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: 2018-12-06

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' allocated to the project, employment rate for the individuals' allocated to the project* and so forth. The dependencies of the different factors (e.g. the salaries of the employees' allocated to a project can't exceed the budget of the project) need to be considered and at the same time the resources should be utilized as effectively as possible. Furthermore, its 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, that also have the complexity issues as described above. To accommodate this, Daniel have been involved in the development of a desktop application that automates all calculations. However, their existing application don't take the dependencies into account and it still requires that the data is manually entered into tables before their backend 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 is 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 Daniels domain there exist a lot of specialized factors (as the list reveals) he need to consider when managing the allocations, which still is a complex task even with their existing application in place. Therefore, Daniels 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, Daniel wants to be able to handle all the allocations for one person. The idea is that a person is first selected form a list of persons that exist in the system, and then a list of all projects that exist in the system is displayed. Each entry of this list of projects must contain a timeline calendar, so Daniel can easily 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 Daniel can easily see the utilization of that person.

2. Background

In this section, we will elaborate on a few topics that are needed to fully understand the document.

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 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 stat. The different components can be implemented in different JavaScript files, which usually isn’t 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 don't make any assumptions of the backend to be used and is therefore highly flexible in creating GUI for web applications.

2.3 Material design for bootstrap

[insert text of the concept of material design and a brief intro the bootstrap]

3. System

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

3.1 Design

We have used a lot of different programming languages to be able to connect all parts of the desired system. Since we have received only a database to begin with, our main focus at the beginning was to retrieve and send data. Through the scripting language php, we managed to retrieve and send data as planned. Now that we have done our first task, our second one was to deliver a more user-friendly interface design. With the help of a convertible language such as Json, and an interactive language react.js, we will be able to develop a better design of the system.

