A Ticket System that Actually Works

Group 22

Project Report 3

FOLKE&**PARTNERS**

Carl Bentzer (cbentzer)
Erik Betzholtz (erikbet)
Josefina Häkkinen (hakki)
Umut Sina Kenes (kenes)
Marcus Nilszén (nilszen)
Adrian Salamon (asalamon)
Klara Sandström (klarasan)
Harry Thulin (hadth)

Mentored by Anoud Alshnakat

April 17, 2024

Contents

List of Figures

List of Tables

1 Introduction

In the course *DD1369 Software Engineering in Project Form* our project group was coupled with our stakeholders Folke&Partners. The objective of the project was to design and develop a ticket system that actually works, without many of the problems present in existing ticket systems. The issues in these ticket systems include many unnecessary steps in the transaction process and poor scalability, leading to users losing their tickets during popular releases. To combat this, our system was designed to enable a shorter transaction flow that allows users to obtain tickets to events quicker and more seamlessly. For this project, we started from a blank slate and developed the whole system from scratch.

Because of this, our initial requirements that we agreed on together with our stakeholders were very basic. Based on these requirements we developed a Minimal Viable Product (MVP) consisting of a responsive web user interface, a database, and an API that enabled users to purchase tickets for a single event. The system design and architecture are described in detail in section 5 and section 7, while the tools used can be found in section 6. The MVP could then be iterated upon by developing an Alpha version with more functionality and features, such as having multiple events, being able to log in, seeing the tickets you have previously bought, and providing events with multiple ticket prices and release times. The improvements chosen for the new version were decided after examining existing ticket systems and creating a Competitor Analysis.

The project timeline was mapped out and planned for using a Gantt chart, as can be seen in section 9. When developing our system we followed the Scrum methodology, where we compiled a list of short tasks to be completed within a sprint which has the time frame of about two weeks. Each week two sprint meetings were held where we updated each other of our progress. Before we started the development process we also compiled a list of possible risks and the impact they would have on the project outcome which can be seen in section 11. This was done in order to sufficiently plan for how to minimize any negative consequences they would have. The risks mainly consisted of problems within the team, but also issues with new code and software.

After finishing development of the Alpha version, our website was tested on our user group by asking them to complete short tasks on our website. Full description of the test methodology and results can be seen in section 13. When analyzing the test results, we considered both the time it took to complete test tasks, as well and thoughts and opinions expressed by our testers. In general, we found that the system was intuitive to use and had a consistent design. We also identified a few usability issues that were fixed before completing the project. We consider the project to be successful in the sense that the final result fulfilled all of our initial requirements, and that we as a group learned and developed over the course of the project.

2 Project Charter - A ticket system that actually works

2.1 Purpose Statement

Using a ticket system today usually means long payment processes that are fragile specifically during popular event releases. Anyone who has used the services of the most popular ticket sellers is probably familiar with the experience of somehow losing their tickets in the process. The aim of this project is therefore to create a system that provides a faster and less complex payment process as well as improved performance during high load. Through evaluation of different competitors on the market, it will also enable the creation of a product that contains a more conscious design

compared to other systems existing today. Likewise, unreached target audiences in need of a ticket system will be able to be identified. Moreover, the team members carrying out the project will learn a lot about project management and the technologies used for the project, this is described further down in the Benefits subsection.

2.2 Objectives

The objective of the project is to create a user-friendly service allowing people to purchase tickets to various events. The purchasing process should require as few steps as possible to make it simpler and faster for the user. The system should also be able to handle a surge of users during popular event releases so that the website doesn't go offline. If possible, the system should also include a fair second hand market, making it possible for users to sell tickets to other users. This market should ideally protect against inflated prices.

2.3 Constraints

The project is mainly constrained by the timespan of the DD1369 course and the amount of time each team member can put in each week, depending on fluctuating workload in other courses. Therefore sprint tasks are decided and appointed together at the beginning of each sprint in order to adapt to the needs of each team member. Also, standup meetings are only held twice a week and therefore if any urgent decisions need to be made it might require setting up extra meetings. Furthermore, the scope of the project will constrain the amount of time that the team members can put on various parts of the project. It is important that the MVP is completed and as a result further ideas are not the top-priority.

Moreover, the project will be carried out during a global pandemic limiting our ability to communicate efficiently both within the group, with our mentor, and with stakeholders. If there is a lack of communication this might have an effect on the quality of the project or certain aspects of it.

2.4 Benefits

There are many perks of accomplishing the project, for instance the team members will learn how software engineering projects operate and function. This includes project management and software engineering methods. They will also learn about the different system design phases, including identifying requirements, modeling and adapting these processes to new findings during the project cycle. Moreover, the team will gain experience using different technologies used for the project. This will favor the team members in future studies, work, and projects. Also worth mentioning is that they will gain more experience communicating and coordinating with multiple parties, such as collaborating with stakeholders, making internal decisions, and presenting the progress.

Furthermore, the stakeholder company will gain contacts and recognition amongst computer science students, which might benefit future hiring and they will also receive a ticket system they can pick up for further development.

2.5 Risks

The majority of the risks revolve around complications within the team, for example team members being unable to put in enough work. New code or software can also cause problems such as bugs. The solution is in general to keep communication open and effective. A summation of risks, severity and possible measures can be seen in section 11.

2.6 Roles

Since we are using the scrum method as explained in section 10, we will have a scrum master, Adrian and a project owner, Klara. The primary responsibility for communicating our progress to the stakeholders is given to Carl. We have decided against assigning specific roles or responsibilities for frontend/backend development. Instead, freedom to choose different tasks during the project depending on experience and interest is prioritized. This enables us to have an efficient workflow and gain experience with various aspects of software development, as well as increasing the overall understanding and insight into the project.

2.7 Stakeholders

Our stakeholders are Folke&Partners, where Mattias Folke and David Masko will set the goals and influence the project's timeline. Their input and requirements will be considered during the whole development process. However, these ideas are limited by the course objectives in DD1369 and the advice of teachers and our project mentor.

