview: Highlevel components and their interaction

The Travlander+ system has a three tier architecture:

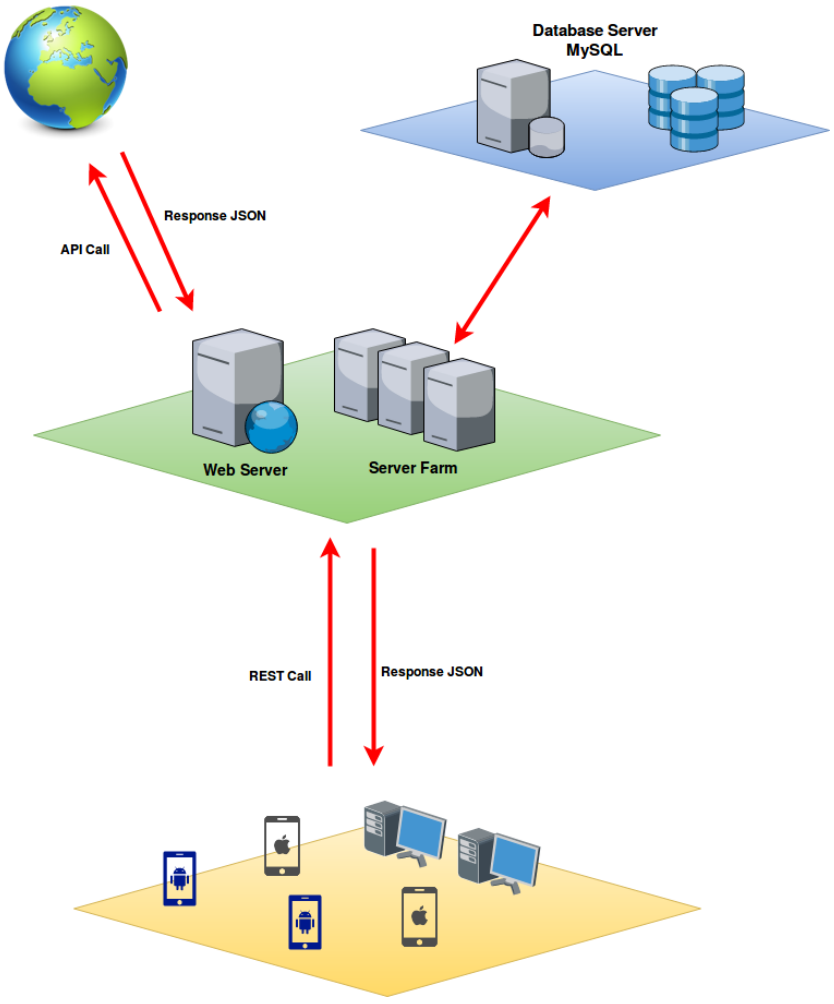

Figure 2.1: Graphical description of the High Level Architecture and relationship of the tiers

The client has only the presentation layer realized with a dynamic GUI, created due

to information retrieved by the server. The communication between client and server happens due to REST calls and JSON response.

The second tier contains the application logic layer and the methods for the communication with the Database Server.

The third layer stores all the usefull data to the user which allows the system to perform operations and manage separatly different users.

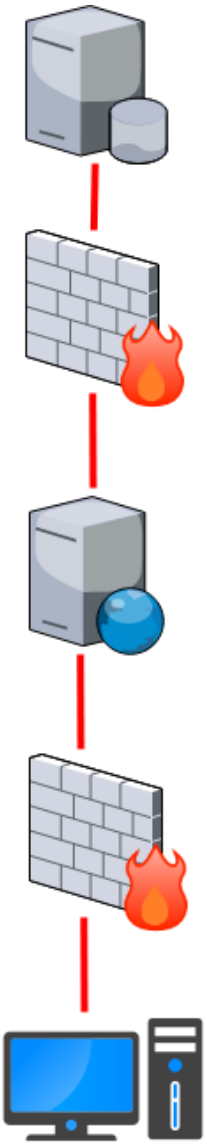

Figure 2.2: Graphical representation of the position of the firewalls in our network

Between all the layers there are firewalls in order to increase the safety of communi-

cation and increase the reliability of the overall system.

High level component

The high level components architecture is composed of four different elements types. The main element is a singleton, the web server. Clients communicate only with the web application server and this kind of communication can be performed throught a PC, a mobile device and all devices which allow the execution of JavaScript.

When the client has to perform a request, it sends a well formatted JSON string to the web application that will computate it, interacting as well with third part services and with the database if needed to save or to request usefull information, and provide a response.

The client and the server communicate using both synchronous messages and asynchronous ones based on the purpose. For example, if the user wants to save some information, a synchronous communication will be done. Otherwise if, for example, a client requires the information of a map (that is retrieved with the Google's API) an asynchronous communication will be done.

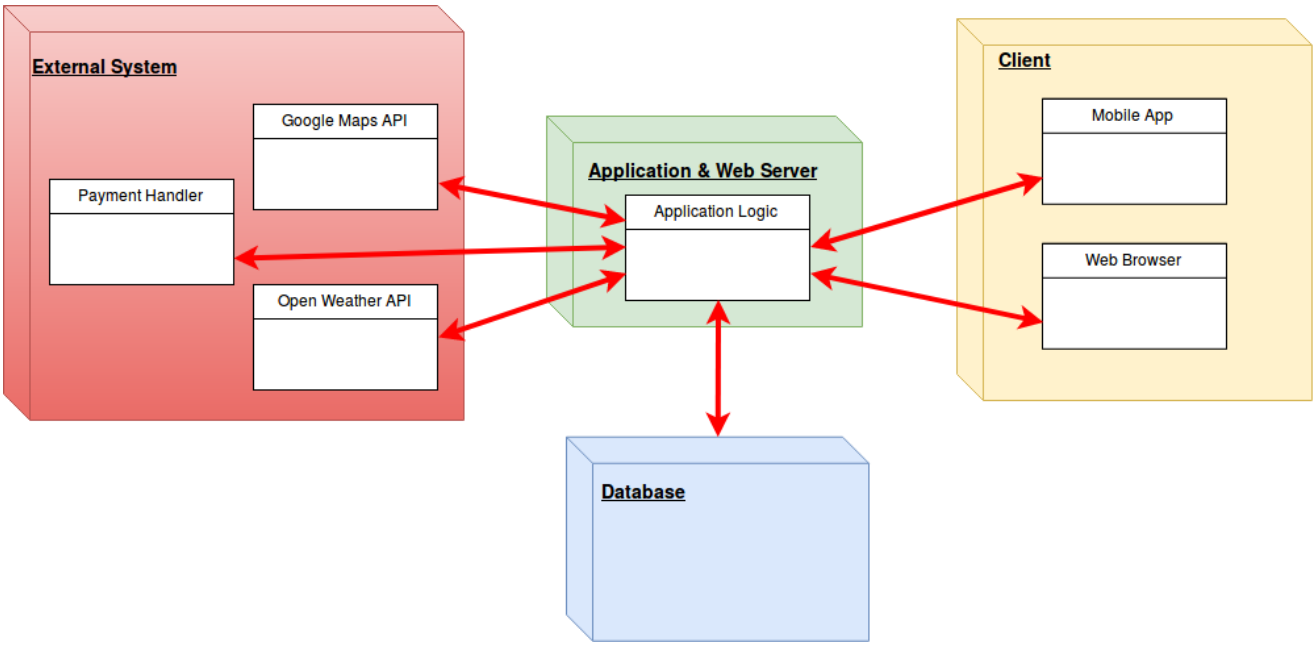

Figure 2.3: Graphical representation of the High Level components

2.2 Component View

In this section we will explain in detail all the major sub-part of the system and the way they interact each other.

DBMS Server

Into the database server we want to use a relational DBMS component, like MySQL, to manage the interaction with the database. In that way we can guarantee the ACID properties of the transactions executed.

As seen in the previous schema (Fig 2.2) the DBMS server can communicate only with the Application server through a firewall. Sensible data such as passwords and personal information must be encrypted properly before being stored. Users must be granted access only upon provision of correct and valid credentials.

The DBMS will use a high level schema like the one represented in the following Entity-Relationship diagram.

The main components of the DBMS server are:

- **DBMS**:

  This component deals with the operations of read and write of the data stored into the Database. It's implemented as OLTP, which is used to refer to processing in which the system responds immediately to user requests.

- **DBRequestHandler**:

  This module converts requests from the Application Server into SQL queries which are executed by the DBMS.

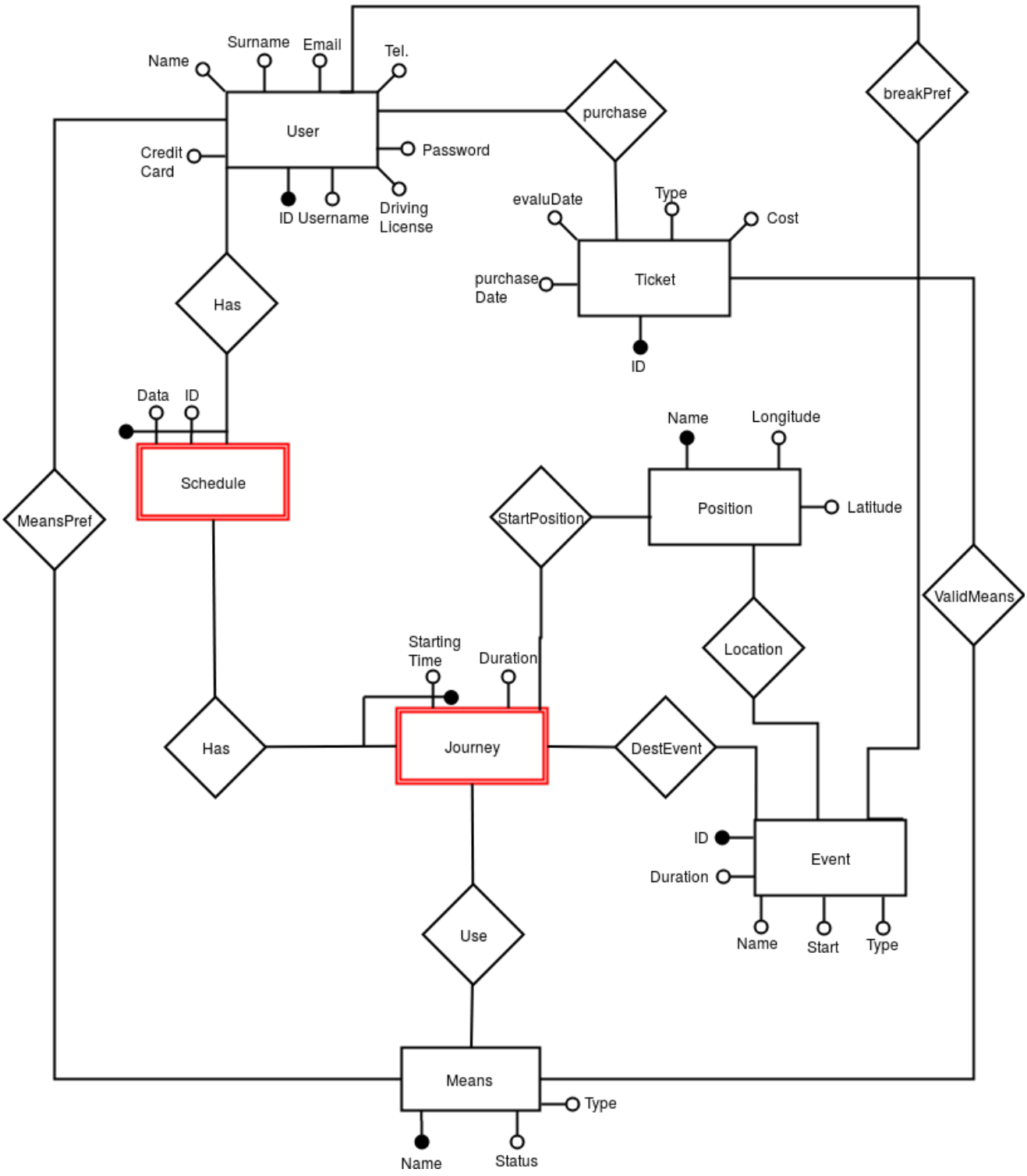

Figure 2.4: E-R Diagram of the DBMS

Application and Web Server

For our purpose we decided to merge together the Application and Web server. This choice is based on the three-tier architecture model. The Application server is the middle-tier which implements the business logic of the WebApp. It also uses a Script Engine mechanism to provide dynamic real-time contents to the users.