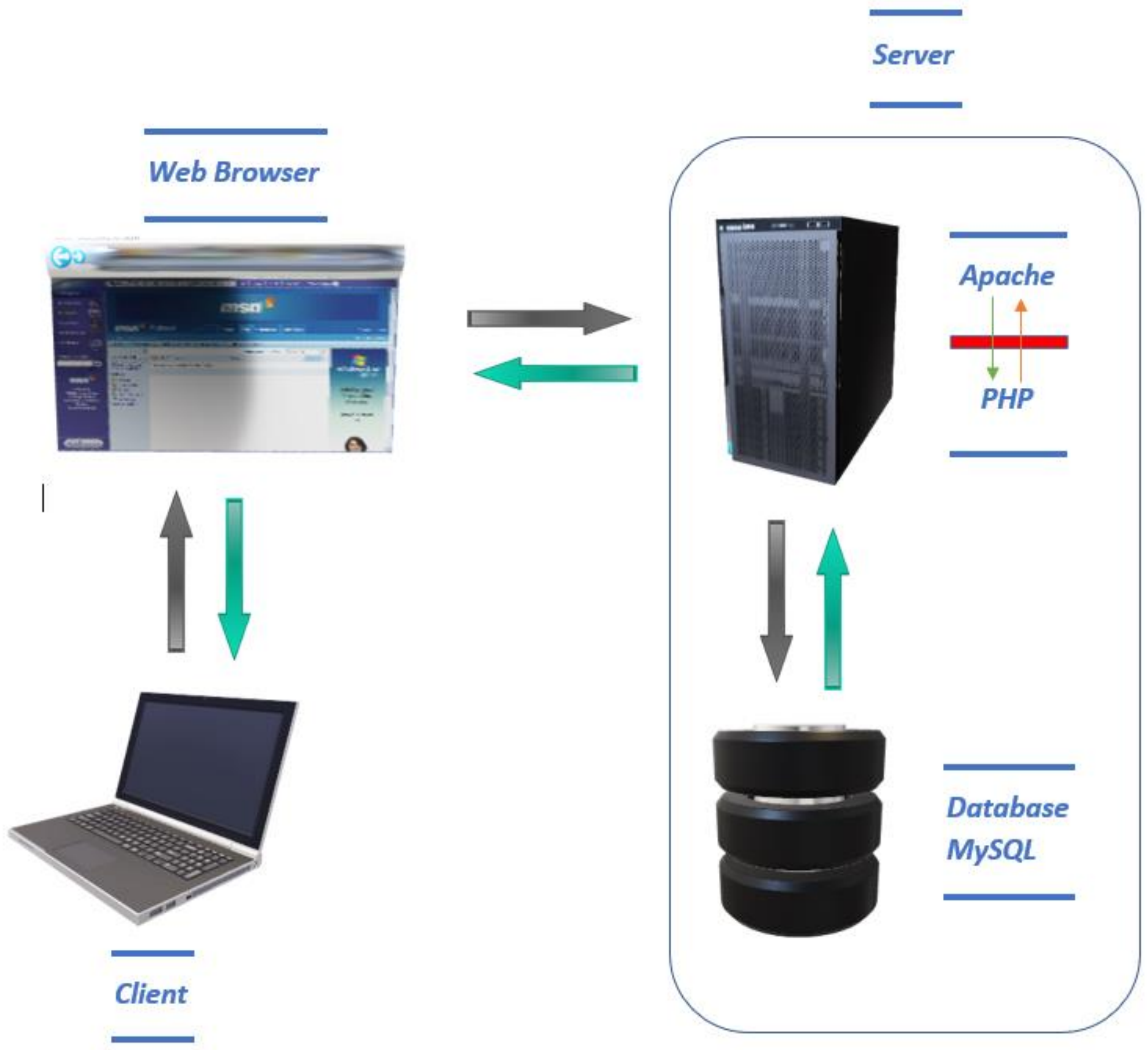
3.2 Main parts

Our system will be web-based application and it will run on a web server. Main parts of the system are:

- graphical user interface,
- database,
- PHP Server and
- a client.

In the following text we describe all the parts of the system and how they communicate. The client, the user of our application, would access the application through a web browser. The application would be running on a PHP Server and it would include the database. In the database we want to store data about persons, allocations, projects and the user. SQL is the language that we use to get a connection between the database and PHP files. With SQL we want to send orders ("queries") to the database in order to fetch and update some data. In order to achieve communication such as connecting to the database server, querying the database and the other database related function between application and database we will write PHP code. One of the PHP files function would be to retrieve data from the database and save it in a JSON array as a list of values. JSON arrays are very similar to the arrays in JavaScript. We would present data stored in JSON array to the user using React. React is JavaScript library which would help us to build the user interface. The user would have the possibility to manipulate with data by creating, updating and deleting entries in the database. Additionally, if we decide to add another user to the application in later phase of development, user would not have possibility to make changes to database.

Following image shows diagram how the system work and its containing parts:



Picture 1: System Architecture Diagram

3.3 Communication with external systems

The client can interact directly with the system by the website we are implementing. The website and the required database can be hosted locally or on a server, depending on how the client wants to work with the system. Apart from that, the system does not transmit information to any other system in the external environment.

3.4 System manipulation:

The system 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, but we thought to fix it if we have enough time. At this moment the client can use the website without login by using Google Chrome. Chrome is the browser that client wanted to browse the website through it. Chrome is a free browser program which can be installed on computer, tablet or mobile.

3.5 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. However, we need to ask client if tablet is needed. 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 (if wanted by client) an add tab which contains a drop-down where it is possible to add project and staff.

The main 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.