2.8 Target audience

The target audience for the product are both people who attend events or places where tickets are required, and those who need to sell tickets for those events. We are focusing mainly on smaller events, since it is more likely that we would find a market for our product in smaller businesses. One example might be events organized by student associations at KTH.

2.9 Timeline

Project time frame: 17/01/22 - 10/05/22.

Below are the most significant milestones of the project, in regards to course related deadlines. For a detailed project plan, see section 9.

- First submission of project report. (17th of January)
- First seminar review. (3rd of February)
- MVP finished. (2nd of March)
- Second submission of project report. (14th of March)
- Half-time presentation. (21st of March)
- Seminar presentation. (29th of March)
- Second seminar review. (14th of April)
- Third submission of project report. (2nd of May)
- Finished product. (9th of May)
- Final presentation of project. (10th of May)

2.10 Deliverables

- A responsive user interface.
- A database to store necessary information about users and their tickets.
- An API to communicate with the database.
- Project report, three iterations.
- Seminar, half-time and final presentations.

3 Requirements

3.1 Business Requirements

In the first stage of the project an MVP was developed that could later be expanded upon by adding different features. This product had few requirements, focusing mainly on getting the frontend, backend and database in place and integrated. There needed to be a user interface designed mainly for smart phones but usable on desktop as well (BR1). This interface needed to display sufficient descriptions and attributes for events (BR2), which included title, location, date, description, price and a maximum amount of tickets. Finally, a user needed to be able to purchase tickets (BR3). These business requirements came mainly from our stakeholders, and were fulfilled on March 2nd when the MVP was finished.

For further developments, new requirements were formulated. The main goal of this stage is to create a more useful application with multiple new features. This version is referred to as the Alpha version.

The system should in this stage be able to host multiple events and display them so that users can find events they are interested in (BR5). Events should be able to have different types of tickets with different prices (BR7). Events should also be able to have specific release times, where users are only able to buy tickets after a certain time (BR8). Tickets should be delivered as e-mails (BR9), and a user should be able to see what events they purchased tickets to and how many tickets they bought in the system (BR6). This also requires that users should be able to log in (BR4). All business requirements can be seen in Table 1.

ID	Description
BR1	Should exist a user interface, designed both for smart phones and desktop
BR2	Events have detailed descriptions and attributes
BR3	A user should be able to buy tickets
BR4	A user should be able to log in
BR5	A user should be able to find multiple events
BR6	A user should be able to see their own tickets
BR7	Events should be able to have different types of tickets
BR8	Event tickets can have a release time
BR9	Tickets can be delivered by email

Table 1: Business requirements

3.2 Functional and Non-Functional Requirements

Each business requirement has functional and non-functional requirements related to them. The functional requirements are related to the functionality and behaviour of the system, while the non-functional on the other hand are related to qualitative aspects of the system, such as usability, aesthetics, and reliability. These two types of requirements are more detailed and specific than the business requirements which makes it easier to test whether they have been fulfilled or not. In Table 2 and Table 3 the functional and non-functional requirements are showed respectively, as well as their priority. The priorities for the requirements are based on how important they are for the system and range from low, medium, to high. Priorities for specific requirements were determined by us in collaboration with the stakeholders.

FR ID	Description	Priority
FR1	An event has: title, description, location, price, time	High
FR2	A user can buy 1 ticket	High
FR3	A user can buy multiple tickets	High
FR4	A user should not be able to buy negative amount of tickets	Medium
FR5	A user should be able to navigate on the website on a phone	High
FR6	A user should be able to navigate on the website on a desktop	Medium
FR7	A user can log in via Google	Low
FR8	Failed Google authentication means that the user can not log in	Low
FR9	A user can see a list of events	Medium
FR10	A user can click one event to see more details	High
FR11	A user can see events they bought tickets for	Medium
FR12	A user can see how many tickets they have bought	Low
FR13	There is support for events with different ticket prices	Medium
FR14	A ticket release can be displayed	Low
FR15	A user can not buy a ticket before a release	Low
FR16	A user can buy a ticket after a release	Low
FR17	After a completed purchase, tickets are sent by email	Medium
FR18	Tickets delivered by email contain a unique order number, payment, and event info	Low

Table 2: Functional requirements

NFR ID	Description	Priority
NFR1	A user should quickly be able to determine if they are interested in an event	Medium
NFR2	It is easy to find the button to buy a ticket	High
NFR3	It is easy to understand how to buy multiple tickets	High
NFR4	It is easy to understand the price to pay for tickets	Medium
NFR5	The product should have a consistent design	Low
NFR6	The login process should be intuitive	Medium
NFR7	A user can determine if they are interested in an event from the explore page	Low
NFR8	It is clear why tickets cost differently and by how much	Medium
NFR9	A user understands that tickets are unavailable and when they will be available	Low
NFR10	A user understands when a ticket is available	Low
NFR11	Emails with tickets are clear and easily readable	Low
NFR12	Emails should look professional	Low

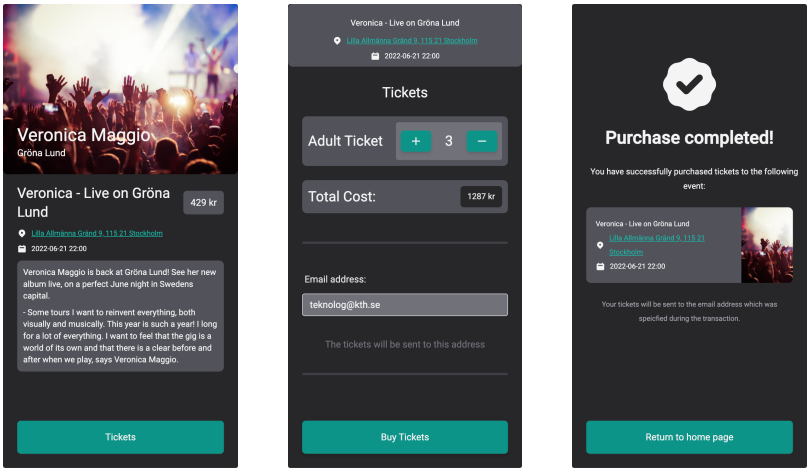
Table 3: Non-Functional requirements

4 Product Description

The product is a website for purchasing tickets to events. The website is usable on smart phones and on desktops.

