Politecnico di Milano

AA 2017-2018

POLITECNICO
MILANO 1863

Computer Science and Engineering

Software Engineering 2

DD

Design Document

version 1.0 – 23.11.2017

Table of Contents

1. Introduction

1.1.Purpose

The purpose of this document is to give more technical details than the RASD about Travlendar+ application system. While the RASD presented a general view of the system and what functions the system is supposed to execute, this document aims to present the implementation of the system including components, run-time processes, deployment and algorithm design. It also presents in more details the implementation and integration plan, as well as the testing plan.

More precisely, the document presents:

- Overview of the high level architecture
- The main components and their interfaces provided one for another
- The Runtime behavior
- The design patterns
- The algorithm design of the most critical parts of the application
- Implementation plan
- Integration plan
- Testing plan

The purpose of this document is to provide an overall guidance to the architecture of the software product

1.2.Scope

The project Travlendar+ is calendar-based application which aims to remind the user for his/her meetings, to automatically compute travel times between the appointments as well as to support the user in his/her trips by identifying the most appropriate travel option and by providing the opportunity to buy public transportation tickets. Users create meetings and Travlendar+ is in charge to warn them if the meeting locations cannot be reached at the desired time. Additionally, it offers the possibility to arrange the trips of the users and allows them to set his/her long term and short term preferences, which have to be taken into consideration when offering routes and travel means. The users of our application have also the opportunity to define some restrictions on particular travel means for a certain time of the days. Furthermore, it has the ability to take into account the weather forecast or some road obstacles that can affect the possible routes. Additional Travlendar+ feature is allowing the user to specify flexible break by indicating the days and time intervals.

Target group are all the people who want to better organize their time, to be reminded for their following meetings and to be recommended the most favorable ways to get to a particular location.

The main purpose of Travlendar+ is to provide centralized system which offers more opportunities to the user at once (i.e. the users can schedule their meetings, set time slots for their breaks, organize their

trips using different transport means, be warned for the obstacles that may possible appear etc. at the same time).

1.3. Acronyms, Abbreviations

Acronyms

API: Application Programming Interface

CPU: Central Processing Unit

DB: Database

DBMS: Database Management System

DD: Design Document

GPS: Global Positioning System

GUI: Graphical User Interface

MVC: Model View Controller is a design pattern used for GUIs

RASD: Requirements Analysis and Specifications Document

Abbreviations

[Gn]: n-th goal

[Rn]: n-th functional requirement

1.4.Document Structure

The first chapter gives an introduction of the design document. It contains the purpose and the scope of the document, as well as some abbreviation in order to provide a better understanding of the document to the reader.

Chapter 2 deals with the architectural design of the application. It gives an overview of the architecture and it also contains the most relevant architecture views: component view, class view, deployment view, runtime view and it shows the interaction of the component interfaces. Some of the used architectural designs and designs patterns are also presented here, with an explanation of each one of them and the purpose of their usage.

Chapter 3 presents the algorithm design. It includes the most critical parts of the application and the algorithms designed to deal with them. All of the algorithms are written in pseudo code in order not to make any restriction on the implementation language and stay focused on the most important parts.

Chapter 4 refers to the mock-ups already presented in the RASD document.

Chapter 5 explains how the requirements that have been defined in the RASD map to the design elements that are defined in this document.

Chapter 6 identifies the order in which it is planned to implement the subcomponents of the system and the order in which it is planned to integrate such subcomponents and test the integration.

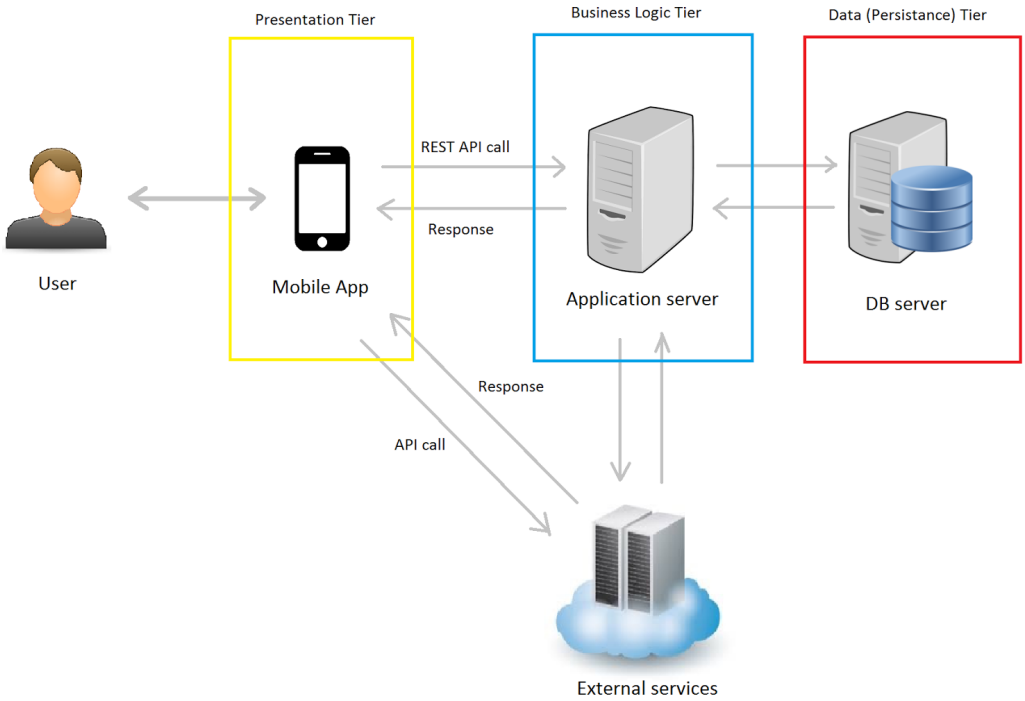
Chapter 7 shows the effort spent by each group member while working on this project.

Chapter 8 includes the reference documents.

2. Architectural design

2.1. Overview

Application architecture design is a process which has to be executed in a defined flow. The flow basically includes three different layers:



• **Presentation tier**: This layer comprises UI components and UI process components (Views and Controllers). The client side has a dynamic GUI, i.e. a module that communicates with the application server, but as well as with the external services. The idea is to have the information such as maps of the city gathered directly from the external source, rather than bypassing through the application server.

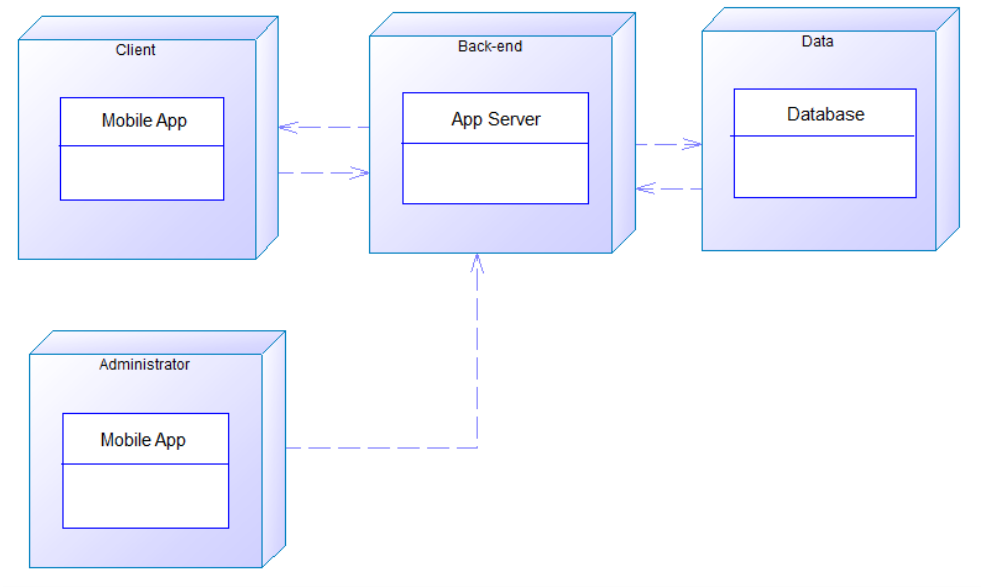
• **Business Logic tier**: As the name suggests, the layer focuses on the business front. In simple language, it focuses on the way business will be presented in front of the end users. This includes workflows, business components, and entities beneath the hood of two sub-layers named service and domain model layers.

While the service layer focuses on defining a common set of application functions that will be available to client and end users, the domain model layer represents expertise and knowledge linked to the specific problem domain. The entire plan is formulated in a way to explore and enhance the future of application.

• **Persistence tier**: At this third stage, data-related factors are kept in mind. This includes data access components, data helpers/utilities, and service agents. Here, the three components sit under the two subheads, precisely, persistence layer and network layer.

While the former provides simplified access to data, which can be stored in a persistent storage or backend, the latter is responsible for networking calls.

2.2.High level architecture

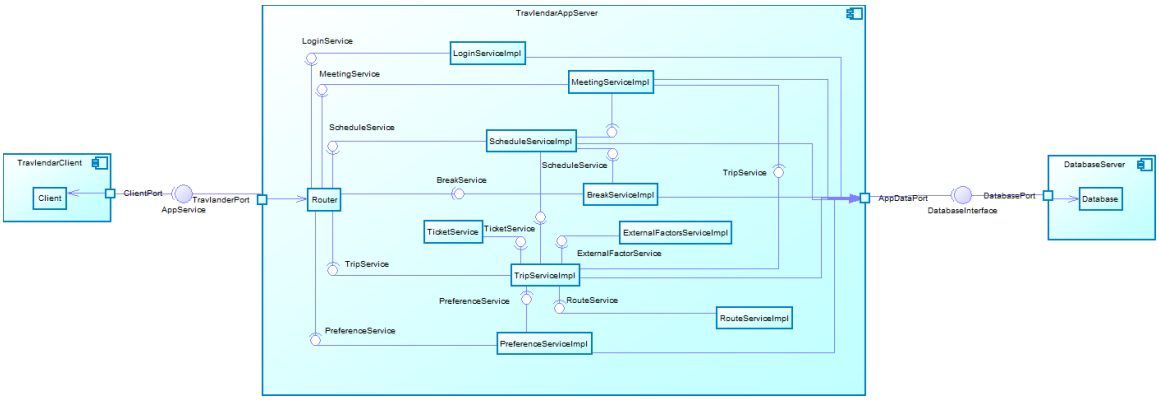


The figure above describes the high level components and their interaction. The App Server is the main component in our application as it is the central point between all the other components. It holds the business logic of the system. It interacts with the client and the administrator, who use the application as a mobile app. It also has an access to the DB server which holds all the data for the application.

