Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Ákos Norbert Sepp

DATA-DRIVEN PROJECT MANAGEMENT WEB APPLICATION

Business Information Systems

ACADEMIC SUPERVISOR

Dr. Csaba Simon

INDUSTRIAL SUPERVISOR

Dr. Márton Szabó

(Mazesoft Technologies)

BUDAPEST, 2020

MSc THESIS

Ákos Norbert Sepp
Business Information Systems Student

Data-driven project management web application

Modern project management strongly relies on data, but the data sources are quite heterogeneous in nature. Project-related data may cover bills, budget plans and reports, tasks and tracking data placed in separated tables, usually listed by different viewpoints. Therefore the evaluation of the current status or analyzing the risks of a given project requires the careful correlation of these sources, generating hidden costs in the project management process. Worse still, since this data handling process is a time consuming one, the project manager lacks updated and timely information at the decision making point. Although an Enterprise Resource Planning (ERP) system is a convenient tool to address this problem, most companies cannot afford the cost and time to implement and customize one. Also, typically the complexity level of smaller projects does not recommend the use of ERP, at all.

The aim of this Thesis is to design and implement a data-driven project management application and administrative support system that only focuses on the project itself and the logically connected tasks, enabling the project manager to evaluate and analyze the current status of the project quickly but thoroughly without the need of an ERP system.

The task described should include the followings:

- Review the literature of project management tasks, evaluate the software solutions available on the market and analyse their functionalities.
- Evaluate the suitable software technologies to implement a project management framework (e.g., Python, Django Framework, jQuery, HTML5, JavaScript, CSS3), select the ones that correspond to the requirements of the Thesis and motivate his selection.
- Plan the business logic and the competitive advantage behind a project managemnet tool that addresses the issues described above.
- Plan and design the main parts of the user interface (front-end) and the data processing module (backend) according to the received database system, provided by the company.
- Implement the following parts of the project management framework: the user interface for desktop and mobile platform (front-end), and the data processing software (backend).
- Test the implemented software with popular web software testing methods.
- Discuss the results and document the work.

Academic supervisor: Dr. Csaba Simon (BME)
Industrial supervisor: Márton Szabó (Mazesoft Technologies)

Budapest, 10. March 2020.

Dr. Gábor MAGYAR
Head of Department

Dept. of Telecommunications and Media Informatics
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics

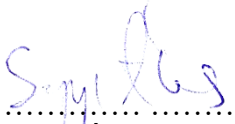
H-1117 Budapest, Magyar tudósok krt. 2.
Phone: (+36 1) 463 2448 Fax: (+36 1) 463 3107
E-mail: magyar@tmit.bme.hu http://www.tmit.bme.hu

HALLGATÓI NYILATKOZAT

Alulírott **Sepp Ákos Norbert**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2020. 11. 30.



…… ……

Sepp Ákos Norbert

Table of contents

Abstract

Modern project management strongly relies on data, but data sources are quite heterogeneous in nature. Project-related data may cover bills, budget plans and reports, tasks and tracking data placed in separated tables, usually listed by different viewpoints. Therefore the evaluation of the current status or analyzing the risks of a given project requires the careful correlation of these sources, generating hidden costs in the project management process. Worse still, since this data handling process is a time consuming one, the project manager lacks updated and timely information at the decision making point. Although an Enterprise Resource Planning (ERP) system is a convenient tool to address this problem, most companies cannot afford the cost and time to implement and customize one. Also, typically the complexity level of smaller projects does not recommend the use of ERP, at all.

The aim of this Thesis is to design and implement a data-driven project management application and administrative support system that only focuses on the project itself and the logically connected tasks, enabling the project manager to evaluate and analyze the current status of the project quickly but thoroughly without the need of an ERP system.

Acknowledgement

I would like to thank the following people, without whom I would not have been able to complete this Thesis.

First of all, I would like to express my gratitude to my academic supervisor, Dr. Csaba Simon, who undertook my topic and helped me not only with insights and deadlines, but encouraged me through difficult times. Secondly, my industrial supervisor, Dr. Márton Szabó, who shared his enthusiasm and his visionary idea of Maze Project with me, and helped me get through road blocks both in development and in task management. Special thanks to Gergely Tálos, who created a unique atmosphere for working and learning during our time together as developers.

And my biggest thanks go to my family for all the support they have shown me throughout these years. For my wife Júlia, to be my partner in crime, for my parents, whom I can always rely on, and for my brother and sister, who are one of the greatest inspirations for me in life.

1 Introduction

Modern day project managers must face the fact that it is almost impossible to stay on track with a project. Nowadays, working in the era of data revolution, it became a number one priority for managers to analyze and evaluate projects' situation and support their decision-making with objective measurement numbers on a daily basis. Financial tables, production or resource plans are just a few of the status reports that are essential for a project to succeed.

However, these reports come together from countless parallel processes, which usually take place in different environments from multiple data sources, undoubtedly creating tons of documents and spreadsheets, being edited and re-edited by co-workers all the time. Moreover, other important details are lost between meetings or post-it notes, never getting a chance to be properly recorded. Organizing this whole documentation mess and filtering out only the relevant data would consume too much time. If a manager tried to get a picture of their project's current situation, at the end of the day, the results were already outdated. Nevertheless, a project manager must get the necessary status reports and risk analysis in one way or another, ending up in a constant chase after real-time data.

The problem is not newborn - companies faced similar problems several times in the past and always came up with a solution. Today, the most common solution for the problem described would be an ERP (Enterprise Resource Planning) system. ERP is about to integrate the whole company into one business network, where all application can work together, and share data in a common database. Although ERP systems connect all functional areas of the company together, implementing ERPs are rather expensive. Moreover, it takes several months only to adjust and customize them to the company's specific needs alone, while getting used to the new environment is another big step that must be taken. On top of all that, most companies' projects are usually not complex enough to make it a beneficial investment.

Due to the aforementioned barriers, in recent years another approach started to unfold as a possible solution, the so-called data-driven project management or 'dynamic scheduling'. Vanhoucke [1] defines it as a 'project management methodology to plan, monitor and control projects in progress. The methodology focuses on the integration of

three crucial aspects: baseline scheduling, schedule risk analysis and project control'. With these three aspects, the method aims to integrate essential data-handling in the core of the management process. That being said, the project manager can harvest the important information during the management process, without the necessity of putting all data source in one place, for instance, with the implementation of an ERP.

After two years of business planning, Mazesoft Technologies started Maze Project in 2018 in order to create a modern, web service business solution that can enhance data-driven project management. As decision-support software, it was specially designed for project managers who want to deliver projects on time and within budget by implementing dynamic scheduling in their work. Although in recent years successful competitors appeared on the market, Mazesoft Technologies believes that their business model can lead them to success. As a software developer I was given the chance to participate in the project and became a part of the Maze Project-journey. In the following pages I would like to present the work I have been doing in the process of business planning and software development.

Section 2 is divided into three main parts. Firstly, it summarizes the main theoretical background for project management. Secondly, it aims to get a deeper understanding on project management tasks, related to the functionalities of the application. Thirdly, the section looks into the market's current situation, focusing on the three, differently selected applications and their main attributes.

Section 3 reviews the core ideas and business logic behind the product, presents the most important visions and frames that build up the software. After the business plan, section 4 translates business language into informational technology terms. The first part presents the possible methods and technological solutions for the development. The second part chooses between technologies and selects the final development tools.

Finishing theoretical work, section 5 walks through the system plan provided by the company. Then it overviews the three most important technological parts of the software: the application, the business and the data layers.

Starting the actual development work, section 6 is about front-end development. It presents how the HTML skeleton and the basic user interface was created, the steps how most important client-side functionalities were built in, and how UI development was finished with additional stylesheets.

Section 7 presents the data processing module. It shows the implementation of most important data transportation methods and functions, and finally, the finishing of major functionalities for server-side actions, such as signing or logging in, or saving to the database.

After the most important development steps, section 8 introduces some of the most popular website testing applications for both the front-end and the backend, and presents the testing results. As a second step, it also shows the evaluation process on the received performance results.

Section 9 discusses future possibilities of the application, both in business and development level. Last, but not least, section 10 summarizes the Thesis.

2 Project Management

2.1 Project, management

2.1.1 The project

First of all, to understand the concept of project management systems, we have to start from the very beginning, by defining the term ‘project’ itself. The PMBOK Guide [2], one of the most fundamental books in the field of project management, describes that ‘a project is a temporary endeavor undertaken to create a unique product, service, or result. The temporary nature of projects indicates that a project has a definite beginning and end. The end is reached when the project’s objectives have been achieved or when the project is terminated because its objectives will not or cannot be met, or when the need for the project no longer exists’.

To examine that definition from a software development point of view, we could follow Kerzner’s understanding of the project [3], as he highlighted several important attributes. Among others, he emphasizes, that projects

- Have a specific objective, with a focus on the creation of business value, to be completed within certain specifications
- Have specific start and end dates
- Have funding limits
- Consume human and non-human resources.

Later, in section 2.5, we will see how these four attributes become the cornerstones for project management systems.

2.1.2 Managing projects

Project management, defined by the Project Management Institution (PMI) in the PMBOK Guide is ‘the application of knowledge, skills, tools and techniques to project activities to meet the project requirements.’ The PMBOK Guide also provides a list of actions that are typical project management processes, such as identifying requirements addressing needs and expectations of stakeholders, creating project deliverables, or most

importantly for our scope: balancing the competing project constraints like scope, quality, schedule, budget, resources, and risk.

The real challenge lies in the relationship among these project constraints, meaning a change in only one factor can affect several others as well. For example, if the budget is decreased, either the quality of the product will suffer, the resources must be re-arranged or the schedule must be adjusted. Obviously, a change in quality, resources or schedule affect other factors, instantly creating a chain of reactions. Due to that characteristic, project management is an iterative activity that needs continuous improvement and detailing.

2.2 Project Management Tasks

2.2.1 **Process Groups**

In this section, the term Process Group will be presented. The following subsections overview the five group of project management tasks that the PMBOK Guide defined, explaining how tasks are related to each other and in which part of the project are they usually performed. The Guide's official Process Groups are: Initiating, Planning, Executing, Monitoring and Grouping, and Closing. Figure 1 represents the interaction among Process Groups. Before the individual groups would be presented, it is essential to look at the whole chain of actions and observe the relations between management processes (darker dotted lines).

Project management processes are connected to each other (with dotted lines), creating a flow of interactions among Process Groups and some specific stakeholder (lighter dotted lines represent the external participants). The link between processes are inputs and outputs where the result of one process becomes the initial input for another process. However, the figure cannot be set in parallel with project life cycle (PLC) phases. Moreover, it is likely that each Process Group could be executed only in one PLC phase.

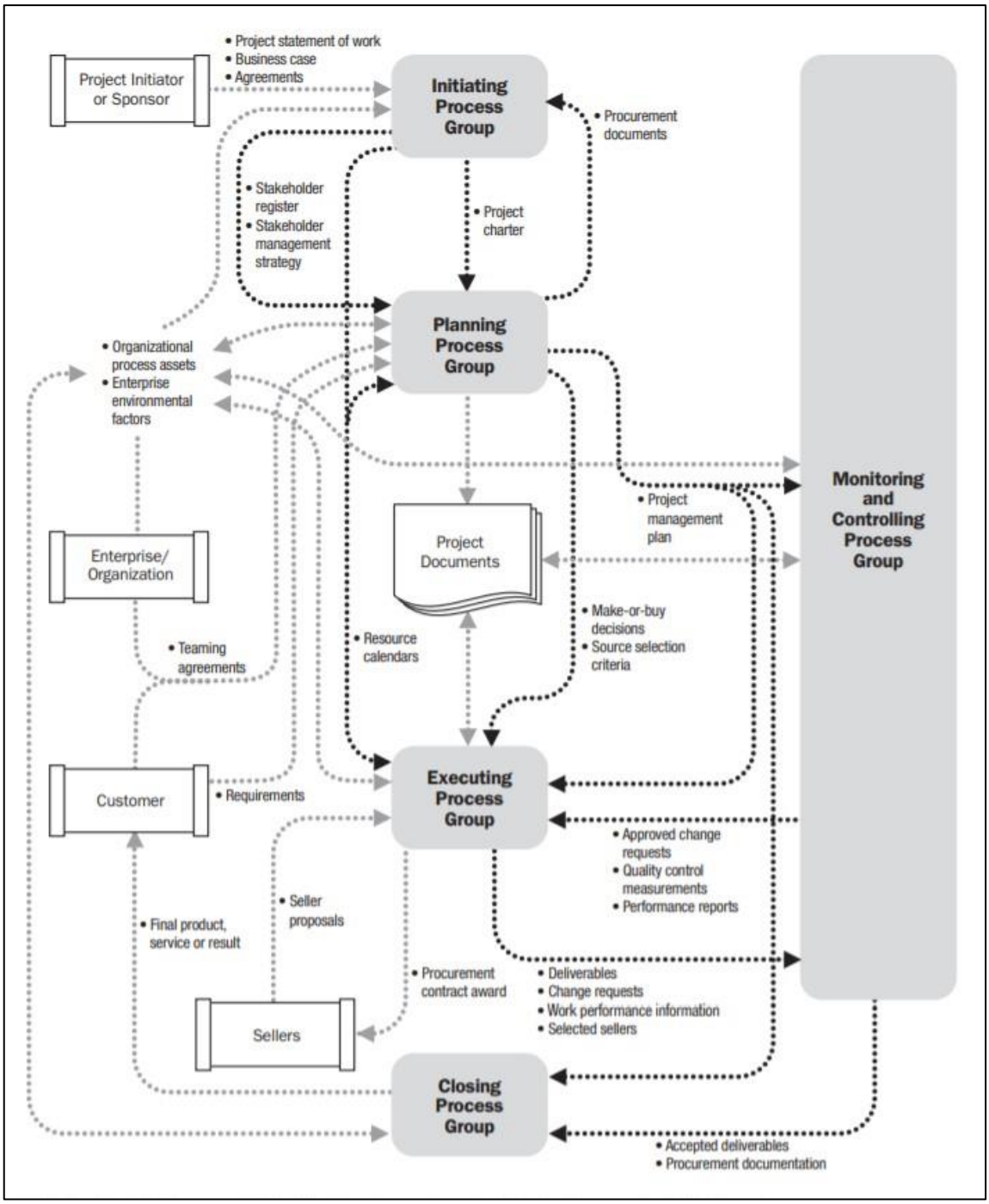


Figure 1 – Project Management Process interactions (Source: The PMBOK Guide, p53)

2.2.2 **Initiating Process Group**

Those processes belong here, which are performed in order to create a new project or a new phase of an existing project. Figure 2 – Project boundaries set by the Initiating Process Group (Source: The PMBOK Guide, p. 54) shows how this Process Group affects the whole project or phase. Typical processes are:

- Stakeholder identification

- Project Boundary definition
- Project Deliverables definition
- Project Vision evolution
- Project Charter development

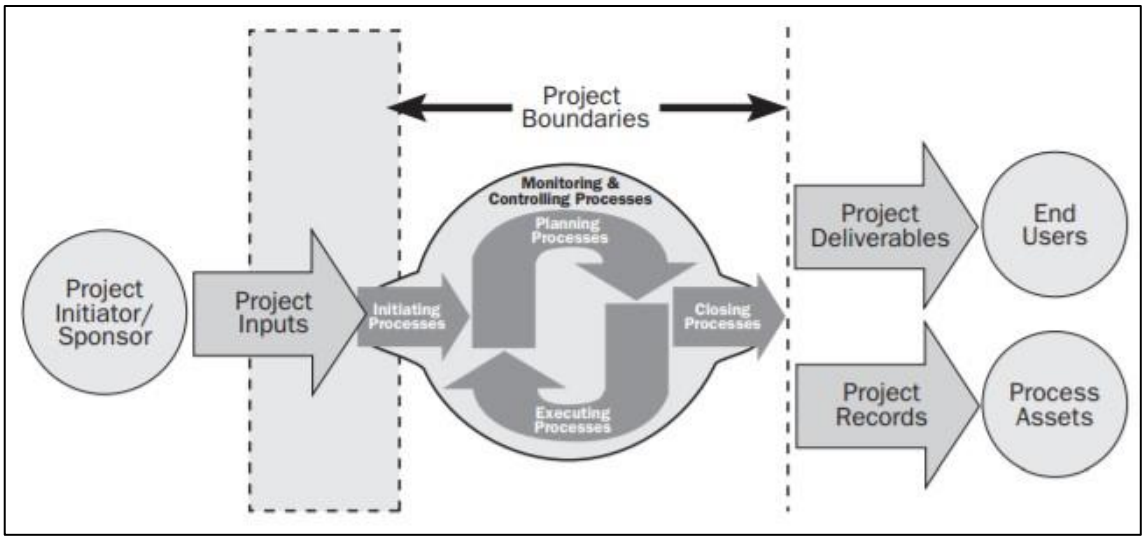


Figure 2 – Project boundaries set by the Initiating Process Group (Source: The PMBOK Guide, p. 54)

2.2.3 **Planning Process Group**

Activities in this Process Group aim to establish the scope of the project with defined and refined project objectives and the way they will be measured. They also create a logical chain of actions in order to meet the objectives. It is likely, that processes of this Group may be repeated several times along the project duration, as the complexity of these processes and the sensitivity for significant changes can lead to great differences between plans and actual results. Typical project management tasks of this Process Group are:

- Project Objectives definition
- Project Management Plan development, included but not limited to:
  - Scope, Schedule and Cost baseline, Communication technique definition
  - Scope, Requirements, Quality, Process, Human resource, Communication, Risk, Procurement and Change management planning
  - Project life cycle and Project management methodology selection

2.2.4 **Executing Process Group**

The Executing Process Group contains those processes that are performed in order to complete project objectives. Most of the actions are connected to the coordination of human and non-human resources, the communication among stakeholders and the handling project management plan changing events. Typical project management tasks of the Process Group are:

- Managing risks and implementing risk response actions
- Managing stakeholders, sellers and suppliers
- Establishing communication channels inside and outside the project team

2.2.5 **Monitoring and Controlling Process Group**

Monitoring and Controlling Process Group summarizes processes required to track, overview and measure progress and performance status of the project, or which processes focus on identifying changes and initiating corresponding actions. Typical project management tasks of this Process Group are:

- Controlling changes, taking adaptive actions
- Performing Risk Analysis
- Generating and measuring performance data
- Coordinating project phases
- Collecting and documenting experiences related to possible improvements

2.2.6 **Closing Process Group**

The last Process Group includes all processes that are performed to end all activities in every other Process Group, leading to finish all projects, phases or contracts formally. Performing this Process Group verifies that every single process in the whole project will be closed or terminated. Typical project management tasks of this Process Group are:

- Formally closing the project based on official acceptance contracts
- Performing post-project or phase-end reviews
- Documenting gathered knowledge on the project

- Closing all contracts
- Archiving related project documents
- Evaluating project team performance and releasing resources

2.3 Agile project management

Project management methods can vary based on the nature of the project. For well-defined works like mass production in a factory, there are clear and efficient methods that proved their value during the decades. However, there are projects with great variability, often in rapidly changing environments, where traditional methods cannot succeed effectively. For instance, software development, where technological progress is the most important driving force.

In 2001, software industry leaders formed the agile movement with the publication of the Agile Manifesto [4]. In this document, they declared the following four main principles how project management should approach software development projects:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan.