4.1 MVP

The first version, the Minimum Viable Product (MVP) started off with 3 different pages and very basic functionality. It allows users to view an event and find a detailed description, event time, price, and location as shown in Figure 1a. Users can then decide to buy tickets for this event. They can choose a number of tickets to buy and see the total cost as can be seen in Figure 1b. After a successful purchase, users are greeted with a confirmation message as shown in Figure 1c.



(a) Event page (b) Purchase page (c) Purchase complete

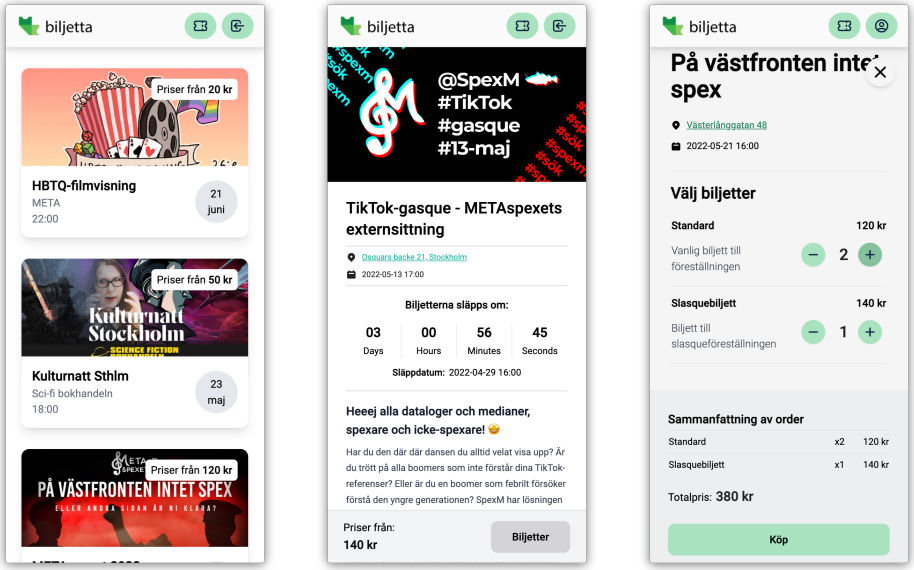
Figure 1: MVP user interface

4.2 Alpha version

In the next phase, the Alpha phase, more functionality was added to the website. The alpha version is a superset of the MVP and contains the same base functionality.

In addition, users are able to browse through and find multiple different events in the explore page as can be seen in Figure 2a. Users can buy different types of tickets for an event as is showed in Figure 2c, and be able to view which events they have previously purchased tickets for. Other features were also be added, for example that events can have ticket releases at specific times as can be seen in Figure 2b. Tickets can not be purchased before this time, and there can also be limits on the amount of tickets that can be purchased. Event descriptions can also be customized with different headings, fonts, emojis etc. The whole website was also heavily redesigned to make it easier to use and make the interface more beautiful, especially on the desktop version. Bought event tickets are now delivered as emails to the user, and there is functionality for users to be able to log in and have accounts across multiple sessions and devices.

Future development of the system could include creating a fair second hand market for tickets, adding more functionality for discovering new events, and creating a robust system that can handle surges of users during popular releases.



(a) Explore page (b) Event page (c) Purchase page

Figure 2: Alpha version user interface

5 System architecture

The system consists of a frontend connected to a backend with access to a database. The website is a Single Page Application (SPA) and is implemented using the React framework. The system consists of three separate parts. A drawing of the system can be seen in Figure 3.

The frontend is served by a simple Node.js server, that builds and sends all necessary HTML, CSS, and JavaScript files to the web browser of the user. The frontend is implemented with 3 major technologies. The application uses React as a front-end framework for building an interactive user interface. In addition, in order to keep track of the application state, we use the Redux framework. We also use the Tailwind CSS framework for designing the UI.

The frontend communicates with a backend API using a RESTful API. The API is implemented using Node.js and the Express web framework. The API server is a separate component from the frontend server, which could facilitate scaling the API to multiple servers in the future. The API enables us to interact with our database, as well as performing tasks such as sending emails. The API is documented using Swagger.

The backend is using the Object Relational Mapping (ORM) library Prisma. The database schema is therefore defined in a Prisma schema which in turn automatically generates the database schema and mappings for all database tables to JavaScript objects in our Node.js backend. The database used is PostgreSQL, but all interactions with the database is handled by Prisma.

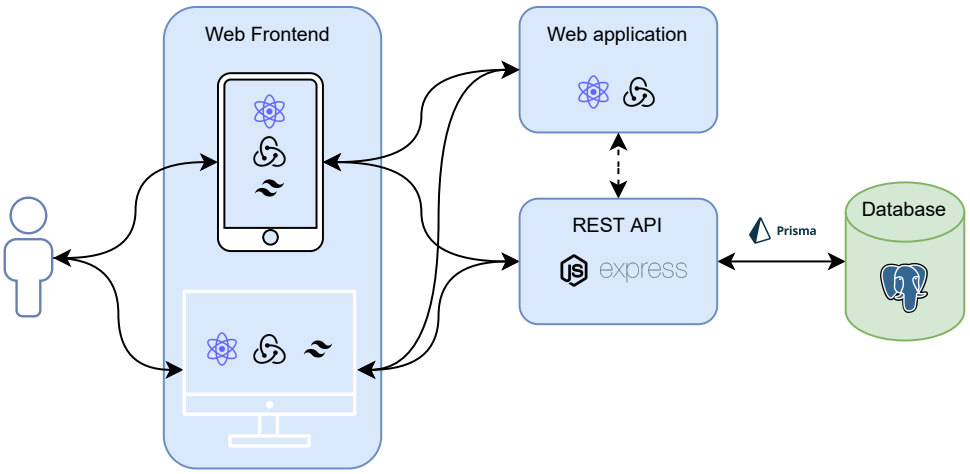


Figure 3: System Architecture

6 Tools and Frameworks

One of the most crucial steps when starting a software project is to decide beforehand what tech tools are going to be used, as the choice of tools and their compatibility with each other have a significant impact on the advancement of the project. Hence, we have discussed in detail to make this choice wisely and minimize the chances of getting stuck during the development process. Mainly two points were taken into account during the decision-making process. The tool should either be a popular choice among the software development community, to make sure that the tool is reliable and there are many resources to learn from; or we use tools that team members already are comfortable or experienced with.

