

Politecnico di Milano

A.Y. 2017/2018

Software Engineering 2: ***Travlendar+***

**Design Document**

Matteo Biasielli - Emilio Capo - Mattia Di Fatta

v. 0.1

**Table of Contents**

**1. Introduction**

1.1. Document purpose..………………………………………………...………………….3

1.2. Definitions, Acronyms, Abbreviations………………………………………….3

1.2.1 Definitions……………………………………………………………………….3

1.2.2 Acronyms………………………………………………………………………..3

1.2.3 Abbreviations…………………………………………………………………..3

1.3. Reference Documents…………………………………………………………………4

1.4. Document Structure……………………………………………………………………5

1.5. Revision History…………………………………………………………………………5

**2. Architectural Design**

**3. Algorithm Design**

**4. User Interface Design**

**5. Requirements Traceability**

**6. Implementation, integration and test plan**

**7. Effort Spent**

**1. Introduction**

* 1. **Document purpose**

This document has to be intended as a general guide for the correct development of the Travlendar+ application. The content of this document follows and is based on the content of the RASD document. This document is meant to be a reference for any person who has an interest in the project. This includes, but is not limited to, development team members, stakeholders and end users.

* 1. **Definitions, Acronyms, Abbreviations**
     1. **Definitions**
* **User**: actor that is using the application and may want to access all functionalities.
* **Application**: with the term application we are talking about the desktop version, the website and mobile version of the Travlendar+ system.
* **Scheduling**: action performed by a user that is adding a new activity to his personal calendar.
* **Flexible Activity**: An activity with starting and ending time larger than the duration.
* **Fixed Activity**: An activity with fixed starting and ending time.
  + 1. **Acronyms**
* **RASD:** Requirements Analysis and Specification Document
* **DD:** Design Document
* **UI:** User Interface
* **API:** [Application programming interface](https://en.wikipedia.org/wiki/Application_programming_interface)
* **UXD**: User Experience Diagram
* **UML**: Unified Modeling Language
* **GPS**: Global Positioning System
  + 1. **Abbreviations**
* **[Gn]:** the n-th goal
* **[Rn]:** the n-th requirement
* **[NFRn]:** the n-th non-functional requirement
* **[An]:** the n-th assumption
* **[Cn]:** the n-th constraint
* **[UIn**]: the n-th user interface example
  1. **Reference Documents**
* Mandatory project assignments for the A.Y. 2017/2018 available on the beep’s page of the Software Engineering 2 course.
* Projects examples and other documents available on the beep’s page of the Software Engineering 2 course.
  1. **Document Structure**
* **Introduction:** This is the very first part of the document.

In this section it’s possible to retrieve general information about the Design Document. The purpose and intended audience of the document are specified here.

In addition, Acronyms, Definitions and Abbreviations are defined in this section in order to make it easier, more concise and clearer to read the rest of the Design Document.

* **Architectural Design:** This part represents the second chapter of the document. Here the reader can find the architecture of the system components at various levels and contexts.

First of all, a high-level overview of the components and the way they’re connected is provided in this section. Following this, some components will be analysed in detail and their internal architectures will be showed for a matter of clarity.

* **Algorithm Design:** The most important algorithms that will be implemented in our application are described here, both with natural language and with java code/pseudocode.
* **User Interface Design:** Some User Interface samples have already been provided in the RASD document but they’ll be extended and some will be added in this section of the Design Document.

In addition, further explanation about the already existing UI will be added here, together with a detailed mapping of the User Interfaces into functional requirements and non-functional requirements.

* **Requirements Traceability:** Design choices are mapped into functional and non-functional requirements here.
  1. **Revision History**
* **v. 0.1 [28 Oct 2017]**: added the whole “Introduction” section.