The mobile application communicates directly with this server to operate.

2.3.Component view

In the following diagram, the mentioned components are more closely examined, with the main focus on the application server. While describing the parts, the notation will be the interface they are providing in order to be more general, because there may be one or more different implementations of the same service within the server. It should be noted that, except for the database server and its DBMS, the other external services are not depicted here. That is because they are observed as black-boxes, their structure does not concern us, only the services that they provide do.



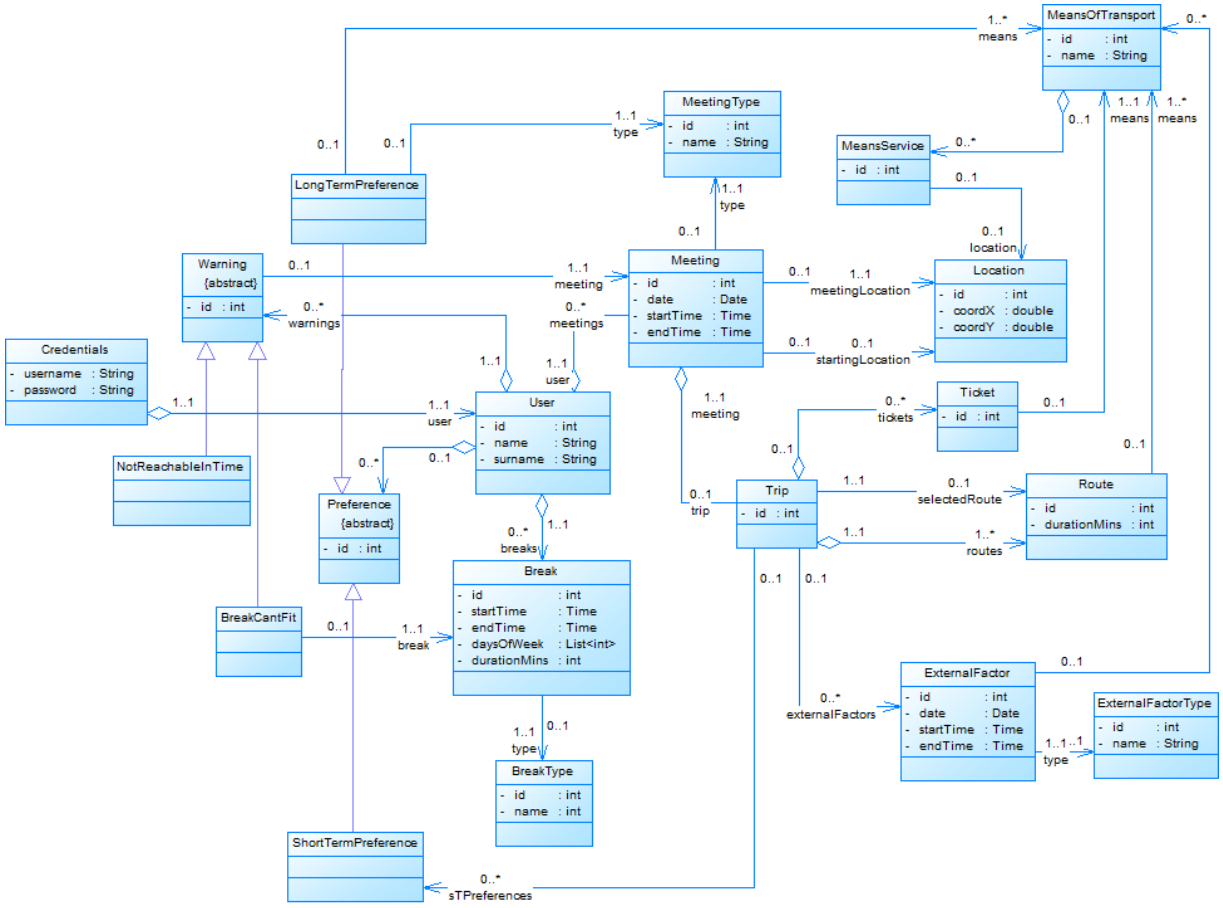
As we can see, the application server can consists of the following parts:

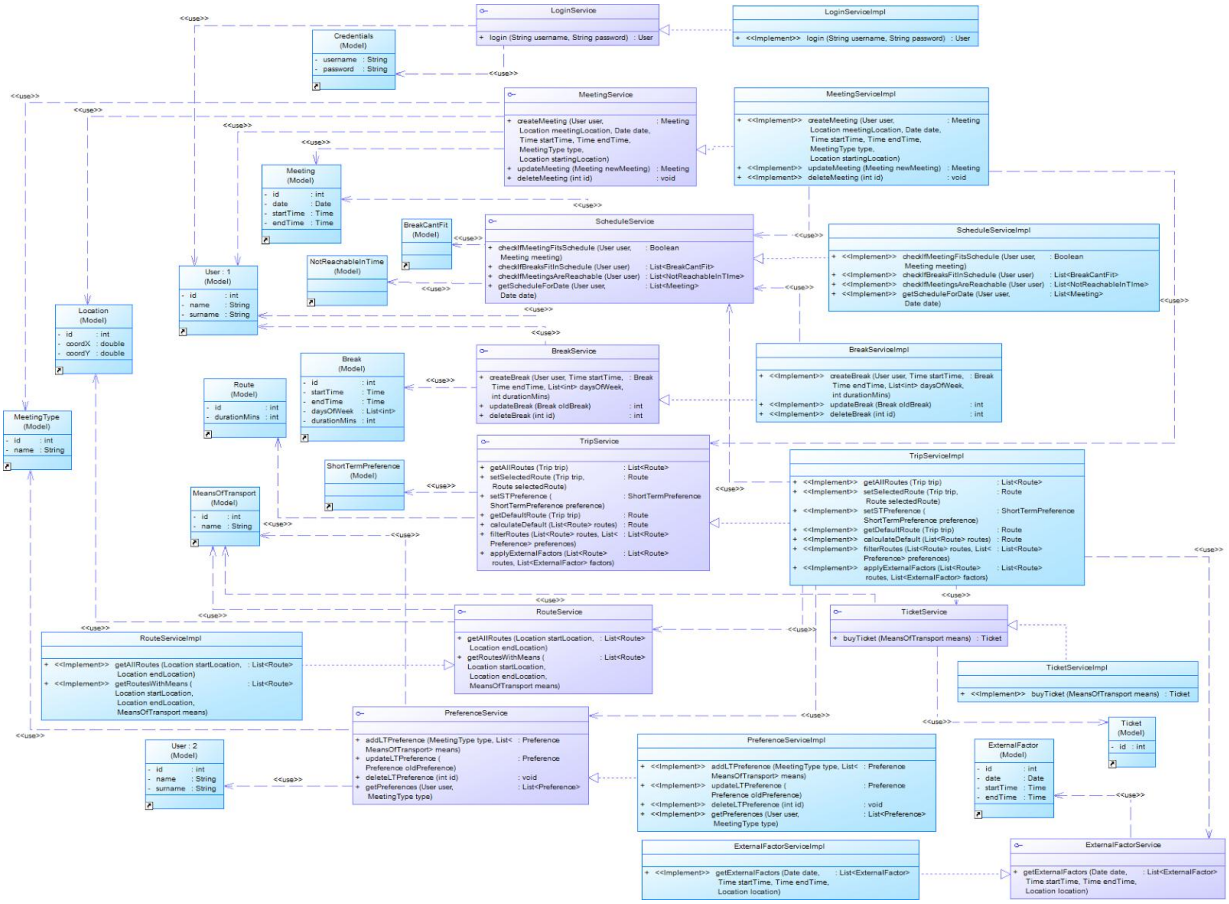
- LoginService - responsible for the authentication of the user
- MeetingService - provides functionalities concerning single meetings
- ScheduleService - provides functionalities concerning groups of meetings, schedules, while also providing the mechanisms to check the feasibility of the schedules (overlappings of meetings, meetings reachable in time with current trip and route, spcified breaks can be completed or not)
- BreakService - provides the functionalities concerning breaks
- TicketService - responsible for the purchasing of tickets for public transport
- ExternalFactorService - responsible for the acquirement of information on any external factors that may affect a route or trip (weather, strikes etc.)
- TripService - combines the effects of multiple other services to offer the functionalities concerning trips between meetings and the routes that can be taken, with regards to the previously specified preferences and any relevant external factors

- RouteService - provides routes between two locations using the possible transport means; also takes into account the sharing services and their locations
- PreferenceService - provides the functionalities concerning the preferences (their creation and their availability to be used while calculating the probable routes)