6.1 Figma

Having to rewrite code is usually more time-consuming than making a redesign. For this reason we are using a design prototyping tool to design our user interface prior to working with frontend development. Among many popular alternatives, we decided on using Figma[1] as our UI design platform, as it is free, relatively easy to use and some team members have experience using it. When developing our MVP, we first designed paper prototypes, and then combined them into a design in Figma. This prototype can be seen in Figure 11, whereas the completed MVP can be seen in Figure 1.

6.2 ReactJS

For our users to be able to interact with our system, we need to implement a user interface. The frontend has a significant impact on how our target users perceive our platform, since it is the part of the system they directly interact with. The UI focuses on the aesthetics of the website and also expands its responsiveness, accessibility, and ease of use. We have decided to use ReactJS[2] as our UI framework, and Javascript[3] as the programming language. The framework allows creation of single page applications and reusing already built components written by the team or other developers,

[1] `https://figma.com/`
[2] `https://reactjs.org/`
[3] `https://www.javascript.com/`

installed via npm (Node Package Manager). It is also easy to learn thanks to its HTML-like declarative nature.

6.3 Tailwind CSS

One of the things that differentiates the web today from its early days is that web pages were very static back in the day, meaning that they were limited in both form and functionality. Along with the development of CSS, the era of having dull and unstructured content was put to an end, and today it is possible to add colors, layouts, animations, and a lot more with some CSS. Coding plain CSS is, however, usually time-consuming and unexciting. Since we want to spend less time while still being able to create good-looking websites, we have decided to use a CSS utility framework called Tailwind CSS[4], which relieves the developers from having to write loads of CSS. It is also very convenient, because it can be written directly inside HTML files, or in our case our JSX files. The framework is very easy to use and the documentation is well-written.

6.4 Redux toolkit

While it is possible to facilitate communication and sharing data across different components by passing them into other components as arguments (that we usually call *props*) by using plain React, it gets messy very quickly, especially when building a fully functioning application with many components. React has a hierarchical tree structure with all its components, and with plain React, props are passed from one part of a tree to another by going through other parts in between that do not actually need the data. This is called "prop-drilling" and it is not recommended to attempt. Thanks to Redux, the state of an application is kept in a store and each component can access any state when needed in a much cleaner fashion. While Redux solves the problem of managing states, it is usually considered as a highly complex framework to use with its structure and boilerplate code, especially for beginners. Redux Team, however, launched a much simpler version of their tool, called "Redux Toolkit"[5], which is now their officially recommended version. This is great news for our team, as we do not want to spend a lot of time struggling with outlandish code.

6.5 NodeJS and Azure functions

For the server side of our project, we aim to build a RESTful API, meaning an Application Program Interface based on representational state transfer, where we are using HTTP requests to access and use data. We originally planned to write the backend in C#, but after using it for a while we decided to switch to NodeJS[6]. Since NodeJS uses Javascript (same as our frontend) we thought it would be easier for everyone to get comfortable fast. With this change, we now, quite incidentally, use PERN stack for our application, which is a technology stack for web development that consists of PostgreSQL, Express, React and Node.js. This is also great, because it is a widely used web development stack, and there are many resources on the internet on how to efficiently build PERN stack applications as a whole.

We are also planning to use Azure functions[7] to add a lot of power to the application with little effort. Azure functions is a cloud service providing regularly updated infrastructure and resources needed

[4] `https://tailwindcss.com/`
[5] `https://redux-toolkit.js.org/`
[6] `https://nodejs.org/`
[7] `https://azure.microsoft.com/en-us/services/functions/`

for our app to run. It is quick to build, easy to maintain and adds great scalability to projects. The ability to write code without setting up any infrastructure makes this tool particularly attractive to the team.

6.6 PostgreSQL and Prisma

Data management is essential in our project and is needed for event details, user information, or possibly for authentication tokens in our application. For this reason, we need to use a Database Management System (DBMS). We decided to use PostgreSQL[8] as our DBMS software due to its popularity and because all team members utilized it during a recent course. Furthermore, there is a big SQL community on the internet to potentially help us improve our database skills and fix errors.

After writing our own PSQL for a while we got a tip from another group to use an Object Relational Mapping (ORM) library Prisma[9]. Prisma allows us to easily and collaboratively update the database schemas.

6.7 Git and GitHub

To work collaboratively on the project, we are using the well-known version control system Git[10]. Git allows us to create files with revision history called repositories, track changes, branch out for adding new features, manage merge conflicts and much more. We also use Issues with labels and milestones on GitHub [11] to stay organized, updated and potentially to ask for help from other team members.

7 UML diagrams

7.1 Activity Diagram

The activity diagram is meant to illustrate in what steps the user can interact with the website and what activities can be performed (Seidl et al. 2015). The connection between the various steps are shown in the diagram where the user's actions and respective consequences can be seen. Actions are represented as rounded rectangles describing what action the user can take, with its connections to other activities being the black lines. Choices with different outcomes presented to the user are represented as diamonds having one connection leading into it and two leading out to respective consequences of a decision. Finally are events, where the user either awaits or performs an action that is not guaranteed such as waiting for a purchase to complete or exiting an ongoing action. Events are represented as rectangles with a triangular indentation.

The user first enters the website and is presented with the list of available events. From there the user can pick an event to see details about on a separate page. While viewing an event the user can either go back to the list of events or proceed to the page for configuring ticket purchase. On the purchase page the user can adjust the amount of tickets they want to purchase after logging in. At any time during this stage the return button can be used to take the user back to the previous page, the Event page. When the user has configured their tickets they can decide to purchase the

[8] `https://www.postgresql.org/`
[9] `https://www.prisma.io/`
[10] `https://git-scm.com/`
[11] `https://github.com/`

ticket(s) taking them to a page with confirmation of the ticket(s) being purchased from where the user has the option to return to the explore page or exit the website.