**2 Architectural Design**

In this section, we first provide a general overview on Travlendar+

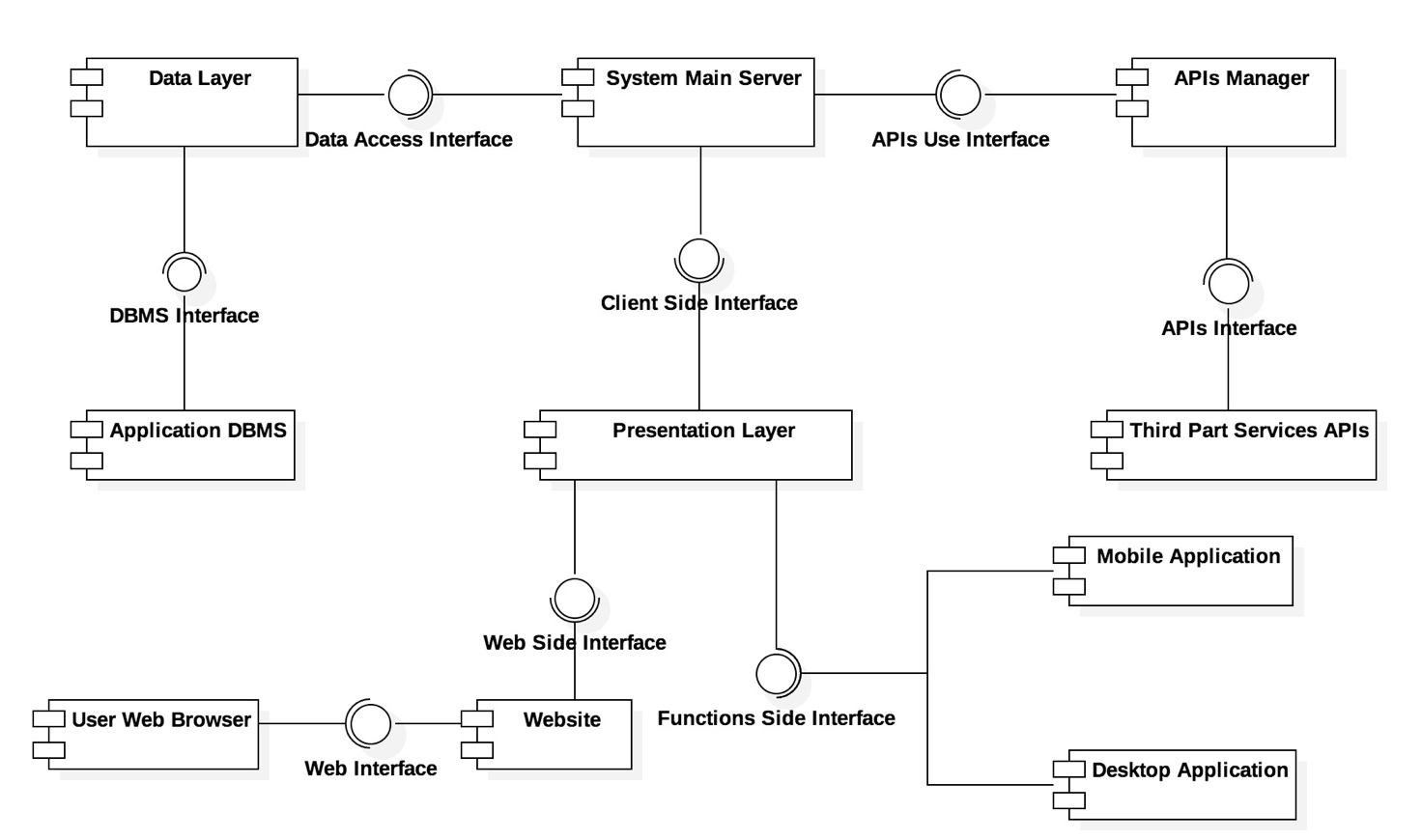
system by means of a general Component Diagram. Then it is given a Component View on some important components again by means of Component Diagrams.

Moreover, a Deployment View and a Runtime View of the system is provided in order to describe nodes with their components, protocol for their interaction and how they are expected to work (by means of Sequence Diagrams).

Eventually, we describe in detail all components’ interfaces that can be found in the general Component Diagram.

**2.1 System Overview**

The following diagram describes the whole Travlendar+ system with all its components (i.e. software modules).



***Data Layer***

This component deals directly with system’s DBMSs allowing other components to write, read and update data in the DBs properly (i.e. in a secure, consistent way) using the proper DBMS. It provides a single interface outward in order to systematize access to data stored in DBs and hide the internal complexity and implementation of DBMSs and DBs.

***Application DBMSs***

This particular component is used as generalization of the two DBMSs used in Travlendar+: one to manage users’ personal data and calendar, one to manage all relevant internal information used by Travlendar’s server to works properly (for further information see Component View section).

It provides an appropriate interface to allow the Data Layer component, and only it, to query DBMSs.

***System Main Server***

This is the core component of the server side of the system and of the application in general. It’s composed by three essential components: the Computation Unit, the User Side Unit and the Dynamic Event Check System. It manages, with the aid of the APIs Manager and the Data Layer, all the Travlendar’s functionalities described in the RASD document, except for the presentation side, managed entirely by the Presentation Layer (see below for further information).

***APIs Manager***

This component is used to homogenize the different kinds of APIs provided by third part services, in other words it adapts the external APIs to the system in order to make them easily usable. By doing this, the APIs Manager tries also to optimize the access and the usage to the APIs by means of an internal optimizer component. It provided the System Main Server with the APIs Use Interface, used to exploit third part services while masking their implementation.

***Third Part Services APIs***

This component is just a collection of all needed APIs with their internal representation, tools and external references.

***Presentation Layer***

The Presentation Layer component is the one appointed to show the front-end of the Travlendar+ system. Making use of the Client Side Interface provided by the System Main Server, it grants access to system’s data to users w.r.t. confidentiality (i.e. only authorized users/clients will have access to data they’re allowed to access and that are made accessible) by means of a website and the mobile and desktop application.

It provides two different interfaces, the Web Side Interface and the Functions Side Interface, as the user can interact in two ways with the system: through the website where he can download the application, find useful information (e.g. FAQs, forum) and additional documentation, or through the desktop/mobile application by means of which he can exploit Travlendar+ functionalities.

***Website***

This component models the website page of the project where a user can download both the desktop and the mobile version of the application and where he can find support documentation provided by developers themselves. It makes use of the Web Side Interface to connect to the system.

***Mobile Application***

The Mobile Application component represents an abstraction of the mobile version of Travlendar+, used in this diagram to show its interaction with the whole system. It makes use of the Functions Side Interface to connect to the system.

The core function of this component is to keep track of the deadlines contained in the travel options provided by the Computation Unit in the System Main Server in order to send notification to the user and notify him of possible/critical changes in their travel options.

