Detailed Design Description

Mälardalen University

Academy of Innovation, Design and Engineering

Project name: Graphical Project Portfolio Management Web Application

Project group: 3

Course: DVA313 – Software Engineering 2

Publication date: 2019-01-117

Table of Content

1. Introduction

The task of allocating a set of employees to a set of concurrent projects, each with its own budget and time constraints can often be tedious and highly complex. This complexity arises from several varying factors that depend on each other, such as; *budget for the project, end date for the project, salaries for the individuals' allocations to the project, employment rate for the individuals' allocations to the project* and so forth. The dependencies of the different factors (e.g. the salaries of the employees allocated to a project cannot exceed the budget of the project) needs to be considered and at the same time the resources should be utilized as effectively as possible. Furthermore, it is often the case that more specialized factors exist in the domain, increasing complexity even more.

Our client Daniel Sundmark, is responsible for the management of several concurrent projects at Mälardalen University, which also has the complexity issues described above. To accommodate this, the client has been involved in the development of a desktop application that automates all calculations. However, their existing application do not take the dependencies into account and it still requires that the data is manually entered into tables before their back-end server can do the calculations and return the results.

Their existing application consist of different tabs all containing tabular data that are either entered manually or calculated. The different tabs with their columns are as follows:

- **People**
  (name, salary, social factor, increment factor)
- **Projects**
  (name, end date, external salary, external overhead, external other costs, internal salary, internal overhead, internal other costs, overhead constant)
- **Allocation**
  (person, project, percentage, start date, end date)
- **Spending**
  (name, spending date, external salary, external overhead, external other costs, internal salary, internal overhead, internal other costs)
- **Remaining**
  (name, spending date, external salary, external overhead, external other costs, internal salary, internal overhead, internal other costs)

- **End Balance**
  (Project, external salary, external overhead, external other costs, internal salary, internal overhead, internal other costs)

In the list above, the data in the *people*, *projects*, *allocation* and *spending* tabs are entered manually, and the *remaining* and *end balance* tabs are calculated from the data in those tabs. The list depicts a set of people, a set of projects and the ability to allocate a person to a project, and by doing so, one is provided with calculated data describing the status of the budget for a specific project.

1.1 Project Proposition

In the domain of the client, there exists a lot of specialised factors (as the list reveals) which he needs to consider when managing the allocations - which still is a complex task even with their existing application in place. Therefore, the clients project proposition is to create a web application of their current system with enhanced functionality in the allocation tab, so allocations can be drawn instead of entered manually in a table.

In the allocation tab, the client wants to be able to handle all the allocations for one person at a time. The idea is that a person is first selected from a list of people that exists in the system and then a list of all projects that exist in the system will be displayed. Each entry in this list of projects must contain a timeline calendar so that the client easily could click on the timeline to allocate that person to that project. Furthermore, the allocations must be extendable horizontally and vertically, indicating the amount of time the person will be working on that project and at which employment rate. If a dependency issue manifests, a warning should be raised and prevent the allocation. An overall summation of all the allocations should also be displayed on a separate timeline so that the client easily could see the utilisation of that person.

2. Background

To make the website as efficient as possible we decided to make it as a “single page application” (SPA). In short, it has the same behaviour as a regular desktop application. Since we want the application to be up to date with the web application market today, we decided to work with a JavaScript framework named React. Not only is React very attractive on the market, but it also suitable for developing fast and easy maintainable SPAs. Since we are eight people on one project, developing could become very difficult when dividing the page into different sections. React makes this problem easy to overcome. In react it is possible to split the work into different components which easily could be put together later in the process. The framework makes it also easy to build demos with the components. These demos make it easier to show the client the current state of the application.