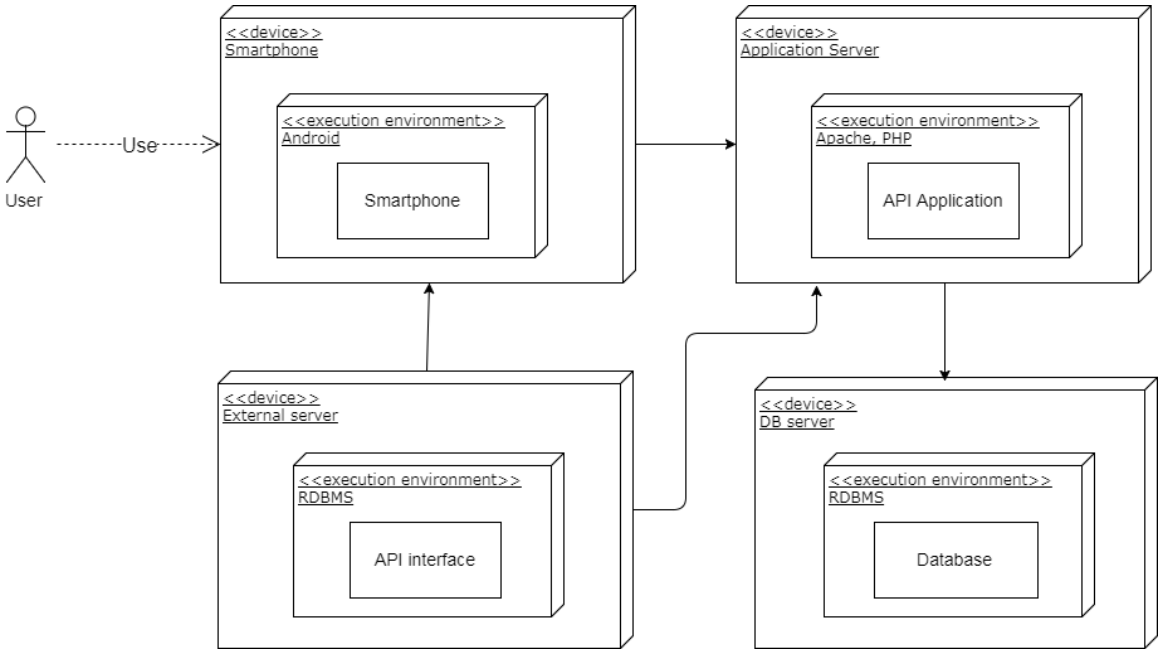
The components shown here communicate with each other and work together to complete certain user requests. The Router has the job of determining whether a request that is received is valid or not and, if it is, to dispatch it to the relevant service component.

The following two diagrams are provided to further describe the components in the application server in more details. Considering that the Model is not a component per se, it and its relationship to the service components in the application server is shown here. It is assumed that each service component will be able to invoke operations on the selected database concerning the part of the Model that it is using.




LoginService
+ login (String username, String password) : User
LoginServiceImpl
+ <<Implement>> login (String username, String password) : User
Credentials (Model)
- username : String
- password : String
MeetingService
+ createMeeting (User user, Location meetingLocation, Date date, Time startTime, Time endTime, MeetingType type, Location startingLocation) : Meeting
+ updateMeeting (Meeting newMeeting) : Meeting
+ deleteMeeting (int id) : void
MeetingServiceImpl
+ <<Implement>> createMeeting (User user, Location meetingLocation, Date date, Time startTime, Time endTime, MeetingType type, Location startingLocation) : Meeting
+ <<Implement>> updateMeeting (Meeting newMeeting) : Meeting
+ <<Implement>> deleteMeeting (int id) : void
Meeting (Model)
- id : int
- date : Date
- startTime : Time
- endTime : Time
BreakCantFit (Model)
ScheduleService
+ checkIfMeetingFitsSchedule (User user, Meeting meeting) : Boolean
+ checkIfBreaksFitInSchedule (User user) : List<BreakCantFit>
+ checkIfMeetingsAreReachable (User user) : List<NotReachableInTime>
+ getScheduleForDate (User user, Date date) : List<Meeting>
ScheduleServiceImpl
+ <<Implement>> checkIfMeetingFitsSchedule (User user, Meeting meeting) : Boolean
+ <<Implement>> checkIfBreaksFitInSchedule (User user) : List<BreakCantFit>
+ <<Implement>> checkIfMeetingsAreReachable (User user) : List<NotReachableInTime>
+ <<Implement>> getScheduleForDate (User user, Date date) : List<Meeting>
NotReachableInTime (Model)
User : 1 (Model)
- id : int
- name : String
- surname : String
Location (Model)
- id : int
- coordX : double
- coordY : double
BreakService
+ createBreak (User user, Time startTime, Time endTime, List<int> daysOfWeek, int durationMins) : Break
+ updateBreak (Break oldBreak) : int
+ deleteBreak (int id) : int
BreakServiceImpl
+ <<Implement>> createBreak (User user, Time startTime, Time endTime, List<int> daysOfWeek, int durationMins) : Break
+ <<Implement>> updateBreak (Break oldBreak) : int
+ <<Implement>> deleteBreak (int id) : int
Break (Model)
- id : int
- startTime : Time
- endTime : Time
- daysOfWeek : List<int>
- durationMins : int
Route (Model)
- id : int
- durationMins : int
MeetingType (Model)
- id : int
- name : String
TripService
+ getAllRoutes (Trip trip) : List<Route>
+ setSelectedRoute (Trip trip, Route selectedRoute) : Route
+ setSTPreference (ShortTermPreference preference) : ShortTermPreference
+ getDefaultRoute (Trip trip) : Route
+ calculateDefault (List<Route> routes) : Route
+ filterRoutes (List<Route> routes, List< Preference> preferences) : List<Route>
+ applyExternalFactors (List<Route> routes, List<ExternalFactor> factors) : List<Route>
TripServiceImpl
+ <<Implement>> getAllRoutes (Trip trip) : List<Route>
+ <<Implement>> setSelectedRoute (Trip trip, Route selectedRoute) : Route
+ <<Implement>> setSTPreference (ShortTermPreference preference) : ShortTermPreference
+ <<Implement>> getDefaultRoute (Trip trip) : Route
+ <<Implement>> calculateDefault (List<Route> routes) : Route
+ <<Implement>> filterRoutes (List<Route> routes, List< Preference> preferences) : List<Route>
+ <<Implement>> applyExternalFactors (List<Route> routes, List<ExternalFactor> factors) : List<Route>
ShortTermPreference (Model)
MeansOfTransport (Model)
- id : int
- name : String
RouteService
+ getAllRoutes (Location startLocation, Location endLocation) : List<Route>
+ getRoutesWithMeans (Location startLocation, Location endLocation, MeansOfTransport means) : List<Route>
RouteServiceImpl
+ <<Implement>> getAllRoutes (Location startLocation, Location endLocation) : List<Route>
+ <<Implement>> getRoutesWithMeans (Location startLocation, Location endLocation, MeansOfTransport means) : List<Route>
TicketService
+ buyTicket (MeansOfTransport means) : Ticket
TicketServiceImpl
+ <<Implement>> buyTicket (MeansOfTransport means) : Ticket
Ticket (Model)
- id : int
PreferenceService
+ addLTPreference (MeetingType type, List< MeansOfTransport> means) : Preference
+ updateLTPreference (Preference oldPreference) : Preference
+ deleteLTPreference (int id) : void
+ getPreferences (User user, MeetingType type) : List<Preference>
PreferenceServiceImpl
+ <<Implement>> addLTPreference (MeetingType type, List< MeansOfTransport> means) : Preference
+ <<Implement>> updateLTPreference (Preference oldPreference) : Preference
+ <<Implement>> deleteLTPreference (int id) : void
+ <<Implement>> getPreferences (User user, MeetingType type) : List<Preference>
User : 2 (Model)
- id : int
- name : String
- surname : String
ExternalFactor (Model)
- id : int
- date : Date
- startTime : Time
- endTime : Time
ExternalFactorServiceImpl
+ <<Implement>> getExternalFactors (Date date, Time startTime, Time endTime, Location location) : List<ExternalFactor>
ExternalFactorService
+ getExternalFactors (Date date, Time startTime, Time endTime, Location location) : List<ExternalFactor>
<<use>>

2.4.Deployment view



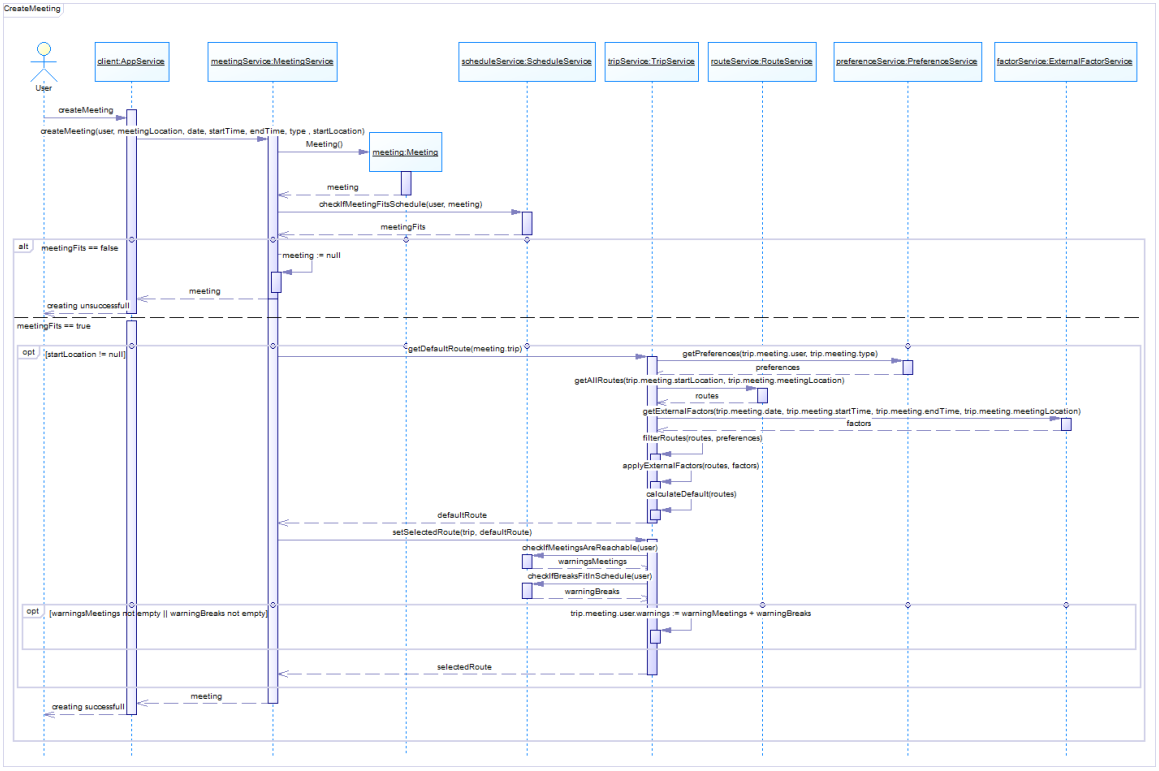
The deployment diagram shows the architecture of the system as deployment (distribution) of software artifacts to deployment targets. Travlendar requires deployment of software on the following nodes:

1. Smartphone: this is the application that will be used by the user. The user will be able to get information from the main Application server, as well as from the external services, that will provide information that does not to go through the main server.