***Desktop Application***

The Desktop Application component represents an abstraction of the desktop version of Travlendar+, used in this diagram to show its interaction with the whole system. It makes use of the Functions Side Interface to connect to the system.

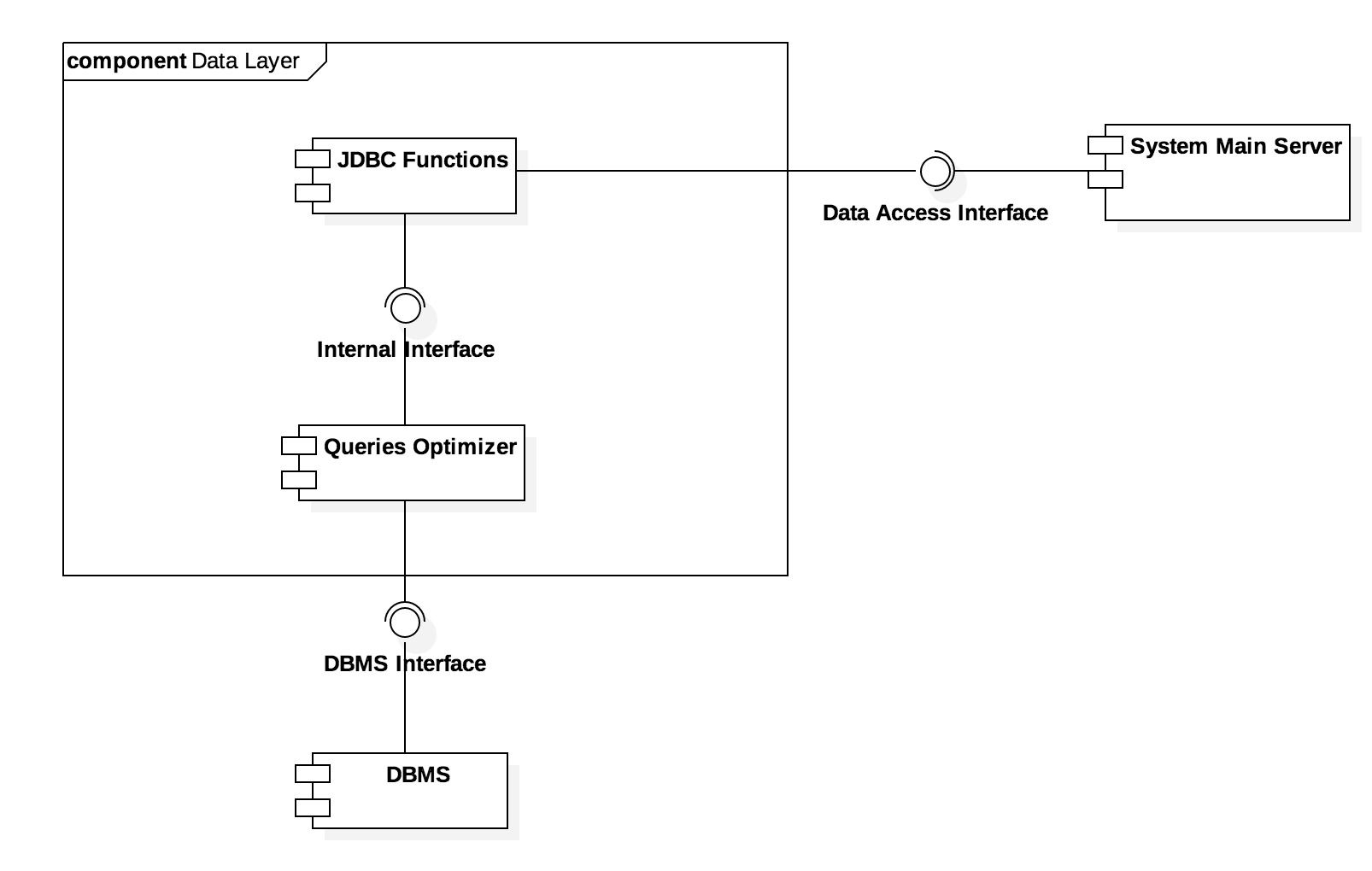
The core function of this component is to keep track of the deadlines contained in the travel options provided by the Computation Unit in the System Main Server in order to send notification to the user and notify him of possible/critical changes in their travel options.

***User Web Browser***

This component models the web browser used by the user to access the website. It’s an external object with respect to our system, but its corresponding component is necessary in order to model its interactions with the website.