2.1 Single Page Application

A single page application (SPA), is a web application that behaves and looks as a desktop application. When the page has been initially loaded from the server, it never refreshes. Regular websites often implement different webpages for different content, meaning when a user browses the website and requests new content, the browser will ask the server for the new content and then refreshes itself to display it. In the SPA implementation, new content is fetched in the background through AJAX calls and then JavaScript is dynamically recreating the HTML-DOM structure to represent the new content.

2.2 Frameworks

To make the web application more interactive, we decided to include some frameworks rather than working with plain HTML, CSS and JavaScript. It took plenty of time to decide which frameworks to use since there is quite many to choose from. After looking in to which frameworks are most commonly used today, we concluded that ReactJS and Bootstrap would suit the project. Not only for its interactive parts but also keeping the application up to date with what is on the market today.

2.2.1 React

React is a JavaScript library for building user interfaces [1]. React enables programmers to think more in an object-oriented manner and to create encapsulated reusable components with its own state. The different components can be implemented in different JavaScript files, which usually is not an easy task in plain JavaScript without a lot of code that merges the files (with the risk of causing namespace pollution). React also connects the instantiated component with the HTML-DOM element the component creates and provides the ability to rerender the GUI automatically when the components state changes. React do not make any assumptions of the back-end to be used and is therefore, highly flexible in creating GUI for web applications.

2.2.2 MDBReact - Material Design for Bootstrap

Within React it is possible to use several add-ons for specific tasks. One of the add-ons used for the web application is the MDBReact library. It contains bootstrap, which is a front-end framework making it faster and easier to develop web applications. It contains pre-made elements such as buttons, tables and other forms which could be useful, especially for our project. For example, the navigation bar found in our web application uses this library.

3. System

The system itself is not the largest of systems. It consists of an interface which handles the interaction with the client and the user of the system. Furthermore, it is connected to a PHP server which does the communication between the database and the interface.

3.1 Design

To be able to connect the different parts of the desired system, plenty of different programming languages were taken into the project. Since we only received a database to begin with, our primary focus at the beginning of the project was to retrieve and send data. Through the scripting language PHP, we managed to retrieve and send data as we wanted to. Now that we have done our first task, our second task is to deliver a more user-friendly interface design. With the help of a convertible language such as JSON, and an interactive JavaScript framework as React, it will be possible develop a better, maintainable design of the system. Since the project might be taken to further development, it is of highest important that the design of the system is easy to understand and easy to maintain.

3.2 Main Parts of System

Since our project is a web application it will be required that it will run on a server. It is also required to have a good structured graphical user interface. The data will be stored in a database and will be fetched through PHP. To browse the application a client is needed – in our case Google Chrome. From this text it is easy to identify our main parts; server, graphical user interface, database and a client.

3.3 Communication within System

When it comes to communication within the system, the user of the system would access the application through a graphical user interface (GUI). The GUI itself must be accessed through a web browser. Due to clients request the system must be suitable for a Google Chrome browser. Furthermore, the application will be running on a PHP server containing the database filled with all the data that the user enters. The database contains stored data regarding persons, allocations, projects and whom the user is. The database will also do the calculations needed for filling the tables later in the process. To communicate between the database and the PHP files, we use Structured Query Language (SQL). With SQL it is possible to send queries to the database in order to fetch and update the data within it. In order to achieve good communication such as connecting the database server and other related database functions, PHP files will be necessary. This will function as the main communication between the database and the React component files. However, it is not possible to inject React files with pure PHP-code. Therefore, within the PHP files, the data received will be stored in JSON arrays. These arrays are very similar to regular JavaScript arrays, which makes it possible for the React components to access the data. React is the framework we use to build the interface and to present the data from the database. Through the client, the user will be able to manipulate the data by creating, updating and deleting entries in the database.