They also state in the Manifesto, that items on the right are also important, however items on the left carry more value.

So far in the 21st century, agile project management has not only became the most popular way to work in the information technology industry, but the application also spread to several other industries. According to the Agile Practice Guide [5] published by the PMI, the reason behind the wide adoption is that ‘exponential innovation technologies’ along with growing user expectations are demanding more and more immediate delivery of additional values. Agile techniques and approaches can handle this kind of pressure effectively, enabling most companies to stay competitive and preserve their share of the market.

There are two reasons, why agile project management is important in this Thesis. Firstly, Maze Project application aims to incorporate the agile approach in its core.

Therefore both business model and system planning must pay attention on all four main principles. Secondly, Mazesoft Technologies' software development follows the agile approach, and it influenced how the final application was developed.

2.4 Data-driven project management

Vanhoucke [1], who created the term data-driven project management, clarifies that in academic literature, data-driven project management is known as 'dynamic scheduling' or 'integrated project management and control'. He also emphasizes that in the field of project management there has been a focus on scheduling since the middle of the 20th century, when methods like CPM (Critical Path method) or PERT (Program Evaluation and Review Technique) were created. But data-driven project management is not about focusing strictly on project scheduling, but rather integrating crucial management control aspects into one methodology.

Vanhoucke describes the following three dimensions:

- **Baseline Scheduling**: Project activities must be connected to time and budget restrictions, and the project timetable is built upon them. Start and finish times of each project activity are determined within the activity network and resource constraints. Therefore all activities has an expected impact on the project's time and budget objectives.
- **Schedule Risk Analysis:** The baseline schedule must be analyzed from a risk point of view, in order to estimate activities' impact on the project's time and budget with a possible probability. Techniques like Monte Carlo simulation can be used to forecast time and budget deviations.
- **Project Control:** Measuring and analyzing project's performance in real-time must be a frequent action. With monitoring deviations from expected project progress, it is possible to control performance with corrective actions to keep the project on track. Traditional methods like EVM (Earned Value Management) or ES (Earned Schedule) can be used.

For the first look it could seem like agile management and data-driven project management are two totally different ways to handle projects. However, Vanhoucke pointed out in an interview [6] that both management methodologies are built on the same

foundation: adaptively controlling project processes with frequent status checks and the ability of re-planning.

Therefore he is convinced that the data-driven approach can be used along with agile techniques as well. Furthermore PMI in the Agile Practical Guide clearly stated that agile methods can be used in various environments according to the project team's needs. All in all, the only obstacle there could be, is the way how we try to synchronize them in Maze Project software.

2.5 Project Management Software (PMS)

2.5.1 Defining Project Management Software

The project management software term is used for applications that support project management activities. Depending on the size of the software, for instance, it can help to plan, organize and allocate resources, schedule and assign tasks, follow and modify logical dependencies among processes, support quality and time-management, enhance collaboration, communication or administration.

Any company that needs planning, estimating or tracking their service can benefit from investing in a PMS. In the market of PMS software there are countless solutions in a wide range of functionality. Market-leading software comparative website Capterra [9] lists more than 650 different project management software products used only in Northern America. Based on that number, the market is totally oversupplied with software solutions.

2.5.2 Comparing existing solutions on the market

To review the competitive market of project management software, three applications have been selected based on a subject viewpoint: the most popular worldwide, the most familiar for the author and one with quite similar functionalities like Maze Project.

The selection of the most popular application was based on the previously presented website Capterra. Every year, Capterra compiles a Top-20 list of project management software, by a two-dimensional scoring system.

The perspectives for grading are the following:

1) **User reviews from the last 12 months** and

2) **Web-search interest by selected keywords**.

According to their research [10], in 2020 the most popular project management software was Asana [11], with the fourth highest user review point (45) and with the highest web search interest point (50), winning the title with the overall score of 95 points out of 100.

Choosing the most familiar application was a simpler task, based on the author's 2-year experience as user and administrator in the software Jira [12] developed by the company Atlassian. Jira was ranked 5th on the Capterra Top-20 list, with the overall score of 89 points out of 100.

Finally, for choosing the most similar project management system, Capterra's built-in search engine was used. With the possibilities to sort by functionalities, it was possible to narrow down the options for 15 different applications (Figure 3 – Functionality filter on Capterra (Source: https://www.capterra.com)). The selected functionalities were based on Maze Project's business plan, presented in section 3.

The search criteria were the following:

- Agile methodologies
- Budget managements
- Cost-to-completion tracking
- Gantt Charts
- Kanban Board
- Milestone tracking
- Resoure Management
- Time & Expense tracking
- Web-based deployment

From the 15 results, after browsing the applications, Easy Project [13] was selected for comparison.

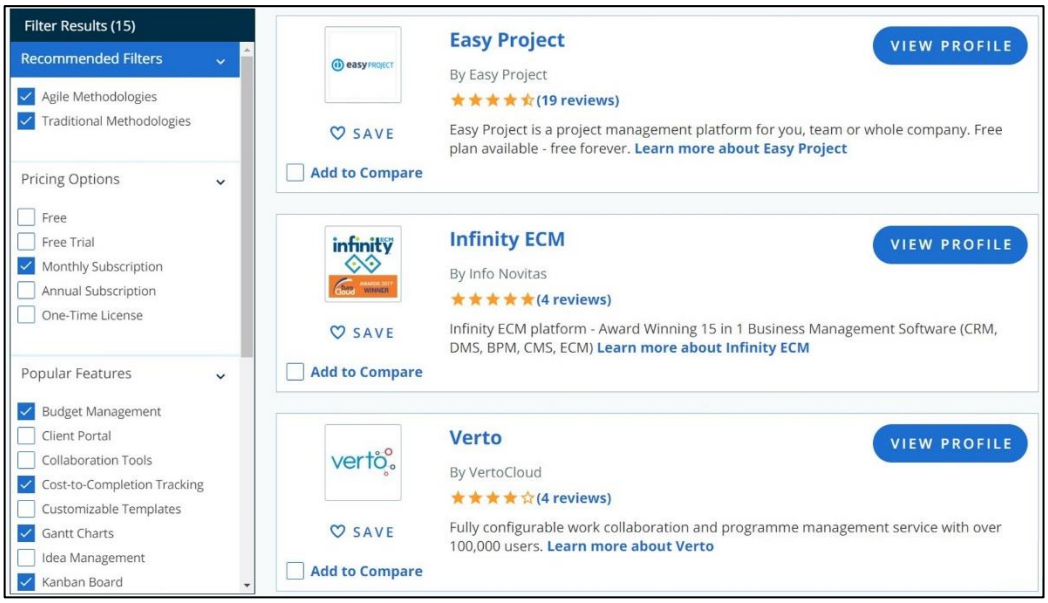


Figure 3 – Functionality filter on Capterra (Source: https://www.capterra.com)

2.5.3 **Comparing applications**

There are endless possibilities to compare software products. However, this comparison focuses strictly on three attributes: product functionalities and features, supported platforms and pricing, as mostly these three aspects define the user base.

Table 1 contains the comparison dimensions (functionalities and features, platforms and pricing), and the findings for the software (ordered in alphabetical order). All related data was gathered from Capterra. Bold features shows comparative advantages and features that not all software have, items marked with italics show optional features (not in the basic software or need add-ons).

Comparison dimension	Asana	Easy Project	Jira
Functionalities and features	- Agile methodologies - **Budget Management** - Collaboration - Gantt/Timeline View - Percent-Complete Tracking - Recurring Task Management	- Agile methodologies - **Budget management** - Collaboration - Gantt/Timeline View - Percent-Complete Tracking - Recurring Task Management	- Agile methodologies - *Budget management* - Collaboration - *Gantt/Timeline View* - Percent-Complete Tracking - Recurring Task Management

	- **Resource Management** - Task Board View - Time Tracking - **To-Do List View**	- **Resource management** - Task Board View - Time Tracking	- *Resource Management* - Task Board View - Time Tracking - *To-Do List View*
Supported platforms	- Cloud-based / Web - **Windows** - **Mac** - **Android** - **iOS**	- Cloud-based / Web only	- Cloud-based / Web - **Windows** - **Mac** - **Android** - **iOS**
Pricing	$10.99/month/user **Free basic version available up to 15 users**	$6.5/month/user Demo version available for 30 days only	$10.00/month **Free basic version available up to 10 users**

Table 1 – Comparison of Asana, Easy Project and Jira software (Source: https://www.capterra.com)

The comparison table shows that there are a number of differences among the three software products. In functionalities and features, Asana has three, Easy Project has two main functionalities that Jira can only provide with add-ons. One feature (To-Do List View) is only provided by Asana by default.

In terms of supported platforms, all three products support a cloud-based web platform, while only Asana and Jira support the main operating systems on desktop and mobile. Comparing pricing strategies, all software solutions provide some kind of free version of the software. All free versions have limited functionality, but the restriction approaches are different. Asana and Jira use user number restriction (15 and 10 user maximum), while Easy Project uses expiration restriction with 30 days.

Summarizing the comparison process, it is clear that all competing software have the main tools built-in for supporting basic project management tasks we reviewed in section 2.2. As all three software were mainly built on web services, it can be considered as the most popular and common approach for modern project management applications. The other common feature is the availability of free versions with some kind of restrictions. For future business planning it is another point that must be taken into consideration.

3 Presenting Maze Project

3.1 Inspiration behind the application

After several years of working as an IT project manager, Mazesoft Technologies co-founder Márton Szabó decided to design a software that enables him to see the project in its real form, as it is. He had his experience with project management software like Jira, he tried to collaborate with co-workers with task management applications like Trello, but every time he had to face the fact: in order to see the ongoing performance and status, he had to go the extra mile to actually integrate project team's data into project progression status, risk analysis or budget reports.

He decided that the most important part is not to collect all documents into databases, like how an ERP would work, but to somehow incorporate data-based thinking and approach into the software itself. The idea formed to its final shape after he read the book *The Data-Driven Project Manager*, written by Mario Vanhoucke, project manager and university professor, who created the term data-driven project management.

The following pages presents the business logic behind Maze Project, the data-driven project management application.

3.2 Business Logic

3.2.1 Vision and confines

Maze Project is a cloud-based SaaS business solution. A project management and administration system, built on separated database structure, supported by business intelligence. The software enhances the planning of project activities and budget with integrated data and automations, multidimensional permission handling and dynamic change management tools. With simple and clean user interfaces, Maze Project makes it easy to follow project progression and cost control in real-time.

It is important to emphasize, Maze Project's main focus is on the project itself. An ERP system connects all departments in the company to work together, as it also handles incomes and revenue. Whereas Maze Project starts with the given budget, as one of the biggest restrictions in projects' life. Daily task and issue management are other

managerial processes that the software do not want to undertake, yet it assigns tasks to administrational and project actions.

The software's target group are project and enterprise managers, CEOs and COOs, financial and project team members, or smaller businesses with their own projects. For education purposes business schools, higher education institutes or post-graduate and other project management training programs.

3.2.2 **Main goals and functionalities**

It is always a challenging task to form visions and thoughts into objectives. For Maze Project, the above presented ideas turned into the following goals and functional requirements:

- Integrate an approach that logically connects atomic project actions with resource-based cost management
- Enable to work with unified project planning and monitoring architecture, based on the software's own database structure
- Terminate parallel data and cost registration in unrelated spreadsheets
- Make it easier to work with project administration data, thereby reducing the unnecessary consumption of valuable time
- Create a systematic risk analysis and progression monitoring approach, based on real-time data
- Prepare automated reports for data-supported decision making
- Create data visualizations and graphical reports about project progression
- Make it easy to implement software into the company's workflow
- Develop an easily customizable platform

Section 3.3 presents the essential modules that build up the application. In every module description, the previously set objectives and functionalities are highlighted, so that it can be seen, how these goals are translated into software elements.

3.2.3 **Competitive advantages**

After researching the market and comparing existing popular solutions, three main competitive advantages can be highlighted. Firstly, the fact that the focus is on the project

itself, not on task or issue management. This point of view indicates a different budget management approach than what the examined popular applications use. This method enables project leaders to easily assess budget limitations and keep within cost in the long run.

Secondly, the ability of seeing the bigger picture for users. As all project and administration actions are strictly connected to budget rows, there is no such thing as lost cost. Not only for the planning management tasks but also for the monitoring and controlling phase it helps to clarify where the project is actually heading.

Thirdly, the continuous and automated reporting. As reports are always necessary, putting a status report is a really time consuming managerial function. With a data-driven project management approach, reports are already served on the fly. What is more, parallel project plan version handling facilitates the global management of resources all along the project.

3.3 Core modules

The Maze Project software has 4 core modules that ensure the most important functionalities. For future development, it plans to provide additional modules, some of which are already in discuss phase, some are only ideas or still waiting for customer review and feedback. In section 3.4 two examples of such potential future modules are presented. The 4 core modules are: Project attributes, Gantt, Budget and Resource management, and Report - Dashboard.

3.3.1 **Project attributes module**

This module contains all basic information about the ongoing project, like start and end date, available budget, project description, or project team. This module is also responsible for the handling of these data. Project team members can be assigned to roles and can be grouped together to work on tasks together. The project calendar also belongs to this module, which displays and handles events from major milestones to meetings and conference calls.

3.3.2 **Gantt module**

Gantt module is basically the great engine of the software, as it connects all other modules together. It handles the project schedule based on WBS (Work Breakdown

Structure), creating a roadmap for the project in greater steps. All project tasks are represented by a single Gantt chart row, marked with a WBS number, which also functions as an ID for that task. As every administrative or cost related data modification is connected to a project task, all other modules have functions that work with this module.

Although Gantt and WBS are often associated with the waterfall development method (and therefore often called untrendy and stick-in-the-mud), modern management methods like Kanban board for task or issue handling, can also work together with these techniques easily. The key is to precisely separate where one project task ends in magnitude. For instance, a rule of thumb can be to break down the project into tasks that has at least a week-long duration. From that point, tasks and issues can be connected to a specific project task, and the agile development approach can be used. Eventually, for the greater project milestones the software uses the Gantt chart, while for daily issue handling, agile methods are implemented.

Being said so, another popular task handling method, a To-Do List is also present to support the Gantt module, for creative thinking and taking notes.

3.3.3 **Budget and resource management module**

The Budget and resource management module contains all functions that are responsible for project resource controlling. Interactive data tables let managers overview free, assigned or used resources. In this module, four submodules can be found: Expenses, Resources, Budget and Acquisition.

Expenses shows which project tasks have been funded and how these are divided among processes. Resources collects and manages the non-monetary instruments. Budget and Acquisition both focus on money consumption, showing exactly how and which processes use their allocated funds. But while Budget works with inner processes, Acquisition collects expenses that at least partially connected to elements outside of the project team. As it was highlighted before, all the assignments above in the seven submodules are connected to Gantt rows. Based on that Maze Project can create a logical data-flow between actions and resources.

3.3.4 **Report – Dashboard module**

The Report – Dashboard module could be considered as two modules, both with the goal to support real-time decision making.

The Report submodule – as the name suggests – is responsible for generating major project management reports. Technically all other modules' status can be reported here, from basic project attributes or Gantt to change or risk management reports. Results are available and downloadable in several different file formats, such as csv, excel or pdf.

The Dashboard submodule enables the project manager to gather valuable information by a glimpse, as both pre-defined and custom KPIs are displayed by this module automatically. Customizing the user interface is possible with easily adding and removing, or creating new performance indicators.

3.4 Additional module ideas

Although several module ideas are in discussion phase or waiting for customer feedbacks, in this section two of the most interesting modules are described on a conceptual level.

3.4.1 **Change and risk management module**

Change and risk management module is all about actual and alternative project plans. Supported with data visualizations, this module aims to integrate project-related decision making in the software, enabling the user to freely edit the selected management plan and schedule or overview possible effects with informative tables and graphs. The user also finds tools for risk analysis and handling, with visually detailed scenarios.

Accepted changes and project activities are stored in this module as well, for the whole project and for smaller phases together. Version numbers are used for version control. Project processes can be marked with status messages, like 'under editing' or 'waiting for acceptance', to make the user interface more intuitive.

Risk management is represented in the module via analytical and predictive methods like Critical Path Method (CPM). Processes marked to be part of the critical path are highlighted for the manager in order to notify them in advance.

3.4.2 **Administration and permissions module**

Administration and permissions module is the odd-one-out among modules, as it is not directly connected to the project, but supports the software management. Being responsible for the whole project's database, the project manager can rely on this module to overview and easily handle the multidimensional permission system that is typical in a project team. This module is also the key for handling multiple projects in the software. It makes it possible to have more than one project or project team, while having the same or totally different permission system.

4 Technological background

4.1 Technological requirements for the project

Web applications have three main parts: the frontend, the backend and the database. For each part, different development tools are used, based on the criteria the project wants to meet.

During the business planning phase, Mazesoft Technologies had to decide, in what environment the application should be developed, and exactly what technological goals should the developer team aim for. As part of the team, for this project the author was given the following list of criteria.

For the Front-end, the following requirements had been set:

- *Build the software on the latest version of the Front-end triangle (HTML5, CSS3, JS)*
- *Use Django's form-based communication between Front-end and Backend*
- *Use a unique library for the user interface that enhances visualization, preferably Plotly.JS*
- *For wireframes, use the latest version of a popular free Front-end element library, preferably Bootstrap 4.0 or higher*
- *Do not base the application on any JavaScript frameworks (like React, Vue or Angular)*
- *Use of jQuery is preferred, but only 3.3 or higher*

For the Backend, the requirement are listed below:

- *According to the latest trends, build the Backend on Python, instead of PHP*
- *Use Django for creating the environment, as it can easily work with additional data-processing software*
- *For platform independence, the application should be run in a container, preferably Docker*
- *Publishing should be done via a repository manager that can solve version control and continuous deployment to the server, preferably GitLab*

For the database, the only requirement was to build it on an open-source, but quick relational database system.

In the following pages, the chosen technologies are presented.

4.2 Front-end technologies

4.2.1 The Front-end triangle (HTML, CSS, JavaScript)

Front-end development is basically built up by the combination of three programming languages: HTML, CSS and JavaScript (Figure 4 – The Front-end triangle (Source: https://www.toughlex.com/technologies/front-end). This vastly evolving group provides a multi-platform opportunity to develop applications for the client side.

Figure 4 – The Front-end triangle (Source: https://www.toughlex.com/technologies/front-end)

HTML (Hyper-Text Markup Language), currently running with its fifth version, gives the structure for a website. Its main function is to create the space and form for every content.

CSS (Cascading Style Sheets), with its third version, is a style describing language. The main purpose of CSS it to create a smoother and more readable, visually enjoyable user experience out of the HTML. CSS3 also handles the majority of website animations and the appropriate cross-platform appearance.

JavaScript, as the third member of the triangle, is responsible for the behavioral part of the webpage. It often contains some of the business logic, as it collects and provides data for the client. It also processes and handles most of the user interactions.

4.2.2 **AJAX**

AJAX stands for Asynchronous JavaScript and XML, and typically it connects the client side front-end with the server-side backend.

AJAX helps to create a more interactive website due to the fact that the site does not need a full reload to show some information. Thanks to the asynchronous property, it optimizes the communication with the server based on the bandwidth and the downloaded data's size (e.g. load on demand). Aside from optimization, it enables the developer to separate the content, the functionality and the appearance on a page, boosting the user experience factor.