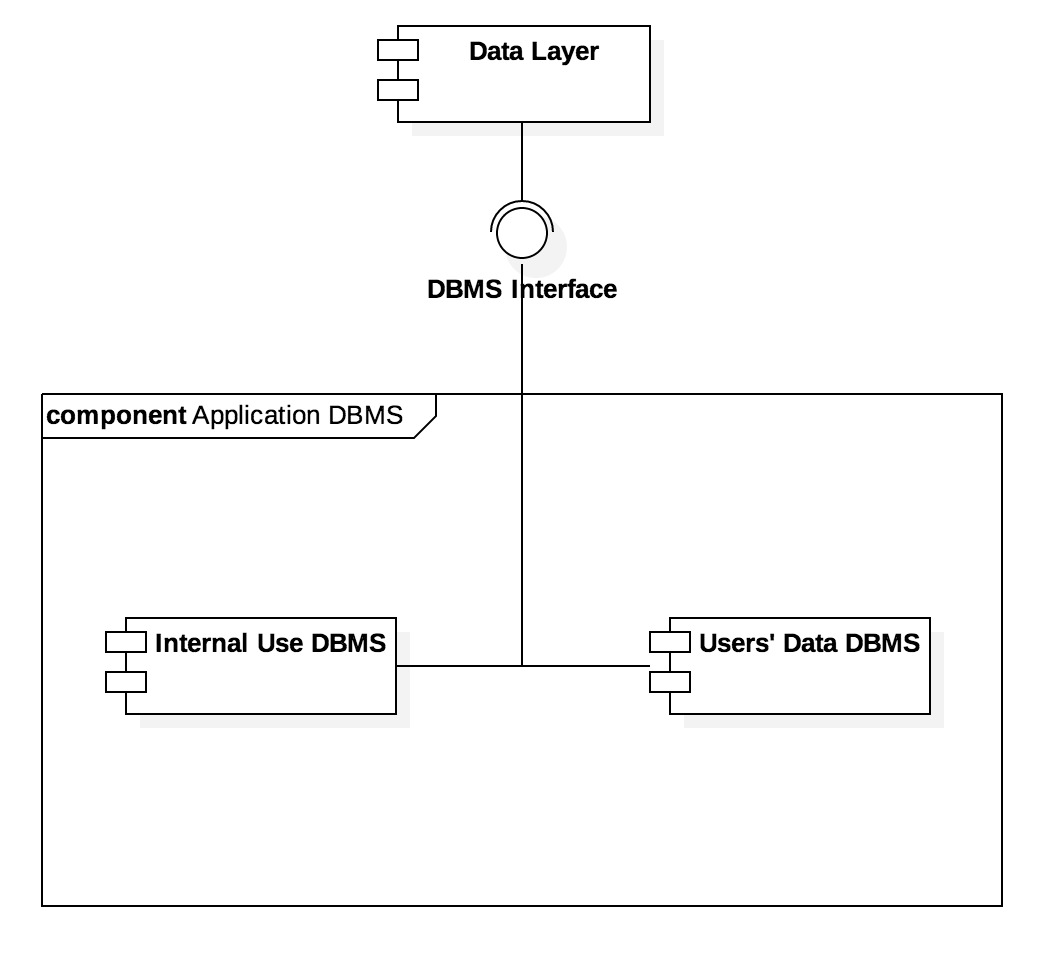
***2.2* ComponentView**

***Data Layer***

******

The Data Layer module is composed by two sub-components: the JDBC Functions Component and the Queries Optimizer Component. The former is a collection of tools and references used to integrate and use the JDBC library (Java library for DBs handling). The latter is used to optimize the formulation of queries by means of JDBC tools.