2. Application server: the main logic of the application will be deployed here. This server will communicate with all the other nodes - it will gather information from the external services, manage user accounts and saved data from the DB server and take requests and send back responses to the user.
3. DB server: it will store all the persistent data for the users such as usernames and passwords, as well as their preferences and the data connected with the meetings.
4. External server: it will only provide services to enrich the application. For example, a map of the city will be used, as well as transport services that operate in the city the application is used in.

2.5.Runtime view

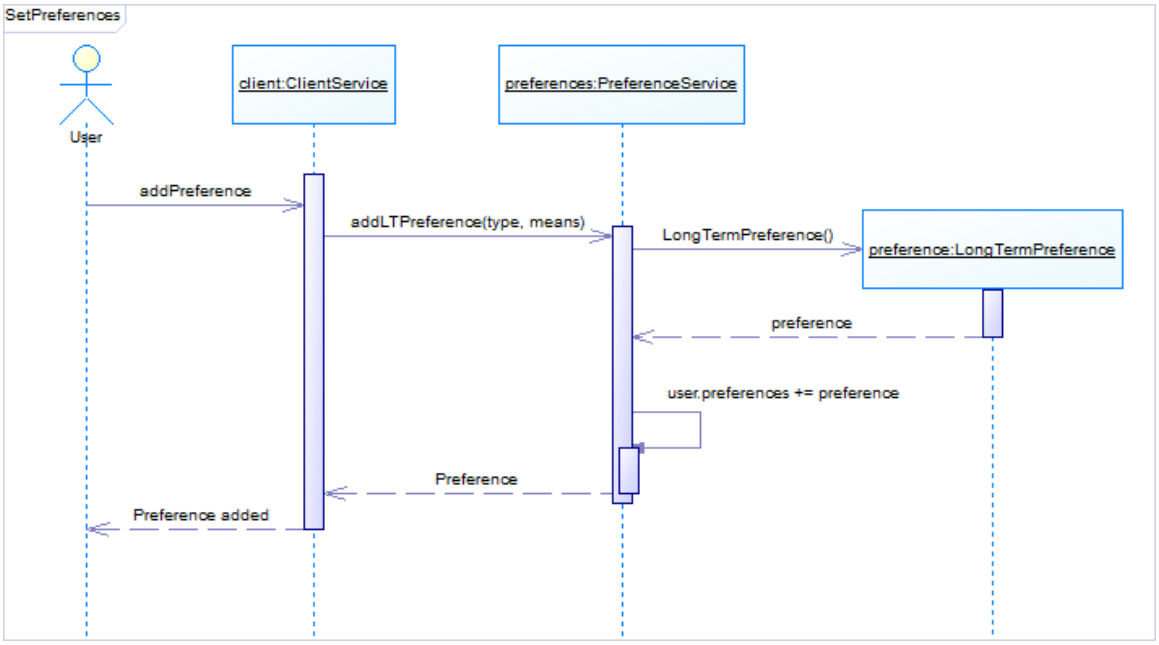
2.5.1. Create a meeting



In this sequence, the process of creating a meeting is shown. The component mainly used for this operation is the MeetingService. It creates a meeting based on the given parameters and then, using the ScheduleService, a check is performed to see whether there will be any overlaps in the schedule with other meetings if the new meeting is inserted in it. If any overlaps are detected, the insertion is not done, and the user is informed that the operation could not be completed. On the other hand, if the meeting can fit without any overlaps then it is inserted in the schedule and it is checked whether in the given parameters the starting location of the trip to the meeting is given in advance (not just chosen when the trip to the meeting starts). If that is the case, then the control is handed to the TripService component. Using the RouteService, PreferenceService and ExternalFactorService, it obtains all the routes that can be taken from the specified starting location to the meeting location, the preferences (general ones and the ones concerning the meeting type of the new meeting) and the external factors that may affect the routes. Using all of them, an ordered list of possible routes is created and the default route (the one by which the user can arrive to the meeting the quickest) is extracted and put as the selected route for the trip to the new meeting. Now that there is a selected route and the time that it will take to arrive to the destination can be roughly calculated, a check is performed to see whether there is any meeting in the schedule that now cannot be reached in time, given the selected route for it.

Another check is also performed to see whether now, that we have an approximate trip duration, there are breaks that can't be completed because there is no enough free time in schedule in the specified break period. If any problems are found during the two checks, appropriate warnings are sent to the user.

2.5.2. Set preferences



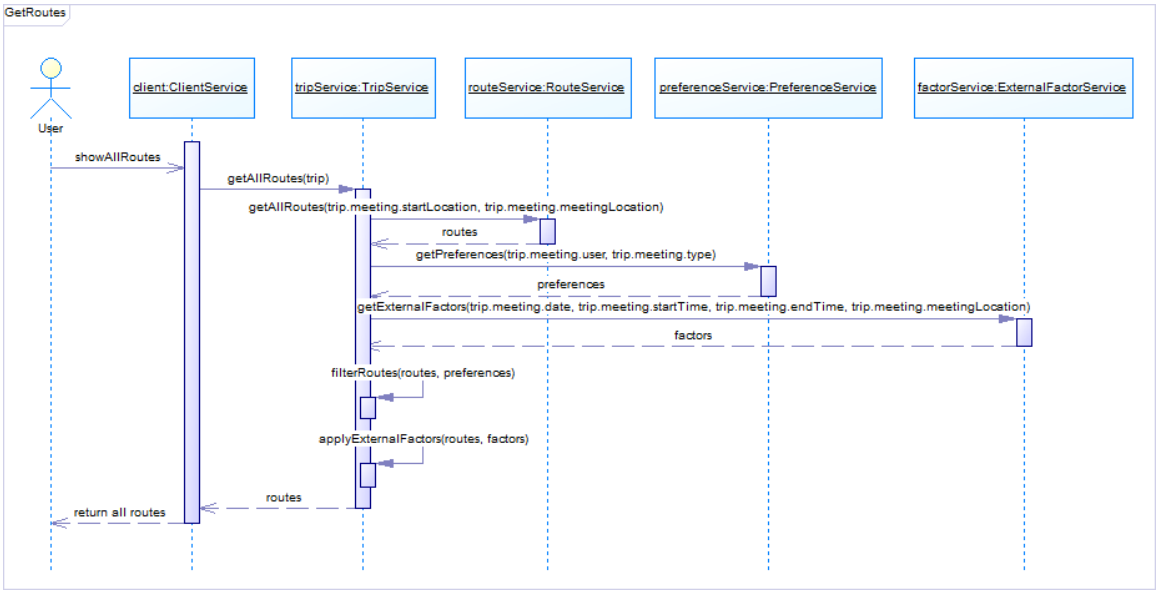
In the first part of the operation, a request for adding a new preference is sent to the server. The PreferenceService component uses the constructor to great a new LongTermPreference which is then added to the list of all the other user preferences in the PreferenceService component. After this, a success message is returned to the client and the user.

2.5.3. Choose route out of the proposed ones

The following two diagrams describe the operation of the user getting the proposed routes and picking one as the selected route for the trip to a meeting. There are two diagrams because the operation can clearly be separated into two distinct parts:

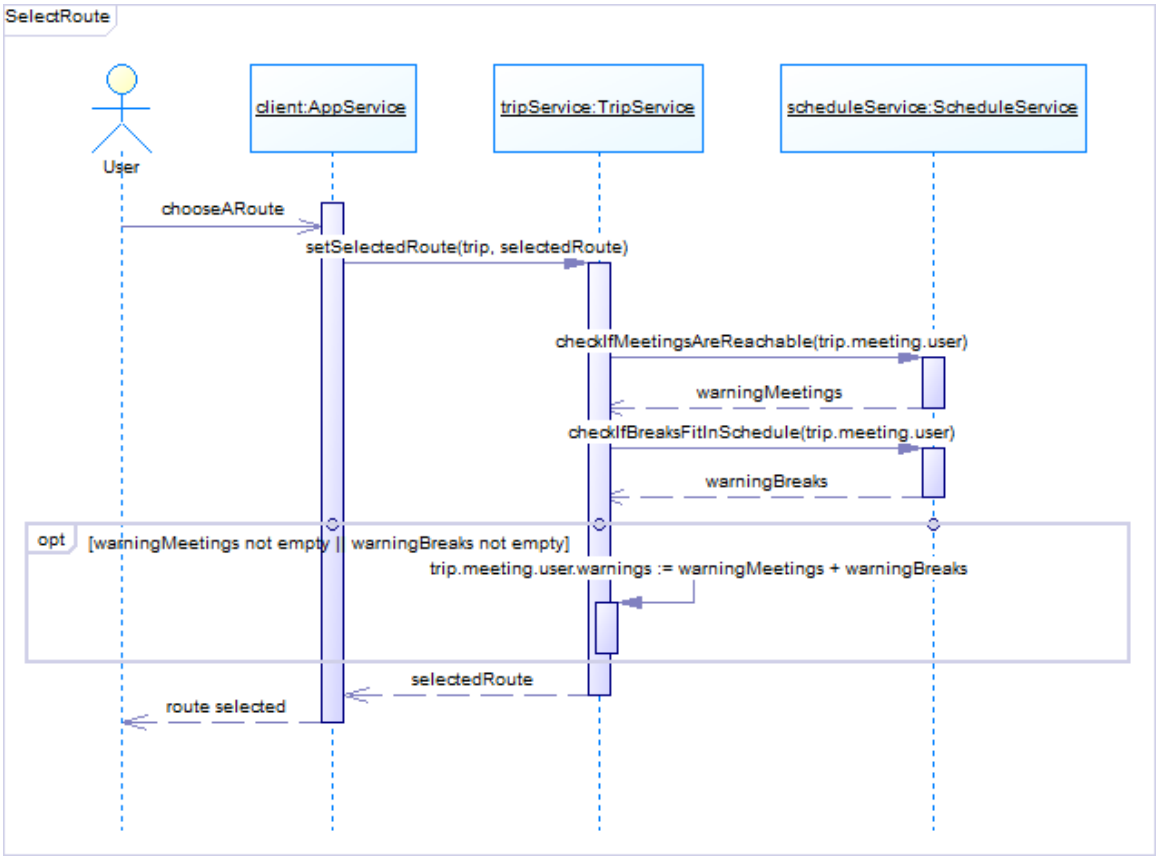
- The user receives all proposed routes from the starting location of the trip to the meeting location, taking into account the user's preferences and possible external effects
- The user chooses one route to be the currently selected trip route