3.4 System Architectural Design

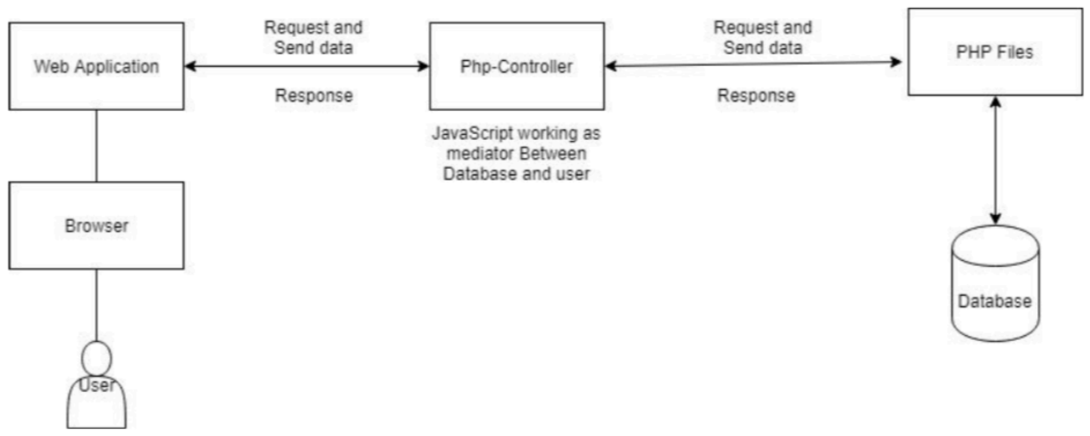

Figure 1: System Architecture Diagram

3.5 Communication with External Systems

When it comes to the communication with external system, by now the system does not communicate with any external systems. Regarding this, the system will presumably not communicate with external systems later on either.

3.6 System Manipulation

The system itself is a webpage which provides a specific service to the client. The website will present a certain data and numbers for different projects. The system which we got from the client is a graphical project which implemented with C# language. Our mission as a group is to implement a website of the same system with extra important additions to facilitate the way of modifying or adding for the user (client). The website will be used directly by the client to retrieve data about people and projects and send it to the server. The client did not want to implement a login page for the system. This is a function which could be added later if needed. At this moment the client can access the website by using any other browser however, Google Chrome is the browser of choice from the client. Chrome is a free browser program which can be installed on computer, tablet or mobile.

3.7 User Interaction

The idea behind the web application is that it is suitable for browsing in Google Chrome, therefore, it should be suitable for use on both computer and tablet. The interaction should be very simple and clear. On the top of the website there is a navigation-bar. The navigation-bar will contain six or seven tabs; People, Projects, Allocation, Spending, Remaining, End Balance and an add tab which contains a drop-down where it is possible to add project and staff.

The focus of the application is the 'Allocation'-tab where most of the interaction will take place. Here the user will interact with a calendar time-line with draggable elements. These elements are the allocation for both which percentage they will spent working on the project but also the duration they will work on the project. By right-clicking the element, the user should be able to split the element into two parts and be able to edit them separately. After changes, the user should be able to press a save-button and then send it to the database.

The other tabs will look slightly similar compared to each other, they will consist of tables filled with data for the specific tab. It should be easy to read the tables. However, they do not need much interaction since they should just be readable. In the people- and project tab the user will also see two buttons for add and delete right above the tables. By pressing these, the user will see a form where they can enter details for either a new person or a new project.

4. Implementation Structure

In this project, we have four main classes. They are the *User*, the *Person*, the *Allocation* and the *Project* class. The User class represents the user of this web application (only one user). This user manages the Person, Allocation and the Project classes. The functions that the user can perform on the person class are: getting person details, adding a person, deleting a person and get all person details. The functions that user can perform on Allocation class are; setting a new allocation, getting details about the current allocation, creating an allocation, deleting an allocation and get all allocation details. The functions that user performs on project class are: creating a project, deleting a project, adding a new project and getting de details of an individual project or all projects. The classes Person, Allocation and Project are also interconnected. Each person has an allocation and each allocation is made for a specific project. The structure and the interactions between the classes have been designed and can be seen in figure 2.