***Application Database Management Systems***

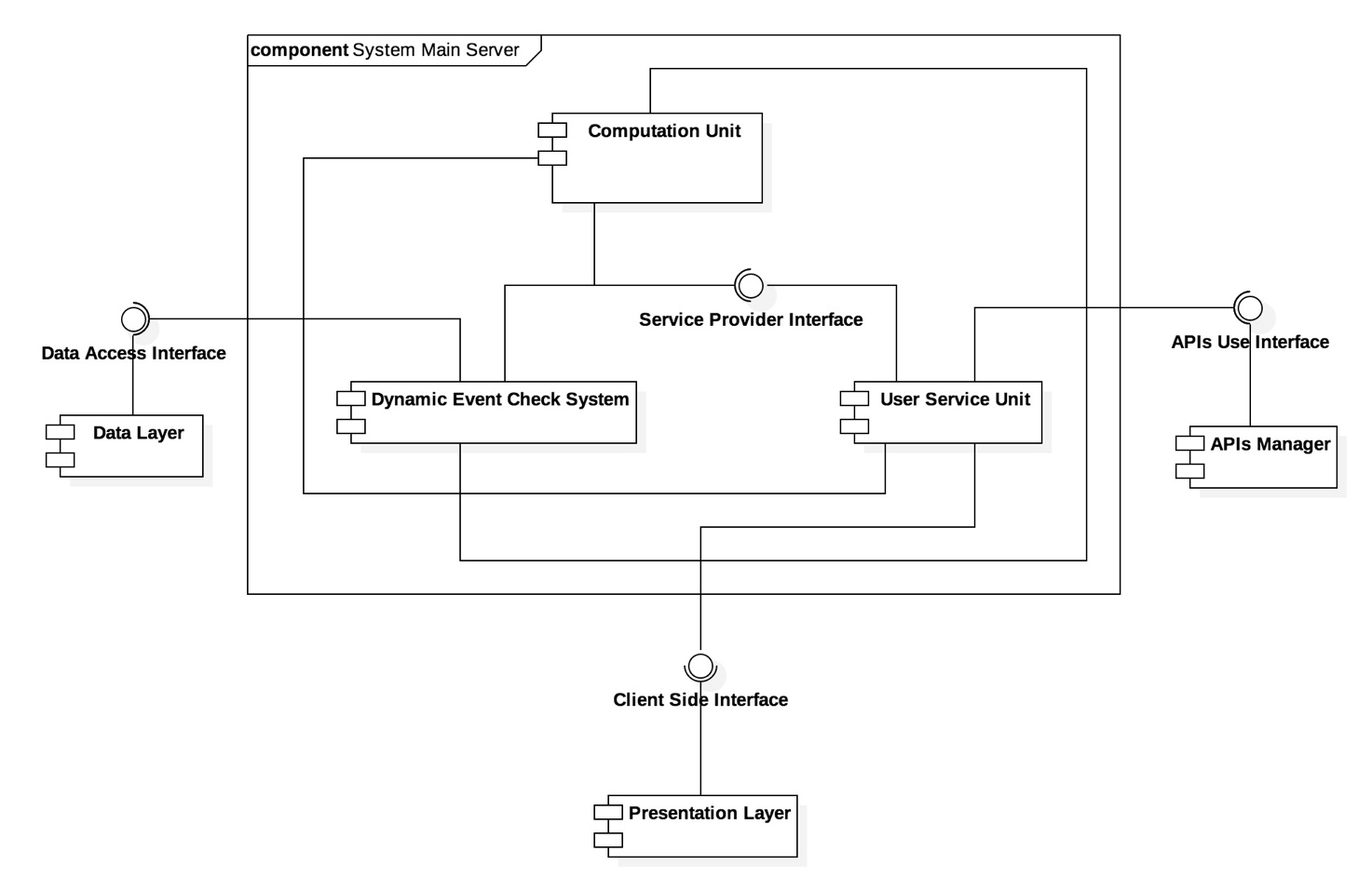
******

The Application DBMSs, as mentioned in the previous section, is a generalization of the two DBMSs used in the Travlendar+ system.

The Internal Use DBMS component models the DBMS that deals with the DB in which the system stores all the data needed to make the Travlendar+ application and server works properly.

The Users’ Data DBMS component models the DBMS used query the DB in which the system stores users’ credentials, calendars and sensitive data. All these data are encrypted.

***System Main Server***

******

The System Main Server is the core component of our server. It acts as “brain” for Travlendar+ since it must compute the best travel option when needed for each user (by means of the its Computation Unit), check dynamically (i.e. when events happen it’s listening) for events which can change travel options for some users and reports these to the local application(by means of the Dynamic Event Check System) and provide all these data (and some more) to the user in a proper way (by means of the User Service Unit).

Since this component has to be scalable and high-performing we could decide to distribute this component over several physical machines.

The communication between its component is allowed by the Service Provider Interface, an internal interface provided by the User Service Unit to transmit data outwards.

All the three internal components makes use of the Data Access Interface through which they can query DBs properly and of the APIs Use Interface through which they can exploits third part services.