2.5.3.1. Get all routes



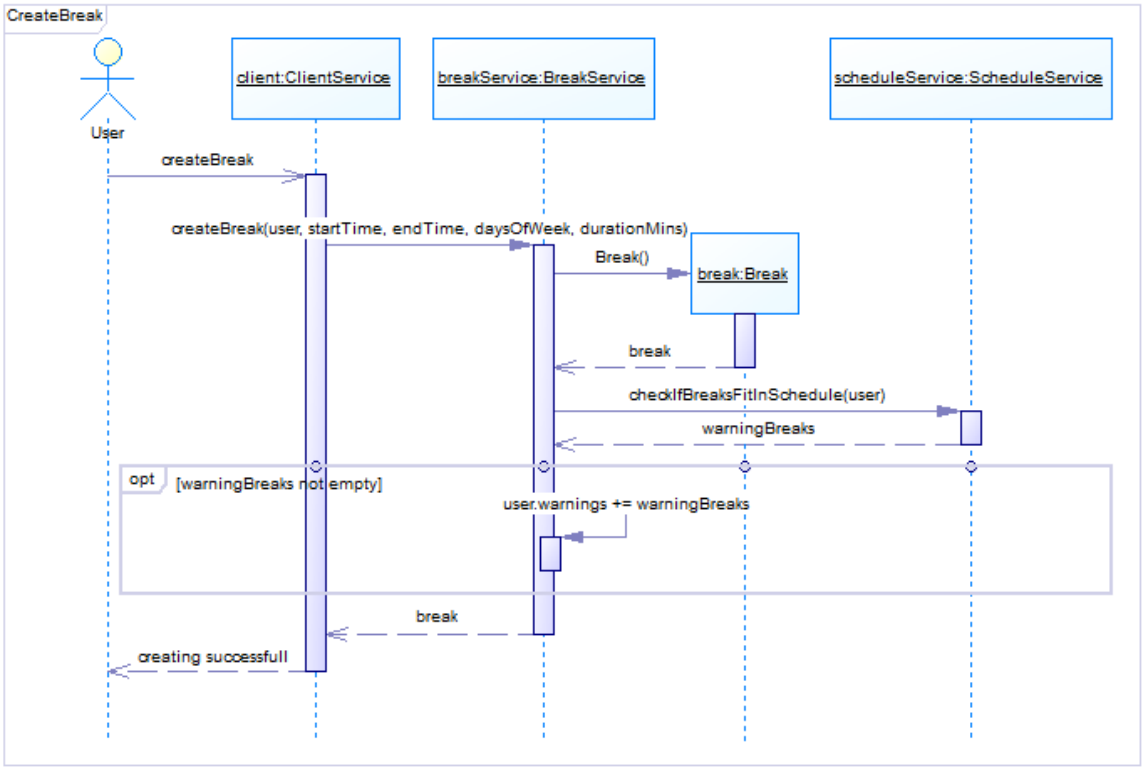
In the first part of the operation, a request for the routes is sent to the server. The TripService component receives information on all possible routes, the user's preferences and the external factors that may have some effect on the routes from the RouteService, PreferenceService and ExternalFactorService components, respectfully. The TripService component then determines which routes satisfy the user's preferences and whether some routes are not recommended due to some external factors. After this, the processed routes are returned to the client and the user.

2.5.3.2. Select a route



In the second part of the operation, the user sends through the client one route among the previously calculated ones to be the selected route for the trip. After the route is set, a check is performed using the ScheduleService component to see whether there are now meetings that are not reachable in time or breaks that can't fit inside the updated schedule. If some are discovered, the appropriate warnings will be shown to the user.

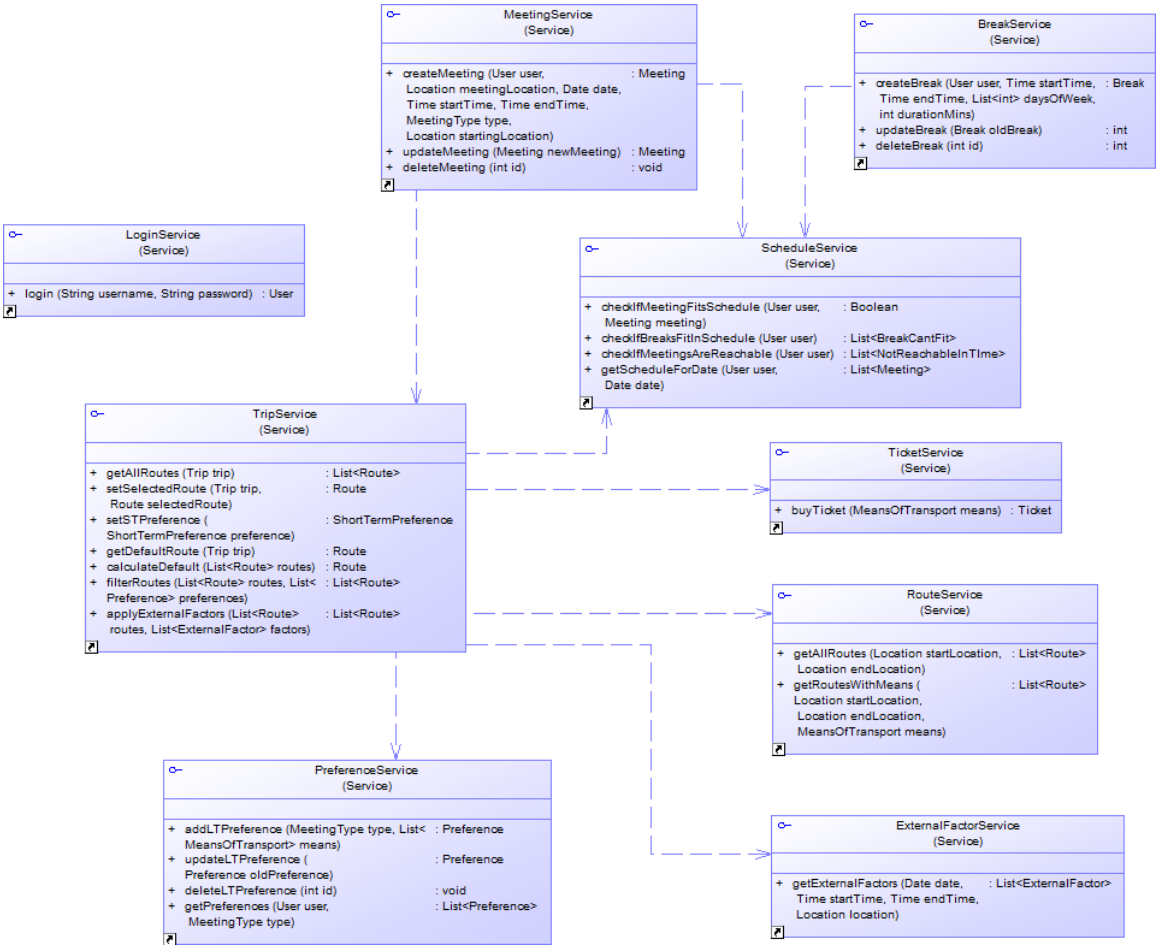
2.5.4. Create a break



In the sequence diagram shown above, the process of creating break is shown. The main component used for this operation is BreakService. It creates a break considering the break preferences specified by the user and then, using the ScheduleService, checks to see whether there will be any overlaps in the schedule with meetings or trips. In contrast to the operation which is performed while meeting is being created, the break is inserted in the schedule no matter if any overlaps are detected or not. If the recently added break overlaps with some previously added meetings or trips, the user only gets a warning. What is checked by the service in order to decide whether to throw a warning to the user or not is the break interval specified by the user, in which it is possible for him/her to have the break, the desired break duration and the day in which the specified break should take place.

2.6.Component interfaces

In the following diagram, the component interfaces are presented and the dependencies between the parts of the application server are shown. This information was already present in the class diagram presenting the services, but here it is shown more clearly.



2.7.Selected architectural styles and patterns

2.7.1. Overall architecture

The most suitable architecture for Travlendar would be three tier architecture. The presentation tier is the topmost level of the application. It displays information related to such services as adding meetings, setting up the preferences etc. It communicates with the other tiers by which it puts out the results to the client. It is the layer which users can access directly (the application's GUI). The application tier is pulled out from the presentation tier and, as its own layer, it controls the application's functionality by performing detailed processing. The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer should provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change.

As with the separation of any tier, there are costs for implementation and often costs to performance in exchange for improved scalability and maintainability.

Apart from the usual advantages of modular software with well-defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology. For example, a change of operating system in the presentation tier would only affect the user interface code.

2.7.2. Design patterns

- Layered architecture that improves reuse and maintainability

The concept of layers is used here in order to maximize separation of concerns, and to improve reuse and maintainability for the mobile application. However, the aim is to achieve the smallest footprint on the device by simplifying the design compared to a desktop or a Web application.

- Consider device resource constraints

Every design decision should take into account the limited CPU, memory, storage capacity and battery life of mobile devices. Battery life is usually the most limiting factor in mobile devices. Reading and writing to memory, wireless connections, specialized hardware and processor speed all have an impact on the overall power usage. The application should be optimized to minimize its power and memory footprint while considering performance during this process.