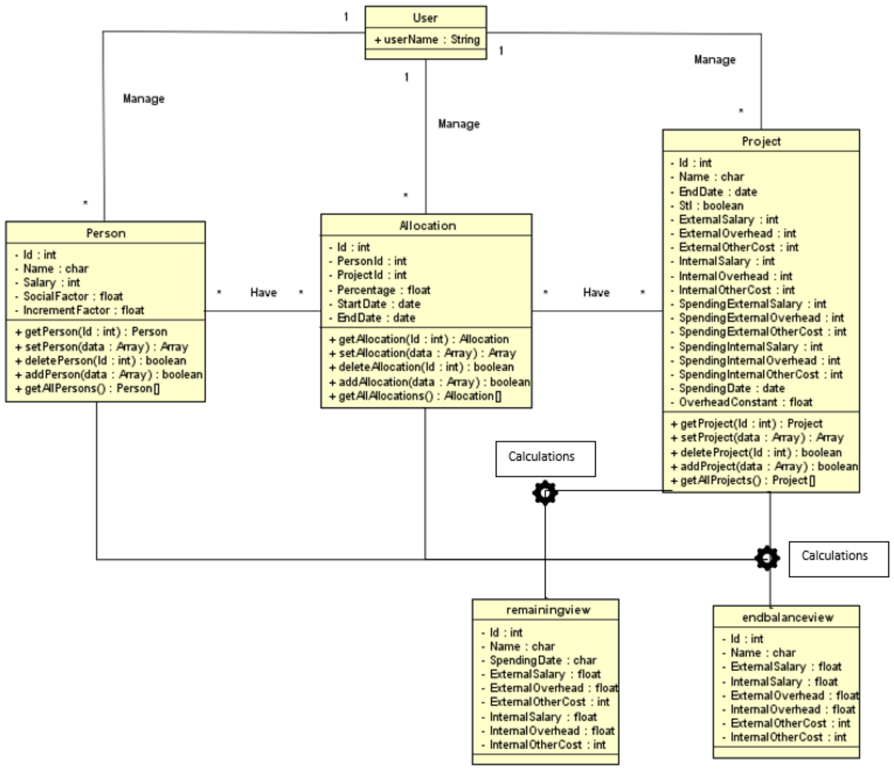
User
+ userName : String
1
1
1
Manage
Manage
Manage
*
*
*
Person
- Id : int
- Name : char
- Salary : int
- SocialFactor : float
- IncrementFactor : float
+ getPerson(Id : int) : Person
+ setPerson(data : Array) : Array
+ deletePerson(Id : int) : boolean
+ addPerson(data : Array) : boolean
+ getAllPersons() : Person[]
* Have *
Allocation
- Id : int
- PersonId : int
- ProjectId : int
- Percentage : float
- StartDate : date
- EndDate : date
+ getAllocation(Id : int) : Allocation
+ setAllocation(data : Array) : Array
+ deleteAllocation(Id : int) : boolean
+ addAllocation(data : Array) : boolean
+ getAllAllocations() : Allocation[]
* Have *
Project
- Id : int
- Name : char
- EndDate : date
- Stl : boolean
- ExternalSalary : int
- ExternalOverhead : int
- ExternalOtherCost : int
- InternalSalary : int
- InternalOverhead : int
- InternalOtherCost : int
- SpendingExternalSalary : int
- SpendingExternalOverhead : int
- SpendingExternalOtherCost : int
- SpendingInternalSalary : int
- SpendingInternalOverhead : int
- SpendingInternalOtherCost : int
- SpendingDate : date
- OverheadConstant : float
+ getProject(Id : int) : Project
+ setProject(data : Array) : Array
+ deleteProject(Id : int) : boolean
+ addProject(data : Array) : boolean
+ getAllProjects() : Project[]
Calculations
Calculations
remainingview
- Id : int
- Name : char
- SpendingDate : char
- ExternalSalary : float
- ExternalOverhead : float
- ExternalOtherCost : int
- InternalSalary : float
- InternalOverhead : float
- InternalOtherCost : int
endbalanceview
- Id : int
- Name : char
- ExternalSalary : float
- InternalSalary : float
- ExternalOverhead : float
- InternalOverhead : float
- ExternalOtherCost : int
- InternalOtherCost : int