In this layer we can also find a Web Server which is the component that handles the requests from the clients. These requests are based both on HTTP standard protocol and on the Web-Socket protocol. Once the page request is completed, a connection is estabilished with the Web-Socket protocol in order to have a full-duplex communication and refresh the web page content without reloading the whole page.

The system must also provide a way to communicate with external systems with the purpose of collecting important data and information which will allow the correct execution of operations.

The main components are:

- **UserManager**:

  This module will manage all the sensitive information about the users. It will also allow the users to see and modify them when requested. It mananages as well the Login and Registration procedures.

- **ScheduleManager**:

  This module contains the business and application logic that performs the computation of the best travel path and also verifies the schedule can fit in the user's timetable. In order to provide the best journey path, this module takes into account the weather forecast as well through the ExternalRequestManager.

- **ExternalRequestManager**:

  This module handles all the communications with third part services and retrives from them important information which will be passed to other system's modules.

- **PaymentHandler**:

  This module deals with all the procedures necessary to perform a payment. This occurs every time a user wants to purchase a ticket or a subscription for a specific journey in his schedule. In some cases, with third part services like car sharing or bike sharing, this module redirects those information directly to the required

service page.

- **SecurityAuthenticator**:

  This module manages all the security policies in order to prevent fraudulent accesses and guarantee the privacy of the users. It also provide a secure way to connect to the web application using a token-based mechanism in a Web-Socket context.

- **NotificationManager**:

  The Notification Manager purpose is to handle all the notifications to the user and communicate them even when the application is closed or in background. It also report the information on different priority levels.

- **RequestController**:

  This module has the task of managing all the user's requests and handle them to the proper system modules in order to fullfill their queries.

- **DataHandler**:

  This module allows the exchange of data and information between the Application/WebServer and the Database Server. While the communication between the Client and the Web Application is made through a HTTPS standard protocol to ensure the connection safety, the communication between the Application Server and the Database is realized with HTTP. In fact, the firewall is configured to deny all the possible connections coming from sources different than the WebApp.

Client

The client in our model architecture is a Thin Client, as the whole business logic and calculations are made on the Application Server side. Therefore the client realizes only the Presentation Layer.

The Client side is meant for the mobile and desktop users and, in order to access to all the system features, it has to have implemented necessary methods to retrieve information from its GPS hardware component.

The client's main components are:

- **PositionController**:

  This module takes into account the position of the used device when the ApplicationController needs it. The user is asked either to insert manually his position or grant to the device the permission of using GPS position data.

- **UIManager**:

  This module is used to manage all the contents displayed to the user in a certain page he visits. It takes data from the ApplicationController and show them. It is also responsible for the handling of user's input and form fullfilling.

- **NotificationMonitor**:

  This software component has the purpose of listen the Application server notifications and display them immediatly or, if expected, store them for a while and then display them when scheduled.

- **ApplicationController**:

  This module acts as core of our client implementation. It contains the client's logic and allows an efficent communication between modules listed above.

  This software component deals with all the user's request and forward them through a specific protocol in a public network to the Application Server. It also handles the retrived data received as responses from the Server side, and send them to the ApplicationController to be shown.

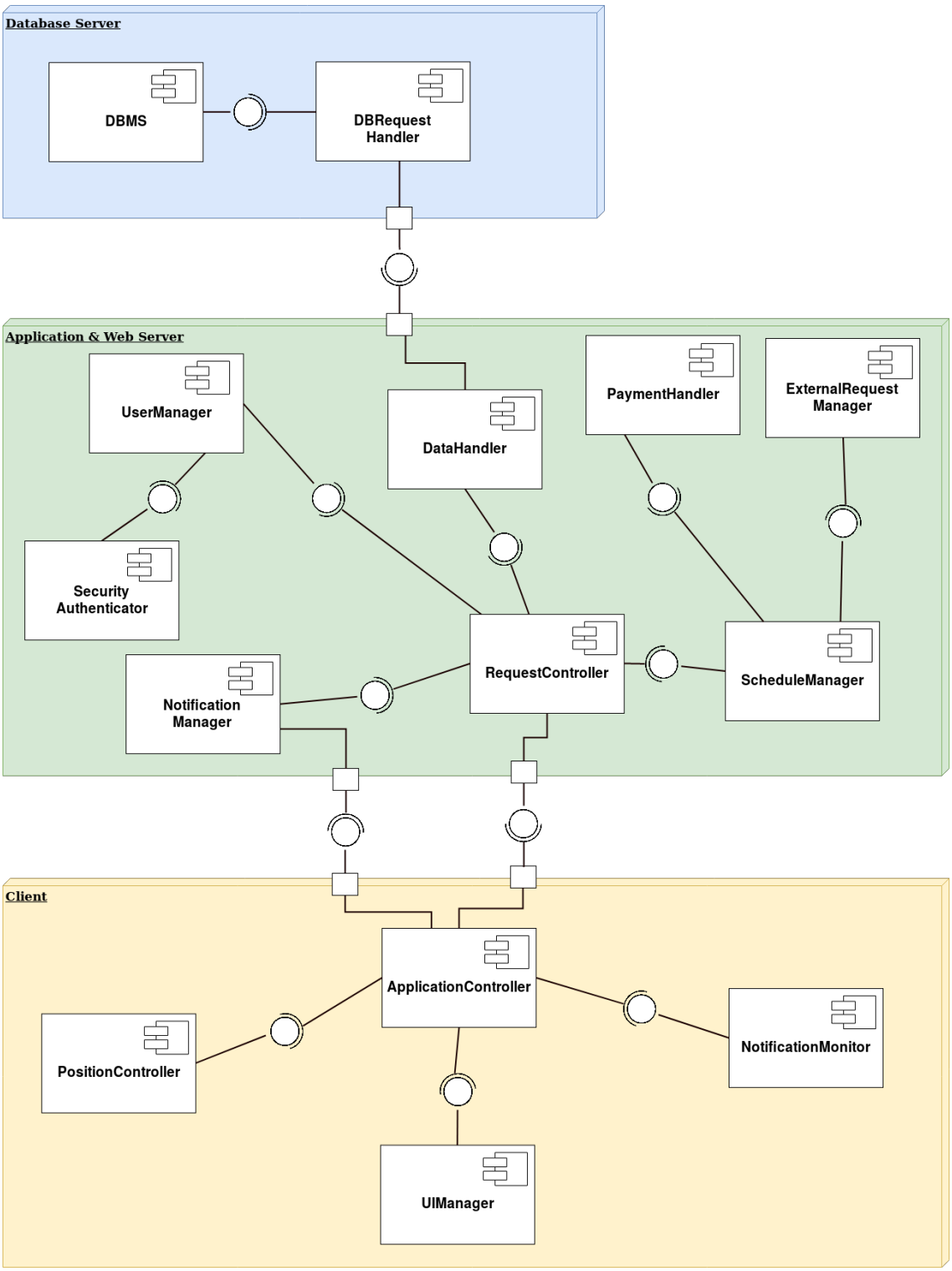

Figure 2.5: Component View of the System

2.3 Deployment View

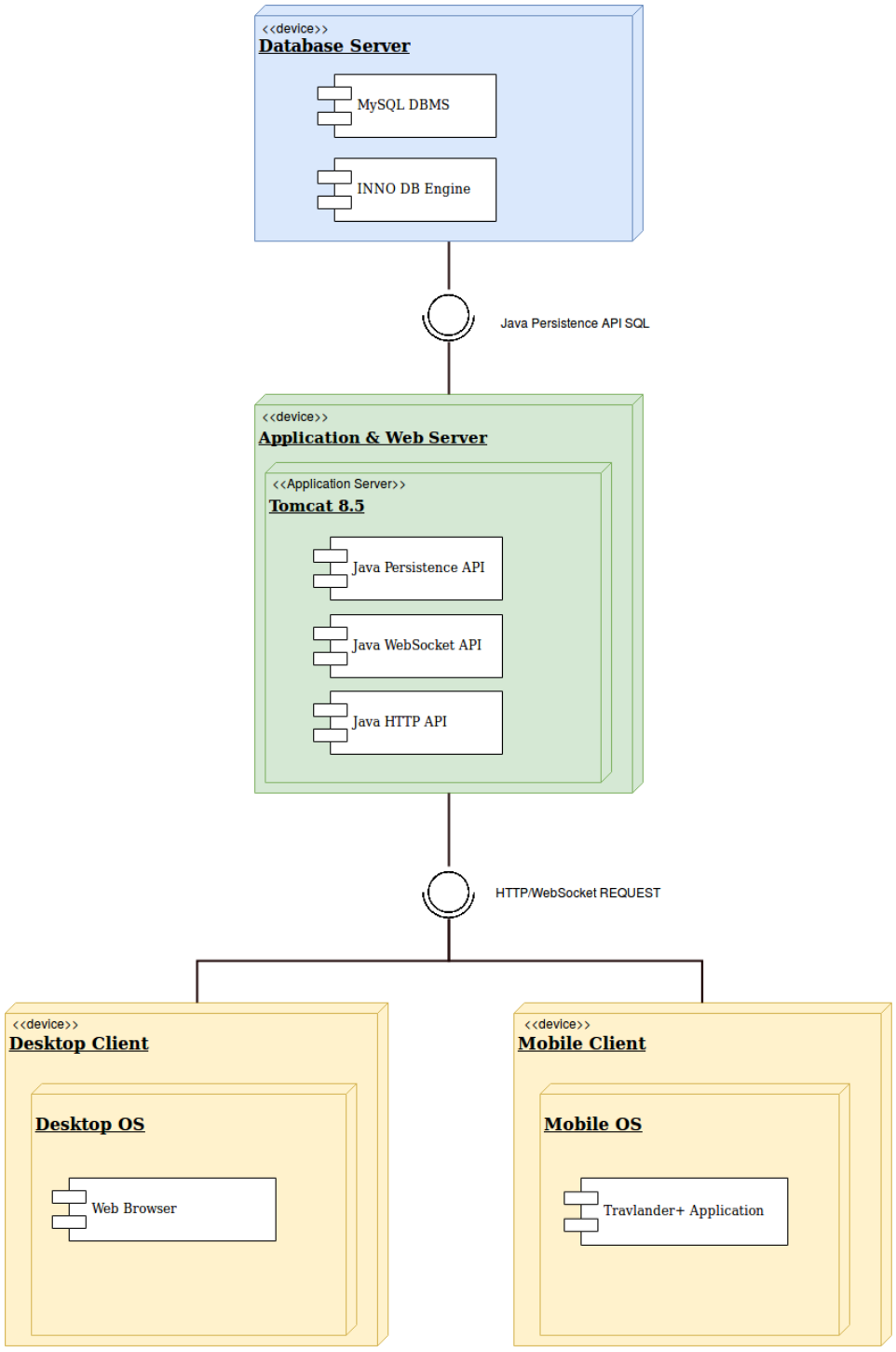

Figure 2.6: Deployment View of the system

2.4 Runtime View

This section represents the dynamic behaviour of the system in the most relevant and already seen cases. The purpose of each of the following sequence diagrams is to explain as much as possibile in the detail, the workflow inside the Application Server and its communication with the Client and the Database Server.