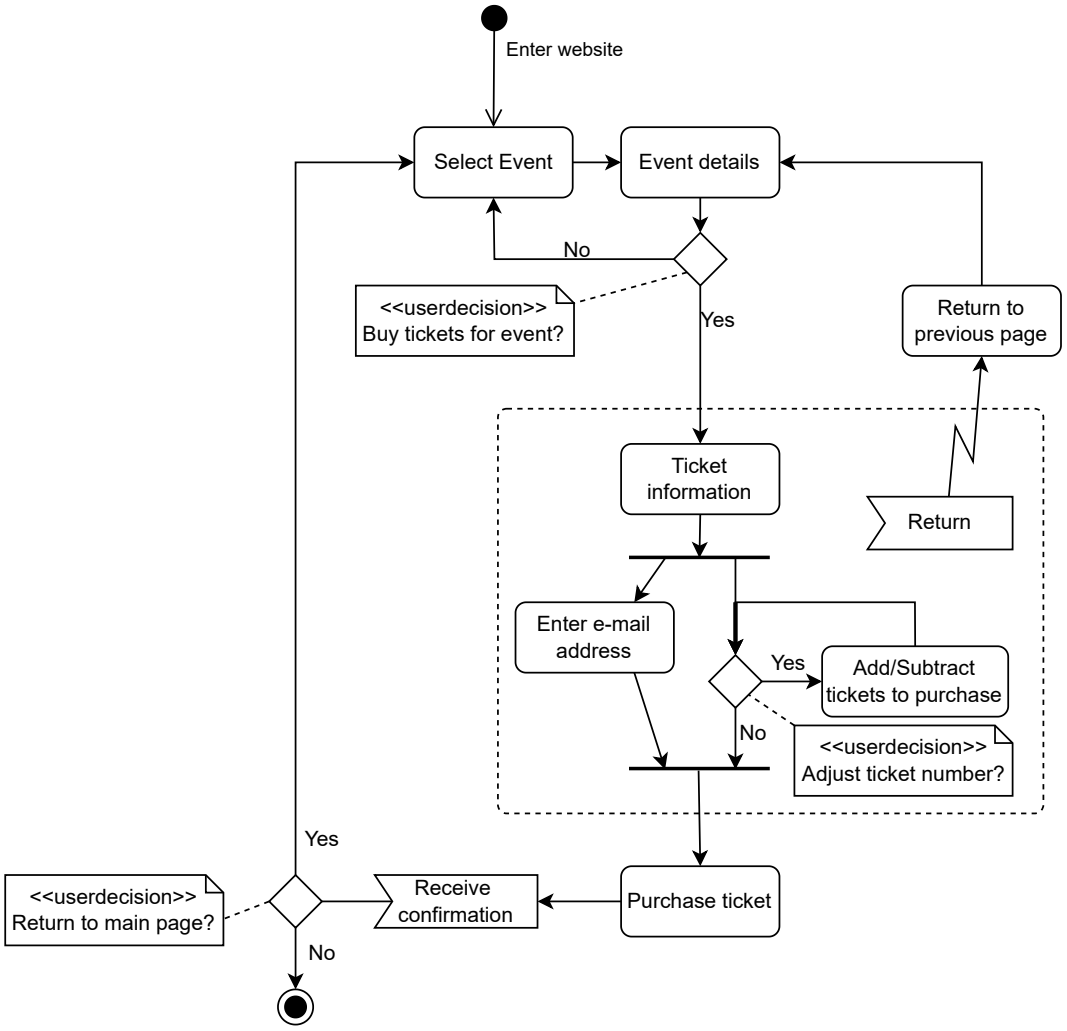


Figure 4: Activity Diagram

7.2 Use Case Diagram

A use case diagram allows us to model possible usage scenarios that the system should be developed for. The different use cases represent what a user expects from the system (Seidl et al. 2015). In our system, the user visits the webpage by entering the url in a browser. In the MVP, the user is presented with an event with general information about the event and the option to proceed to the ticket page. On the ticket page the user can select the number of tickets and enter an email. After buying the ticket(s), an email will be sent to the user from the server with a purchase confirmation.