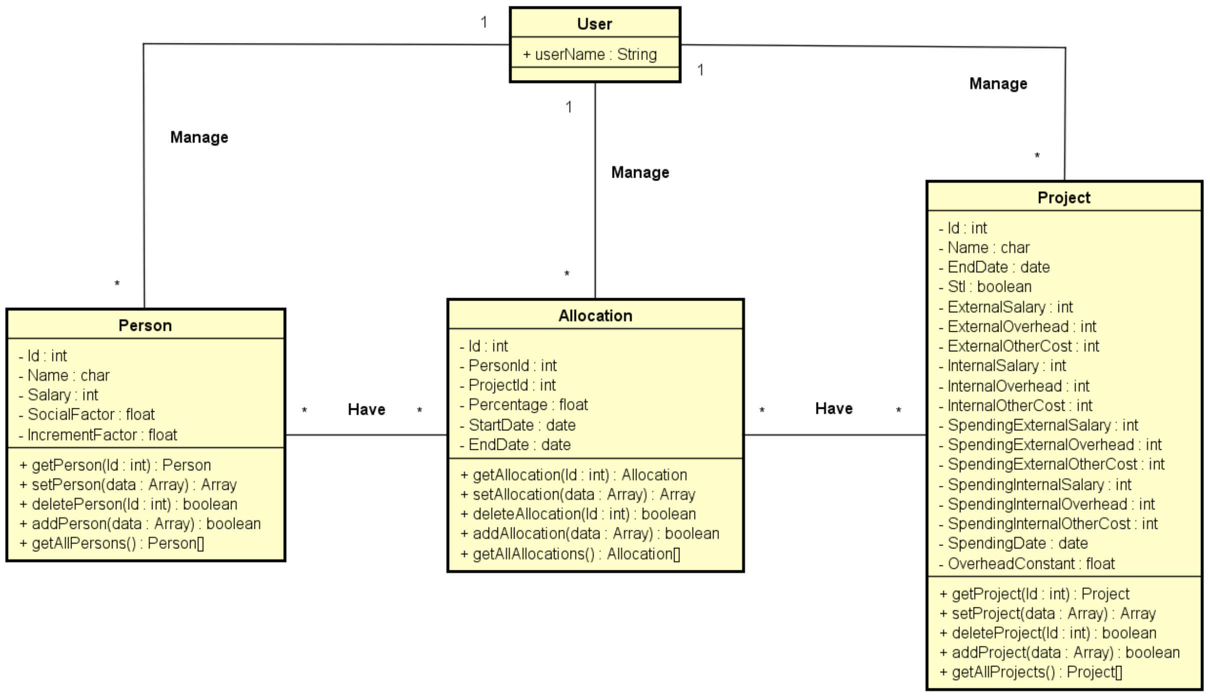
The other tabs will look slightly similar to each other, they will consist of tables filled with data for the specific tab. It should be easy to read the tables. However, they do not need much interaction since they should just be readable. If the client wants an add-tab they should be able to choose whether to add staffing or to add a new project. Depending on which, the application will show either a form to fill out information about staff or project.

4. 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 actually 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.

4.1 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 the class diagram below.


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[]

4.2 Parts Collaboration

Once the user interacts with the system, data flow happens during the operations.

On user side, data 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 the figure bellow.