Registration to Travlendar+ application

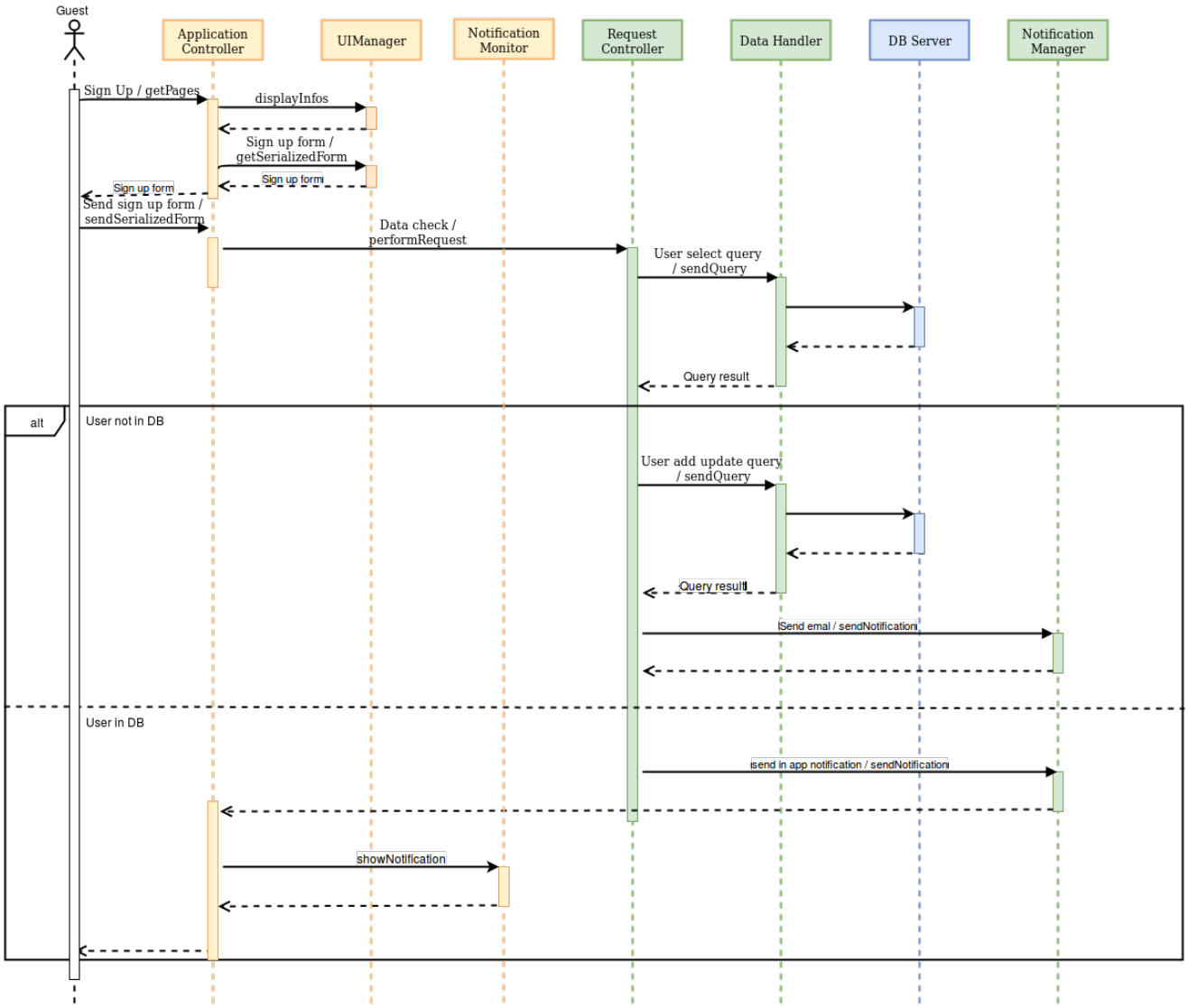

Figure 2.7: Sequence Diagram of the guest's registration to the application

As we can see in this Diagram, the guest can send a Serialized Form from the Application Controller module to the Request Controller, which will entrust the Data Handler

of making a request to the Database Sever in order to know if a user with same data already exists.

If not, the guest is added to the Database server and is sent an e-mail for account verification. Otherwise, the guest is notified that someone with the same data already exists.

Schedule Creation

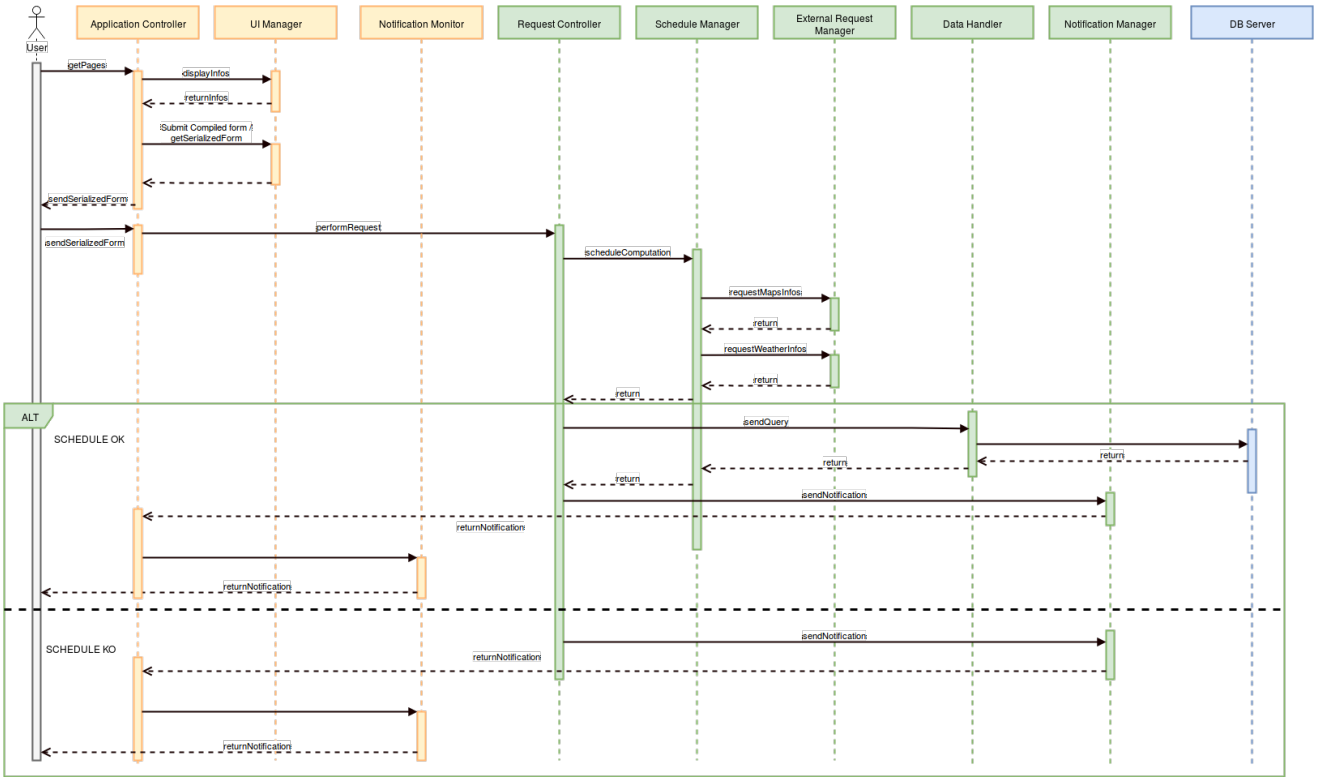

Figure 2.8: Sequence Diagram of the schedule creation

In this diagram is possible to see how the components interact between them in order to create a new schedule. Once the request reaches the Application server, several control are made and the External Request Manager grants information about the Weather and path to be chosen.

Manage User's information

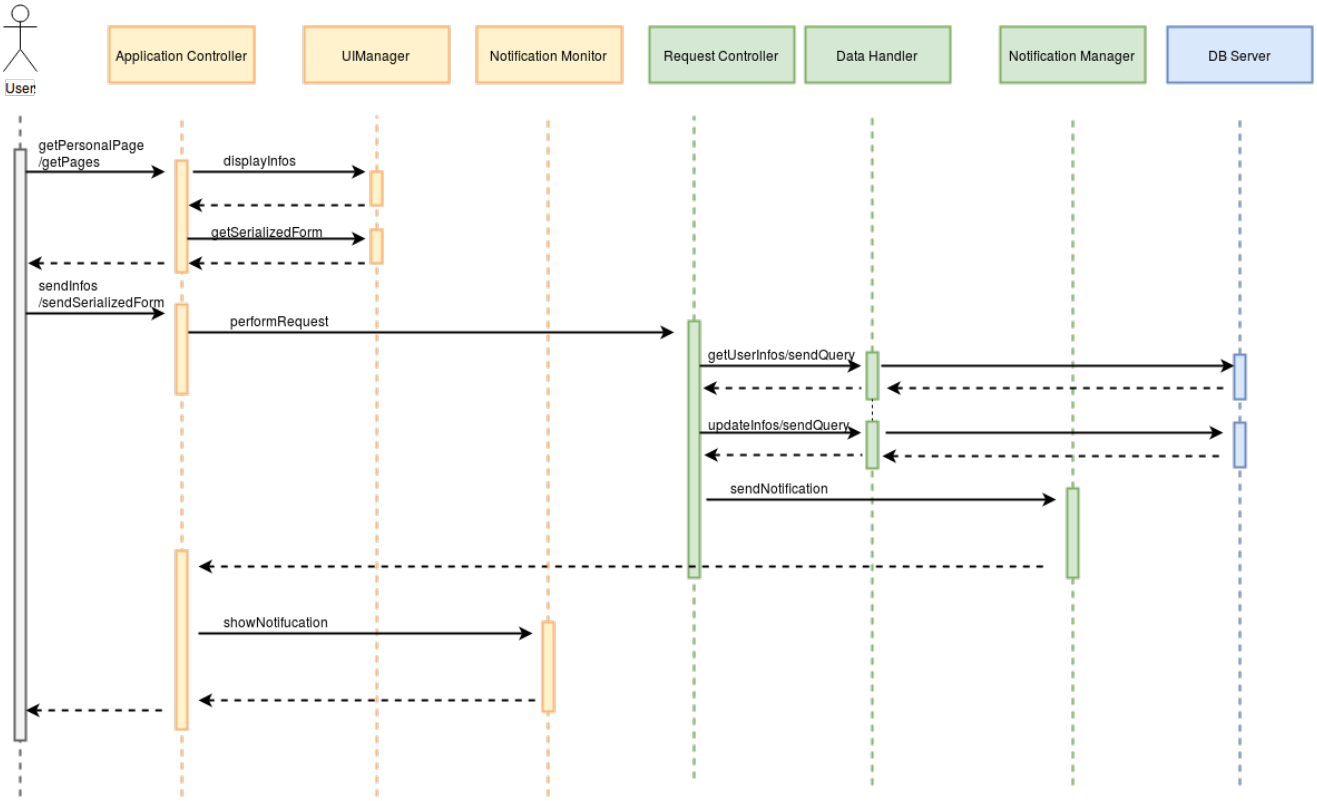

Figure 2.9: Sequence Diagram of the managing of users' information

In this diagram it is possible to see how the Data Handler module deals with a request of the user to modify his data and preferences. All the changes are stored in the DB Server.