4.2.3 **Additional libraries**

For enhancing user experience and making the development process easier, there were additional programs and libraries that could be added to the platform. The first library is jQuery (version 3.3.1), a JavaScript library for event handling, animation, DOM manipulation and document traversal, which can be used as a downloaded project or through an AJAX API.

The second library is Bootstrap, a front-end component library, collecting essential, pre-designed and pre-defined elements. Bootstrap is mostly used for building the HTML and CSS sections, it can be downloaded or acquired through a CDN.

Finally, for data analysis and visualization, additional libraries are used. The most important libraries are Pandas, Numpy and Plotly.JS. The first two libraries are Python based additions to accelerate data processing and analytics, while Plotly.JS is used for visualizing charts in dashboards.

4.3 Backend and database technologies

4.3.1 **Django and PostgreSQL**

Django is an open-source and free framework with a strong developer community and an easy-to-understand and maintain inside logic, called Django MVT Structure. MVT is short for Model-View-Template. These three data structures are the souls of the Django Framework (Figure 5 – Django MVT model).

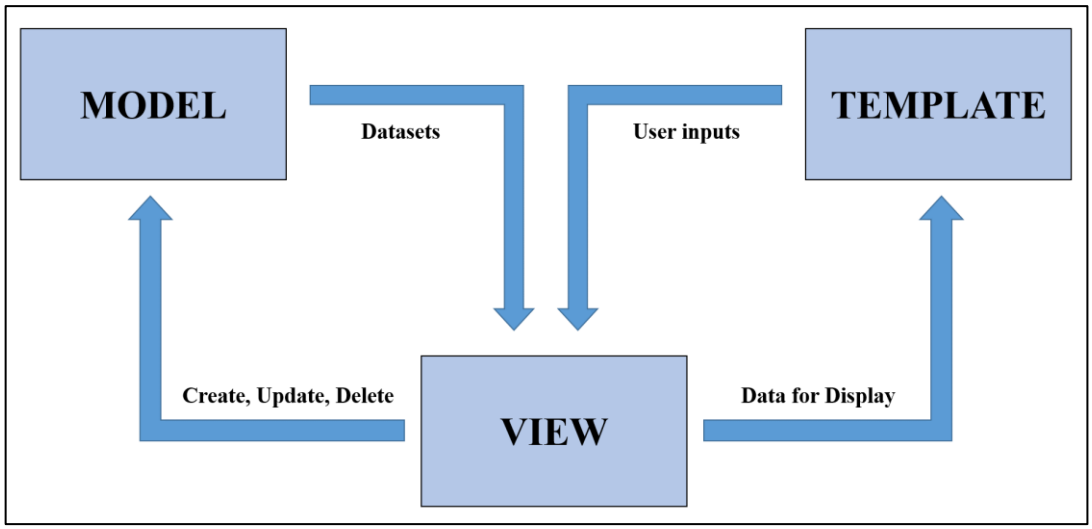


Figure 5 – Django MVT model

Model is the logical data structure, which provides a definition of data formats, called datasets. Views receives data via request methods like POST and GET from the user and it formats the data so it can be stored or requested to or from the database. Data modification is triggered by the View, for instance creating new records, updating or deleting existing parts of datasets. Templates are a convenient way for HTML generation as data can be displayed in a user-defined visual environment. User inputs are received from the templates as well.

Django has strong built-in features such as authorization or administration, which enables the developer to manipulate data in a graphical interface as well (Figure 6 – Django graphical interface).

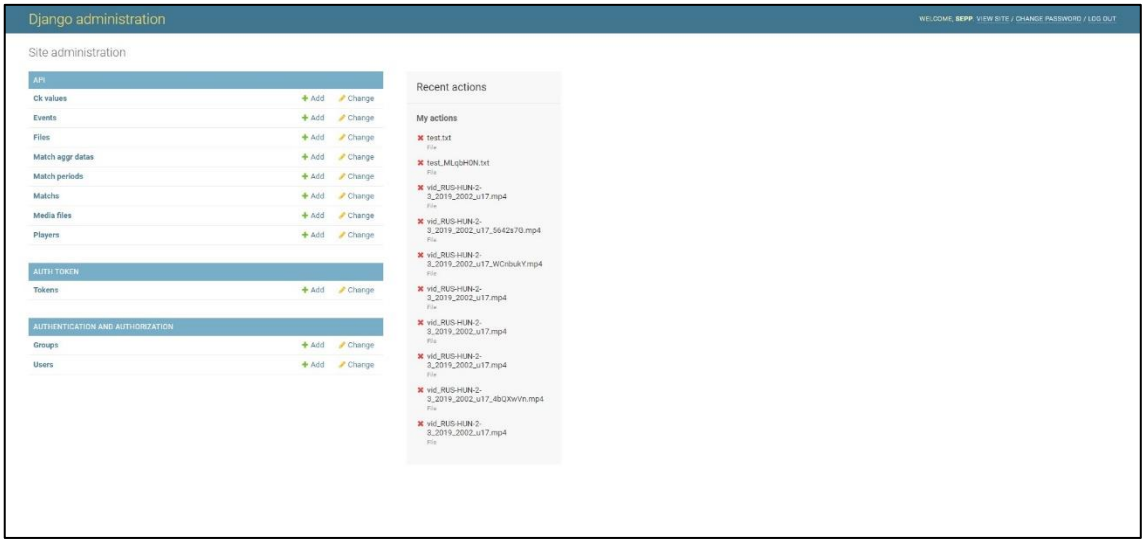


Figure 6 – Django graphical interface

The other important part of the backend is the database. While Django's built-in SQLite database could be good for a starting project, the developer does not have full control over the database to perform any operation on current or future data tables - unlike in MySQL or Oracle. In this project it was a very important factor to handle the database totally, however the project preferred a free solution. That was the reason why an open-source object-relational database system called PostgreSQL was the best choice. PostgreSQL also has a strong reputation for reliability, feature robustness and performance, and the developer can also work in a graphical environment (Figure 7 – Graphical user interface for PostgreSQL).

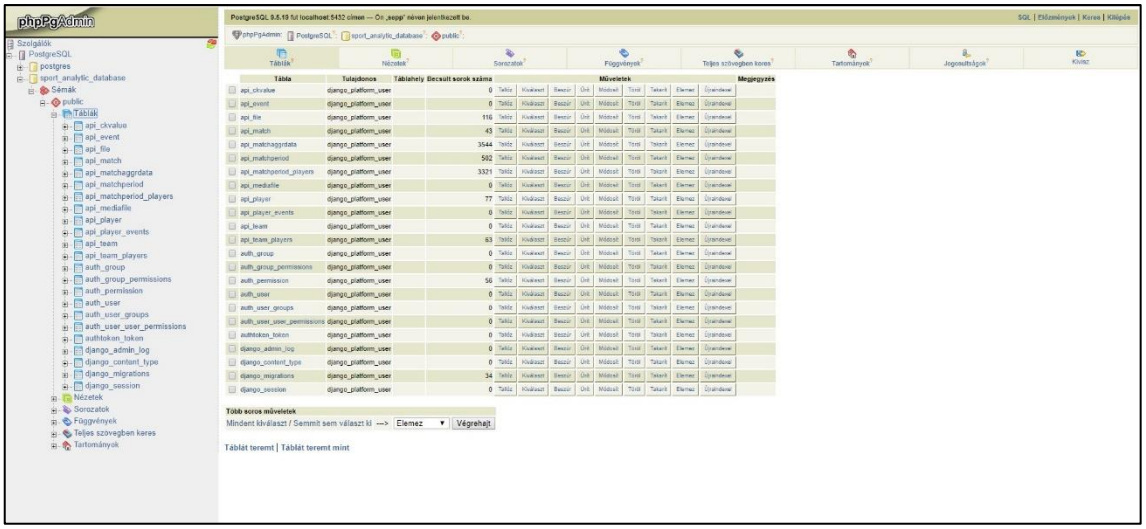


Figure 7 – Graphical user interface for PostgreSQL

4.3.2 **GitLab and Docker**

GitLab is an open-source, web-based DevOps platform that enables developer teams to collaborate on projects. The software stores the code in so-repositories that control multiple parallel versions of code in branches, making troubleshooting, issue handling and maintenance easier. With the tool called CD/CI (which stands for Continuous Integration, Continuous Delivery and Deployment), publishing processes can be connected to any server with secure methods. [15]

As platform independence was a requirement, Maze Project has been built to run in a Docker container. That way, all applications can run in their standalone environment, with its unique dependencies. The approach makes portability more efficient for the application, as well as it reduces maintenance costs. They also isolate the software, ensuring a strong level of security. [16]

5 System requirements

5.1 System build-up overview

As Figure 8 – Technological structure of the client and server sides of the application shows, the platform basically has three important parts: the front-end, which is the part of the platform a user can reach and interact with, the backend, which is practically a logical and communications bridge as it handles and resolves requests, and the database, which contains and provides data.

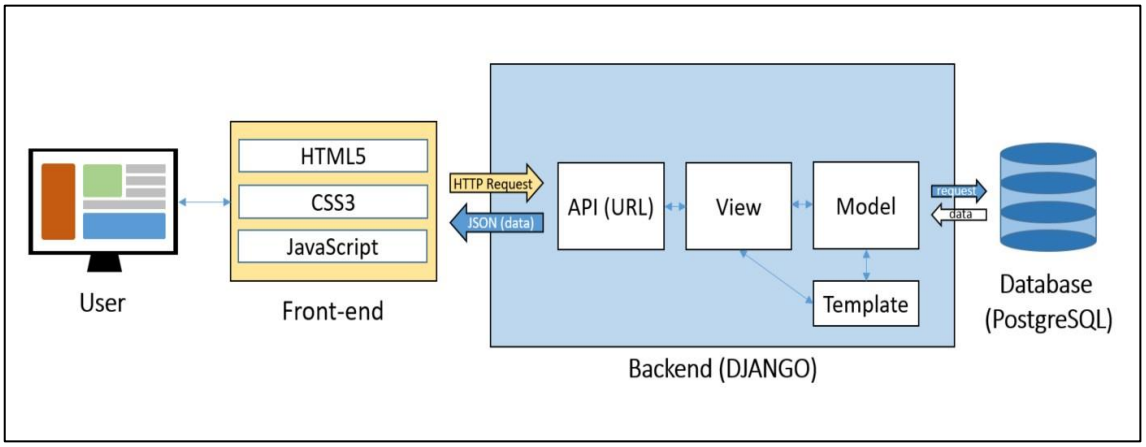


Figure 8 – Technological structure of the client and server sides of the application

The front-end sends HTTP requests to the backend, more precisely to the API application of the backend. The API sorts and forwards the requests to the corresponding Django View. The View is the central processing part of the backend, this is where the backend evaluates the requests. If needed, it sends the requests to the proper database via the respective Model, then returns the data in question to the front-end, typically in JSON format.

The three parts use three different ways to communicate: the front-end works with JavaScript for event handling, the Django backend uses Python as programming and processing language, and the database handles requests with SQL queries. In order to solve all tasks, one should be familiar with all three languages.

In this Thesis work, these three parts are also referenced as layers, but one should not take it as a reference to the Open Systems Interconnection (OSI) model [17]. The OSI model has 7 different abstraction layers, while this Thesis only differentiate three: the presentation layer (front-end), the business logic layer (backend) and the database layer

(database). The term layer however, is similar to the definition of layer by OSI, as it not only represents how layers are built on each other from the user's perspective, but how communication and data flow is served by one to another.

5.2 Database layer

5.2.1 **Received database system**

The database system presented below was designed and created by MazeSoft. I received an exact description of data tables, attributes and connections for all database objects, and relationship graphic. For deeper understanding of the database structure and the model objects, in the following pages I present the received database system.

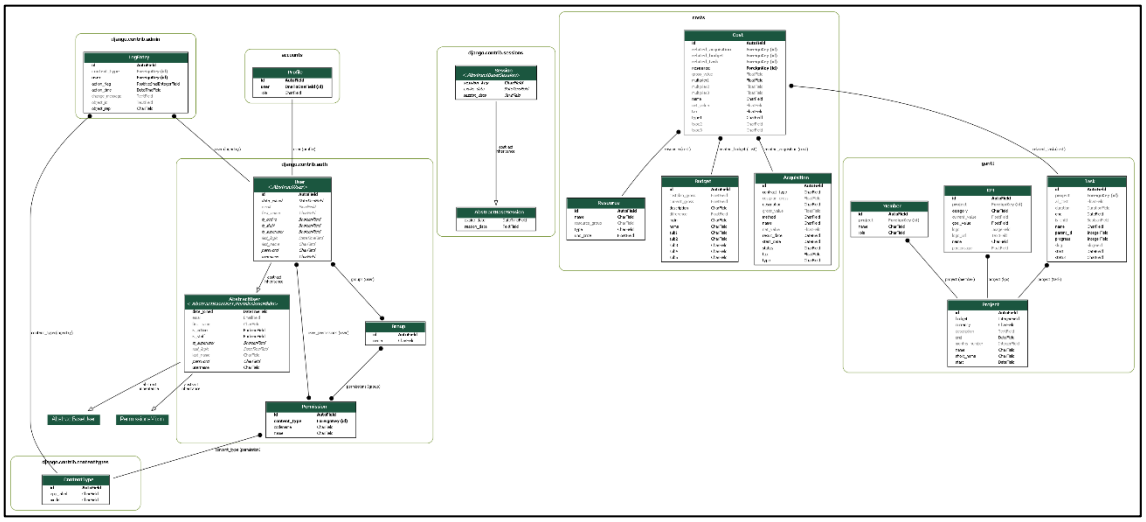


Figure 9 – The whole database structure (Image: courtesy of MazeSoft)

Figure 9 shows the whole database structure, but for the development, only the Costs and Gantt model groups are relevant, as many database build-ups, such as handling administration-related objects like users, had been covered by Django initially.

The two relevant model groups are shown in detail in Figure 10, Table 1 and 2. The two tables contain model object descriptions with the name of the attributes, the type of field and short description.

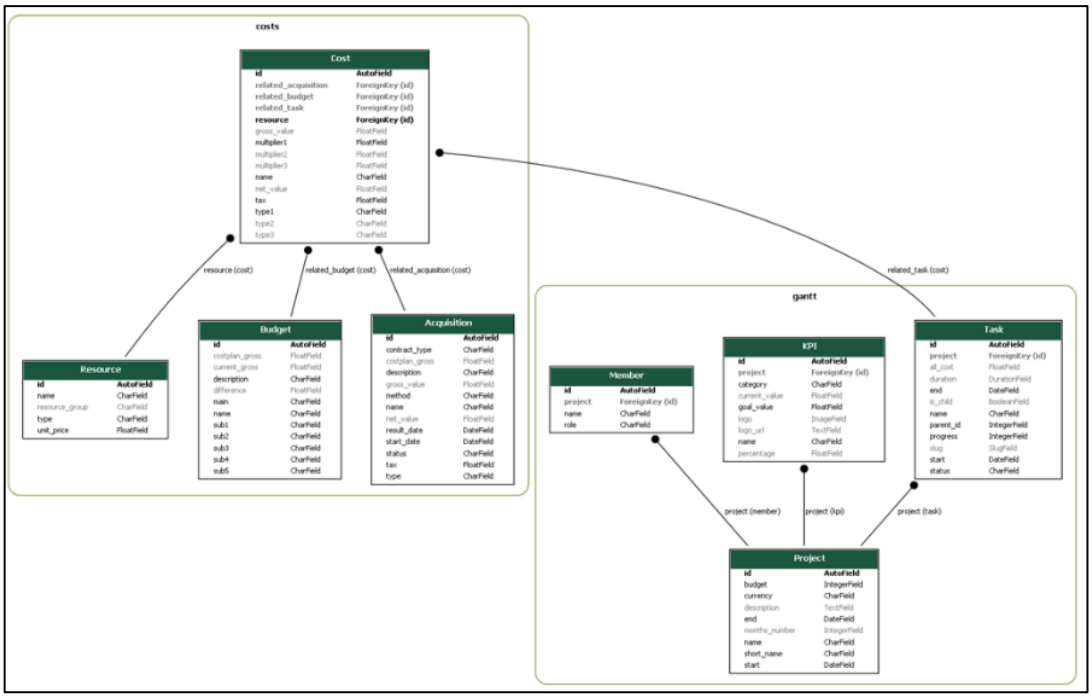


Figure 10 – Costs and Gantt model groups with attributes

Name of model		
Attribute name	Type of attribute	Short description
Cost		
id	ID	Identification number
related_acquisition	Foreign key (id)	Connector to Acquisition model
related_budget	Foreign key (id)	Connector to Budget model
related_task	Foreign key (id)	Connector to Task model
resource	Foreign key (id)	Connector to Resource model
gross_value	FloatField	Gross value of cost (computed automatically)
multiplier1	FloatField	Additional unit field cost value
multiplier2	FloatField	Additional unit field cost value
multiplier3	FloatField	Additional unit field cost value
name	CharField	Name of the cost
net_value	FloatField	Net value of the cost unit
tax	FloatField	Tax percentage
type1	CharField	Type of additional unit (e.g. time)
type1	CharField	Type of additional unit (e.g. quantity)
type1	CharField	Type of additional unit (e.g. sale)
Resource		
id	ID	Identification number
name	CharField	Name of resource
resource_group	CharField	Name of resource group
type	CharField	Type of resource
unit_price	FloatField	Price of one unit
Budget		
id	ID	Identification number

costplan_gross	FloatField	Gross value of cost plan's related row
current_gross	FloatField	Current value of budget row
description	CharField	Description of budget row
difference	FloatField	Difference between plan and current value
main	CharField	Name of main budget category
name	CharField	Name of budget row
sub1	CharField	Name of subcategories
sub2	CharField	Name of subcategories
sub3	CharField	Name of subcategories
sub4	CharField	Name of subcategories
sub5	CharField	Name of subcategories
Acquisition		
id	ID	Identification number
contract_type	CharField	Type of acquisition contract
costplan_gross	FloatField	Value of related cost plan row
description	CharField	Description of acquisition
gross_value	FloatField	Gross value of acquisition (calc. automat.)
method	CharField	Name of acquisition method
name	CharField	Name of acquisition
net_value	FloatField	Net value of acquisition
result_date	DateTime	Date of acquisition results
start_date	DateTime	Start date of acquisition
status	CharField	Status of acquisition
tax	FloatField	Value of tax
type	CharField	Type of acquisition

Table 2 – Object attributes in the Costs model group

Task		
id	ID	Identification number
project	Foreign key (id)	Connector to Project model
all_cost	FloatField	Total cost of task (calculated automatically)
duration	DurationField	Duration of task (calculated automatically)
end	DateField	End date of task
is_child	BooleanField	Boolean, if the task has child task (hierarchy)
name	CharField	Name of the task
parent_id	IntegerField	ID of parent task (if has any)
progress	IntegerField	Integer value of progress percentage
slug	SlugField	Short, slug version of task's name
start	DateField	Start date of task
status	CharField	Status of task
Member		
id	ID	Identification number
project	Foreign Key (id)	Connector to Project model
name	CharField	Name of project member
role	CharField	Role of project member
KPI		
id	ID	Identification number
project	Foreign Key (id)	Connector to Project model

category	CharField	Category of KPI
current_value	FloatField	Current value of KPI
goal_value	FloatField	Goal value for KPI
logo	ImageField	Logo to display in KPI card
logo_url	TextField	Url of logo to display in KPI card
name	CharField	Name of KPI
percentage	FloatField	Calculated value of progress (calc. autom.)
Project		
id	ID	Identification number
budget	IntegerField	Total value of budget for project
currency	CharField	Currency of budget
description	TextField	Description of project
end	DateField	End date of project
months_number	IntegerField	Number of project length month
name	CharField	Name of the project
short_name	CharField	Short name of the project
start	DateTime	Start date of project

Table 3 – Object attributes in Gantt model group

5.2.2 **Costs model group**

The Costs model group contains 4 object models: Cost, Resource, Budget and Acquisition. The main object, as Figure 10 highlights, is the Cost model. It connects all the other three objects, with the following logic: all costs require resources, with multipliers to clarify the exact way of usage. Then all costs can be assigned to a budget row. Simultaneously, a cost can also be assigned to an acquisition. These two ways of organizing cost ensures to find the balance among used resources, planned or performed acquisitions and the pre-planned budget.