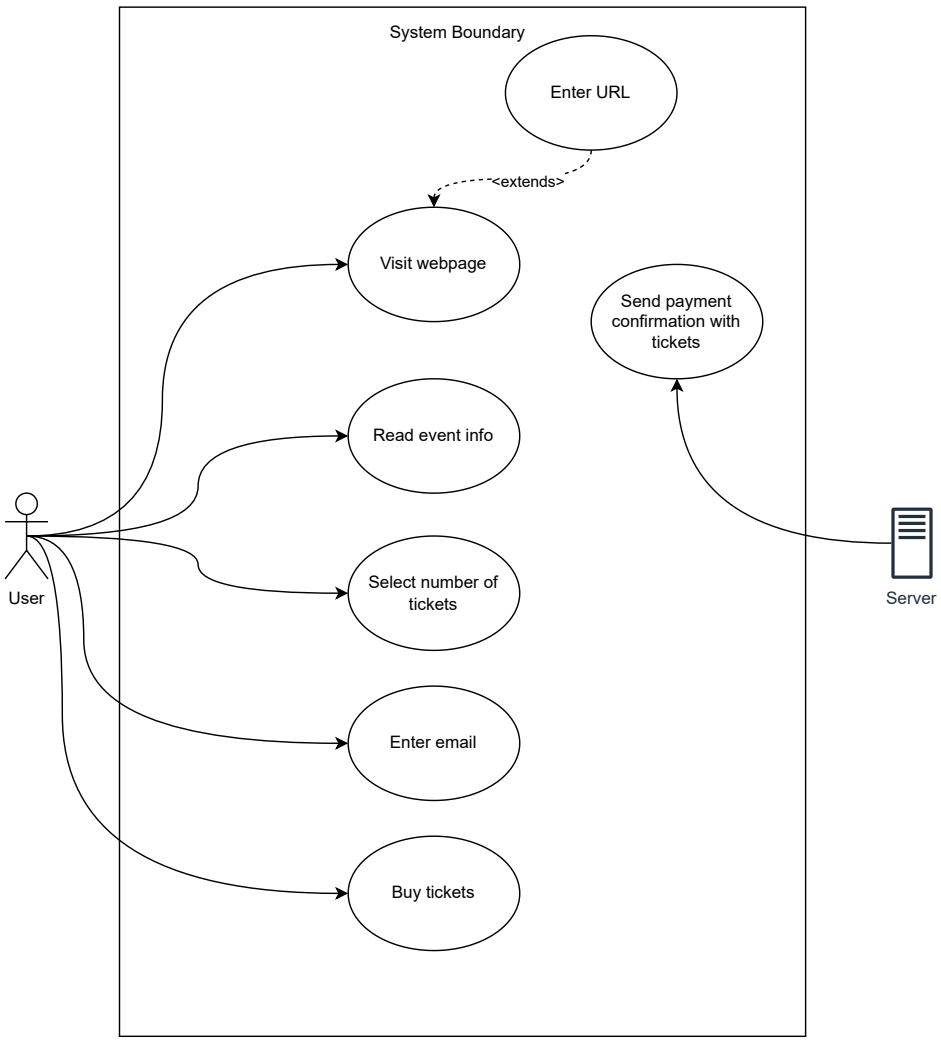


Figure 5: Use Case Diagram

7.3 Sequence Diagram

The sequence diagram aims to model interactions between the objects in a system (Seidl et al. 2015). They are used in a specific situation and show messages and responses within the system. As showed in Figure 6 we have four actors in our system, the user, the UI which is a website, the API backend and the database. The interactions are displayed for the transaction flow of a user trying to buy a ticket.

Firstly the user opens the website, which means the website requests event information from the API which is in turn requested from the database before it is returned all the way back to the user. The user then goes to the purchase page which means the website requests new event information from the API, such as different ticket prices and amount of tickets. The API in turn requests this information from the database, and it is then returned all the way back to the user. Lastly the user attempts to purchase ticket(s). This route has two different alternatives depending on if sufficient amount of tickets are available for the purchase. If tickets are available, the API inserts a purchase and tickets in the database and returns that the purchase was successful to the website which displays it to the user. If tickets are not available the API returns that the purchase was

unsuccessful, which the website displays to the user in the form of an error message.

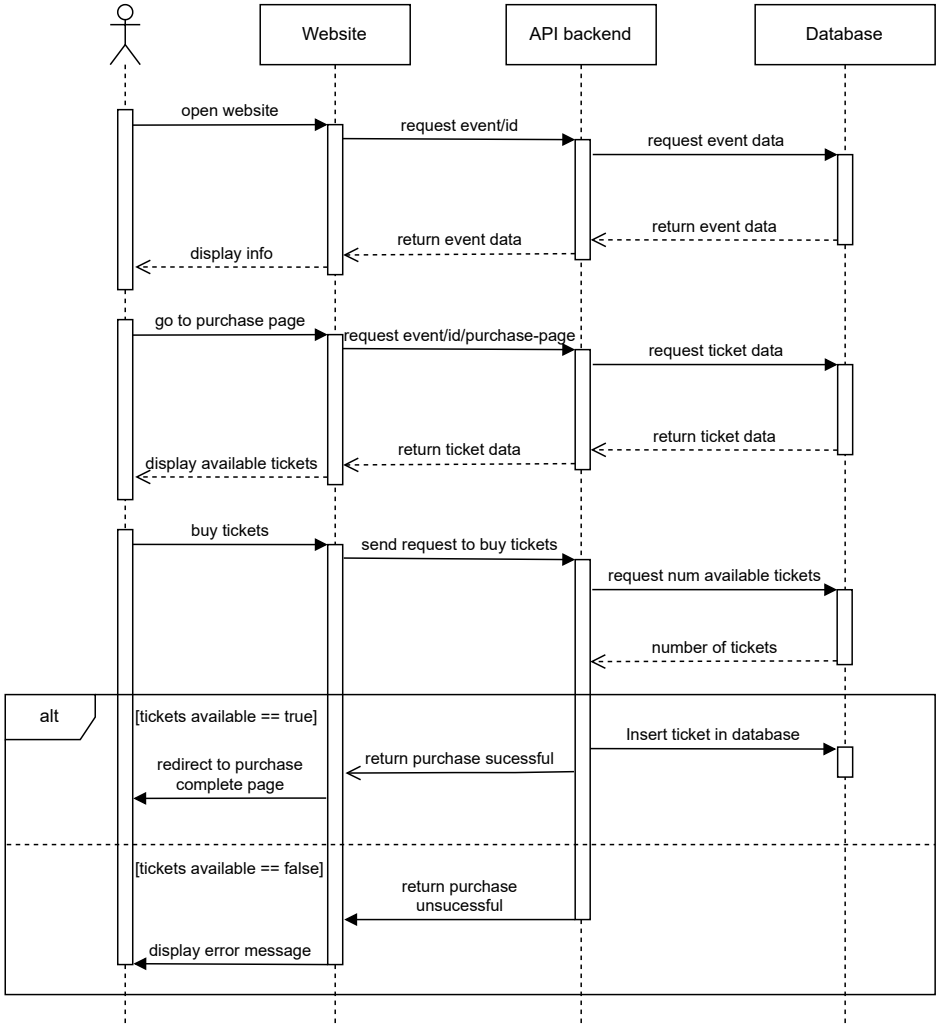


Figure 6: Sequence Diagram

8 Competitor analysis

Conducting a competitor analysis is essential to achieve an unconventional product that can compete with already established brands (Deshpandé and Gatingon 1994). It is vital to compare applications when the ambitions of the project is to create a system with faster and less complex payments, improved scalability and performance during high load. Therefore, the project will be evaluated against the most popular ticketing systems in Sweden, in order to identify the existing gaps, successes and trends on the market as well as disadvantages in the current MVP. Further are the comparisons of the MVP and its direct competitors, regarding mobile versions as well as the objectives of the project.