Figure 2: Implementation classes in Database

4.1 Parts Collaboration

Once the user interacts with the system, data flow happens during the operations. On user side, data has been provided and managed by the user. The React framework is managing these operations, by preforming Ajax requests to the server. Then, these requests are processed in server and exchanged through the database. A main view of the data flow and control are described in figure 3.

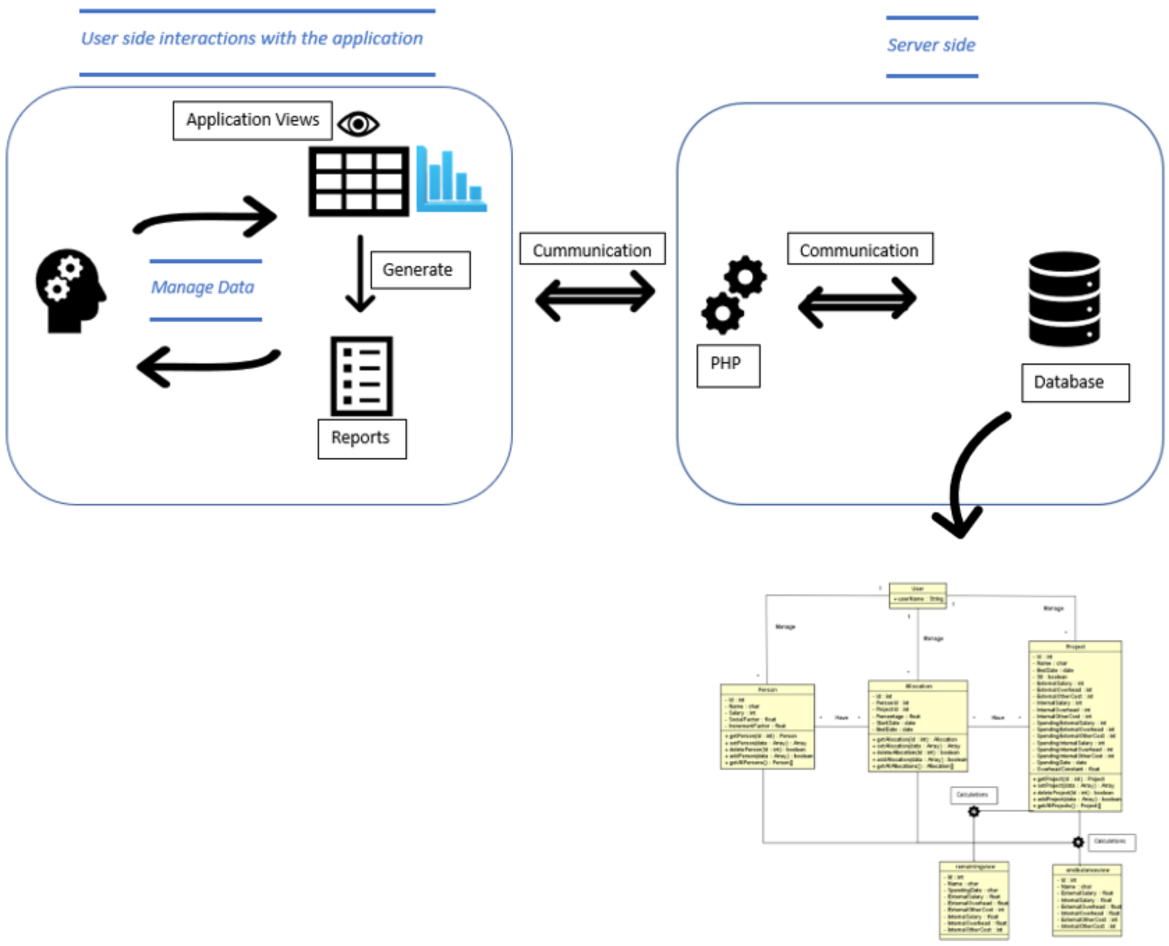

Figure 3- Data flow chart

Validation processes on data occurs in server and client side. The data will be checked from any issues before it has been sent to the server side. Thus, the data dependencies are logically stored and distributed. The collaboration of this parts in this way, provide the functionalities of system. The data goes through the system validated and in a right way.