Retrieve Weather Forecast information

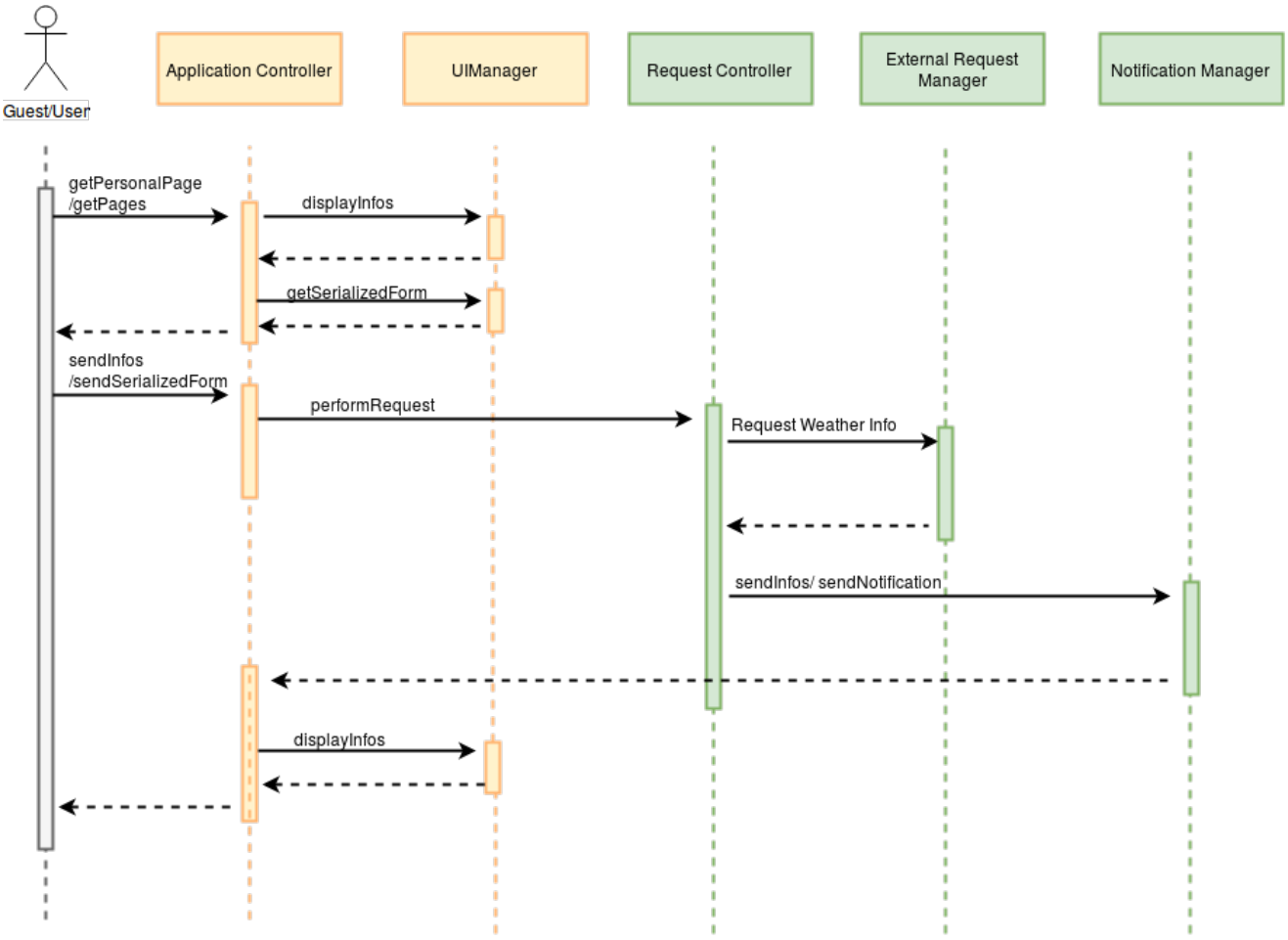

Figure 2.10: Sequence Diagram of the retrieving of Weather Forecast information

Both the guests and users can actually retrieve information about the Weather Forecast. This is done by the External Request Manager that collect data from the Openweather external API. Then these information are sent to the Notification Manager module.

Buy tickets and subscription for public transport

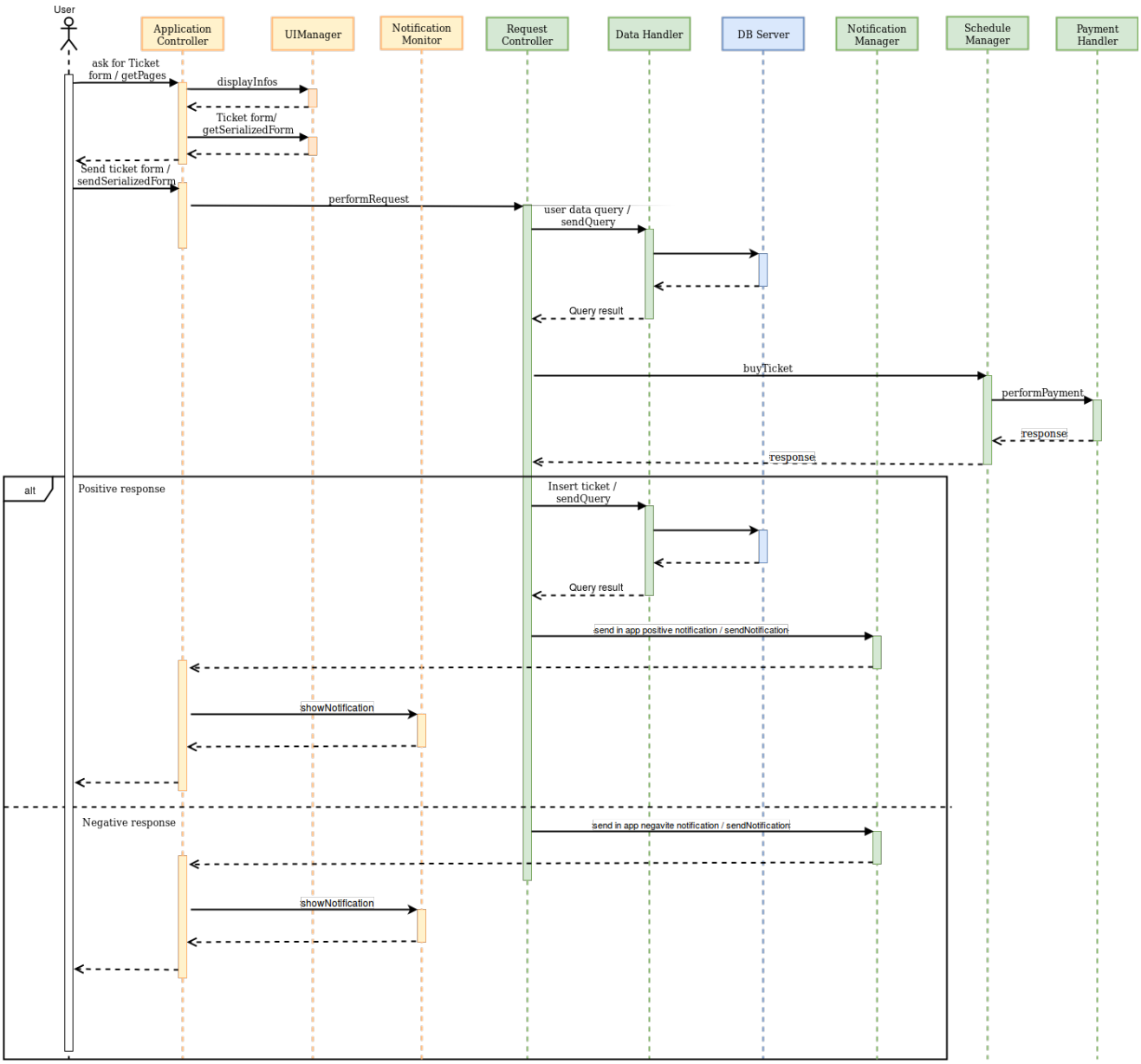

Figure 2.11: Sequence Diagram of the purchase of tickets and subscription for public transport

Here we can observe the procedure that deals with the purchase of tickets by the user. If the transaction done due to the Payment Handler goes well, the information about the bought ticket are saved to the Database Server using the submitQuery method.

Share user's schedule

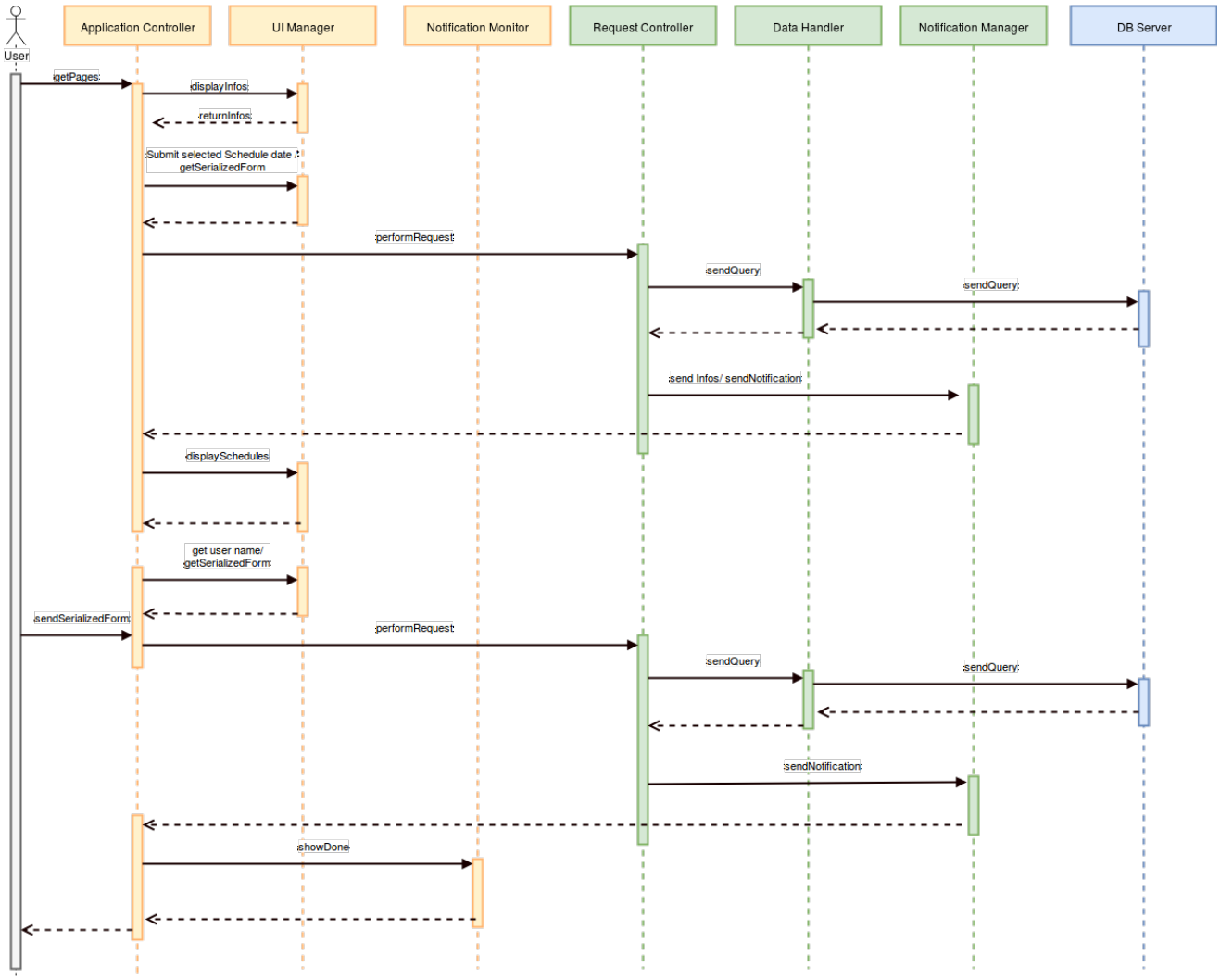

Figure 2.12: Sequence Diagram of sharing user's schedule

Whenever a user wants to share his own schedule of a given day of his choice, the username of the second user is saved to the Database Server. Then the first user is notified by the Notification Manager.