5.2.3 **Gantt model group**

The Gantt model group, similarly to the Cost model group, has 4 object models: Project, Task, Member and KPI. The connector model is the Project, as any entity of all other three objects must be associated with a project. Therefore, Maze Project can handle several projects at the same time. As every project is built-up by tasks, has members to be assigned to tasks and has an ongoing level of progress, the Gantt model group summarizes the action-level processes of the project for the user.

5.3 Presentation layer

5.3.1 Main Graphical User Interface functionalities

The presentation layer or the front-end is what the user actually sees from the full software and can interact with. As the topmost layer it has three main functionality groups: user navigation, data presentation and visualization.

1) User navigation

User navigation contains all functionalities that help the user to define and clarify the exact options and possible actions they can perform on the site. From the headlines and menu items to the used colors and tooltips it summarizes to the user, what is the goal of the presented site and where to go to perform other actions. It also provides information about the other two layers, for instance in case of bad requests or wrong data input.

The main elements of the user navigation toolkit are based on HTML, like texts and buttons, on CSS like colors, size or highlights, and on JavaScript like animations.

2) Data presentation

The second group contains all functionalities that is about dynamic data handling. How data that the user can interact with, is shown or hidden, how data presentation is part of the site and in which form, and how it is handled on the front-end. Django is a form-based backend, meaning almost all data communications are handled preferably via forms. However, forms can be presented not only as a list of input fields, but as tables or inline forms as well. With the ability to customize them in several forms, and the use of invisible, so-called hidden fields, form-based data provisioning can serve every possible need the front-end and the user might expect.

The main elements aside from HTML forms are the JavaScript codes that run on the front-end, and that connects user interactions like a click on a button or on a table row with the proper form element, which later will be handled by the business logic layer to properly transform and validate.

3) Visualization

Last, but not least there are static data, the information the user needs for daily use, but do not necessarily want to interact with, such as the main KPIs or tooltip data on the Gantt chart. Visualization functionalities are mostly based on CSS and JavaScript,

transforming numbers and strings into colorful, animated visuals and charts, providing understandable, easy-to-comprehend information to the end-user.

5.3.2 **Responsiveness**

Nowadays, in the middle of the Industry 4.0 era, users have the natural desire to be able to visit a website not only on their computers but on mobiles or tablets as well. Although, in case of Maze Project's target group, project managers still do most of the hard work on their laptops, the urgent need to check facts or to lookup specific numbers during a meeting or a call is growing rapidly. Therefore, during the development process, it is not only a basic demand to focus on these platforms as well, but a strategic cornerstone.

However, as project managers' everyday workflows showed, the purpose of the two versions (desktop and smaller devices) aims different goals. While the former is the main platform for data modification and long, deep analysis processes, the latter is for shorter, quick fact checks and visual confirmations. That difference should be mirrored in Maze Project as well, and to achieve that, it is important to enhance how content is presented in the website. Later, in section 6, front-end implementation presents the different versions of the site to review, how successful development was from that perspective.

5.4 Business logic layers

1) Navigation (links)

One of the most important functionalities on the Internet, from the very beginning of web1.0, was the ability to move from one location to another. Links are connecting pages together, letting developers to easily change the presented view for the user based on their inputs. In Django, most of the navigation actions are performed by the Business logic layer, where user inputs are evaluated by custom designed codes, in order to serve their needs on a higher level.

While basic HTML navigation is still an option, this layer provides deeper insight of transferred data, with the power of changing routes from one url to another faster and based on predefined scenarios. For instance, it is possible to just by one click, navigate user towards two different sites, relying on previous user actions. Therefore, static HTML navigation can be upgraded to dynamic, Business logic based navigation.

2) Data transfer and transformation

Figure 8, in section 5.1 described the linking functionality of the Business logic layer. Being in the middle of the chain of events, data flow is determined and executed by this layer on a regular basis. As the demand arrives from the end user, request handling and processing defines, what data should be selected for actions like saving, deleting or modifying. Moreover, several times data need not only being transferred but transformed as well.

Request handling scenarios, defined by the Business logic layer handle default and custom developed data transformation on-the-fly, to serve the appropriate data for the normal working process. These functionalities lead us to the last category, which describes which functionality group makes it possible to work.

3) Data validation and error handling

The last functionality group provides the base ground for all other functionalities. As the Business logic layer serves as a gate between the Presentation and the Database layer, this group of functionalities are the guard that makes sure only valid data flow can run through the platform.

Two types of validation should be executed here: the first type works with the Database layer, validates data flow that comes from daily working processes, predefined by developers. It concentrates on data conversions success, missing data and other regular inner requests. The second type works with the Presentation layer, therefore with user requests, coming from outside. Here, data validation focuses not only on validating basic forms, but also on filtering out possible harmful intentions.

Fortunately, Django provides crucial safeguards for form validation by default, and by that and the fact that the framework is based on forms, it serves as a great help for developers.

6 Front-end implementation

6.1 Project attributes module

Project attributes module contains three pages, each responsible for different module functionalities. Figure 11 - Home screen of the application after sign in, shows the home screen after the user successfully logged in the website. The tile design was implemented in order to make responsiveness easier. In the right lower corner, the standalone tile enables the user to create new projects easily and immediately, navigating to a new page.

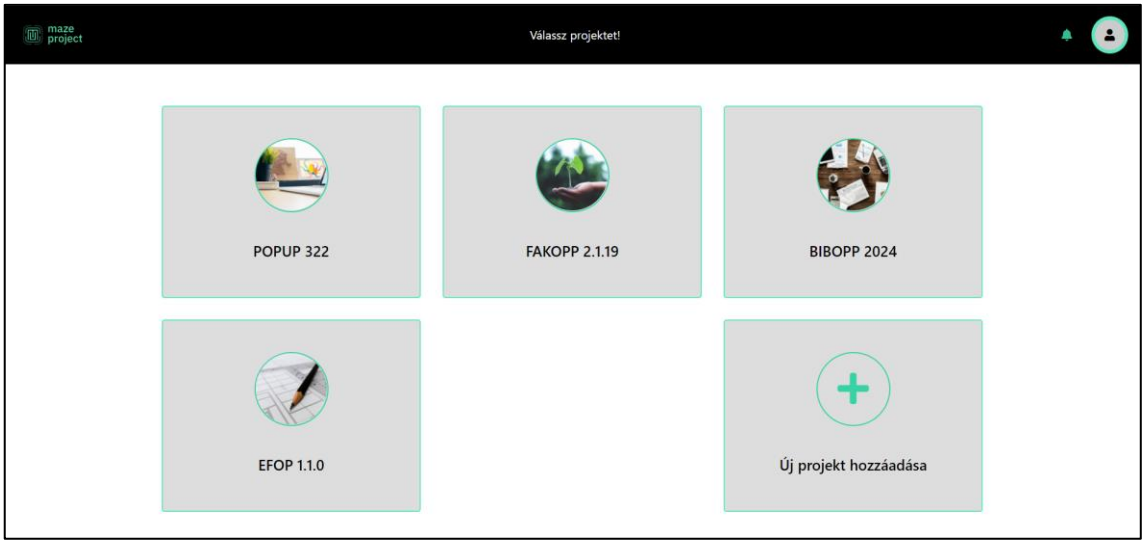


Figure 11 - Home screen of the application after sign in

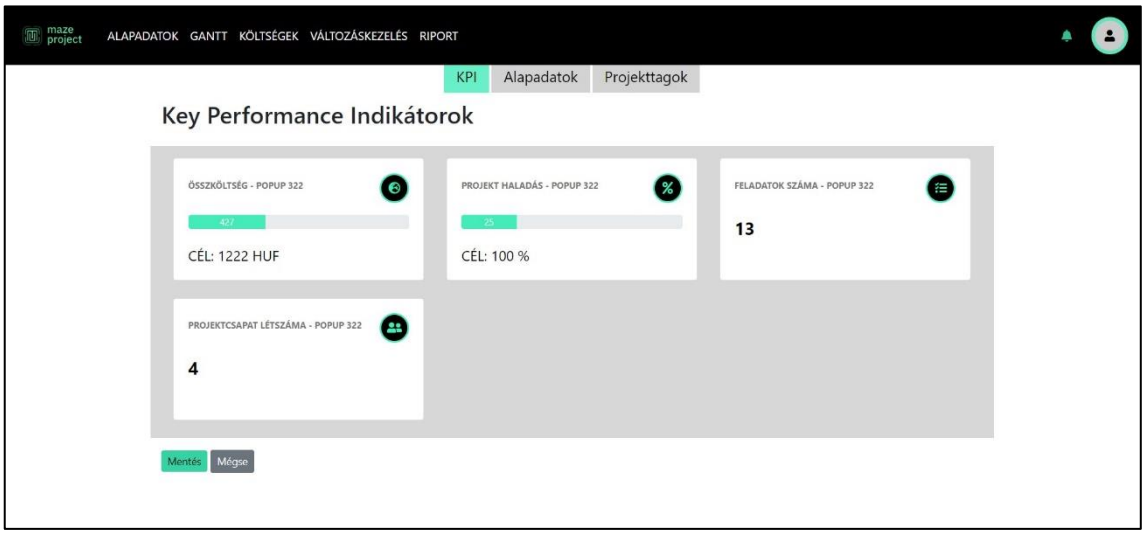


Figure 12 - KPI page, displays the first page that is shown to the user after selecting a project from the main page. All Key Performance Indicators (KPIs) are recalculated at the moment of page loading, in order to ensure the most recent data are displayed about the selected project. KPIs can be added or removed on-the-fly, and can be customized with unique logos or images, texts and measures to calculate. KPI categories are Percentage, Currency, Number, Time or Other. Every category has a different card layout, and a default logo. The main goal is to hand over the user quick but thorough information about the project's current financial position, according to the budget plan.

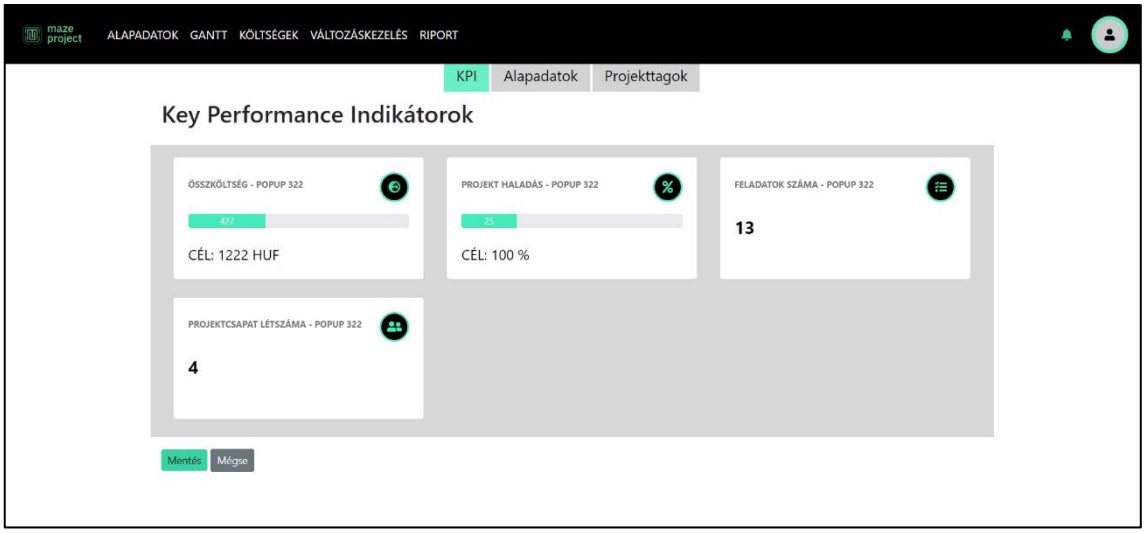


Figure 12 - KPI page

Figure 13 – Project attributes, is showing the attributes of the projects, for instance the name of the project, start and end dates, budget limitation or the used currency. Date inputs are made easy by a popup mini-calendar. All available data can be modified and updated by simply clicking into the texts and typing. All changes can be saved with the green button on the bottom, or the whole project can be deleted as well with the trash icon. However, as it is a crucial step, a second-level confirmation for deletion is necessary, provided by additional JavaScript code, to avoid unintentional delete.

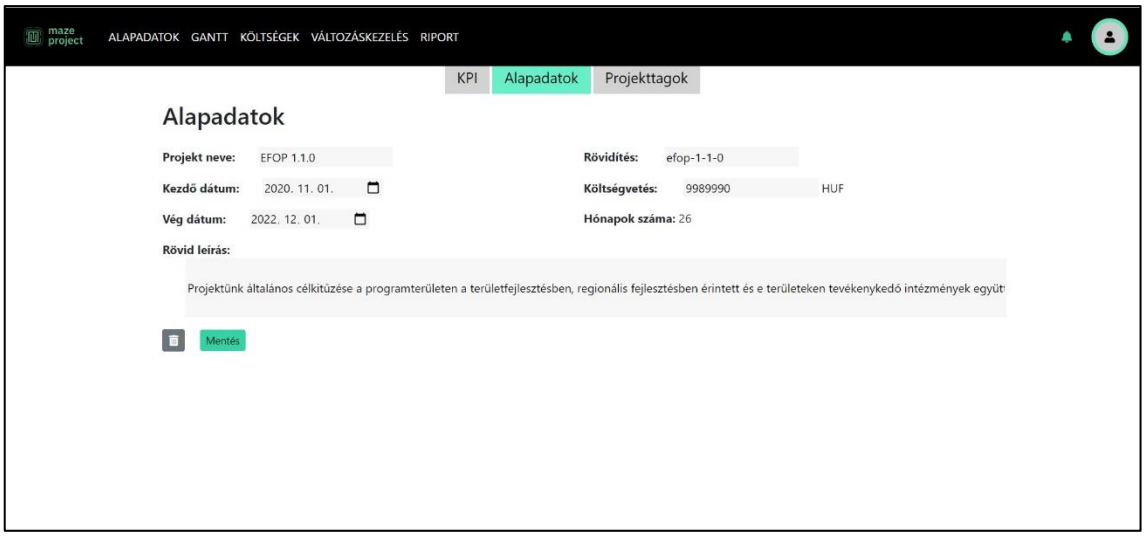


Figure 13 – Project attributes

In Figure 14 – Project members, the project team handling process is visualized. Each card represents a member of the project. With this card tile design, responsiveness is ensured. Project members can be assigned to roles or teams easily with a few clicks, enabling the project manager to have a clear overview of the distribution of human resources. New project members can be added, or existing can be deleted with the two buttons on the bottom.

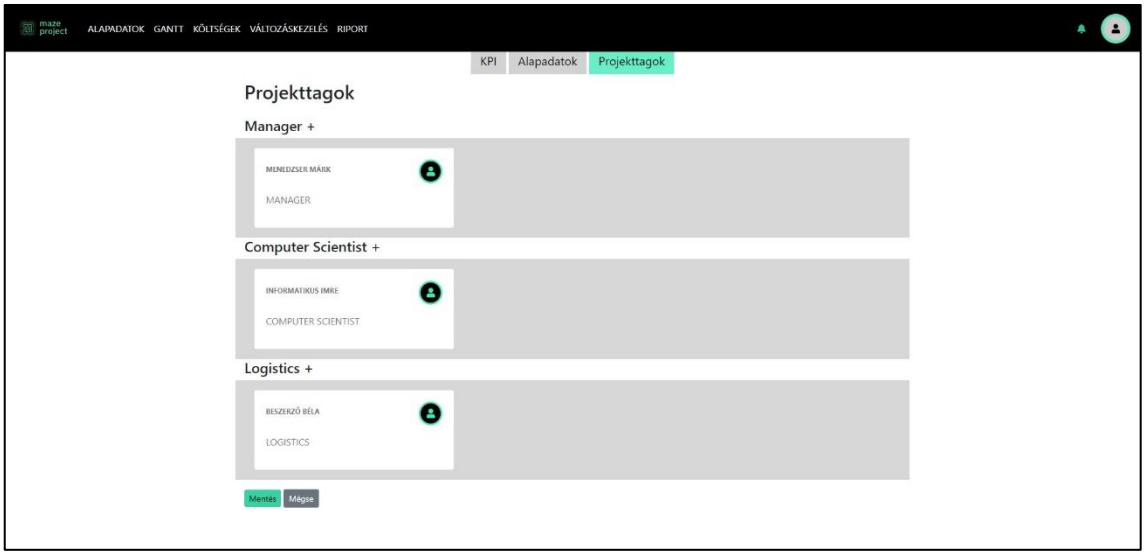


Figure 14 – Project members

6.2 Gantt module

The Gantt module has two subpages, Gantt chart and the To Do. Gantt chart is the core of Maze Project, where tasks can be edited and adjusted real-time. Figure 15 – Gantt view, shows the overview of the page.

The page operates with a split window layout. The window on the left is the table-based version of the Gantt chart, ordered with the WBS structure and detailed in columns, such as task name, start and end date, dependencies, status or percentage. The window on the right is the visual representation of the Gantt chart, with processes for tasks and arrows for dependencies. Both windows can be edited and the actions trigger and change the parallel row as well.

Each row has a pair in the neighbor window, connecting the table and line representations together. Whenever an event such as drag or rewrite happens on any element, the pair is affected as well. The process is also aided from a graphical aspect, as hovered and selected rows in both panel are highlighted with matching background colors during the editing process. Another important functionality is the possibility to update rows by elements, via the save and delete buttons on the left side table. That way, minimal data transfer operation can be guaranteed.

In the left bottom corner, the three buttons enable the user to switch among different Gantt-chart views such as day, week or month view, to get a more detailed or summarized picture of the whole project.