- Model View Controller (MVC)

The most mobile applications' core operation is to retrieve data from a data store and update the user interface with the newly requested information based on the user inputs. The logical sense is to tie the user interface components with the data store components. However, since the user interface

components are updated regularly to accommodate the changing user requirements and new technologies – more so than the data store components, we introduce extra coupling in our system. The aim of this pattern is to separate the components of user interface (View); core functionality and data (Model) and the response to user inputs (Controller).

- MVC in the context of Android applications

This is a brief overview of the components of the Android system. The main components are:

- Activity – represents a single user interface class. Activities are usually packaged together to form the UI components of the application.
- Service – allows tasks to be executed in background threads (such as networking operations) without affecting UI components.
- Content Provider – enables data to be stored within the application with the use of SQLite database or SharedPreferences (data stored in an XML file on the device).
- Broadcast Receiver – responds to announcements from the system (such as low battery warning) and provides notifications to the user.

Each Activity consists of a Java file and a corresponding XML layout file. We define the UI layout of the Activity in the XML file and implement the functionality and behavior in the Java file (separation of user interface with application logic). When we use widget components in our Activity, we first have to define the widgets as XML elements in the layout file to instantiate the widget and its attributes. The handling of the behavior of the widget in response to user actions is then implemented in the Java file.

In terms of the MVC pattern, for applications with large UI components it might be worthwhile following this pattern of design as the constant update and modifications of the user interface due to the rapidly changing nature of user requirements will result in less work maintaining the core business logic of the application.

- Layered Abstraction of the client's part

The Layered Abstraction pattern consists of a hierarchy of layers. At each layer, there are components that work together within the same level of abstraction, with the layer below providing functionality for the layer above.

The Android architecture follows this Layered Abstraction pattern. It consists of four layers of abstraction with direct communication only between a layer and the layer directly above or below it:

- Applications – this layer consists of a set of core applications as well as developed applications.
- Application Framework – layer which contains the main components of the Android system (Activity, Service, Content Provider, Broadcast Receiver) as well as other system components.
- Libraries – consists of core Android system libraries as well as a lightweight relational database management system – SQLite.
- Linux Kernel – contains device drivers which provides communication with the device's hardware.

With the Layered Abstraction pattern, since the Android system follows the abstraction layer architecture as discussed above, we can say that this pattern forms the foundation for all Android applications that makes request to services and receives notifications from them.

2.7.3. Other design decisions

Travlendar+ is an application that combines different transport services and calculates the optimal route from the location of one meeting to the next one. For this reason, external services are being used.

The application primarily needs and integration with a map service. The best option will be chosen depending on the city for which the application is being developed. One of the options is using Google Maps as it also helps with route calculation and location.

Moreover, public transport services are key part of the application. That is why all the services that are offering APIs will be integrated in the application. The services also depend on the city, but they will be integrated in the same way, using their APIs.

3. Algorithm design

3.1. Route calculation - external factors

Route calculation is an important part of our application. However, it will be handled by external services that have already implemented this functionality, for example - Google Maps or any other map service that offers this functionality. The main objective of Travlendar+ is to filter the routes according to the user preferences and check if the meetings, the breaks and the trips fit in the schedule.

3.2. Filter and sort routes (by preferences)

When the information about all the possible routes is gathered, Travlendar+ needs to filter the routes and show only the ones that match the user's long term preferences. Also, the limitations due to external factors, such as public transport strikes, road work or weather conditions, should be taken into account.

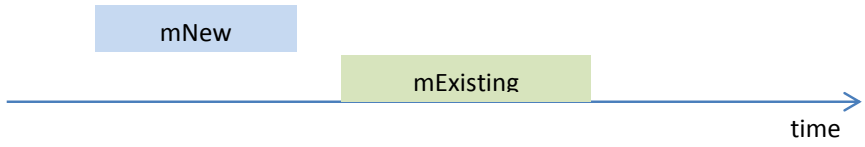
1. Get a route form the external service
2. Check if the user has selected the specific transport mean in his long term preferences
3. Check if the minimum and the maximum trip distance assigned to that transport mean by the user, is in accordance with the trip distance offered by the external service
4. Check if there is info for any external factors that may affect the trip
5. If there is, show warnings
6. If all of the conditions mentioned above are satisfied, add the route to the routes list
7. Repeat steps 1-5 until all the routes provided are checked
8. Show the list to the user

3.3.Check if meeting fits

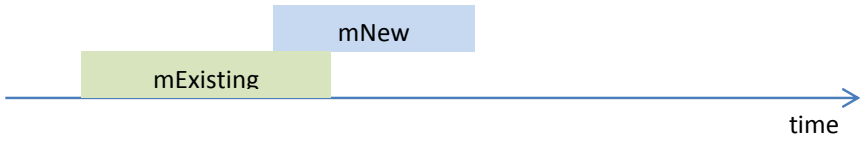
One of the key functionalities of the application is to manage the schedule for the user and check if all the meetings fit in the schedule.

```
Flag = 0 - no overlap meetings
Loop for every meeting already in the schedule for the given day
      Check if the new meeting starts before the other meeting ends
            If yes -> check if the new meeting ends before the other one
starts
                  If yes -> continue; (1)
                  Else -> increment the flag; break; (2) (3)
      Check if the new meeting ends after the other one starts
            If yes -> check if the new meeting starts after the other one
ends
                  If yes -> continue (4)
                  Else -> increment the flag; break; (5) (6)
If flag == 0 -> add meeting to the schedule
```

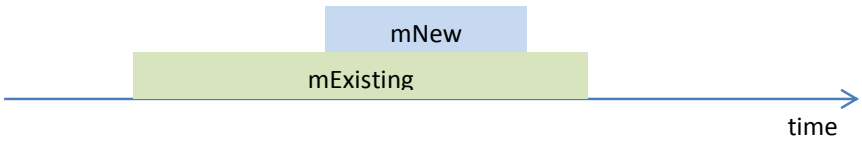
(1)



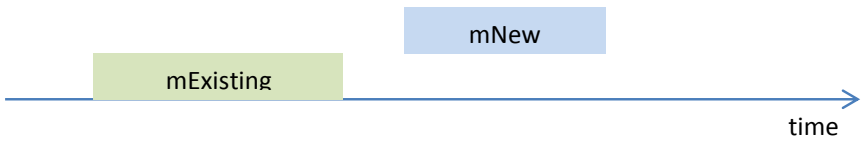
(2)



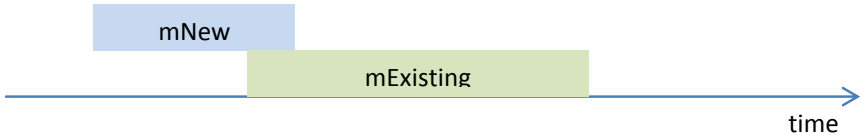
(3)



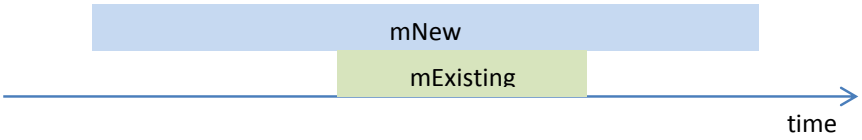
(4)



(5)



(6)

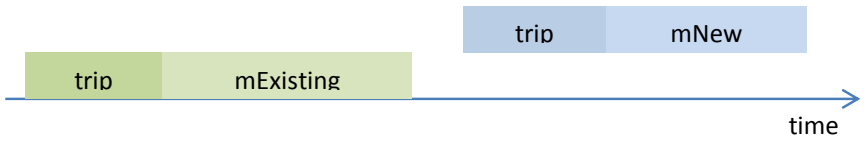


3.4.Add warnings for time needed to reach meetings (trip duration)

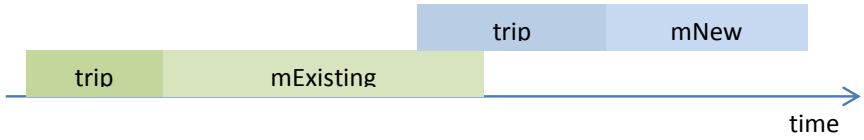
When a new meeting is added, the application should offer an option to the user to arrange a trip to the next meeting location. According to the user choice, it should show warnings if there is not enough time to reach the other location with the selected route. This functionality is applied by the following algorithm:

```
Calculate the start time of the trip by subtracting the trip time from the
starting meeting time;

Flag = 0 -> no meeting overlapping with a trip

Loop for every meeting already in the schedule for the given day, happening
before and except the meeting for which the trip is being calculated

Check if the start trip time is after the end of the meeting
      If yes -> continue; (1)
            Else -> increment the flag; break; (2)
If flag > 0 then show warnings;
```

(1)



(2)

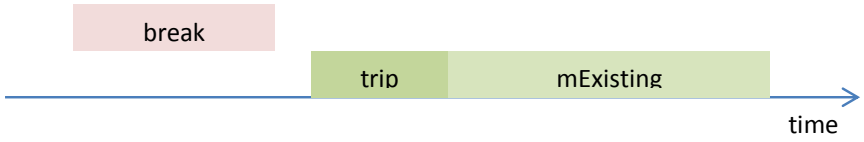


3.5.Check to find slot for break in the schedule

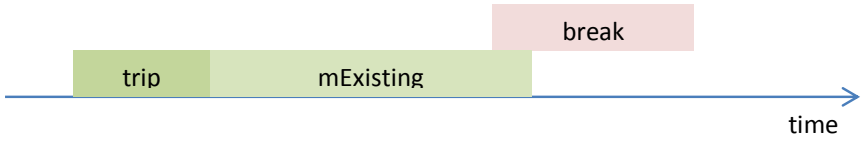
Check all the free slots in the given period and see if there is any long enough to fit the break.
If there are no free slots for a break in the given period - send warnings, but add the break anyway.

```
Flag =  0 - no overlap meetings and breaks
Loop for every meeting already in the schedule for the given day
      Check if the break starts before the meeting ends
            If yes -> check if the break ends before the start of the meeting
trip
                  If yes -> continue; (1)
                  Else -> increment the flag; break; (2) (3)

      Check if the break ends after the meeting trip starts
            If yes -> check if the break starts after the meeting ends
                  If yes -> continue (4)
                  Else -> increment the flag; break; (5) (6)
If flag == 0 -> add break to the schedule
```