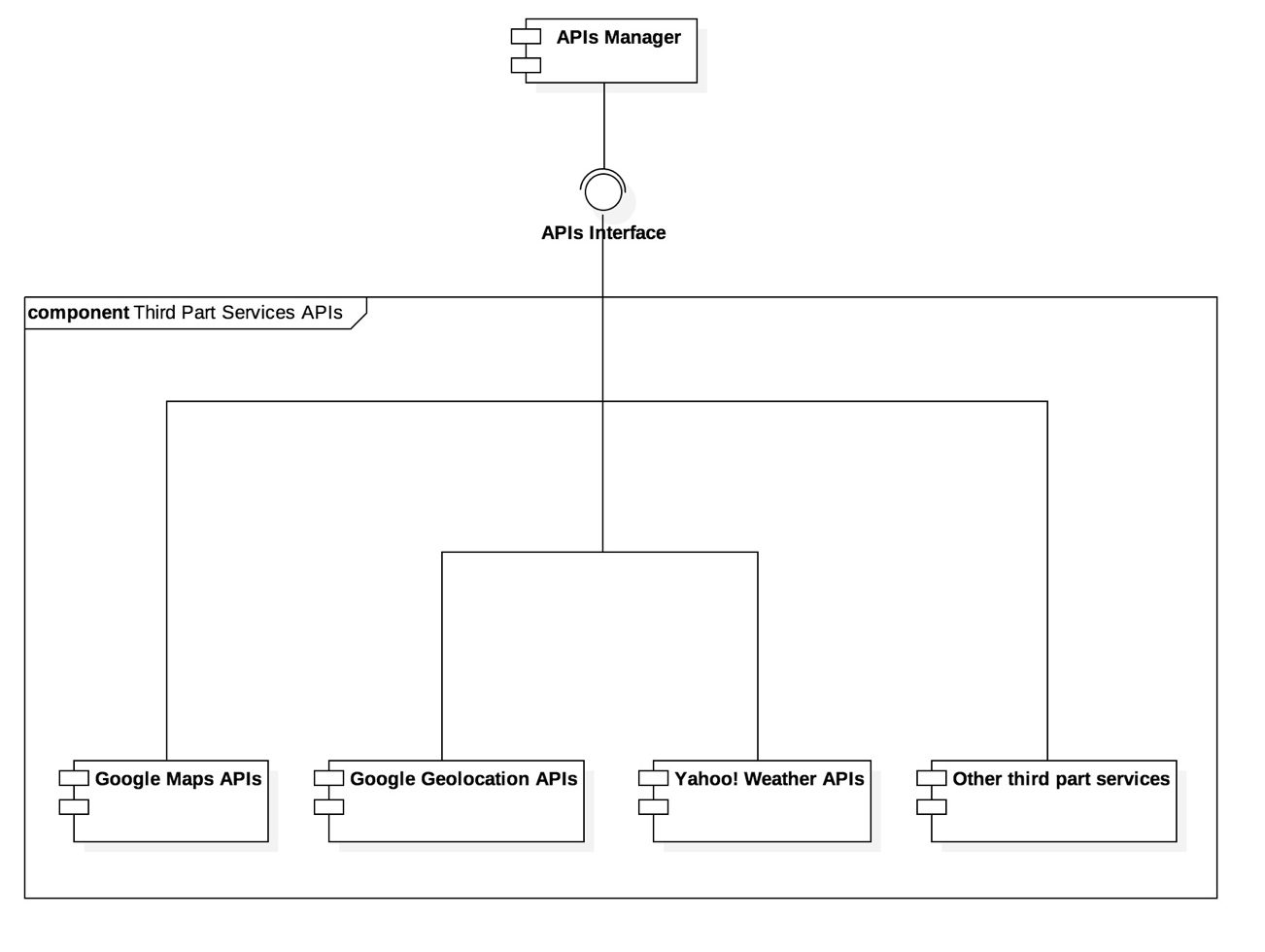
***APIs Manager***



The APIs Manager is internally composed by two components and an interface to let them communicate.

The Adapter component is used to make homogenous all different kinds of APIs exploited by the system by encapsulating them according to OO principles for sake of clarity and order. It also provides the Internal Interface to let other components make use of these APIs.

Moreover, the Optimizer tries to optimize the access to APIs and the computation in using them.

***Third Part Services APIs***

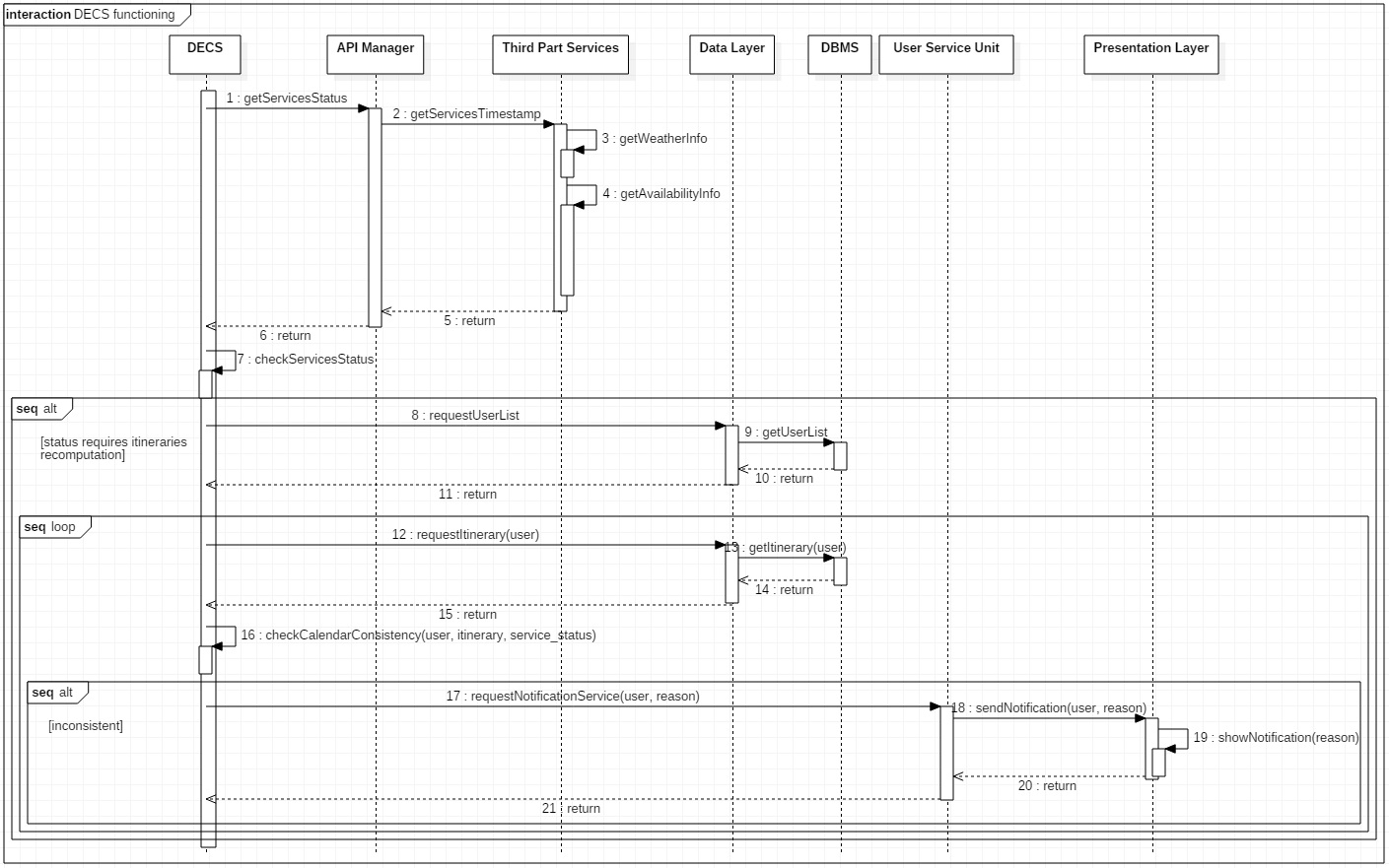
This component is a container of all commercial partners’ APIs. The system makes use of the OO principle of ‘encapsulation’ to efficiently and methodically have a reference to all these external tools. This way, this component of our system is easily extensible.