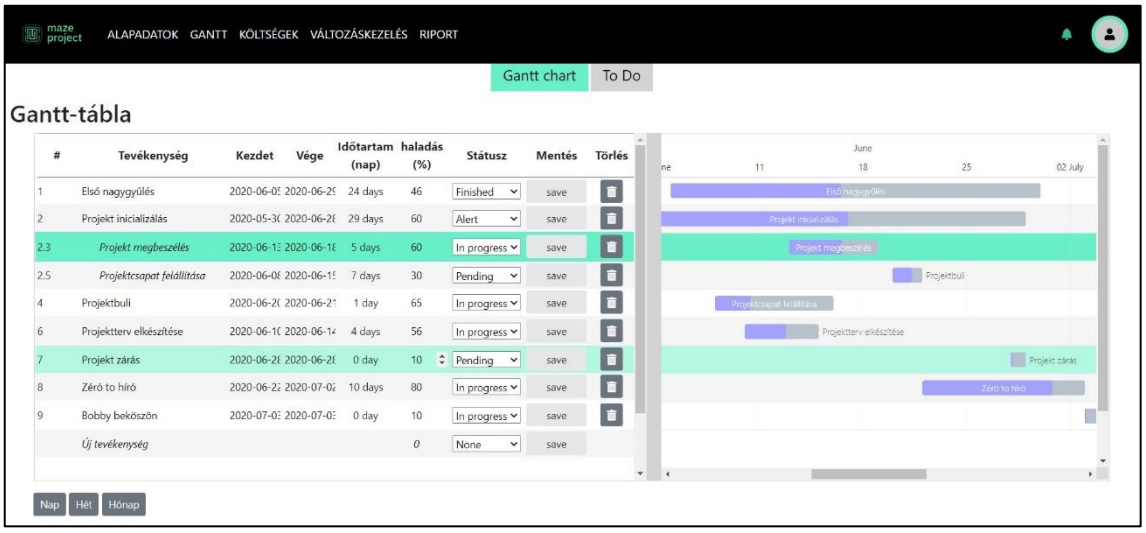


Figure 15 – Gantt view

The other important page in the Gantt module is shown by Figure 16 - To Do list page. That page can list all important items that can be related to any tasks, dynamically be added or removed by the user. Moreover, inline modification is enabled on the site.

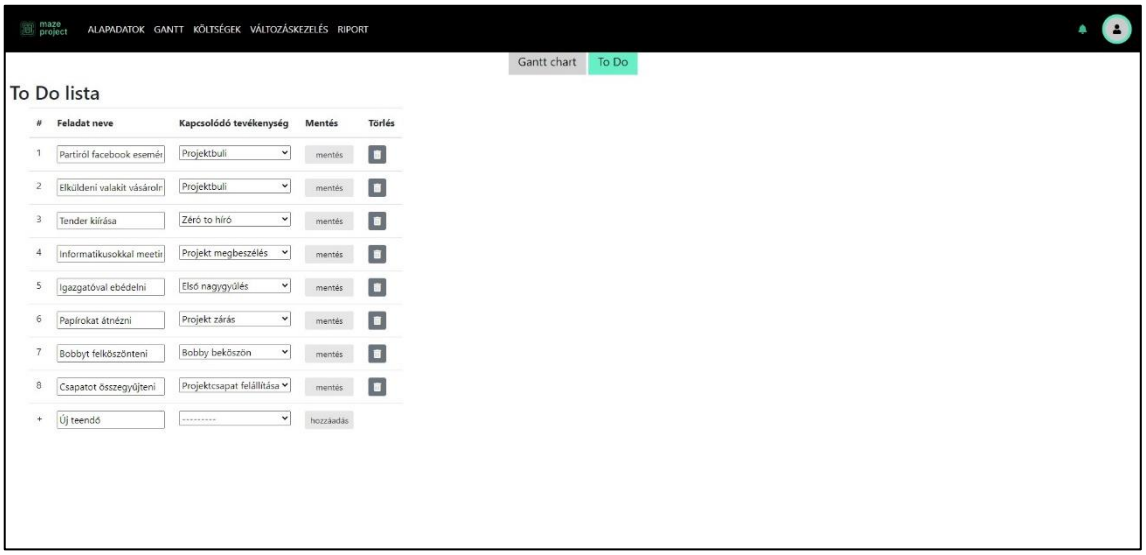


Figure 16 - To Do list page

6.3 Budget and resource management module

Budget and resource management module has four different subpages, each responsible for a different type of resource controlling.

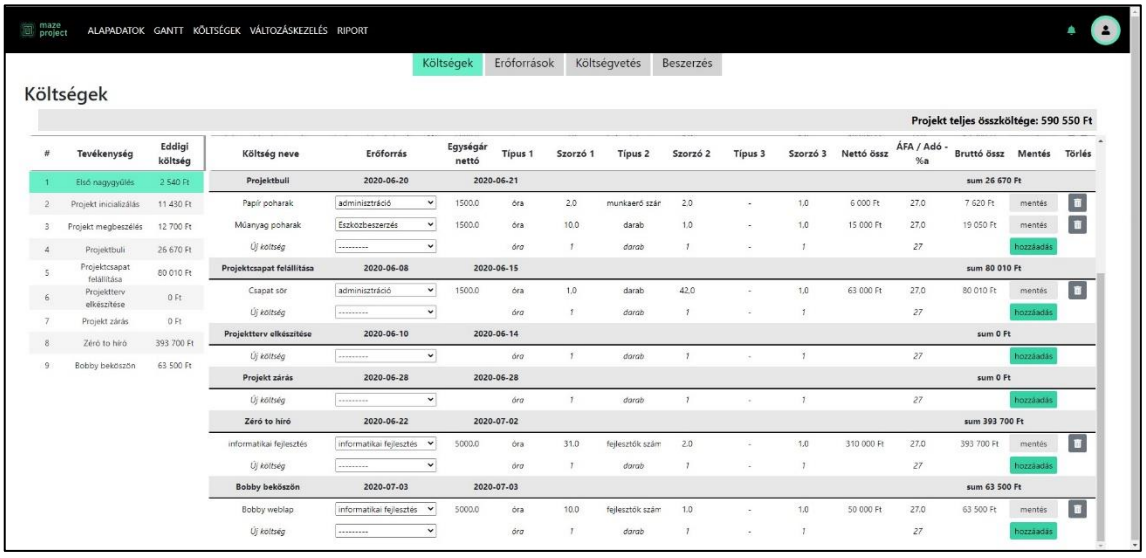


Figure 17 - Expenses page

Figure 17 - Expenses page contains two main elements. The left, smaller panel shows an aggregation of all tasks, related to the selected project, in a list form. The

software automatically aggregates all previously connected expenses, providing an insight on total costs per task.

The right panel operates with a nested table system layout. All tasks are listed in the main rows, highlighted with gray background. Below each task, related expenses can be created, modified or moved from one to another. The total summary of the project's cost is presented above the two panels. Another useful functionality that had been implemented is the scroll-control, based on the left panel. Selecting an item in the aggregated list automatically scrolls the right table to the exact place in the nested table, with an additional highlight. As a result, not only aggregation is presented for the end user for insights, but simultaneous navigation in the larger list is solved conveniently.

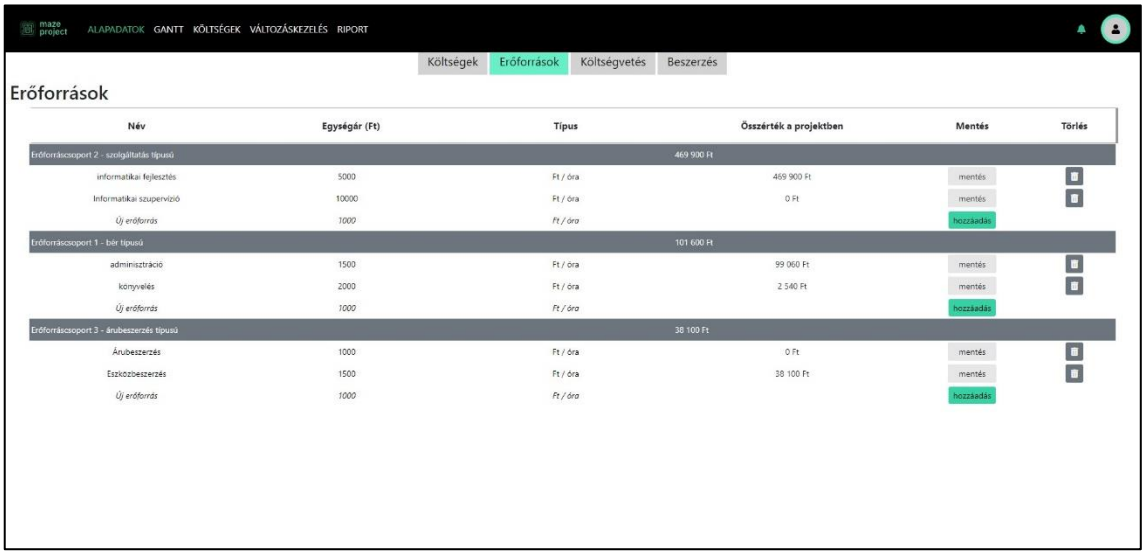


Figure 18 - Resources page

Figure 18 - Resources page, visualizes, how resource categorization works in practice. Every resource belongs to a resource group, highlighted with dark gray background. Groups describe the base aspect of grouping, such as human resources, salaries or procurement of goods. Each resource is listed below the category row, where attributes can be listed via the inline forms.

Total costs are aggregated on both resource and resource group levels, providing deep and insightful information to the user about resource management. As on other pages, immediate editing is solved with inline forms.

Figure 19 - Top half of Budget page, and Figure 20 - Bottom half of budget page, present the longer page of Budget submodule. This submodule not only provides two different functionalities to the user, but implements one of the core business ideas behind the software, how tasks and costs are playing a key role in successful project management. This submodule is, what connects in practice the usually separated method of tracking resources, tasks and costs.

The two functionalities are both represented in the figures. Figure 19 contains the so-called pairing table, while Figure 20 shows the budget plan for the project.

The pairing table has two panels, the left one is the fact panel, with up-to-date connections between costs and rows from the budget plan. The panel on the right is the storage panel, listing costs that are not connected to any budget rows at the moment. The two buttons on the middle are the controllers. The button on top submits all selected costs from the fact panel to the storage panel, the bottom button connects all selected unassigned costs to one, selected row of the budget plan. The whole process is helped with highlights for user understanding. In order to accelerate page performance, most of the switching actions are performed on the front-end, with the help of custom JavaScript functions. The pairing table can be shown or hidden as well, in order to make space usage customizable for project managers.

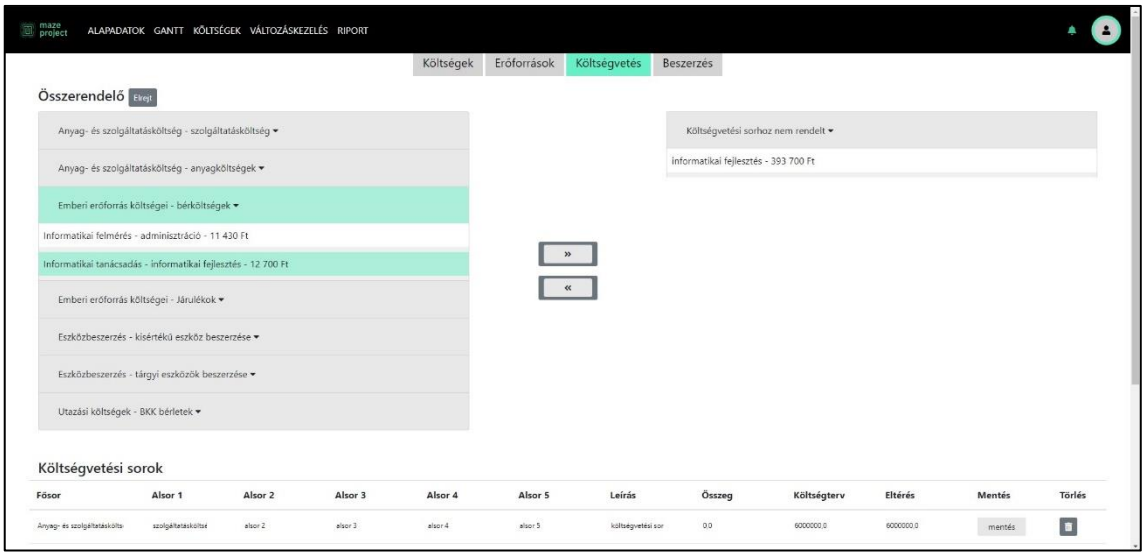


Figure 19 - Top half of Budget page

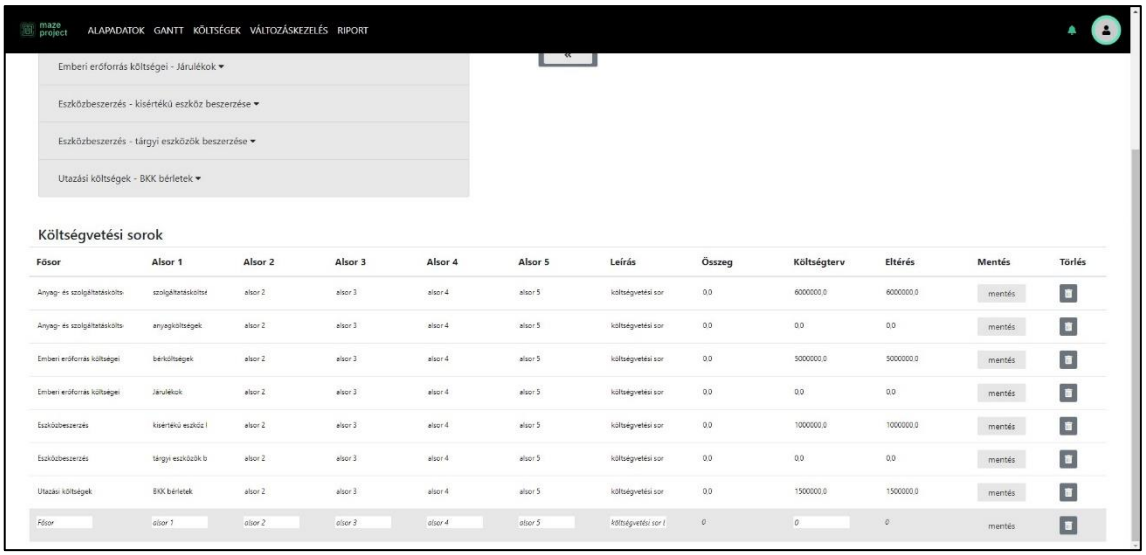


Figure 20 - Bottom half of budget page

The bottom half of the page contains the budget plan for the whole project. For successful project management, budget plan is the main comparison point in all actions. It not only contains all future spending, but also holds a mirror to the project plan, in terms of actual and desired position as well. The modification of the budget plan is delivered by the sorted and organized rows of the budget plan table, which includes a main category and five additional subcategories for high-level and deeper differentiations. Attributes like description, plan total and current summa are also listed in the table. As rows are modified or expanded, automatic calculations ensure that all rows refresh the current difference between actual and planned values.

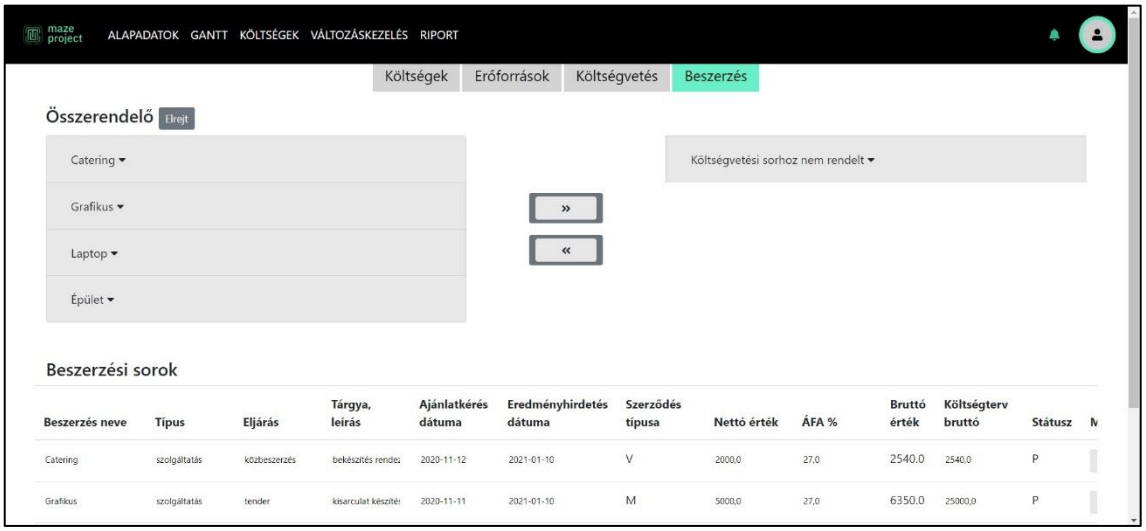


Figure 21 - Acquisition page

Figure 21 - Acquisition page, shows that both the Budget and the Acquisition submodules have a similar structure. Acquisitions are responsible for all the resources that come from outside the project. While Resources lists all inner capabilities, Acquisition groups all resources that need acquisition for the project success.

As both tables, the upper pairing and the lower acquisition listing tables have similar functionalities; as it had been described in the overview of the budget page, it is unnecessary to showcase them one more time.

6.4 Report and Dashboard module

Report and Dashboard module, as the name shows, has two submodules. Report is responsible for creating a status quo snapshot of the project's selected elements. As Figure 22 – Report page, shows, all elements can be selected with a custom designed checkbox, and the data can be exported in several formats, like Excel, CSV or PDF. The site also presents the selected data below the selector panel.

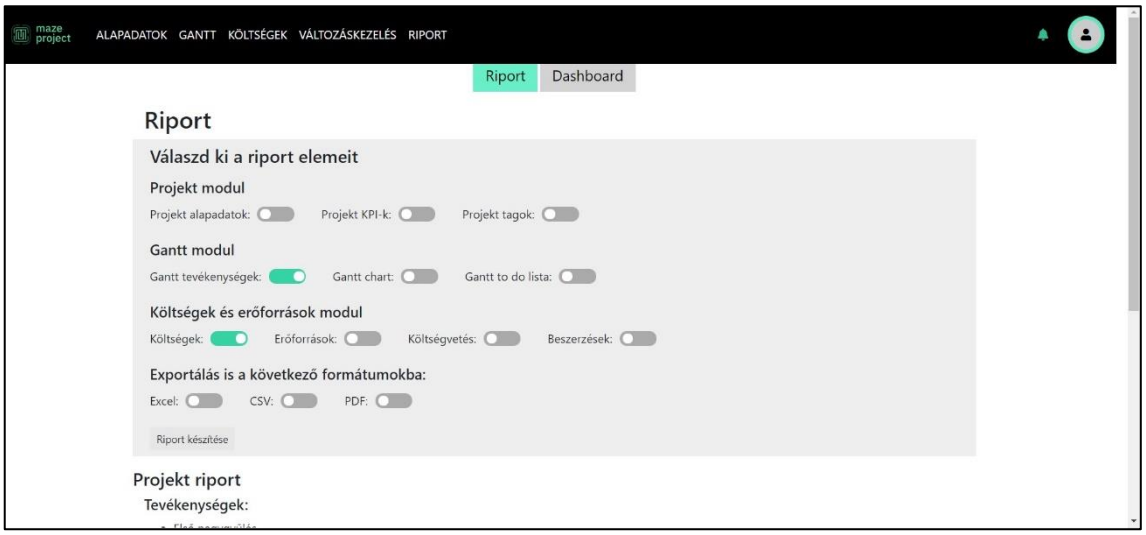


Figure 22 - Report page

In Figure 23, the Dashboard page is presented. These listed charts and visuals enable the project manager to have an insightful understanding of the current project state via visualized data. These visuals had been created with Plotly.JS, an open-source library. The first visual shows a summary of total costs. The second indicates the distribution of acquisitions. The third chart gives an insight on how the budget plan is related to current values. The last visual summarizes the total project spending and incomes.