8.1 Live Nation

8.1.1 Faster Payments

One of the main goals of our project is to shorten the transaction flow in our product in order to achieve faster payments. Because of this we will use the length of the transaction flow as a metric to compare our product against our competitors, aiming towards shortening the time it takes to obtain a ticket. This goal encompasses minimizing the information input required of the user for each purchase such as address, personal details, phone number etc, as it is deemed hindering to the users goal.

Live Nation[12] has two routes when it comes to ticket payment since the website works as both a host for event-info from other sites and a first party seller. The first case is that the website redirects the user to the page of another ticket seller who sells the actual tickets making each purchase different for the user. In the other case, the user is sent to their own payment page which has very clear graphics depicting what stage of the payment the user is currently on.

This is a feature yet to be implemented in our product as the payment system does not handle transactions and is therefore very short. However, in future iterations a payment system must handle transactions and by that point our system should probably follow a similar layout as that of Live Nation. One exception on the other hand is that Live Nation's process requires an account which in turn requires a lot of information about the user, namely: name, address, postal code, phone number, etc. This is an element in the system that goes against our goal and should therefore be reworked in our solution. We could, for example, make the account connected to another service such as google for ease of log-in or use another method of verifying the user like bank-id or similar to spare the user the process of inputting a lot of information.

8.1.2 User Flow

One core feature that differs in Live Nations user flow in regards to ours is that the Live Nation uses filtering to a much lesser extent. For example, Live Nation's website only allows filtering date, location and genre. One reason is that Live Nation only sells tickets to music events meaning they have less filtering to apply as opposed to if they were to sell tickets to more types of events. As an effect, the filtering on the website is very easy to understand and always applies to what the user is searching for. This is desirable in our product as well but will have to be applied to more types of events than music which can complicate implementing it in a similar fashion. While the filtering is concise and understandable, there is no filtering option from the start of the page. This implies that the user has to know what they are searching for or have to go to the list of events before they can begin filtering. In our product we should see the value of letting users filter directly on the first page which simplifies the user flow for those who don't already know what event they are searching for.

8.2 Ticketmaster

8.2.1 Faster Payments

Similar to the Live Nation app, payments in Ticketmaster[13] require that the buyer has an account and the app works as a first party seller, but also directs users to other sellers. In other words, there is nothing original in the payment process in this app compared to the Live Nation app.

[12] `https://www.livenation.se/`
[13] `https://www.ticketmaster.se/`

8.2.2 User Flow

The Ticketmaster app has plenty of subcategories making it easy for the user to be very specific about the type of event they would like to attend. Also, the app displays the most popular events on the event landing page and every category-specific page. These are some features that should be taken into consideration for future iterations of the MVP, since it could speed up the searching process for the buyer and also increase sales. Additionally, an improvement that will be added in future iterations is that the app will recommend events to the user regardless if they have chosen a category or not. Also worth mentioning is that the Ticketmaster app has bigger buttons and more pictures than the MVP. This could serve as inspiration for making the ticket system more user-friendly.

8.3 AXS

8.3.1 Faster Payments

The payment process in the AXS app[14] is similar to the one in the Ticketmaster and Live Nation apps, although the AXS app handles purchased tickets differently. For instance the QR-code of a ticket is a personal QR-code linking to the account of the user, thus the same code is used for every ticket rather than having separate codes for every ticket. The buyer of a ticket also has the ability to share tickets with other AXS users, easing the process of buying tickets for a company of people and splitting the costs. This solution does not necessarily result in a faster payment process, however an increase in transactions between users could decrease the amount of initial buyers of an event, since a single user can buy multiple tickets. This is a possible solution to avoiding surges during popular ticket releases. The MVP does not handle transactions or contain a specific page for purchased tickets with their QR-codes. Our system will handle the purchases simply by sending a ticket to the buyer's email. While developing future ideas on how to handle high load, the AXS handling of the tickets should serve as a great example.

8.3.2 User Flow

The only features that enable fast searches in the AXS app are the search bar and filtering by date, week or month. Creating filtering options with different time frames is unique to this app and it is something that should be implemented in future iterations of the MVP since it is favourable to the user that might wish to attend an event during a specific time period. The app also displays a map for the location of events and it is possible to see upcoming events in specific arenas. Furthermore, the buyer is able to see the different ticket prices of an event before entering the payment process and there is a static menu bar at all pages of the app with links for the user's tickets, profile as well as the events page. All of these ideas will be taken into consideration since it is practical for the user to be able to see ticket prices before deciding if they want to attend an event and the menu bar is a necessity if the system is going to be user friendly.

9 Gantt chart

In order to plan and keep track of the timeline of the project, a Gantt chart has been made, which can be seen in Figure 7. The project started on the 17th of January and we completed the first milestone in the beginning of February which included the research and design phase for the MVP.

[14] `https://www.axs.com/se`

The complete implementation and integration of the MVP was then completed in the beginning of March. The planning of this phase was influenced heavily by suggestions from our stakeholders who suggested we cut down on tasks and features for the MVP as they initially want us to focus on the core of the product.

We then started researching and designing the Alpha version and implemented feature by feature in the following weeks. When this phase was completed we ended the project with a final phase of evaluation and user testing which is not included in the Gantt chart.