Update user's schedule

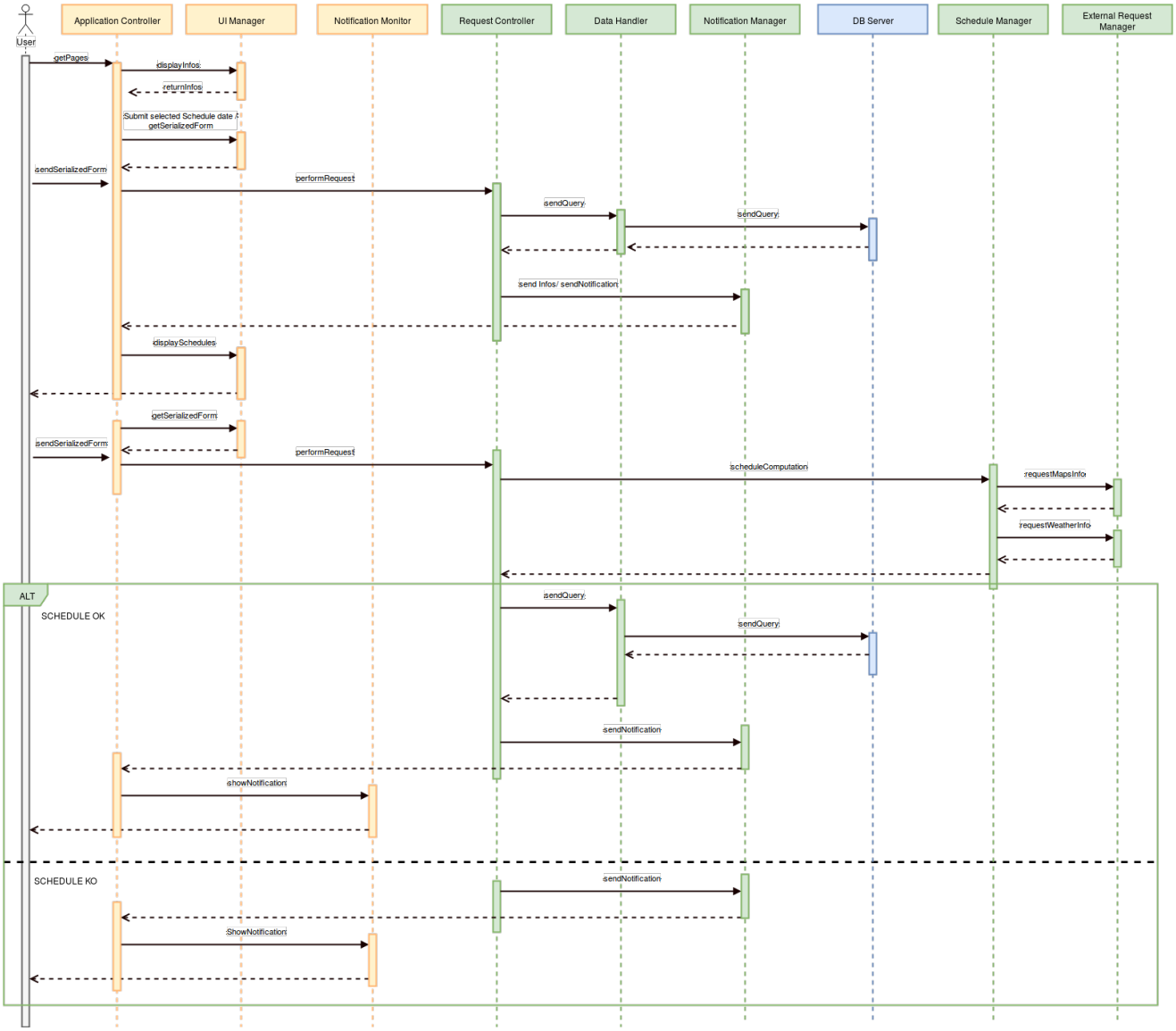

Figure 2.13: Sequence Diagram of updating user's schedule

As the preferences of every user are stored in the Database Server, whenever they want to change them the Data Handler's method submitQuery is called. Then the modifications are saved and updated immediatly in the DB server.

Manage user's preference about time for break

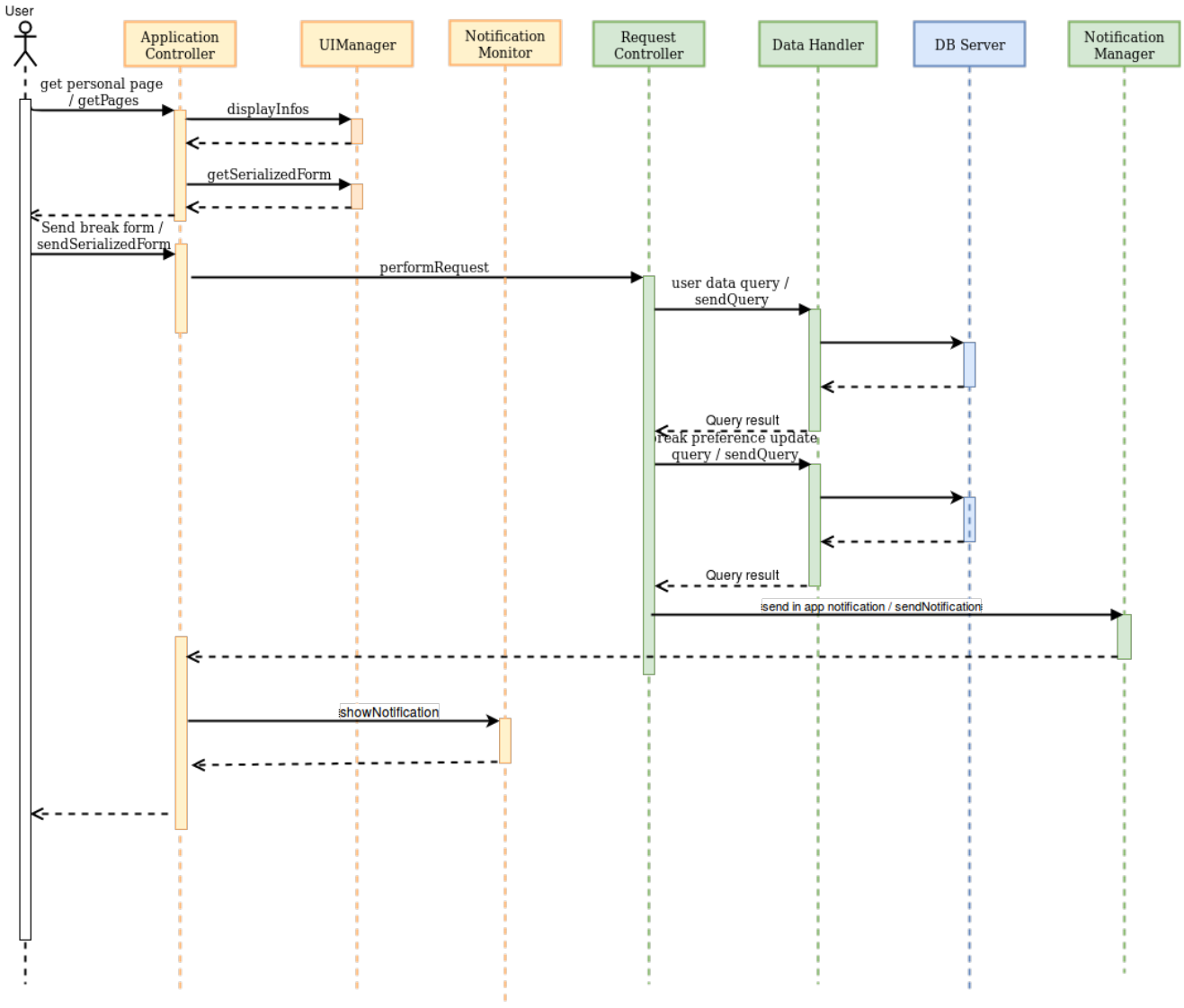

Figure 2.14: Sequence Diagram of managing user's preference about time for break

If a user decides to modify his break preferences, they are modified for all the upcoming schedules he decides to create. Already created schedules are not modified.