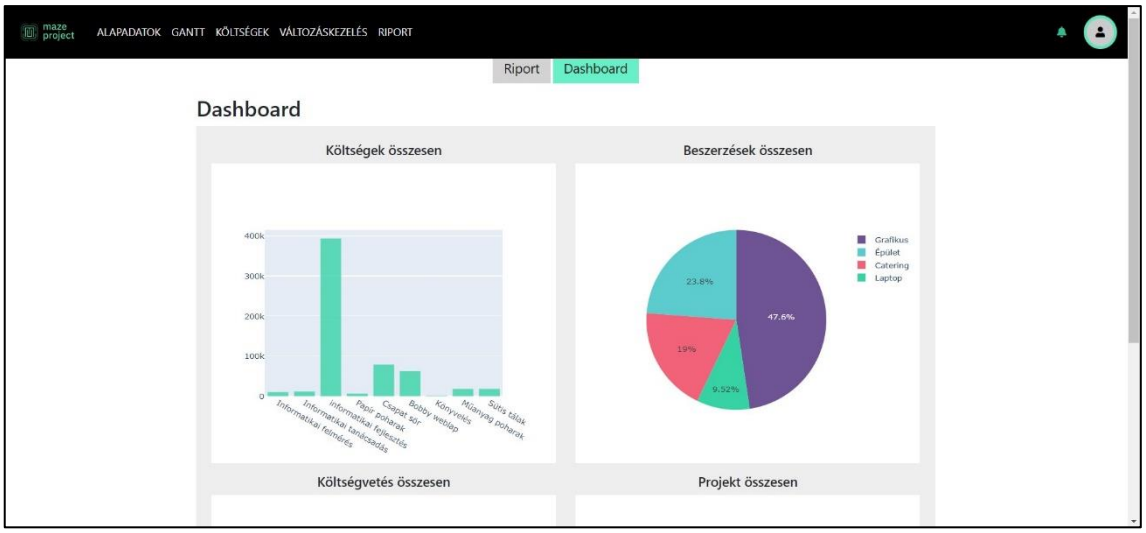


Figure 23 - Dashboard page

6.5 Responsiveness in implementation

In section 5.3.2, it was presented how responsiveness plays a key role in the future success of Maze Project. The implemented responsive pages are presented in this section, with additional description of usage. The test device was an Apple iPhone X for mobile version, and Apple iPad 2 for tablet version.

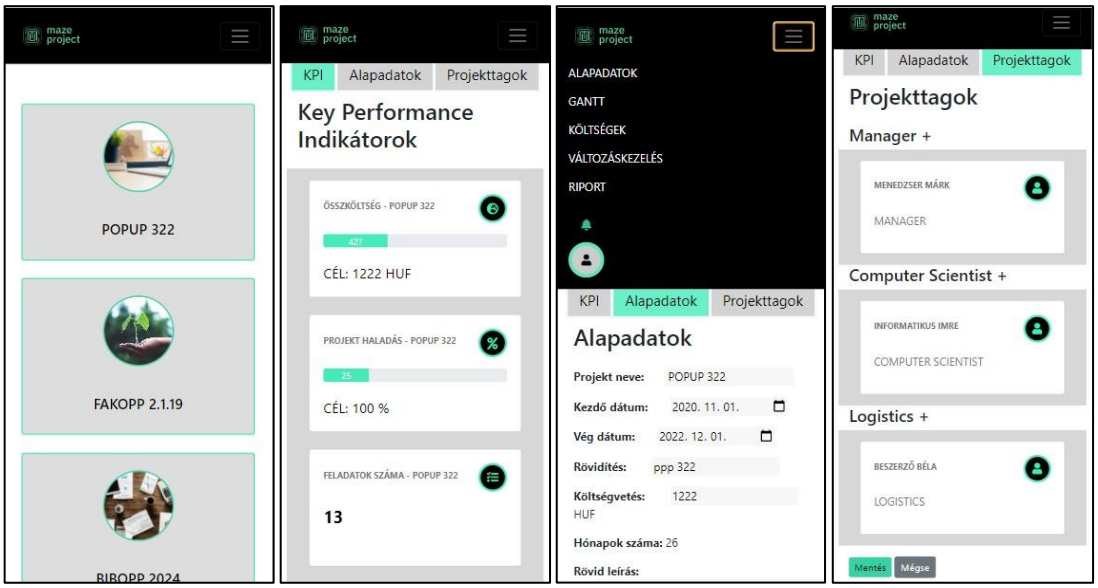


Figure 24 - Responsive version of Project attributes module

Figure 24 presents the responsive view of the Project attributes module. The first figure from the left showcases the landing page after login. The card layout changed direction, from horizontal to vertical, in that way creating a list of project cards. Although

it could be seen as a flaw in design, the fact that the third card has been cut in half, indicates that the list is not finished and more content can be found by scrolling down.

The second figure displays the KPIs that had been defined by the project manager. Similarly to the previous figure, a change in alignment can be detected on the placement of KPI cards.

The third figure from the left presents both the project attributes page and the behavior of the mobile menu in action. The upper black menu row turned into a so-called hamburger menu style, which can be toggled in and out of view with the touch of the icon in the right upper corner. The change of the project attributes form can also be remarked on the figure.

The last figure contains the project members list. Although in that dummy project, only three members had been added, the behavior of each member card is quite the same as it is represented in the landing page. Horizontal tile alignment had been replaced with vertical card columns.

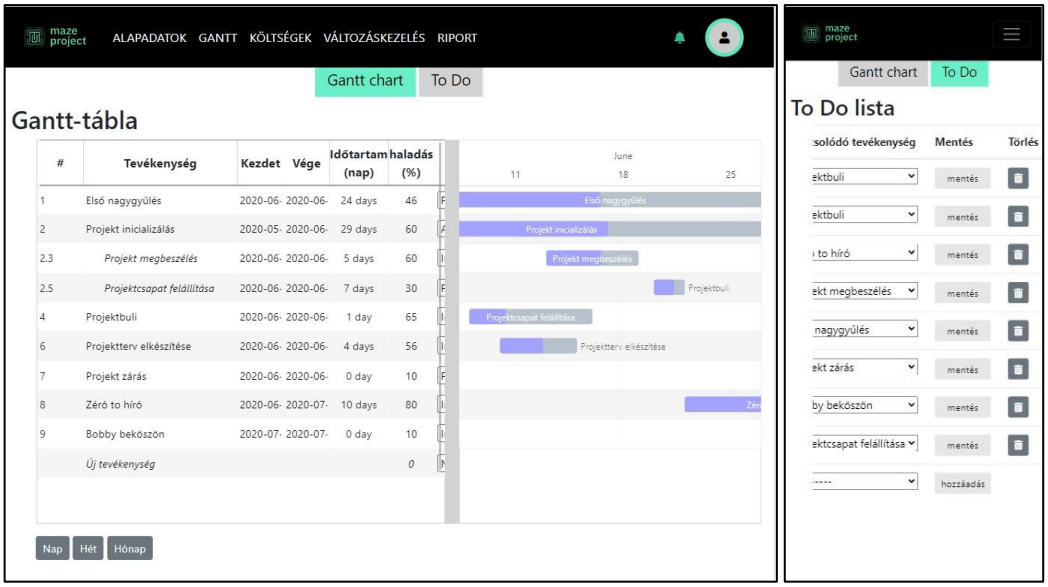


Figure 25 – Responsive version of Gantt module

The Gantt module's two submodule have different layout, based on the fact how the Gantt chart was developed and organized. In the earlier stage, the placement of the right chart was below the table, however, it disturbed the convenient and logical workflow of the test users, therefore it was reversed. In the end, I realized, that one of the main functional restriction on mobile view will be the ability to edit the Gantt-chart. In one hand, as it was discussed in the previous section, mobile view obviously has its limits,

but one can adjust to those limitations, as most of the project managers would not use their portable devices for an action like that anyway. However, on tablets Gantt chart is comprehensible, therefore it has not been excluded from there. The To Do list, on the other hand, is even handier in responsive view, as the list fits in the portable screen just more easily.

In Figure 26, the Budget and Resource management module's responsive version can be seen. The first figure from the left shows the expenses. Unlike on the Gantt chart, the Expenses submodule can be easily used with the vertical version of the two tables. As the left panel shows the aggregated costs, it can spare time for the user in case of the need for quick lookups, but for deeper information, it also behaves like a controller and table of contents. However, the detailed table of expenses are placed in a responsive table, which even though it is scrollable in horizontal direction, cannot show the whole rows. This is another limitation in the responsiveness, and a universal disadvantage of smaller screen sizes. The second figure depicts the Resources submodule, with the aforementioned limitations to the responsive tables, however in this page, most of the content is visible, only the last column with the delete option is missing.

The third and fourth figures shows the Budget and the Acquisition submodules, with the similar design layout. The top and bottom half differentiation remained from the desktop version, along with the ability of hiding the top table. However, the top table's two panels transformed into vertical rows in one column, with fixed heights. Therefore the user is informed that the content continues below the pairing panel as well.

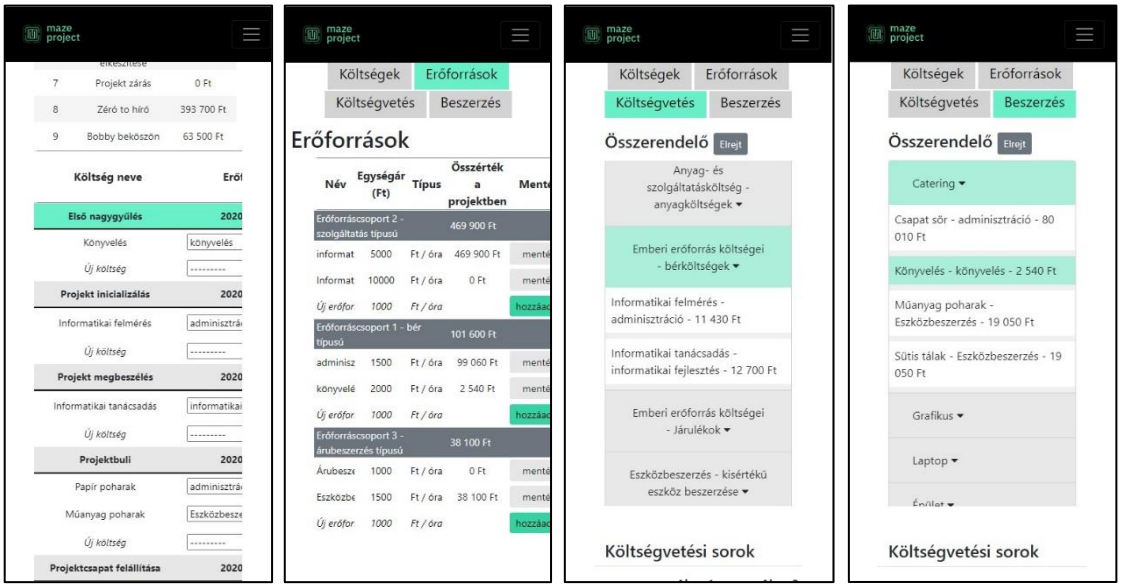


Figure 26 - Responsive view of Budget and Resource management module

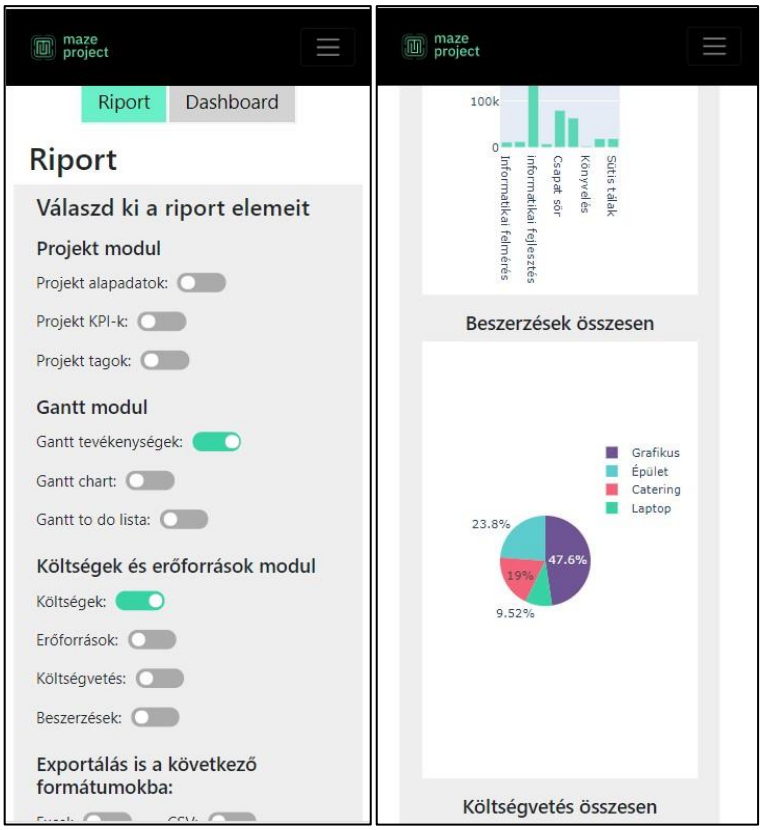


Figure 27 - Responsive view of Report - Dashboard module

It had been highlighted how providing instant information can be crucial for managers. Report – Dashboard module mostly attempts to achieve this very goal, therefore in this module was the success of responsiveness the most important.

Figure 27 shows the responsive view of Report and Dashboard module. The left figure displays, how the different selectors for reporting transformed into a column alignment, enabling users to scroll and select easily on portable devices. Similarly, all visuals in Dashboard now filling the available space to enable users to examine exact data values. Due to the Plotly.JS implementation, several options like zoom, export and highlights are available for the users in every chart, as well as the ability to include or exclude data from the chart by only a touch.

7 Backend implementation

Most of the backend implementations used the developer software Visual Studio Code. Although the dataflow would suggest a different approach, in this section Database layer implementations is presented first, in order to understand the models and attributes. Business layer implementations, as they connect the Front-end with the database, are presented last, as the finishing piece of the full software.

7.1 Database layer implementations

Database layer implementations are the coded pairs of the received database system description and attribute lists.

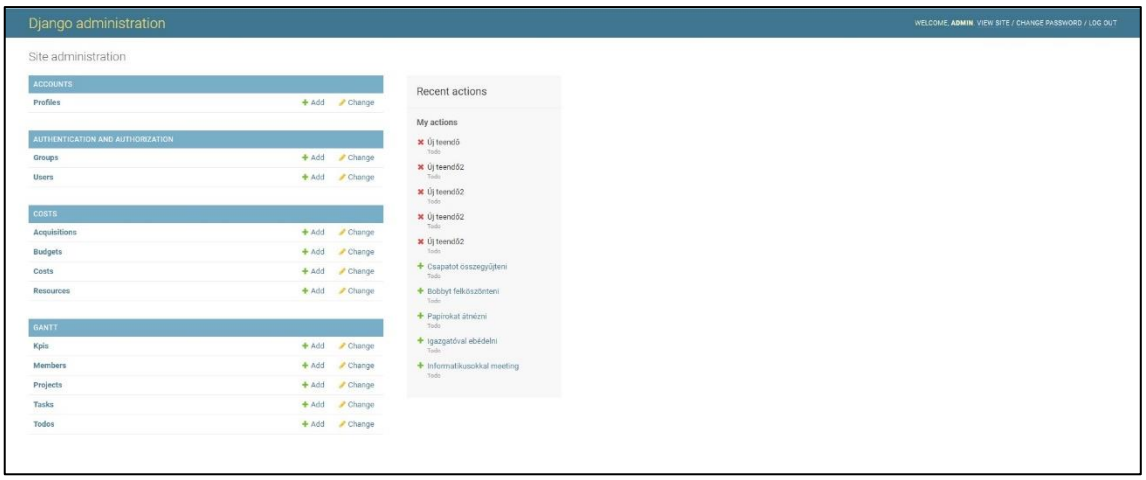


Figure 28 - Admin site of database

Figure 28 - Admin site of database, shows the Django admin page, where the database structure can be edited. It also lists the finished data model objects, with the list of latest database actions on the right gray panel. In this section, the exact implementations of the listed model are presented, categorized by the module they are connected to. It is important to note here, that although the Report – Dashboard module is a standalone module with its own functionalities, it does not rely on any models alone, as it creates reports on the other modules' models. Therefore, only the other three modules serve as model categories.

7.1.1 **Model implementations for Project attribute module**

The Project attribute module, as presented in section 5.2.1, works with three objects: Project, KPI, and Member. Each model had been created with the given attributes, however additional coding was required for proper connections, enhanced user experience and functional saving and overwriting processes. As the attributes had been listed before, in this section only the additional steps are described in detail.

```
class Project(models.Model):
    CURRENCY_OPTIONS = (
        ('HUF', 'Hungarian Forint'),
        ('EUR', 'Euro'),
        ('GBP', 'Great Britain Pound'),
        ('USD', 'United States Dollar')
    )

    name = models.CharField(max_length=100, default='Új projekt')
    short_name = models.CharField(max_length=30)
    budget = models.IntegerField(default=10000000)
    currency = models.CharField(max_length=3, choices=CURRENCY_OPTIONS, default='HUF')
    start = models.DateField(default=timezone.now)
    end = models.DateField(default=timezone.now() + timedelta(days=365))
    months_number = models.IntegerField(blank=True, null=True)
    description = models.TextField(blank=True, null=True)
    image = models.ImageField(blank=True, null=True)
    readonly_fields = ['months_number']

    def __str__(self):
        return self.name

    def save(self, *args, **kwargs):
        self.months_number = diff_month(self.end,self.start)
        super(Project, self).save(*args, **kwargs)
```

Figure 29 - Project model implementation

Figure 29 - Project model implementation, provides an insight, what additional steps were needed in the implementation phase, comparing the final solution to the received model description.

Three main further functionalities can be observed in the code. The first one is the definition of currency options, a yet hardcoded but more convenient solution for the end user to define the currency of the whole project. The second one is the automatic calculation of the project length in months, defined in the save function. The last one is the elimination of the possibility of unintended overwrite of the months_number attribute, which always should be calculated in the backend, and never given by the user.

```python
class KPI(models.Model):
    CATEGORY_OPTIONS = (
        ('%', 'Percentage'),
        ('$', 'Currency'),
        ('T', 'Time'),
        ('N', 'Number'),
        ('S', 'Something else')
    )

    name = models.CharField(max_length=255, default="Új KPI")
    goal_value = models.FloatField(default=100.0)
    current_value = models.FloatField(blank=True)
    logo = models.ImageField(upload_to='kpi_logos', blank=True, null=True)
    logo_url = models.TextField(blank=True, null=True)
    category = models.CharField(max_length=1, choices=CATEGORY_OPTIONS, default='N')
    project = models.ForeignKey(Project, on_delete = models.CASCADE, blank=True, null=True)
    percentage = models.FloatField(blank=True, null=True)

    def __str__(self):
        return self.name

    def save(self, *args, **kwargs):
        self.percentage = self.current_value / self.goal_value * 100
        super(KPI, self).save(*args, **kwargs)
```

Figure 30 - KPI model implementation

Figure 30 - KPI model implementation, depicts two additional functionalities. The first is the categorization of KPI types, such as percentage, currency, time, number and other. In that way, the user can create custom, but pre-defined KPI cards during the workflow, and the overall design of these cards can be ensured during the development. The second is the automatic calculation of KPI progress, which is calculated again in the save function.

```python
class Member(models.Model):
    name = models.CharField(max_length=50, default="New member")
    role = models.CharField(max_length=50, default="role")
    project = models.ForeignKey(Project, on_delete=models.CASCADE, blank=True, null=True)

    def __str__(self):
        return self.name
```

Figure 31 - Member model implementation

Last but not least, Figure 31 shows the implementation of the project member model. This model does not have any additional functionality, and was created only because it had been requested exactly, reserved as an interface for future use.