***2.4* RuntimeView**

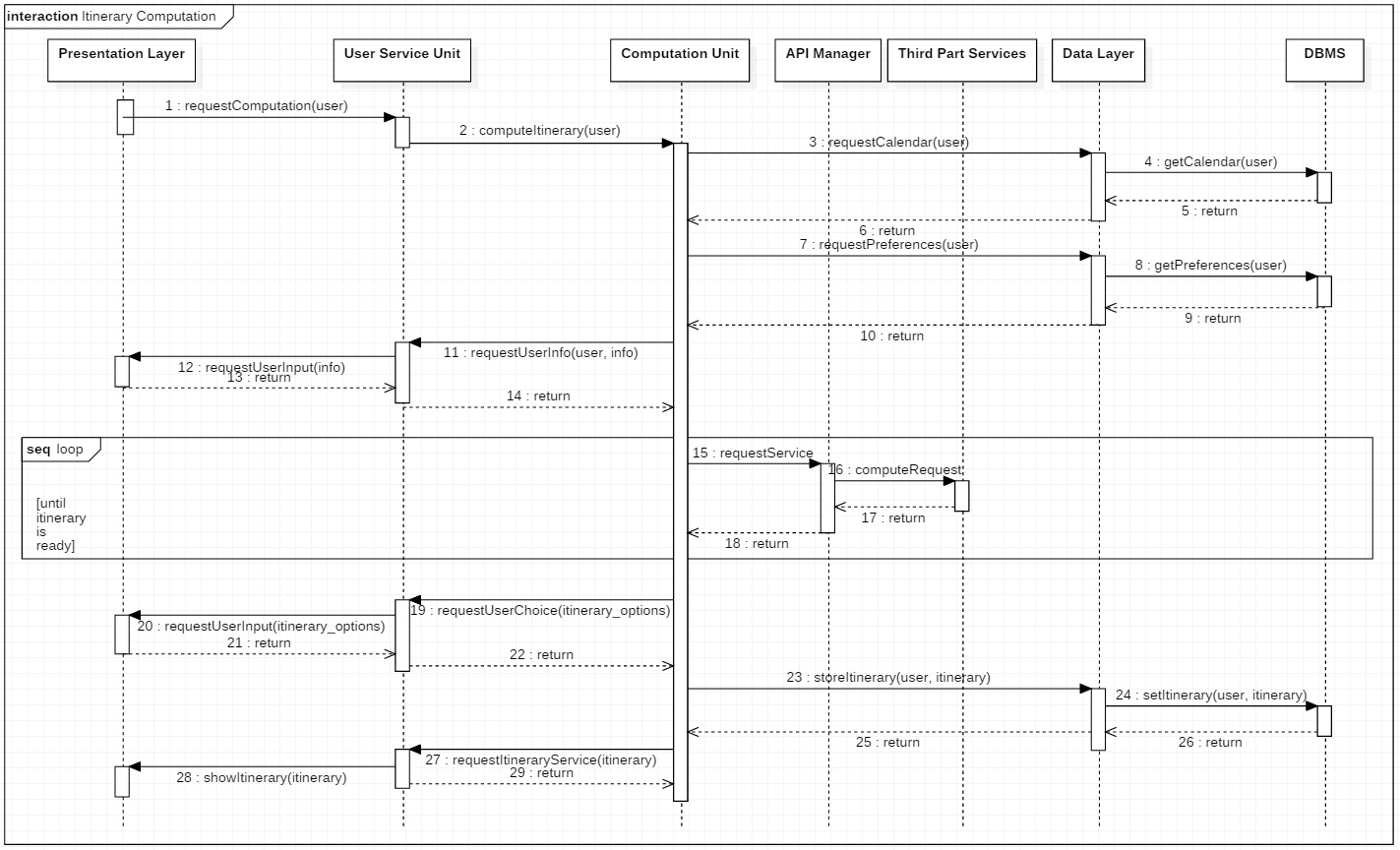
This chapter of the Architectural Design section is dedicated to showing how the previously identified components interact together to guarantee the correct functioning of the different features offered by our system.

Since most interactions are quite redundant, mainly requiring the cooperation of the Computation Unit and the Data Layer services, this chapter only shows the most complex interactions, where the correct functioning of almost every component is required to provide a given feature, while for the simplest just some cases are shown in detail, as the others can be easily derived from these ones.