User wants to see a schedule for specified day

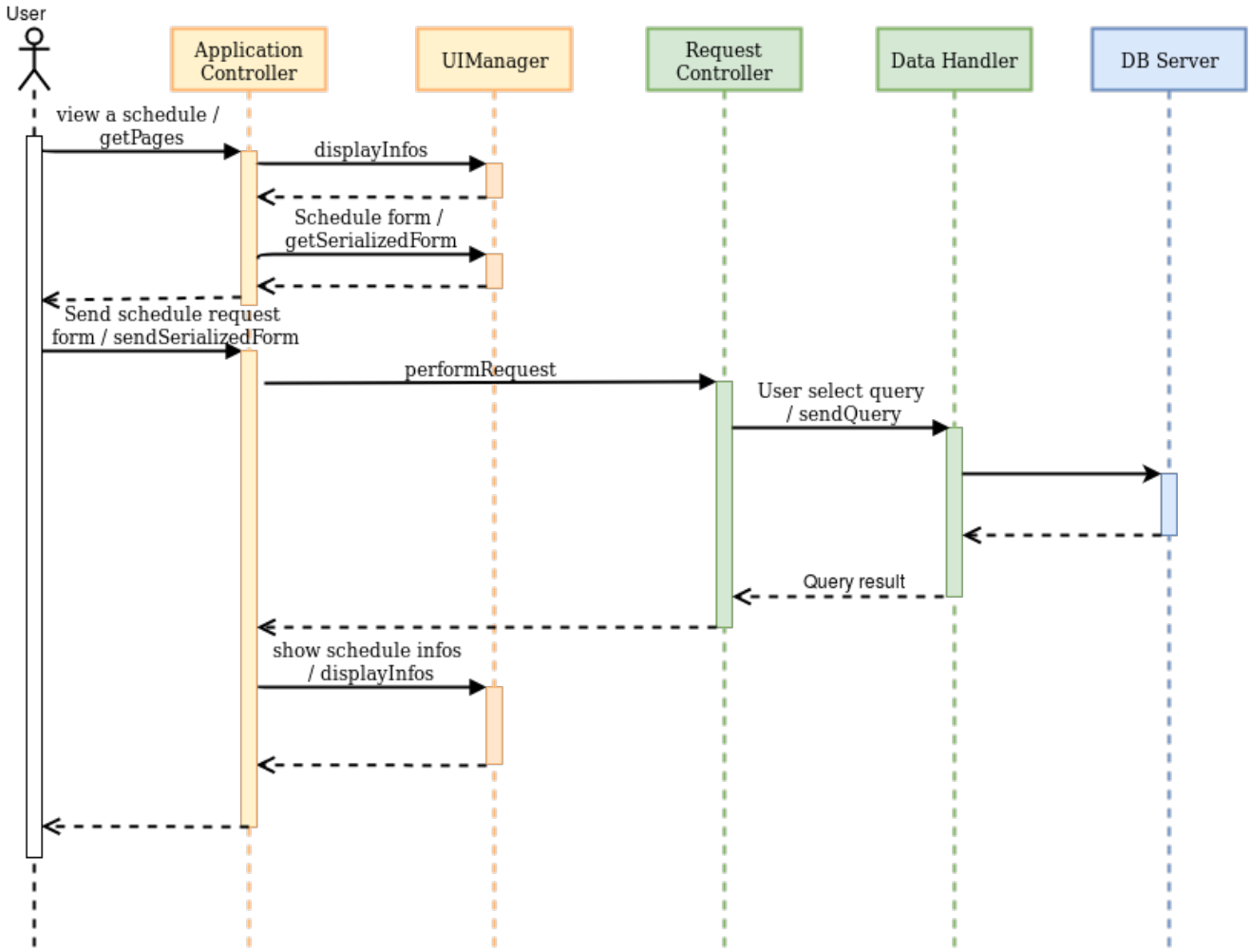

Figure 2.15: Sequence Diagram of a user wants to see a schedule for specified day

If a user decides to see a schedule of his, he has just to insert the data of the day and thus it will be passed to the Application Controller. Then, after a query is successfully executed passing from the Request Controller and Data Handler, the result is shown to the user by the UI Manager.

Other considerations

In all the diagrams above the procedure which check the correctness of the token and the one that explain the submitQuery method to the Database server where deliberately omitted. The reason is mainly redundancy and thus they are explained in the following diagrams.

Check token procedure

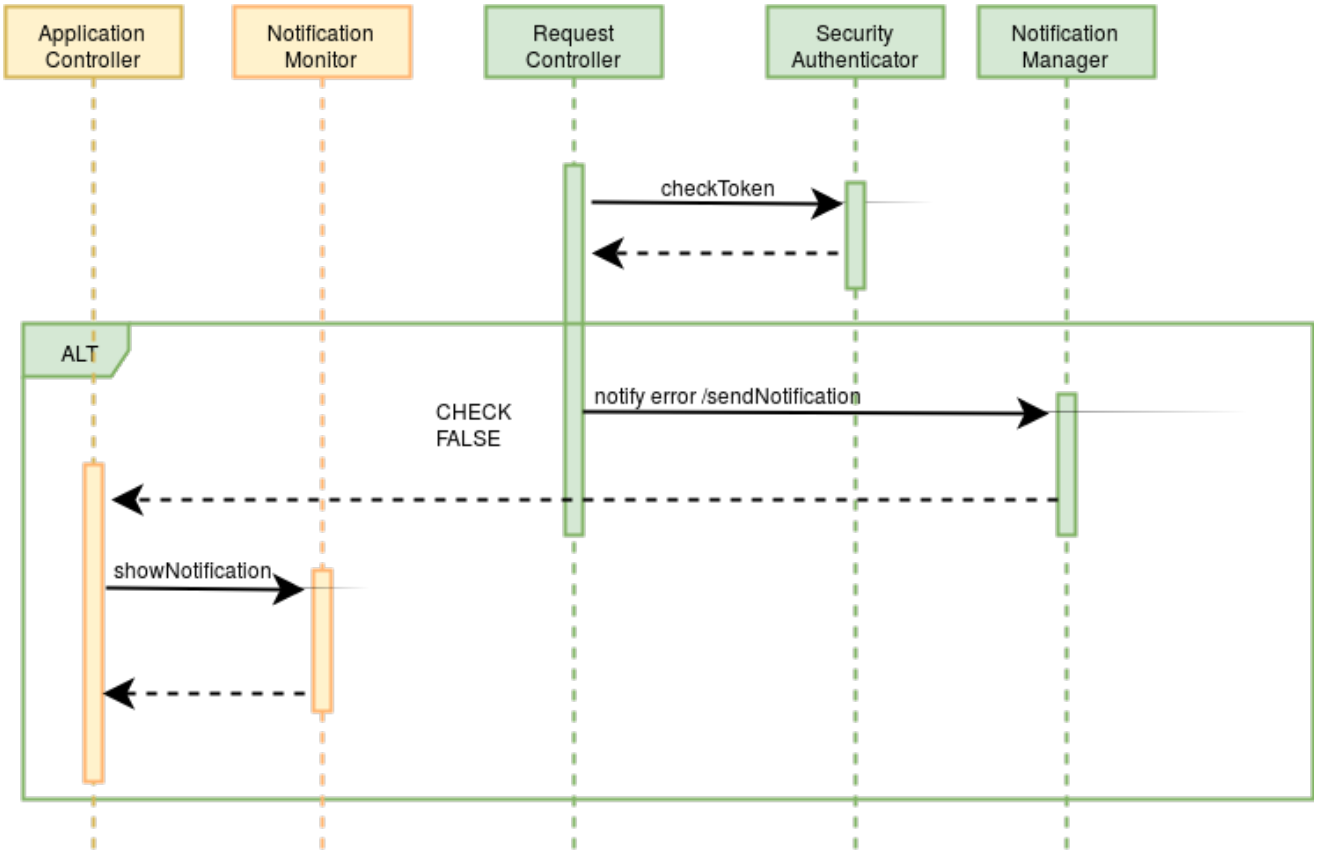

Figure 2.16: The procedure of checking the token

This particular procedure is called every time the user makes a request to the server. The validity of the token is then increased to permit the user an easy flow in browsing contents.

Query DBMS procedure

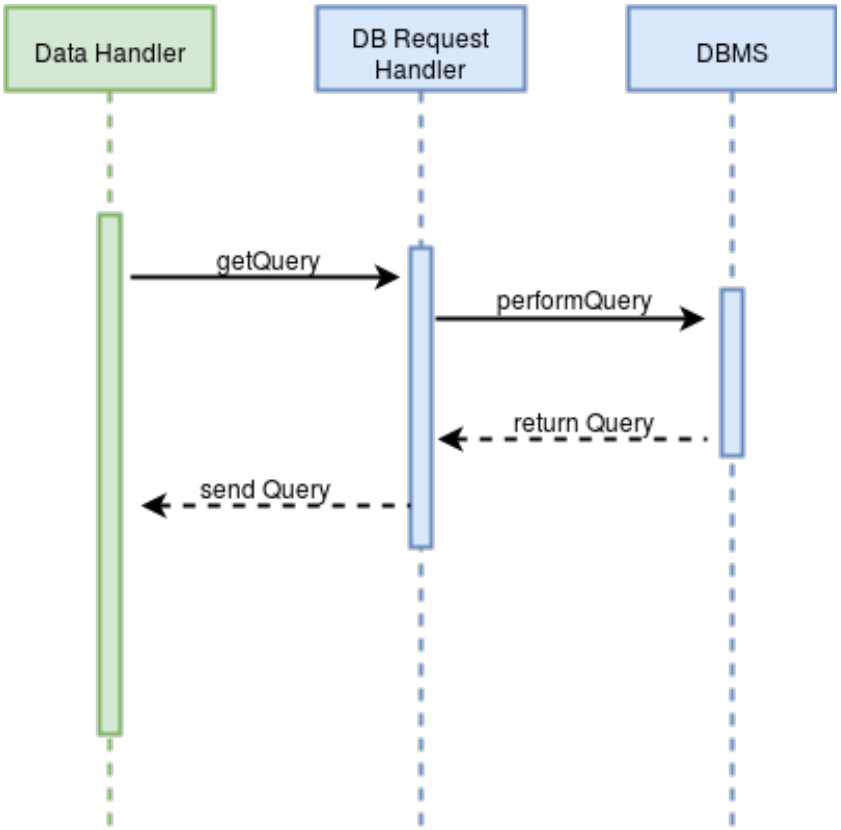

Figure 2.17: The procedure of sending a query to the DBMS

Login procedure

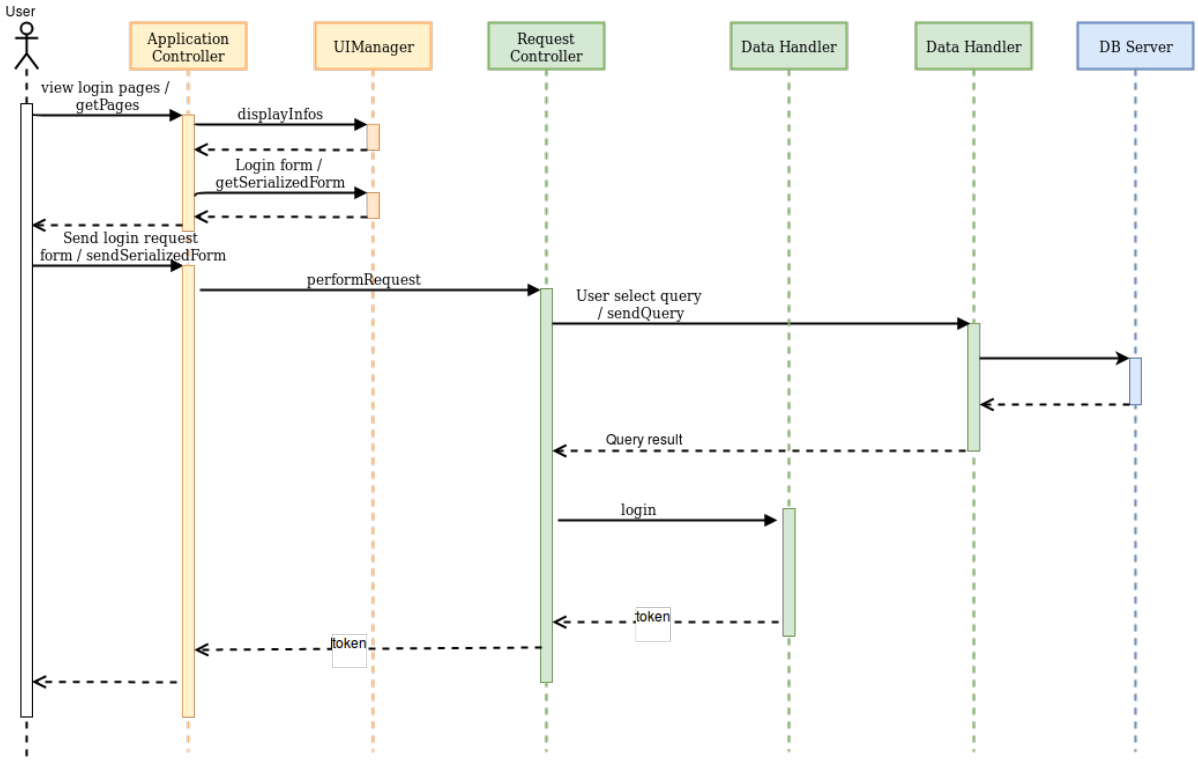

Figure 2.18: The procedure of logging in

2.5 Component Interfaces

This sections includes further details on the interfaces between different components of the system.