7.1.2 **Model implementation for the Gantt module**

The Gantt module operates with two important models, the Task model, which is the main element of the module and the To-Do model, which has no additional functionality, and therefore is shown but not described in more depths.

Figure 32 - Gantt model implementation, represents the list of attributes, the Gantt module works with. As all attributes play a huge role in the visualization, the exact format of the values were another limitation the software had to meet. For instance, start and end

dates were strictly limited to only one format, therefore additional conversion was needed in both the front-end and the backend.

This model has four additional convenience functions. The first one is the status options, in a range from ‘None’ (not started), then ‘Pending’ and ‘In progress’ to ‘Finished’. ‘Alert’, as the fifth status was added to highlight critical or out-of-date tasks for the user.

The second addition is the so-called slugification, which is a string conversion from free text to hyphen-separated, non-capitalized text. Slugified texts can be used later as variables in HTML or JavaScript codes. This addition operates within the save function.

The third one is the ability of hierarchically organizing tasks into parent and child elements, enabling the user to create hierarchic order among project tasks.

The fourth addition is the automatic calculation of duration, which is similar to how the projects’ length were determined in months.

```python
class Task(models.Model):
    STATUS_OPTIONS = (
        ('P', 'Pending'),
        ('I', 'In progress'),
        ('A', 'Alert'),
        ('F', 'Finished'),
        ('N', 'None')
    )

    name = models.CharField(max_length=255, default='Új tevékenység')
    slug = models.SlugField(blank=True)
    start = models.DateField()
    end = models.DateField()
    duration = models.DurationField(blank=True, null=True)
    progress = models.IntegerField(default=0)
    # assigned_to = models.ForeignKey(User, default=None, on_delete=models.CASCADE)
    all_cost = models.FloatField(blank=True, null=True)
    status = models.CharField(max_length=1, choices=STATUS_OPTIONS, default='N')
    project = models.ForeignKey(Project, on_delete = models.CASCADE, blank=True, null=True)
    parent_id = models.IntegerField(default=0)
    is_child = models.BooleanField(default=False, blank=True, null=True)

    def __str__(self):
        return self.name

    def save(self, *args, **kwargs):
        self.slug = slugify(self.name)
        self.duration = (self.end - self.start)
        if self.parent_id:
            self.is_child = True
        super(Task, self).save(*args, **kwargs)
```

Figure 32 - Gantt model implementation

```python
class Todo(models.Model):
    name = models.CharField(max_length=255,default="Új teendő")
    related_project = models.ForeignKey(Project, on_delete=models.CASCADE)
    related_task = models.ForeignKey(Task, on_delete=models.CASCADE)

    def __str__(self):
        return self.name
```

Figure 33 - To-Do model implementation

7.1.3 **Model implementation for the Budget and Resource management module**

The Budget and Resource management module is built on four model objects, Cost, Resource, Budget and Acquisition. As Figure 34 - Cost model implementation shows, entities in Cost play a huge role in Maze Project. This model is not only connected to the other three models, but it also has a relationship with Task, creating a bridge throughout the two modules. The automatic calculation of net and gross values are added functionalities, as well as the default values for attributes like tax (27%) or multiplier types (piece, hours, etc.).

```python
class Cost(models.Model):
    name = models.CharField(max_length=255, default='Új költség')
    related_task = models.ForeignKey(Task, on_delete=models.CASCADE, blank=True, null=True)
    related_budget = models.ForeignKey(Budget, on_delete=models.CASCADE, blank=True, null=True)
    related_acquisition = models.ForeignKey(Acquisition, on_delete=models.CASCADE, blank=True, null=True)
    resource = models.ForeignKey(Resource,on_delete=models.CASCADE)
    type1 = models.CharField(max_length=255, default="óra")
    multiplier1 = models.FloatField(default=1)
    type2 = models.CharField(max_length=255, blank=True, null=True, default="darab")
    multiplier2 = models.FloatField(default=1, blank=True, null=True)
    type3 = models.CharField(max_length=255, blank=True, null=True, default="-")
    multiplier3 = models.FloatField(default=1, blank=True, null=True)
    net_value = models.FloatField(blank=True, null=True)
    tax = models.FloatField(default=27)
    gross_value = models.FloatField(blank=True, null=True)
    readonly_fields = ["net_value", "gross_value"]

    def save(self, *args, **kwargs):
        self.net_value = self.resource.unit_price * self.multiplier1 * self.multiplier2 * self.multiplier3
        self.gross_value = self.net_value * (1+self.tax*0.01)
        super(Cost, self).save(*args, **kwargs)

    def __str__(self):
        return self.name
```

Figure 34 - Cost model implementation

Figure 35 - Resource model implementation, represents the Resource model, without further additions, as it serves the only purpose to hold a base for cost calculations.

```python
class Resource(models.Model):
    name = models.CharField(max_length=255, default='Új erőforrás')
    unit_price = models.FloatField(default=1000)
    type = models.CharField(max_length=255, default='Ft / óra')
    resource_group = models.CharField(max_length=50, default='Erőforráscsoport X - Y típusú', blank=True, null=True)

    def __str__(self):
        return self.name
```

Figure 35 - Resource model implementation

Unlike Resource, Budget has two additions compared to the initial description. The first is the calculation of real-time difference between budget plan and actual values. The second is the clarification of namespacing, as each budget row can be broken down to five subcategories.

```
class Budget(models.Model):
    name = models.CharField(max_length=255, default='Új költségvetési sor')
    main = models.CharField(max_length=255, default='Fősor')
    sub1 = models.CharField(max_length=255, default='alsor 1')
    sub2 = models.CharField(max_length=255, default='alsor 2')
    sub3 = models.CharField(max_length=255, default='alsor 3')
    sub4 = models.CharField(max_length=255, default='alsor 4')
    sub5 = models.CharField(max_length=255, default='alsor 5')
    description = models.CharField(max_length=255, default='költségvetési sor leírása')
    current_gross = models.FloatField(blank=True, null=True, default=0)
    costplan_gross = models.FloatField(blank=True, null=True, default=0)
    difference = models.FloatField(blank=True, null=True)
    readonly_fields = ['difference', 'current_gross']

    def save(self, *args, **kwargs):
        self.difference = self.costplan_gross - self.current_gross
        self.name = self.main + " - " + self.sub1
        super(Budget, self).save(*args, **kwargs)

    def __str__(self):
        return (self.name)
```

Figure 36 - Budget model implementation

Finally, Acquisition has three additional functionalities. Two of them are pre-defined list of contract and status options, the third one is the automatic calculation of gross value.

```
class Acquisition(models.Model):
    CONTRACT_OPTIONS = (
        ('A', 'adásvételi'),
        ('M', 'megbízási'),
        ('S', 'szállítási'),
        ('F', 'fuvarozási'),
        ('K', 'kölcsön'),
        ('V', 'vállalkozói'),
        ('B', 'bérleti'),
        ('L', 'licensz'),
        ('H', 'haszonbérleti'),
        ('T', 'biztosítási')
    )

    STATUS_OPTIONS = (
        ('P', 'terv'),
        ('F', 'tény')
    )

    name = models.CharField(max_length=255, default='Új beszerzés')
    type = models.CharField(max_length=255, default='szolgáltatás')
    method = models.CharField(max_length=255, default='meghívásos')
    description = models.CharField(max_length=255, default='szolgálatás vásárlása')
    start_date = models.DateField(default=timezone.now)
    result_date = models.DateField(default=timezone.now() + timedelta(days=60))
    contract_type = models.CharField(max_length=1, choices=CONTRACT_OPTIONS, default='A')
    net_value = models.FloatField(blank=True, null=True)
    tax = models.FloatField(default=27)
    gross_value = models.FloatField(blank=True, null=True)
    costplan_gross = models.FloatField(blank=True, null=True)
    status = models.CharField(max_length=1, choices=STATUS_OPTIONS, default='P')
    readonly_fields = ["gross_value"]

    def save(self, *args, **kwargs):
        self.gross_value = self.net_value * (1+self.tax*0.01)
        super(Acquisition, self).save(*args, **kwargs)

    def __str__(self):
        return self.name
```

Figure 37 - Acquisition model implementation

7.2 Business layer implementations

As it was described, the Business layer connects the other layers into the dataflow. Therefore, almost all processes are targeting request handling from the front-end, transformed into data requests to the database, then data-handling from the backend, ending with a transformed data sent back to the user. These processes are built on three main types of Business layer tools: forms, views and urls. Forms create the vessel and shape for data transport, usually in the form of model forms. Views handle the requests and data coming from forms. And urls are the endpoints of all communications, pointing from one view to another. In this section, these three tools and their most common implementations are described with examples from the source code.

7.2.1 Forms in the Business layer

Figure 38 - Form implementations for Gantt module, shows all implemented form models for the Gantt module. Although it only represents form entities from this specific module, the syntactical and structural build-up is similar for the other modules as well. It differentiates three types of forms: simple forms, model forms and form sets.

Simple forms, like 'ProjectRedirect' form contain pre-defined, but custom fields, like name, id or category. In validation, the only rule Django examines is whether a field was mandatory or not, and what value type was associated with it. Model forms however, are connected to models like Project. For instance, 'SaveProject' form uses the Project class for meta-definition, creating there a reference point. It does not mean, that it cannot be supplemented with further fields, but during validation, it will be always compared to the Project model as well, along with all the model's attributes. Finally, form sets are, or at least could be, the combination of two, as these sets can be put together by any of the two former categories.

One of the main advantages of forms is the built-in validation process before submission. Submissions require not just token validation, provided by Django itself, but also data validation, to make sure, only the requested data was provided.

Forms, as Figure 38 showcases, can also be used to flag user intentions like delete inside the forms. That way, the form incorporates that piece of information in itself, making views' data handling job way more comprehendible.

```python
from django import forms
from . import models
from django.forms.formsets import BaseFormSet

class SaveData(forms.ModelForm):
    id = forms.IntegerField(required=False)
    to_delete = forms.BooleanField(required=False)

    start = forms.DateField(
        widget= forms.DateInput(format='%Y-%m-%d'),
        input_formats=('%Y-%m-%d',)
    )
    end = forms.DateField(
        widget= forms.DateInput(format='%Y-%m-%d'),
        input_formats=('%Y-%m-%d',)
    )


    class Meta:
        model = models.Task
        fields = ['name','start','end','duration','progress','status']

class ProjectRedirect(forms.Form):
    project_id = forms.CharField(max_length=3)

class SaveProject(forms.ModelForm):
    id = forms.IntegerField(required=False)
    to_delete = forms.BooleanField(required=False)

    class Meta:
        model = models.Project
        fields = ['name', 'short_name', 'budget','currency', 'start', 'end', 'months_number', 'description']

class SaveTodo(forms.ModelForm):
    id = forms.IntegerField(required=False)
    to_delete = forms.BooleanField(required=False)

    class Meta:
        model = models.Todo
        fields = ['name', 'related_task', 'related_project']
```

Figure 38 - Form implementations for Gantt module

7.2.2 **Views in the Business layer**

Views can be categorized by their main goals in four categories: data listing views, data modification views, data calculation views and mass data update views.

Figure 39 - Data listing views showcases a basic listing method for project members. As the GET request arrives, the view gets the active project and the related members from the backend and the session, defines the existing roles for categorization, then returns the request with a page template, rendering the received data in the form of two object queries. Simple data listing views only make short queries to the database, usually with filtering or sorting the received data, but not modifying it before sending it to the front-end.

```python
def project_members(request):
    p_id = request.session.get('active_project')
    project = Project.objects.get(id=p_id)
    members = Member.objects.filter(project=project)
    roles = []
    for m in members:
        if m.role not in roles:
            roles.append(m.role)
    return render(request, 'gantt/project_members.html',{'members':members, 'roles':roles})
```

Figure 39 - Data listing views

Data modification views however, are built to achieve that goal in the first place. As Figure 40 displays, these views differentiate GET and POST requests. While in GET method, they are similar to the listing views, in the POST branch data validation and transformation is processed with exception or success handling. In case of the latter scenario, the successful process is handed over to another, listing type view via an HTTP redirect, therefore it keeps data transformation in one place, and data listing in another.

```
def todo(request):
    todos = Todo.objects.all()
    tasks = Task.objects.all()
    active_project_id = request.session.get('active_project')
    active_project = Project.objects.get(id=active_project_id)
    form = forms.SaveTodo()
    message = "GET method"
    if request.method == "POST":
        form = SaveTodo(request.POST)
        if form.is_valid():
            todo_pk = form.cleaned_data['id']
            try:
                todo = Todo.objects.get(pk=todo_pk)
                todo.name = form.cleaned_data['name']
                related_task_pk = form.cleaned_data['related_task']
                todo.related_task = Task.objects.get(pk=related_task_pk)
                todo.related_project = active_project
                to_delete = form.cleaned_data['to_delete']
                if not to_delete:
                    todo.save()
                    message = "save success"
                else:
                    todo.delete()
                    message = "delete success"
            except:
                todo = form.save(commit=False)
                name = form.cleaned_data['name']
                related_task = form.cleaned_data['related_task']
                related_project = active_project
                todo = Todo.objects.create(name=name, related_project=related_project, related_task=related_task)
                todo.save()
            return HttpResponseRedirect('todo_update_success')
        else:
            message = form.errors
    todos = Todo.objects.all()
    form = forms.SaveTodo()
    return render(request, 'gantt/todo.html', {'todos':todos, 'form':form, 'message':message, 'tasks':tasks, 'active_project':active_project})
```

Figure 40 - Data modification views

Although most of the business logic is wired in the construction of the Business layer itself, some important calculations are happening in the data calculation views. A shorter example is presented by Figure 41, where total costs are calculated based on related tasks, right before data is sent back to the front-end in the Expenses submodule.

```
def cost_update_success(request):
    form = forms.SaveCost()
    costs = Cost.objects.all()
    tasks = Task.objects.all()
    grouped_costs = []
    for task in tasks:
        grouped_costs.append(0)
    for i in range(0,len(tasks)):
        for c in costs:
            if c.related_task == tasks[i]:
                grouped_costs[i]+=c.gross_value
    message = "update successful"
    total_project_cost = np.sum(grouped_costs)
    return render(request, 'costs/costs_table.html',{'costs':costs, 'tasks':tasks, 'form':form,
    'message':message, 'grouped_costs':grouped_costs, 'total_project_cost':total_project_cost})
```

Figure 41 - Data calculation views

The last category contains mass data update views. These views had been created to enable the software to accelerate the performance on the front-end, while creating more convenient user experience and easy-to-use environment. As Figure 42 depicts below, many Acquisitions had been sent back via a request, and they are dealt with simultaneously in the view. Even though it is not so far in construction from the simple data modification view, the fact that it handles several model instances during the workflow makes it a different challenge the software needs to face.

```
def save_many_acquisition(request):
    if request.method == "POST":
        form = SaveAcquisitionMany(request.POST)
        if form.is_valid():
            direction = form.cleaned_data['direction']
            id_list = form.cleaned_data['ids'].split(",")
            new_related_acquisition = form.cleaned_data['new_acquisition']
            for i in id_list:
                cost = Cost.objects.get(pk=int(i))
                if direction == "right":
                    cost.related_acquisition = None
                    cost.save()
                if direction == "left":
                    cost.related_acquisition = Acquisition.objects.get(pk=int(new_related_acquisition))
                    cost.save()
        else:
            message = form.errors
            form = SaveAcquisition()
            acquisitions = Acquisition.objects.order_by('name')
            costs = Cost.objects.all()
            return render(request, 'costs/acquisition.html', {'message':message,'form':form,'acquisitions':acquisitions,'costs':costs})
    return HttpResponseRedirect('acquisition_update_success')
```

Figure 42 - Mass data update views

7.2.3 **Urls in the Business layer**

The last one of the three tools is the usage of urls. These links are the connectors among views, each responsible for different scenarios and user behaviors. As Figure 43 presents, the Budget and Resource management module alone has more than ten different urls, each to handle an event like listing expenses, saving acquisitions or updating budget plans. Even though other modules may have different urls, this figure efficiently represents the idea behind the position and role they have in the Business layer workflow.

```
from django.urls import path
from . import views

app_name = 'costs'

urlpatterns = [
    path('costs_table', views.costs_table, name='costs_table'),
    path('acquisition', views.acquisition, name='acquisition'),
    path('acquisition_update_success', views.acquisition_update_success, name='acquisition_update_success'),
    path('costs_pairing', views.costs_pairing, name='costs_pairing'),
    path('cost_update_success', views.cost_update_success, name='cost_update_success'),
    path('save_many_budget', views.save_many_budget, name='save_many_budget'),
    path('save_many_acquisition', views.save_many_acquisition, name='save_many_acquisition'),
    path('budget_update_success', views.budget_update_success, name='budget_update_success'),
    path('resource_update_success', views.resource_update_success, name='resource_update_success'),
    path('resource_table', views.resource_table, name='resource_table'),
    path('', views.costs_table, name='costs-home')
]
```

Figure 43 - Urls in the Business layer implementation

8 Testing and evaluation

8.1 Front-end testing

Performance testing is one of the most important part of review from the user experience and the business success aspect. Slow and continuously loading pages can easily evolve into low conversions and platform leaving in a medium time horizon. For that reason, it is important that all segments of the page are tested and audited via measures like performance, quality and correctness.

Google's Lighthouse is one of the most popular automated testing tools, integrated within the browser. Lighthouse runs several tests against the page and generates a performance report with key indicators to reveal inner weaknesses and directions to improve. In this section, Maze Project is tested with Lighthouse. The tool examination can be categorized into four groups: performance, accessibility, best practices and SEO.

Each category is calculated separately, regarding the used metrics. Performance result is based on metrics like First Contentful Paint, Speed Index or Cumulative Layout Shift. Accessibility takes measures like Navigation, ARIA, Names and labels into account. Best practices examines Trust and Safety, User Experience or Detected JS libraries. Finally SEO considers metrics like Mobile Friendliness, Content BPs or Crawling and Indexing. While Lighthouse provides a list of details and observations as Figure 45 shows, it also provides an overall score for each page, displayed in Figure 44.