***DECS functioning***

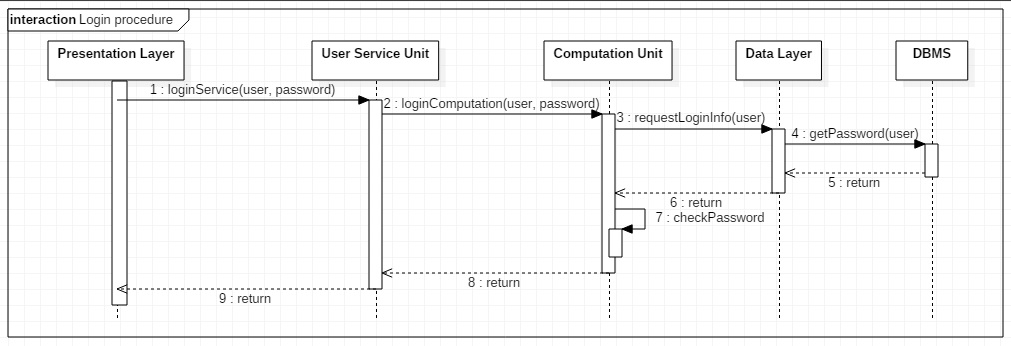
******

The DECS asks to the API Manager to retrieve the information regarding the services status. The API Manager gets a timestamp of the services, that is, the status of all services at a given point in time. In the case of weather forecast, it means getting the current weather condition, while in case of other services, it simply gets the information on the availability on those services (e.g. if the underground metro service is available, if there is a strike, etc.). This information is returned to the DECS, which analyses it to see if it reveals possible threats for the user itineraries (e.g. if it’s raining, users who were supposed to use the bicycle must be warned). If that’s the case, the DECS requests the Data Layer to hand a copy of the users list, which is fetched from the DBMS. At this point, for each user, the consistency of their itinerary is verified. If their itineraries require modifications due to a change in the weather condition or availability of a given service, then the DECS prepares a notification to the user and sends it through the Notification Service provided by the User Service Unit. At this point it’s up to the user to ask the system to compute a new itinerary.

***Itinerary computation***

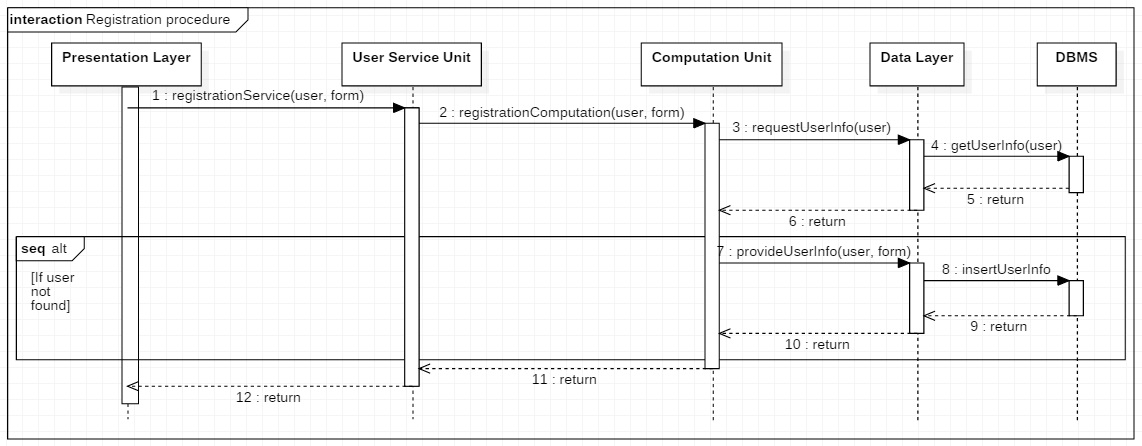
Starting from the Presentation Layer (either from the desktop application or the mobile application, as it’s irrelevant), a request of computation is forwarded to the User Service Unit, which notifies the Computation Unit of the request to compute a new itinerary coming from a certain user. At this point, the Computation Unit begins an information gathering phase: first, through the Data Layer, the Computation Unit asks to retrieve the calendar and preferences related to that user. Then, if necessary, the Computation Unit might ask the user for some details, through the User Service Unit. like specified in the algorithm section. At this point, the Computation Unit has all it needs, so a computation phase starts where, in loop, many different services are used to compute the different possible paths. When a list of potential itineraries is ready, it is immediately forwarded to the user through the User Service Unit. After making his choice, the itinerary is saved into the DBMS through the Data Layer and the selected itinerary is showed to the user.

***Login Procedure***



This procedure is one of the simplest in our system. Through the login service provided by the User Service Unit, a login request is sent to the Computation Unit. Upon receiving the password retrieved from the DBMS through the Data Layer, the Computation Unit checks the validity of the login and the result is sent to the Presentation Layer through the User Service Unit.

***Registration Procedure***



This procedure is quite similar to the login procedure, except it also requires storing data onto the database. In fact, after receiving the registration computation request forwarded by the User Service Unit, the Computation Unit looks for the username of the user who’s trying to register onto the database. If the username is not found, the user can be registered, otherwise the user is notified of the issue.

**7. Effort Spent**

This section will provide detailed information about the number of hours spent on this document.

**Matteo Biasielli**, matr. 893590

|  |  |
| --- | --- |
| **Section(s)** | **Number of hours** |
| 28-oct-17 Introduction | 1 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **TOTAL:** |  |

**Mattia Di Fatta,** matr. 893608

|  |  |
| --- | --- |
| 28-oct-17 | 1 |
| 29-oct-17 | 2.5 |
| 30-oct-17 | 2 |
| 31-oct-17 | 2 |
| 1-nov-17 | 2 |
| 4-nov-17 | 2 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **TOTAL:** | 11.5 |

**Emilio Capo,** matr. 899842

|  |  |
| --- | --- |
| Date + Section(s) | Number of hours |
| 4-nov-17 Runtime View | 1.5 |
| 5-nov-17 Runtime View | 1.5 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **TOTAL:** | 3 |