Component Interface of the client

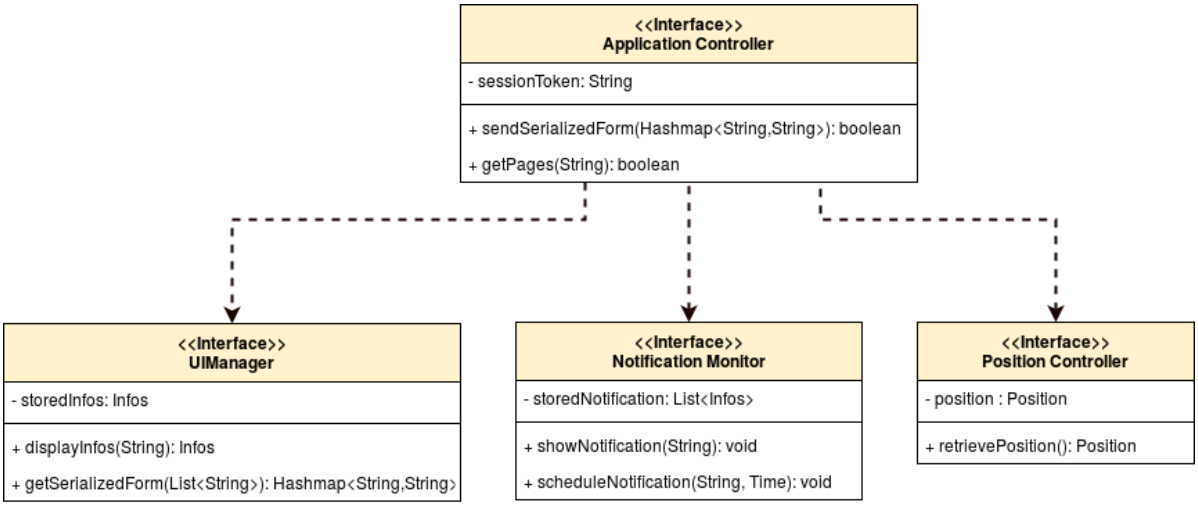

Figure 2.19: Component Interface of the client

Component Interface of the Application & Web Server

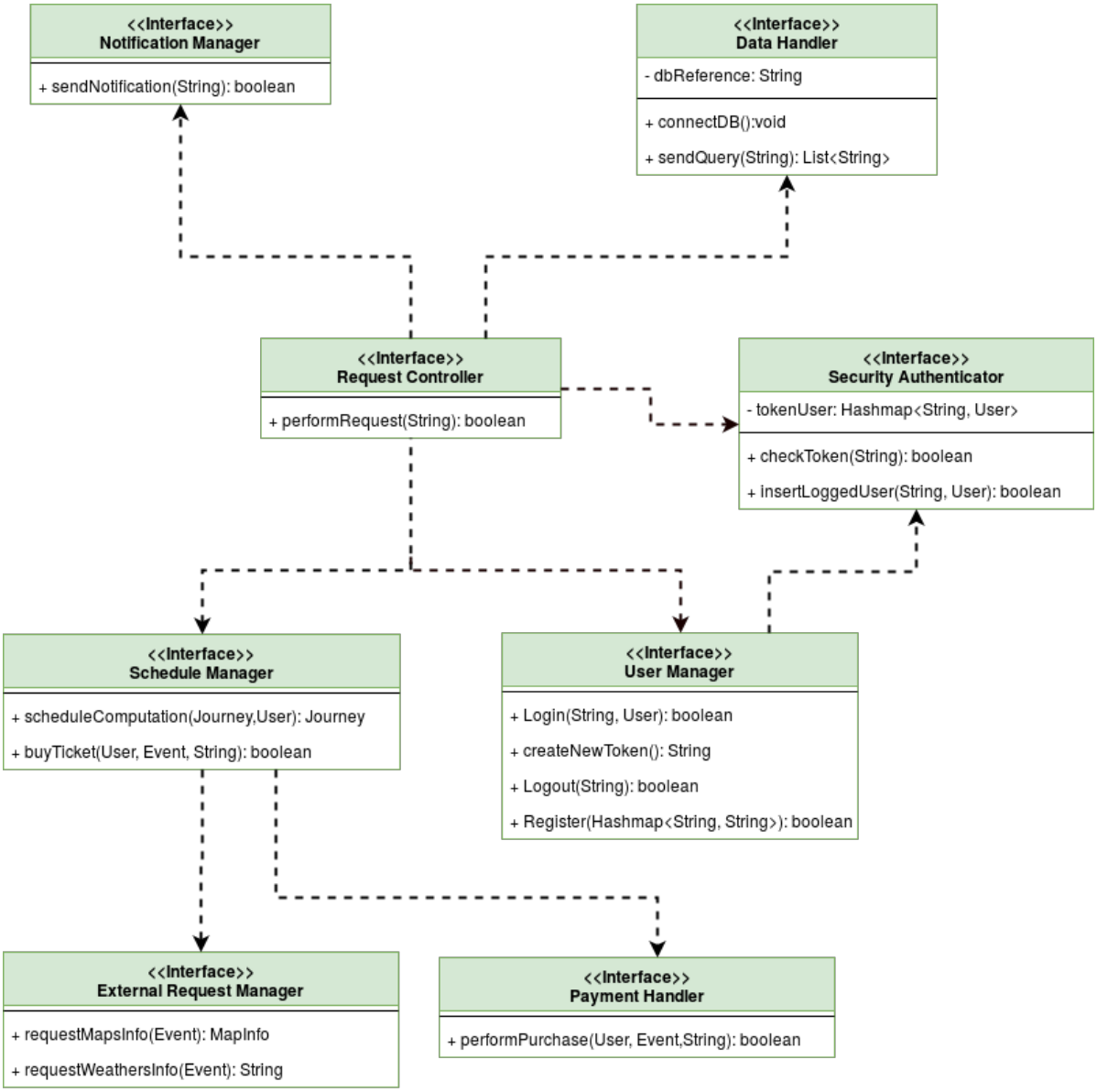

Figure 2.20: Component Interface of the Application & Web Server

Component Interface of the Database Server

Figure 2.21: Component Interface of the Database Server

2.6 Selected Architectural styles and patterns

We decided to use a three-tier architecture. The client will be represented by the application installed on the device of the user or his web browser, which only contains the presentation layer. Therefore it is a so called Thin-Client.

On the other hand, the server is represented by a mainframe which computate all the business logic and computation to fullfill the users' requests. The reason behind this choice are the willing of keeping thin the client to improve the user experience and the reserved channel on which communicate the Application server and the Database server. As there is only this single channel, we can guarantee the confidentiality and privacy of the users.

Here are listed the main pattern we chose to implement:

- **Model View Controller**:

  The model role is covered by the Data center which mantains the data related to the users. The controller instead is covered by the Application & Web Server which manage the events handling with both Model and View. Finally, the View is covered by the Client Application which displays data and information in a way that is possibile to interact with them due to a User Interface.

- **Proxy**:

  On the Application side we keep a copy of the tokens which allow the user to correctly login. In this way it is not necessary to request every time the database server in order to know if a user is currently logged in or not. Therefore we can actually even double check if a user is really connected with a random control over time.

- **Singleton**:

The Application Server's module RequestController is a Singleton, as it is istatiated only once and handle all the possibile request by different users.

2.7 Other design decisions

We decided to display on the client side a map which displays the journey the user is willing to do. In order to achive this, we decided to use the Google Maps API.

For the Weather Forecast instead we decided to requests information to the OpenWeather API. In this way we can also display data about the forecast whenever the users' request them.

To keep stored the session of any current user, we decide to implement the login procedure based on a token-exhange and verification. Therefore is possibile to keep alive a session up to an estabilished time (for example 5 min) and refresh it whenever a correct HTTP GET is performed.

To ease the developement of a cross platform application, we decided to use Phonegap (https://phonegap.com/), which allows the creation of different kind of native applications starting from a WebApp. In that way it is possible to use it from iOS, Android and Windows Phone and from Desktop browsers

3 Algorithm Design

In this section we will explain the algorithms that concern the critical part of the implementation on the system.

Creation of a Journey/Event

The algorithm used for the creation of a journey has to take into account every aspect of a user profile. It has to follow these steps:

1. Consider the schedule of a chosen by user day
2. Take information about the Weather Forecast
3. Take the user's preferences about the means of transport
4. Retrieve the information about the best path to take from Google Maps API
5. Verify that the combination of created Event and Journey in the Schedule does not overlap with another currently existing Event or Journey
6. Verify that there is always possibile to have an Event of type Break in the current Schedule
7. If the previous control is satisfied we can save the update Schedule into the Database Server
8. Else, the Application server notifies the user that something went wrong

Token Creation

This is an example of a piece of code which can be used to apply the algorithm.

```
/**
* Generate a random string.
*/
public String nextString() {
   for (int idx = 0; idx < buf.length; ++idx)
      buf[idx] = symbols[random.nextInt(symbols.length)];
      return new String(buf);
}
```

```
public static final String special = "!%&/=#@";

public static final String upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public static final String lower = upper.toLowerCase(Locale.ROOT);

public static final String digits = "0123456789";

public static final String alphanum = upper + lower + digits;

private final Random random;

private final char[] symbols;

private final char[] buf;

public RandomString(int length, Random random, String symbols) {
   if (length < 1) throw new IllegalArgumentException();
      if (symbols.length() < 2) throw new IllegalArgumentException();
      this.random = Objects.requireNonNull(random);
      this.symbols = symbols.toCharArray();
      this.buf = new char[length];
}

/**
* Create an alphanumeric string generator.
*/
public RandomString(int length, Random random) {
   this(length, random, alphanum);
}

/**
* Create an alphanumeric strings from a secure generator.
*/
public RandomString(int length) {
   this(length, new SecureRandom());
}

/**
* Create session identifiers.
*/
public RandomString() {
   this(21);
}
```

User Registration

This is an example of a piece of code which can be used to apply the algorithm.

```
public void register(HashMap<String, String> newUser) {
   try {
      PreparedStatement prepared = connection
      .prepareStatement("insert into User (Surname, Name, Age, Email) values (?,?,?,?)");
      prepared.setString(1, newUser.get("Surname"));
      prepared.setString(2, newUser.get("Name"));
      prepared.setString(3, newUser.get("Age"));
      prepared.setString(4, newUser.get("Email"));
      prepared.executeUpdate();
   } catch (SQLException e) {
      e.printStackTrace();
   } finally {
   try {
      if (connection != null)
         connection.close();
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }
}
```

Payment

In order to perform the purchase requested by the user, the system has to follow these steps:

1. Consider the mean of transport found in the event
2. Allow the user to choose the ticket or subscription referred to the mean of transport of his choice
3. Consider the credit card data of the user
4. Send the purchase information to third part services like PayPal
5. If the purchase ends well, save the receipt of the ticket and add it to the stored ticket of the user in the Database Server
6. Else, the system notifies the user that something went wrong with the transaction

4 User Interface Design

UX Diagram

The purpose of the UX diagram is to show the different screen provided by the user interface. Moreover it points out the interactions among the screens themselves and the presence of input form and required data in a specific screen.