5. Graphical user interface

The GUI is structured such as that the top will consist of a navigation-bar. In that specific navigation-bar the user will be able to swap between the different pages of the application. Here we find, as mentioned above; People, Projects, Allocation, Spending, Remaining, End Balance and possible also a add function. Below the navigation-bar, the focus of the specific page will take place. Having this said, the structure of the GUI will be very simple. As the image below shows, it only consists of the navigation-bar and the functionality of that specific page.

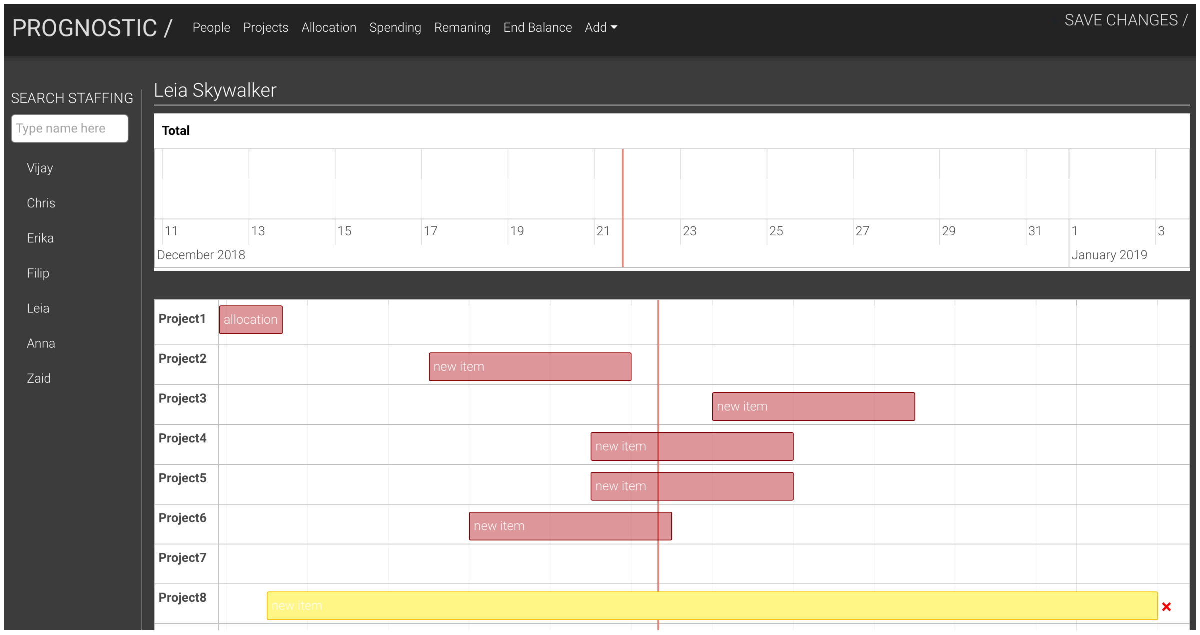

Figure 4 – Screenshot from the allocation-page

REFERENCES

[1] "React – A JavaScript library for building user interfaces", *Reactjs.org*, 2018. [Online]. Available: https://reactjs.org/. [Accessed: 06- Dec- 2018].