Figure 44 - Summarized metrics provided by Lighthouse

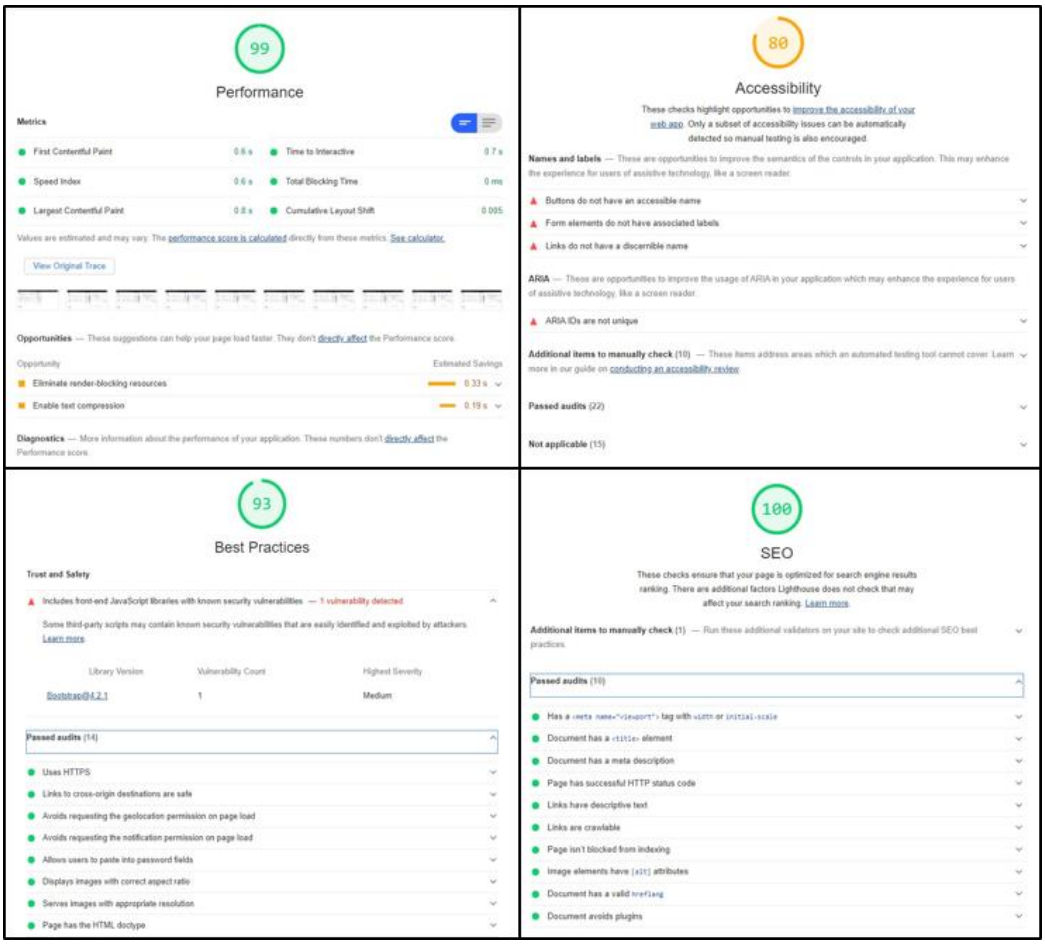


Figure 45 - Detailed metrics provided by Lighthouse

As the provided detailed descriptions show, aside from the average result of the metrics, a long list of further improvements are presented for the developer. Also the list of passed audits are given, to ensure the user about objective observations. Although it can be reassuring that a page scores high points in a category, it is recommended to work on the listed possibilities of improvements, as a few percent of change can deeply affect user experience and user engagement.

To get an overall picture, every existing page on Maze Project within the models had been tested against the automated tool. The results of Lighthouse testing are presented below.

Page name	Performance	Accessibility	Best practices	SEO
Home page	78	85	93	100
KPI	100	83	93	100
Project data	100	90	93	100
Project members	100	81	93	100
Average of PA module	**94,5**	**84,75**	**93**	**100**
Gantt chart	99	80	93	100
To Do list	100	80	93	100
Average of Gantt module	**99,5**	**80**	**93**	**100**
Expenses	100	77	93	100
Resources	100	79	93	100
Budget plan	100	80	93	100
Acquisition	100	80	93	100
Average of BRM module	**100**	**79**	**93**	**100**
Report	100	83	93	100
Dashboard	66	83	86	100
Average of RD module	**83**	**83**	**89,5**	**100**
Total average of Maze Project	**95**	**82**	**92**	**100**

Table 4 – Lighthouse testing results for modules and overall

As it can be seen, most of the pages performed exceptionally on the test, scoring a perfect 100 points in several aspects. However, there are crucial points of the website, where further development will be needed. The two weakest sites are the Home page and the Dashboard page, both scored in the medium (yellow) zone, and have a long list of possible improvements. The Dashboard page, with its visual-based structure, obviously will never have the same result, as the text-based Report page, however the listed fixes are stacked in the next development package. What is remarkable all-in-all, is the overall

score of 100 percent on SEO, which means that the site is fully prepared to compete in for the interest of search engines.

All things considered, Lighthouse has revealed some issues to fix, but provided a reassuring total value on Maze Project's current performance.

8.2 Backend testing

Regarding the backend, there are several ways to perform testing. From unit testing, through integration and functionality testing, the possibilities are almost endless. However, this Thesis work concentrates only on those elements of the backend that had been created within its frames, therefore the thorough testing of each unit is presented here. The main elements of the backend, as it was described in the theoretical part before, are models, forms and views, and this section concentrates on the testing process of these units.

8.2.1 Model testing

Model testing targets the elimination of implementation flaws within model declarations and database entity creations. Therefore, in a test environment, test entities can be created by automatic tests, and can compare the results with the expected outcomes. At the end of the test, the result is a Boolean value, whether the given criteria were meet or not.

Figure 46 shows the testing definitions of Project model, which similarly to other model tests, had been created to ensure object entities can be created and saved properly to the database.

```
class ProjectTestCase(TestCase):
    def setUp(self):
        Project.objects.create(name="Random project",
            short_name="rnd-proj",budget=1000,currency="USD",
            description="Random description")
        Project.objects.create(name="Another project",
            short_name="anth-proj",budget=10000000,currency="HUF",
            description="Random description 2")

    def test_has_value(self):
        first_proj = Project.objects.get(name="Random project")
        second_proj = Project.objects.get(name="Another project")
        self.assertEqual(first_proj.budget,1000)
        self.assertEqual(second_proj.budget,10000000)
```

Figure 46 - Model testing in practice

In case of a successful test, the message ‘OK’ is prompted to the command line, as Figure 47 displays. Although all models ought to be tested during development, in this section only this example is presented, due to the similarity among other tests.

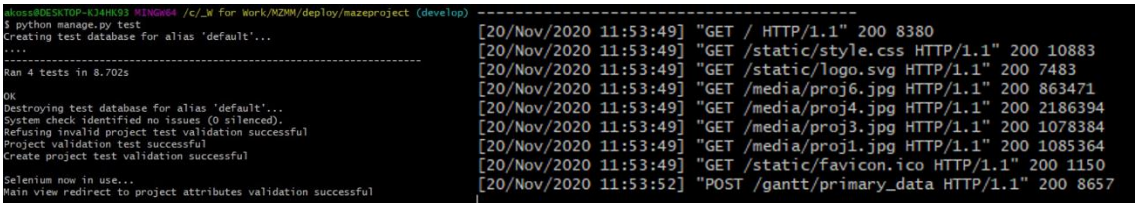


Figure 47 - Successful testing result in command line

8.2.2 **Form testing**

Form testing’s most important goal is to ensure that the user cannot input unintended data type or format, and that the form validation actually prohibits users from object miscreation. Even though all data types, input fields and possibilities can be tested, the logic is all the same everywhere: accept correct, but only correct input data, and deny all others.

```
class CreateProjectTests(TestCase):
    def test_valid_form(self):
        p = Project.objects.create(name="Random project",
            short_name="rnd-proj",budget=1000,currency="USD",
            description="Random description")
        data = {
            'name':p.name,
            'short_name':p.short_name,
            'budget':p.budget,
            'currency':p.currency,
            'start': p.start,
            'end':p.end
        }
        form = SaveProject(data=data)
        self.assertTrue(form.is_valid())
```

Figure 48 - Form validation with a successful branch

Figure 48 displays a form validation with a successful branch, where the created Project instance successfully gets validated in the test environment. It is reassuring to find that even though start and end dates are not explicitly given to the instance, the Project object creation works successfully with given default data.

On the other hand, Figure 49 presents the parallel failure branch, where only one difference can be observed, the change of currency. As the currency attribute is limited

to a given list of selected values, the input of improper data value causes a denied validation process.

```
def test_invalid_form(self):
    p = Project.objects.create(name="Random project",
        short_name="rnd-proj",budget=1000,currency="JYN",
        description="Random description")
    data = {
        'name':p.name,
        'short_name':p.short_name,
        'budget':p.budget,
        'currency':p.currency,
        'start': p.start,
        'end':p.end
    }
    form = SaveProject(data=data)
    self.assertFalse(form.is_valid())
```

Figure 49 - Form validation with the failure branch

8.2.3 **View testing**

For testing views, functionality is the main objective to focus on. Figure 50 showcases a typical view test with the use of Selenium, where a WebDriver opens Maze Project, selects a project from the main page and navigates to another site. The result returned by the client is compared to the expected values to decide the success of the view.

```
class ViewTestCase(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Chrome()

    def test_main_view(self):
        driver = self.driver
        driver.get("http://localhost:8000/")
        driver.maximize_window()
        elem = driver.find_element_by_id("project-form-1")
        elem.click()
        driver.implicitly_wait(5)
        validation_url = "http://localhost:8000/gantt/primary_data"
        self.assertEqual(driver.current_url,validation_url)
        print('Main view redirect to project attributes validation successful')
        driver.quit()

if __name__ == "__main__":
    unittest.main()
```

Figure 50 - View testing with Selenium

8.3 Evaluation

In this final section of testing and evaluation, the whole process of development is overviewed in the light of the initial, provided criteria. To be able to generally have an impression on the success of development procedure, it is essential to have a look at the initial requirements, and evaluate the final result by comparing the two to each other.

Table 5 summarizes the evaluation process, giving room for each criteria of the requirements in a single row. Beside the requirements, the presented implemented results are described shortly and a final evaluation is provided.

Front-end evaluation		
Initial requirement	Implemented Result	Evaluation
Build the software on the latest version of the Front-end triangle (HTML5, CSS3, JS)	The software uses HTML5, CSS3 and JS for front-end development, the current latest versions	Criteria fulfilled
Use Django's form communication between Front-end and Backend	Data communication between the front-end and the backend is based on forms	Criteria fulfilled
Use a unique library for the user interface that enhance visualization, preferably Plotly.JS	For the implementation of visuals and charts, Plotly.JS library was used	Criteria fulfilled
For wireframes, use the latest version of a popular free Front-end element library, preferably Bootstrap 4.0 or higher	Bootstrap 4.1, the current latest version serves as template skeleton	Criteria fulfilled
Do not base the application on any JavaScript framework (like React, Vue or Angular)	None of the listed or other JavaScript frameworks had been used during development	Criteria fulfilled
Use of JQuery is preferred, but only 3.3 or higher	JQuery 3.3.1 was used for AJAX and UX enhancing functionalities only	Criteria fulfilled
Backend evaluation		
Initial requirement	Implemented Result	Evaluation
According to the latest trends, build the Backend on Python, instead of PHP	The implemented Django framework is a Python-based backend providing tool	Criteria fulfilled
Use Django for creating the environment, as it can easily work with additional data-processing software	As showcased, Django provides the backend, and several Python libraries had been added during the development	Criteria fulfilled
For platform independency, the application should be run in a container, preferably Docker	The current version of the software runs on a private server, but in a Docker container	Criteria fulfilled
Publishing should be done via a repository manager that can solve version control and	GitLab had been used for repository management throughout the development	Criteria fulfilled

continuous deployment to the server, preferably GitLab	process, deployment is provided via CD/CI technology	

Table 5 - Evaluating initial development requirements

All things considered, the final software meets the initial requirements, therefore the development can be considered successful.

9 Future development possibilities

It is clear, that the current version of Maze Project can be improved in several aspects. Although all three presented layers are functioning, in many cases these functionalities are limited due to the fact that the project is still in the early stages of market testing. Having a working version of the software, however, can be a true advantage for further possibilities, such as upgrading, differentiating starter and premium versions or even to involve investors. In the following sections, a few ideas are presented about future development possibilities, categorized by the area where development would have the greatest impact.

9.1 User interface development possibilities

In this section, two interesting user interface ideas are presented. The first and more important is the implementation of Lock mode. Lock mode would be a system-wide possibility for the project manager to play around the numbers, to experiment. When in Lock mode, the user could make temporary adjustments in any area on the site, without the fear of modifying real data. Although charts, reports and forecasts would show the difference, the lock on the site would not save any modifications to the database. Therefore, the user could have a greater freedom to use the site for testing scenarios or new ideas. To be able to implement that, temporary duplication of the database system would be necessary, as well as wider use of sessions throughout the page.

The second idea for possible development is Dark mode. As recent trends show, most of the popular platforms, regardless to be web-based or not, are implementing this power-saving, eye-friendly design in their product, and users are fond of it. With just one click, users can switch between light colors or darks, including backgrounds, logos or visuals. To achieve that, mostly just JavaScript and CSS modifications would be necessary, however a successful Dark mode should be planned carefully and with the help of a graphic designer in order to create harmony within the page. This latter would be crucial, as the developer team lacks a designer right now.

9.2 Possible business logic improvements

In case of business logic, a few ideas had been already presented in section 3.4. The implementation of these two more modules could be useful additions, probably as part of the premium pack or later as a total update.

Change and risk management module would create an option for several alternative project plans, a little bit like Lock mode, but supported with tools for risk analysis and handling, with visually detailed scenarios. Change management compared to Lock mode, focuses on changes and project activities globally, as all modifications would be stored in this module, based on version numbers used for version control. The manager would be able to mark processes with status messages. To be able to develop that module, the database should be prepared for alternative plans and scenario handling. Modified data structure and bigger project scope would be necessary as well, therefore the implementation of this module requires lot of effort with planned changes. The second part of the module idea is Risk management.. Analytical and predictive methods like Critical Path Method (CPM) would be implemented, with user experience enhancement features like critical path highlight.

Administration and permissions module, as the second possible future module, could also be part of the premium pack, as some useful tools would widen the possibilities of the end user in analysis. Although it is not directly connected to the project, but supports the software management. Being responsible for the whole project's database, the project manager could overview and easily handle the multidimensional permission system that is typical in a project team. As for feasibility, an important key for handling multiple projects is the aforementioned change of database system and global session handling. To make it possible to have more than one project or project team, while having the same or totally different permission system, a major upgrade would be necessary.

9.3 Database development possibilities

As the sections above highlighted, several additional modifications would be crucial in the database system in order to successfully move further in enhancing the application. Regarding the database, data maintenance and regular safety backups would improve the user experience in short-term already. User satisfaction often relies on issue handling and recoveries, therefore this development would be highly rewarding.

10 Summary

In this Thesis work, Maze Project – a new tool for modern day project managers, was presented. Starting from the motivation behind the ideas, through the business and technological planning, to the implementation phase a whole development project was showcased. This summary concludes the findings and experiences of each section in this Thesis work.

Project Management

One of the first steps was to review the relevant literature on project management paradigms and software tools. The basics of project management, and the theorem of process groups were presented, then the idea of data-driven management by Vanhoucke was introduced. After analyzing the main functions offered by different project management software tools, at the end of the section an evaluation of the found results was presented from the point of view of this current project's goals.

Presenting Maze Project

In this section, the motivation and vision of the Maze Project was discussed in details, showcasing an innovative project management and administration software solution. Maze Project's core idea is built on separated database structure, supported by business intelligence, and provides a strong budget-oriented project control mechanism to its users, the project managers. The competitive advantages and the core modules were described in details, with additional module ideas alongside.

Technological background

For a short bypass, the fourth section concentrated on the technological background of web development. Both requirements and used technologies were presented, displaying front-end technologies like HTML, CSS and JavaScript, and backend technologies like Django framework or Docker.

System requirements

The fifth section presented the system's structural overview, explaining the main modules of Maze Project, and how these descriptions were used as a blueprint later, for the implementation phase. The term layer was redefined for this Thesis, and three main layers were distinguished: Presentation, Business logic and Database layer.

Front-end implementation

The sixth section focused on the implementation of the front-end, or as discussed, on the Presentation layer. One by one, all modules had been presented with figures of the finished pages. Responsiveness, one of the key elements of the software requirements was also showcased in details, with exact changes pointed out in the user interface.

Backend implementation

Accepting the premise that a solid basis is crucial for building a house, it is clear why the implementation of the backend defines the course of the whole platforms success. For that very reason, both the Business logic and the Database layer were presented in detail, to enable the thorough understanding of the platform's underlying structural system. After the created database models were presented, the additional, mostly user experience enhancing functionalities were described with illustrations. At the end of the section, the significance of business layer elements was discussed, with concrete examples for further extension.

Testing and evaluation

Although the implementation phase finished for the platform, one of the most crucial steps in development is always testing. To reveal underlying flaws in both design and programming, the software was tested against several popular modern testing methods. With Google Lighthouse and Selenium, a thorough testing was executed on the written code regarding both the front-end and the backend. The results of the testing was presented, as well as an overall evaluation of the site.

Future development possibilities

Regardless of finishing the first working version of the software, future possibilities is always a topic that should be discussed. In this last section, additional functionalities, possible future modules and existing boundaries were presented, alongside with the idea of a starter and a premium version of the software in the future.

References

[1] Vanhoucke, M. (2018): *The Data-Driven Project Manager: A Statistical Battle Against Project Obstacles.* Apress, 2018.

[2] Project Management Institute, Inc. (2013): *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)—Fifth Edition.* PMI, 2013.

[3] Kerzner, Harold (2017): *Project management: a systems approach to planning, scheduling, and controlling.* John Wiley & Sons, 2017.

[4] Beck, K. et al. (2001): *Manifesto for Agile Software Development.* Downloaded from: https://www.agilemanifesto.org (Last download: 12 May, 2020).

[5] Project Management Institute, Inc. (2018): *Agile Practice Guide*. PMI, 2018.

[6] Vanhoucke, M. (2013): Project Management Using Dynamic Scheduling: Baseling Scheduling, Risk Analysis & Project Control. In: *The Measurable News.* 2013, Issue 2.

[7] Vanhoucke, M. (2018), Interviewed by Nader Rad on YouTube on 13 July, 2018. Available at: https://www.youtube.com/watch?v=mDkR7r5J3Xc (Accessed: 12 May, 2020)

[8] Özkan, D. – Mishra, A. (2019): Agile Project Management Tools: A Brief Comprative View. In: *Cybernetics and Information Technologies.* 2019, Vol 4, 17-25.

[9] CAPTERRA (2020). Available at: https://www.capterra.com. (Last download: 12 May, 2020)

[10] CAPTERRA (2020): *Best Project Management Software – 2020 Reviews of the Most Populat Tools & Systems.* Available at: https://www.capterra.com/project-management-software/#top-20 (Downloaded: 12 May, 2020)

[11] ASANA (2020). Available at: https://www.asana.com (Last downloaded: 12 May, 2020)

[12] JIRA (2020). Available at: https://www.atlassian.com/software/jira (Last downloaded: 12 May, 2020)

[13] EASYPROJECT (2020): Available at: https://www.easyproject.com (Last downloaded: 12 May, 2020)

[14] Turban, E. – Volonino, L. – Wood, G. R. (2016): *Information Technology for Management – 10th Edition.* John Wiley & Sons, 2016.

[15] GitLab (2020). Availbale at: https://about.gitlab.com/what-is-gitlab/ (Accessed: 19 May, 2020)

[16] Docker (2020). Available at: https://www.docker.com/resources/what-container (Accessed: 19 May, 2020)

[17] Briscoe, N. (2000): *Understanding the OSI 7-Layer Model*. In: *PC Network Advisor*. 2000, Issue 120, 13.16.