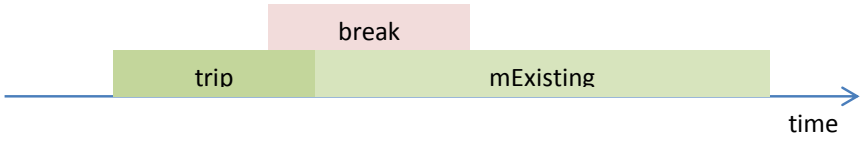
(1)



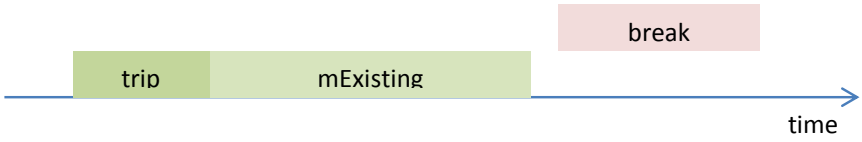
(2)



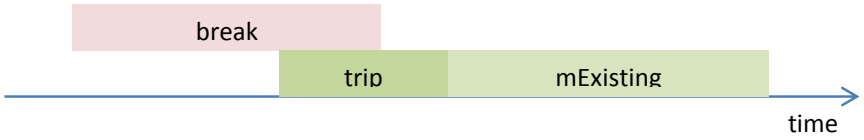
(3)



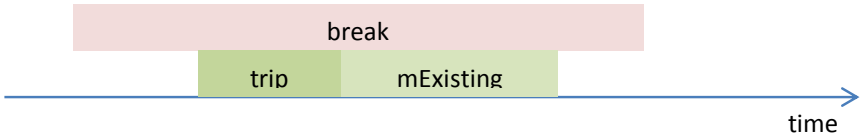
(4)



(5)



(6)



4. User interface design

The mock-ups for this application were presented in the RASD Document in section 3.1.1. User interfaces. At this point, there are no new functionalities that can be presented with new mock-ups for the application.

5. Requirements traceability

The design of this application aims to meet all the goals and requirements that have been previously specified in RASD. Below are listed the design components to which Travlendar+ requirements and goals are mapped:

[G1] - The user can be recognized by providing a form of identification.(Requirements **[R1] - [R2]**)

- LoginServiceImpl

[G2] - Allow the user to specify meetings in their schedule. (Requirements **[R3] - [R5]**)

- MeetingServiceImpl
- ScheduleServiceImpl
- RouteServiceImpl
- TripServiceImpl

[G3] - The user can see the schedule with the time periods for the meetings and the estimated time needed to get to them, while being notified if some meetings can't fit into the schedule or are not reachable in time. (Requirements **[R6] - [R8]**)

- MeetingServiceImpl
- ScheduleServiceImpl
- RouteServiceImpl

- PreferenceServiceImpl
- TripServiceImpl

[G4] - The user is provided with multiple routes to reach every meeting. (Requirements **[R9] - [R12]**)

- RouteServiceImpl
- TripServiceImpl

[G5] - The application allows the user to specify preferences that should be taken into account while calculating and suggesting the routes. (Requirements **[R13] - [R17]**)

- PreferenceServiceImpl

[G6] - The routes provided to the user take into account external factors that may affect them during the travel time. (Requirement **[R18]**)

- ExternalFactorsServiceImpl
- RouteServiceImpl

[G7] - Allow the user to specify breaks that should be considered in the schedule, in between meetings, in a certain time period. (Requirements **[R19] - [R20]**)

- BreakServiceImpl
- ScheduleServiceImpl

[G8] - The user can buy public transport tickets through the application for trips that include these means of transport. (Requirements **[R21] - [R26]**)

- TicketService
- PreferenceServiceImpl
- TripServiceImpl

Please note that all of the listed components use appropriate 'provided' or 'required' interfaces which are explained in details in the class diagram added as a supplement to the component diagram shown earlier in this document.

6. Implementation, integration and test plan

6.1. Implementation plan

The implementation of the Travlendar+ system will be done module by module and component by component. The order in which it is be carried out depends on a number of factors like the complexity of the modules and services, the dependence of other modules on the component being implemented and to the system as a whole, and it should also take into account the possibility of discovering flaws with the proposed design. The later should be dealt in a way that, if such an unfortunate event does

happen, the flaws should be found and corrected as soon as possible, to limit the cost of the change of design.

In this sense, the components of the Travlendar+, could be grouped in the following way, with the order specifying the order of implementation:

1. Model
2. ScheduleService and TripService
3. RouteService, TicketService, ExternalFactorService
4. PreferenceService, MeetingService, LoginService
5. Client and Router

(note that by specifying the names of interfaces of components, we are also considering the concrete implementations, in which ever number they exist)

The Model is proposed to be the first component that is implemented because all parts of the application server will be using some element of it and its role in allowing some service to communicate with the DBMS in the DatabaseServer component is crucial to the whole application. This also eliminates the possibility of every team and developer implementing a part of the Model on the fly, while implementing some service, which could lead to problems with integration and definitions of the same data concepts in different ways.

The second group consists of the ScheduleService and TripService components. They were identified as the most high risk part in the system and it would be advisable to focus on their implementation as soon as possible because there is a high chance that their implementation will take longer than of the other components. They are singled out for the following reasons:

- ScheduleService is one of the more complex components of the system, due to its functionalities of keeping track of all of the schedules and their feasibilities (the functionalities it performs and the way it does them were more closely analyzed in the Algorithms section of this document).
- TripService is the component in the system that has the most connections to any other part and so, in a way, it has a vital role of grouping many functionalities of the system and providing access to them to the user. Therefore, it is crucial that the implementation of this specific component is carried out in a way to guarantee very high tolerance to fault and good performance because it represents a possible bottleneck of the application.

The third group consists of modules that heavily depend on external, already implemented services and components (not including the DBMS component). These components, named in general considering the functions they will provide, are: MapExService, PublicTransportExService, FactorsExService, SharingSystemExService. Because these components already exist, the parts of the Travlendar+ system responsible with the task of communicating with them (the ones belonging to this group) have to be adapted to these external services. Although this fact has already been considered in the design of the system, this still represents a point of possible problems that require slight updates and changes to the design. In that sense, their implementation should be done earlier than others.

In the fourth group are services whose functions, while being some of the most crucial elements of the system, are CRUD operations that are relatively simpler than others that were already mentioned. Considering this, these components are not as risky as others because the time needed to complete them can be relatively accurately predicted.

The fifth group consists of components that are mainly used to redirect request and offer the user with a way to communicate with the modules and services which will be carrying out the main functions of the system.

6.2.Integration and testing

6.2.1. Entry Criteria

The integration of components and its testing should start as soon as possible, but before they can commence, some conditions must be met. First of all, the external services and their APIs that are going to be used in the application should be available and ready. This applies to the already mentioned MapExService, PublicTransport-ExService, FactorsExService, SharingSystemExService and to the DBMS and the server on which it will be running on.

Next, the modules which are being integrated should have at least the operations concerning one another created, if not completed completely. The operations that have been developed should pass the unit tests in order to be sure that the components are working fine on their own and that if an integration test fails, the problem lies in the in the integration itself.

To be precise, in order to have i integration tests that have some meaning, the completion of components before the start of testing should at least be:

- MeetingService - 50% (at least one of the operations must be completed)
- BreakService - 50% (same, at least one operation must be fully developed)
- ScheduleService - 90% (the checking operations should be complete, because the dependant services need them)
- TripService - 80-90% (in most cases, more than one operation of the component is used together with each other)
- TicketService - 90-100%
- ExternalFactorService - 90-100%
- RouteService - 90-100%
- PreferenceService - 70% (it has to be able to complete the sending of preferences and, at least one other operation concerning the data persistence)

6.2.2. Elements to be integrated

The Travlander+ system is composed of a number of components, as already shown, and the integration process can be grouped in the following way:

- Integration of components with the DBMS
- Integration of components with the (other) external services
- Integration of the components of the application server
- Integration of the client (mobile application) and the application server

Integration of components with the DBMS - This group includes the integration of every part of the application server that uses the external server with the database, in any way. The specific integrations in this group are the following ones:

  - LoginService, DBMS
  - MeetingService, DBMS
  - BreakService, DBMS
  - ScheduleService, DBMS
  - TripService, DBMS
  - PreferenceService, DBMS

Integration of components with the (other) external services - In this group, we include the integration of every part of the Travlander+ system with an already existing and functional external service (without counting the DBMS). They are the following ones:

  - RouteService, MapExService
  - RouteService, SharingSystemExService
  - TicketService, PublicTransportExService
  - ExternalFactorService, FactorsExService
  - Client, MapExService

It should be noted that every part of the system will be connected to a metric data gathering service in order to keep track of the performance.

Integration of the components of the application server - Here, the integration between the parts of the application server is conducted. It is composed of:

  - MeetingService, ScheduleService
  - BreakService, ScheduleService
  - TripService, TicketService
  - TripService, ExternalFactorsService
  - TripService, PreferenceService
  - TripService, RouteService

Integration of the client and the application server - This is carried out to enable the sending of requests to the server via the mobile application which the user will be utilizing.

6.2.3. Integration Testing Strategy

Considering the implementation plan and the overall architecture of the Travlendar+ system, the chosen strategy for the integration testing is the bottom-up one. This allows us to start the integration and it's testing while not waiting for the completion of the development and the unit testing of each component in the system. Considering the integration of two components, we would assume that, in best case, they have been implemented fully and that their respectful unit tests pass. However, the integration can, in some cases, start, if necessary, before the implementation has been completed. This can be allowed if the part of the component needed for that specific integration has been completed and tested.