The diagram provided below follow the User Interface requirements stated in the RASD. The Web and the Mobile application implement the same functionalities and the same screen.

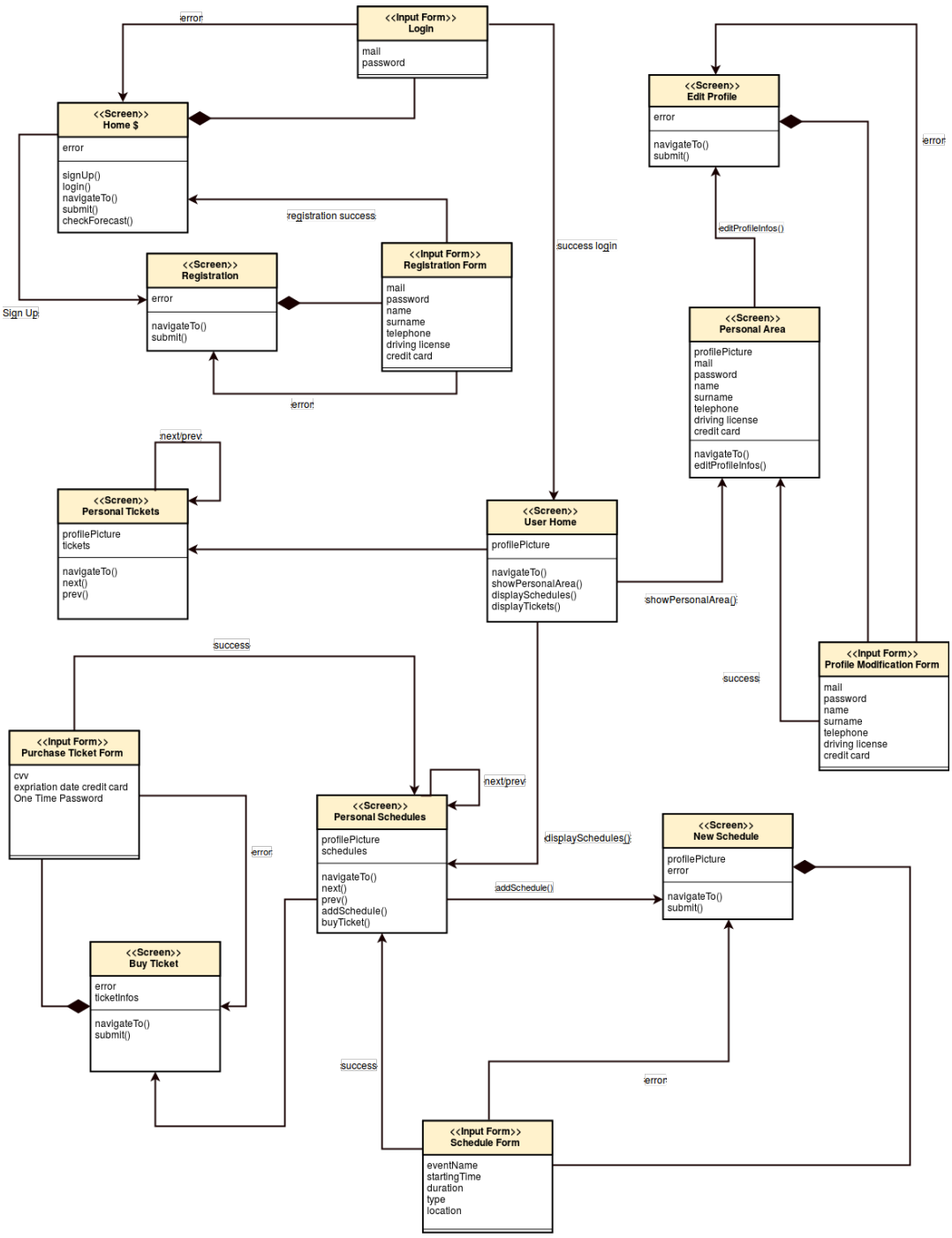

Figure 4.1: Component Interface of the Database Server

5 Requirements Traceability

In this section we create a bridge between the functional and not functional requirements defined before in the RASD with the sections in the DD.

DD Component	**Functional Requirements**
Request Controller	Register, Create Schedule, Data Management, Weather Forecast, Buy Ticket, Share Schedule, Update Schedule, Manage Break, Journey Infos
Data Handler	Register, Create Schedule, Data Management, Buy Ticket, Share Schedule, Update Schedule, Manage Break, Journey Infos
Schedule Manager	Create Schedule, Buy Ticket, Update Schedulet
Payment Handler	Buy Ticket
External Request Manager	Create Schedule, Weather Forecast, Update Schedule
User Manager	Login

DD Component	**Functional Requirements**
Notification Manager	Register, Create Schedule, Data Management, Weather Forecast, Buy Ticket, Share Schedule, Update Schedule, Manage Break
Application Controller	Register, Create Schedule, Data Management, Weather Forecast, Buy Ticket, Share Schedule, Update Schedule, Manage Break, Journey Infos
UI Manager	Register, Create Schedule, Data Management, Weather Forecast, Buy Ticket, Share Schedule, Update Schedule, Manage Break, Journey Infos
Notification Monitor	Register, Create Schedule, Data Management, Buy Ticket, Share Schedule, Update Schedule, Manage Break

RASD Requirements	**DD Section**
Security	Highlevel components and their Interaction
Parallel Operation	Client Server Architecture
Maintainability	Selected Architectural styles and patterns
Portability	Other design decisions

6 Implementation,Integration and Test Plan

Once we defined the whole software and hardware architecture, the next step is the Implementation and Testing. During the testing phase we have to cover a certain amount of code:

- **100%** of the DBMS Server's components
- **80%** for the Web & Application Server's component
- **50%** for the Client's component

We want to approach the Integration phase with a **bottom-up** paradigm, which starts with the developement of minor modules. Then we extend the modules integrating them each other.

Meanwhile we realize the presentation layer and data layer as they are separated each other, so can be developed in parallel. Therefore we implement on the server side:

1. Data Handler
2. User Manager and Security Authenticator
3. Payment Handler and External Request Manager
4. Schedule Manager
5. Request Controller and Notification Manager

Done that, we must create the Application Controller for the Client. Once completed, it is finally possible to integrate the single components together like shown in the following schemas:

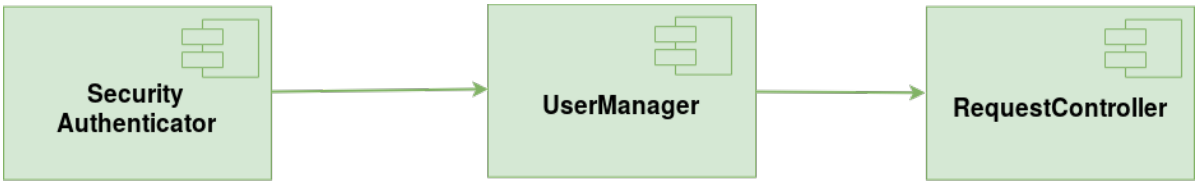

Figure 6.1: Integration of the Request Controller with the User Handler

Figure 6.2: Integration of the Request Controller with the Data Handler

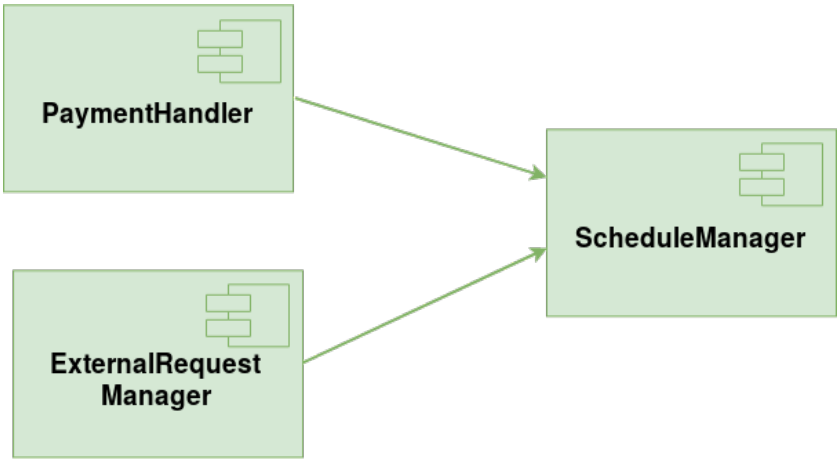

Figure 6.3: Integration of the Schedule Manager with Third Part Services

Figure 6.4: Integration of the Schedule Manager with the Request Controller

Once we integrate all these components, we can let them communicate in order to make the application work. Then, as in the scheme down here, we implement all the Client's modules.

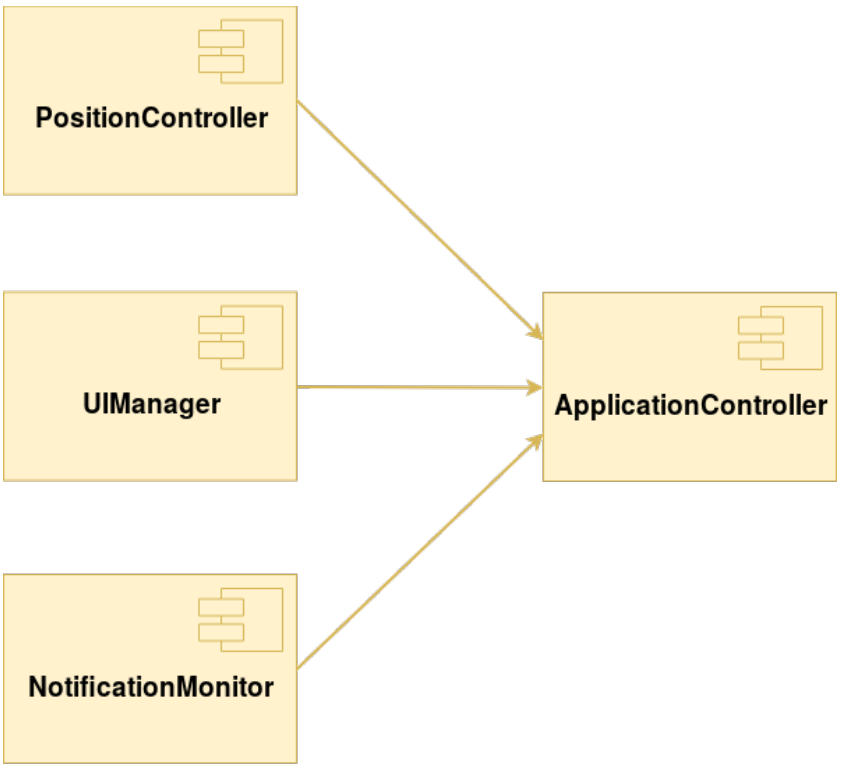

Figure 6.5: Integration of the various components of the client

When all those implementations are done, it is effectively possibile to try and test the system as a whole.

7 Effort Spent

While working on the project, we always met to discuss main topics to provide more consistence to the project:

Name	**Effort**
Lukasz Moskwa	Group 30h and 5h alone
Marco Mussi	Group 30h and 5h alone
Gianluigi Oliva	Group 30h and 5h alone

8 References

- The orginal Travlendar application:
  http://score-contest.org/2018/projects/travlendar.php
- The revised document of the assignment:
  https://goo.gl/9m1ojy
- IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
- IEEE Std 1016tm-2009 Standard for Information Tecnology-System Design-Software Design Descriptions.
- Phonegap:
  https://phonegap.com/