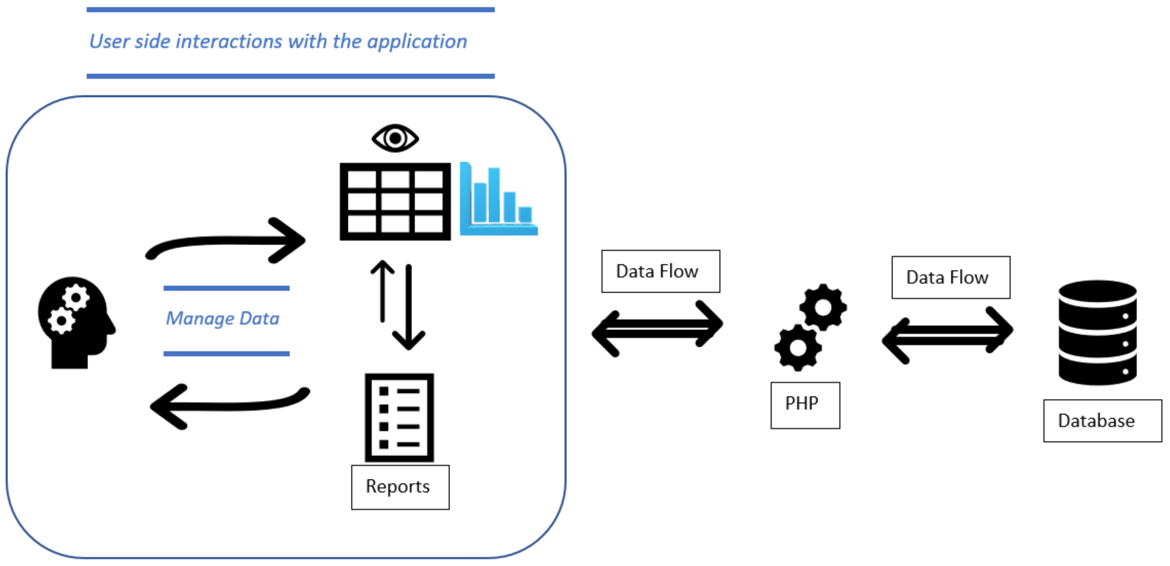


Figure - Data flow

Validation processes on data occurs in server and client side. So, the data is been checked from any issues before it's 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 main 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.

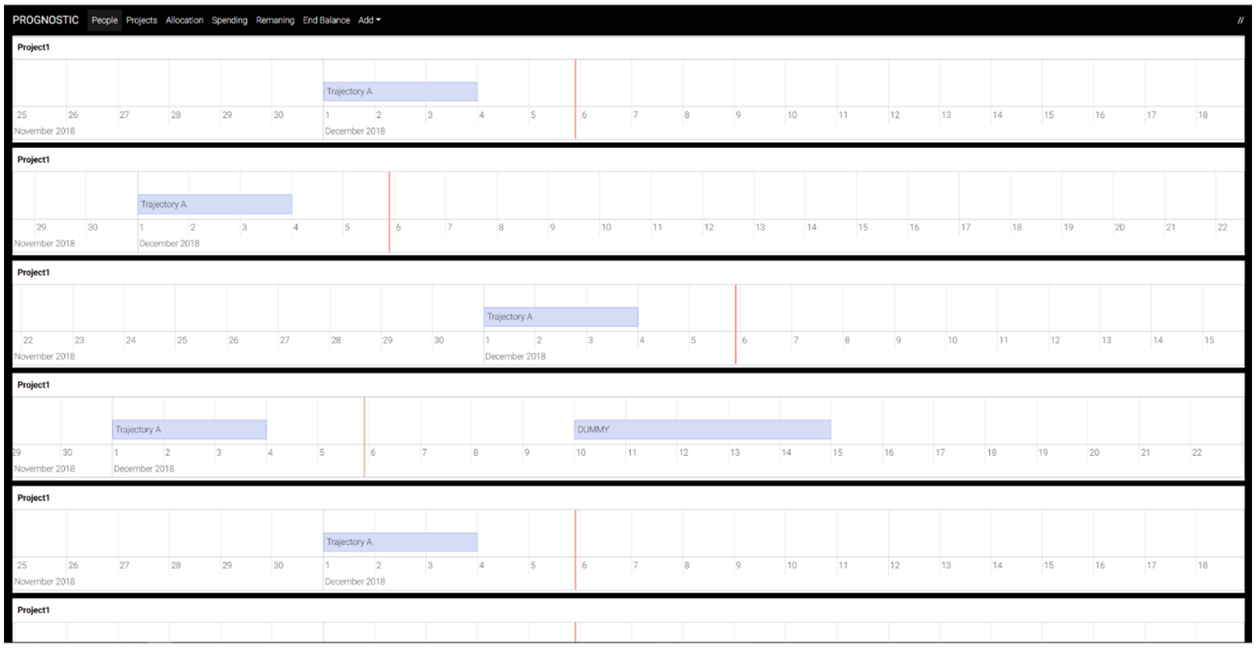


Image X – Screenshot from the current allocation-page

REFERENCES

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