Since the opted solution is to start from the bottom-up, that means that the among the first integrations performed will have the already built external components in them. Since the application rests on these services and the communication with them, this order of integration and testing will enable the earlier detection of errors in these critical parts.

It should be noted that it can be assumed that each integration in the same level of hierarchy (defined by the groups of integrations in the previous chapter) is independent and there is no specific order in which to complete them. In this way, the integration process and its testing are more flexible

6.2.4. Sequence of Component/Function Integration

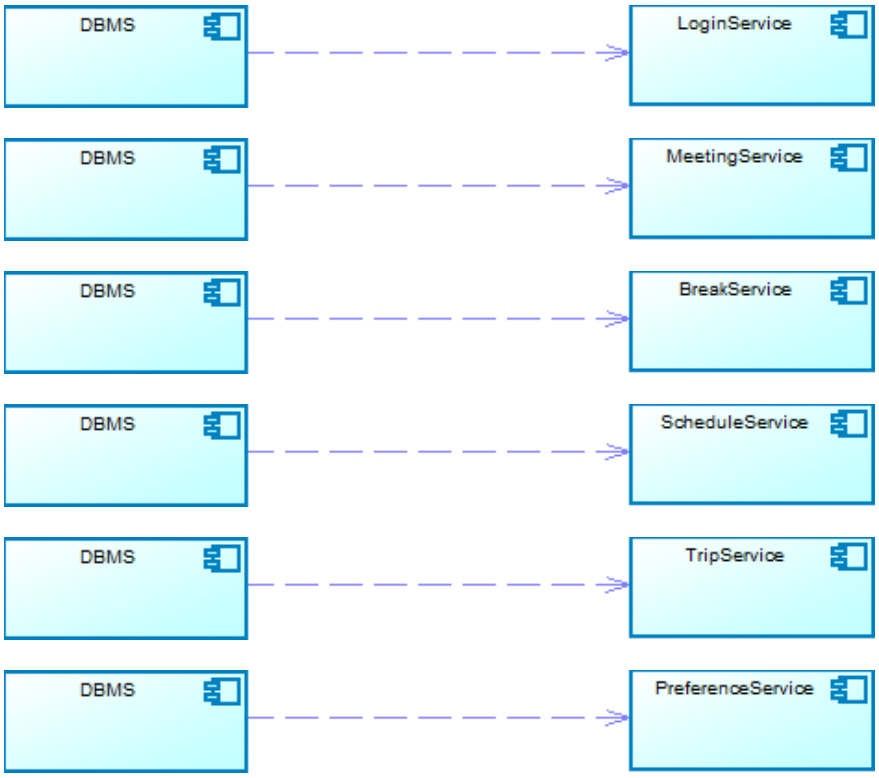
In this section, the list and order of every integration that is performed is shown. As already stated, the integration will be performed from the bottom-up. As a notation used, the arrow pointing from the first component in the integration to the second means that the second one is using the services of the first component.

It should be noted that there will be no explicit integration of the Model with any of the other components. This is because the nature of the component, the extent of the usage and dependency of other components on it and the implementation plan, that clearly states that the Model will be the first part that is implemented, mean that the integration itself is already being done during the implementation phase of the depending components and its correctness will inherently be tested by the unit tests of each component.

It should also be noted that the "DBMS" notation refers to the external server with the database and its DBMS and that every "service" component considers all of the implementations, whatever they may be, of the specified interface.

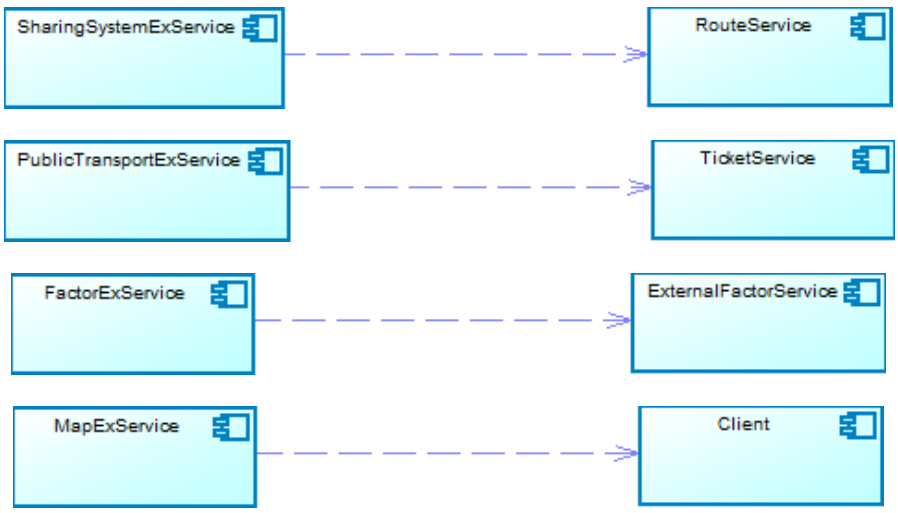
Integration of components with the DBMS

The integration in this section is carried out between the database server and a specific component of the application server of Travlendar+. Each of the components shown here has the operations needed to work with the database, concerning the parts of the Model that they are using. In that sense, these integrations have the goal to successfully apply the specified operations on the database, using the DBMS.



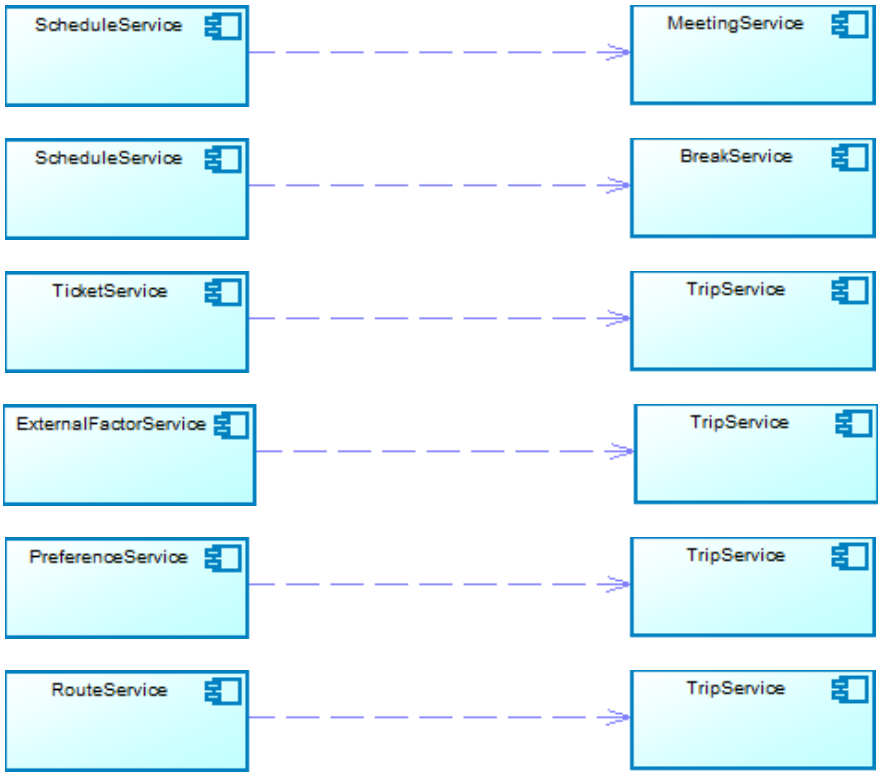
Integration of components with the (other) external services

In this section, the other external service component, not considering the database, are integrated with either the components of the application server that dedicated to their usage them or the client mobile application. Considering the external service components are already developed and that the proposed integration strategy is bottom-up, it makes sense to carry out these specific integrations as soon as possible.



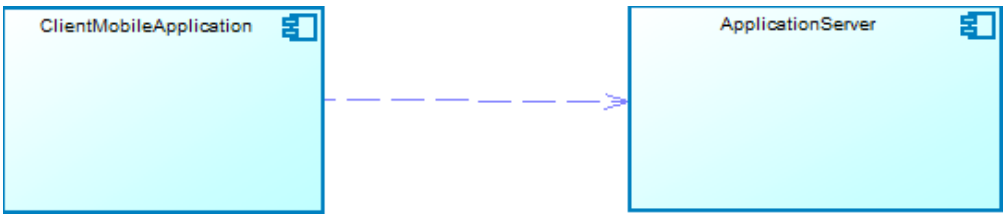
Integration of the components of the application server

Moving upwards through the hierarchy, we integrate higher level components of the application server that use other parts of it. After this phase of integration and testing is complete, the application server should be integrated fully, while being connected to the database server.



Integration of the client and the application server

The final phase of the integration and testing, with the highest level components. It should be done last, after at least most, if not all, of the application server parts have been developed and integrated.



7. Effort spent

7.1.

Description of the task	Hours
Purpose, scope, definitions	1
Component view	6
Class view	6
Runtime view – sequence diagrams	7
Component interfaces	1
Algorithm design	3
Implementation, integration and test plan	9

7.2.

Description of the task	Hours
Purpose, scope, definitions	2
High-level components and their interaction	3
Deployment view	4
Runtime view – sequence diagrams	4
Selected architectural styles and patterns	8
Algorithm design	9
Implementation, integration and test plan	2

7.3.

Description of the task	Hours
Purpose, scope, definitions	3
Runtime view – sequence diagrams	7
Selected architectural styles and patterns	4
Algorithm design	8
Other design decisions	2

Requirements traceability	5
Implementation, integration and test plan	4

8. References

- Specification document “AA 2017---2018 Software Engineering 2—Mandatory Project goal, schedule, and rules”
- “Architectural patterns for Mobile Application Development” - https://blog.inf.ed.ac.uk/sapm/2014/02/13/architectural-patterns-for-mobile-application-development/
- “architecture of mobile software applications” - https://www.slideshare.net/hassandar18/architecture-of-mobile-software-applications
- Slides – “Design part I and II.pdf”
- Slides – “Architecture and Design in Practice”
- Slides – “Verification and Validation”