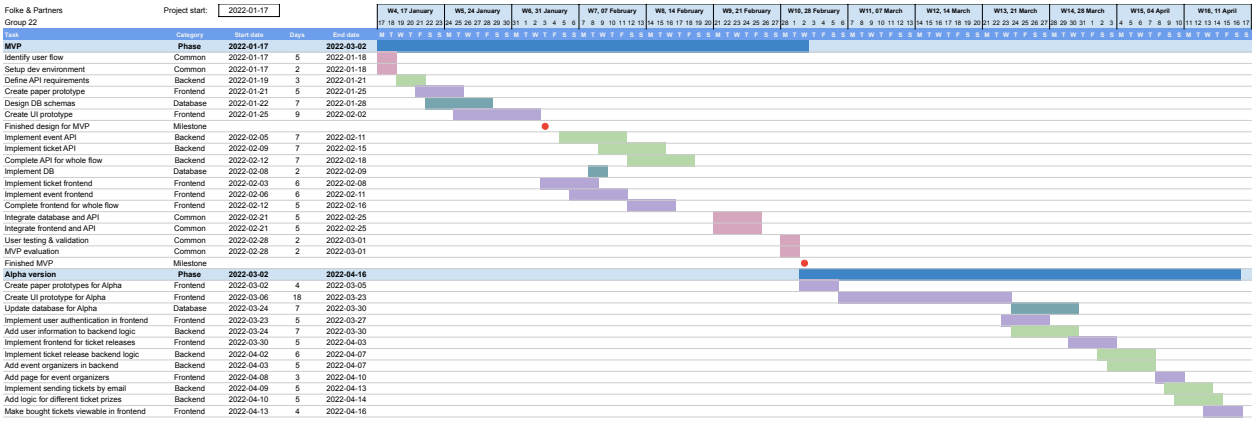


Figure 7: Gantt chart outlining the work for the MVP and Alpha development.

10 Software Engineering Methodology

The project was performed using an agile approach based on the scrum methodology. We used digital scrum boards on GitHub to manage issues and keep track of development. An overview of completed development tasks categorized by sprint can be found in Appendix A. Sprints had a time frame of about two weeks.

Adrian had the role of scrum master. As outlined in The Scrum Guide (Schwaber and Sutherland 2020), the scrum master has the responsibility of making sure everyone knows what to do, running scrum meetings and helping the product owner to manage the backlog. Klara served as the product owner, which entails managing the backlog of tasks to be done and prioritizing what tasks should be done in each sprint. She was also accountable for developing and communicating the product goal to the rest of the team.

At the beginning of each two-week sprint, sprint planning meetings were held where tasks were collectively assigned to all team members. We also identified any dependencies between tasks, set deadlines for urgent tasks, and estimated the time it would take to perform each task. For example, if a member estimated a task to take two hours to complete, and another member estimated it would take them eight hours to complete the same task, the person estimating it will take less time would be assigned that particular task. Of course, the time it takes for a person to complete a task was not the only criteria for dividing up the work. For example, if a member was interested in taking on a particular task, this would be taken into account. A sprint could also be cut short if we were done earlier, or made longer if we had less time to spend on working with the project. This was always a group decision where the opinion of every group member was taken into account. The workload was divided evenly so each member had approximately 13 hours of work per week which

corresponded to the amount of work expected given the extent and timeframe of the course.

Tasks were intended to be small, ideally under eight hours of total work. We wrote short user stories for each task to be performed to make sure that each task had a clear purpose and goal. For all tasks, a review was also needed before the task could be considered complete. A review entailed that another member who was not involved in completing the task reviewed the solution and identified potential issues. This helped catching bugs and gave each team member a broader view of the project. When assigning tasks during sprint planning meetings, time for performing reviews was also considered.

It was clear after some time that we preferred to work in pairs, as working with new technologies alone can be daunting. Therefore, we decided to start with pair programming. This meant that development tasks were assigned to pairs of people that collectively were responsible for completion of the task. The intent was to have the pairs work collaboratively but the pairs were free to work out the details of how this should work by themselves. It was for example possible to sit together physically and write code or use remote collaboration tools such as Visual Studio Code Live Share[15]. This has had the advantage of more direct communication about issues and that team members have a natural and efficient way of asking for help and helping each other. This contributed to an environment that facilitated learning, which was one of the main goals of the project.

Short standup meetings were held two days every work week where team members gave a brief status update on how their tasks were going. Each team member was expected to be present. These meetings were held digitally over Zoom. After each sprint, a sprint review was held together with our client Folke&Partners where we demoed the progress made during the sprint. We also performed a sprint retrospective meeting after each sprint where we as a team reflected on how the work was progressing and if there were any ways to optimize and improve our workflow. At each of these meetings, every team member was expected to be present. For the sprint review with our client, Carl was responsible for ensuring we were ready to present. This workflow was developed using feedback from our stakeholders.

In addition, we had access to an hour each week from our mentor, Anoud. Since we had quite extensive support in regards to development from our client, most of the support received from Anoud was in regards to documentation, and getting a clear view of what was expected from us as a team in respect to the course. As such, we reached out to Anoud when we needed help with issues our client could not help with.

11 Risk analysis

The following risk analysis is created to assess risks and their severities to properly prepare for potential detrimental issues in our future work. The importance of producing a risk analysis for a project lies in identifying risks and their severities by judging both the probability and impact of each risk. This information can then in turn be used to create a rating of how severe each problem is and to decide upon what measures can be taken to either prevent or handle the problem should it arise (Sommerville 2011).

Further categorization of the problem can be done through problem type. In this risk analysis the problem types *Project* and *Result* are used to indicate whether the problem will impact the processes in the project development or the result of the project. The severity ranges between five

[15] `https://code.visualstudio.com/learn/collaboration/live-share`

levels. The following levels are used for the probability of a risk happening: Rare, Unlikely, Possible, Probable, and Almost certain. For the impact, a risk can have the following levels: Insignificant, Minor, Moderate, Major, and Catastrophic. All risks identified can be seen in Table 4.

Table 4: Identified risks in the project. Probability is rated in a 5 point scale - ranging from Rare, Unlikely, Possible, Probable to Almost certain. Impact is also rated on a 5 point scale - ranging from Insignificant, Minor, Moderate, Major to Catastrophic.

	Description	Type	Probability	Impact	Measure
1	A team member is unable to work on a task	Project	Probable	Minor	Give said task to another member. Who becomes responsible is based on their role and/or availability
2	A critical member such as the scrum master is unable to continue with the project for a longer period of time	Project	Rare	Major	Conduct a team meeting and decide who will assume the duties of the absent person
3	The company does not have enough time to help, for instance in answering potential questions	Project	Possible	Major	If it can wait, ask said questions in the planned meetings with the company. Look for help elsewhere in the meantime such as online resources, Anoud, or team members
4	New public health recommendations require us to work at home, increasing communication issues	Project	Possible	Minor	Increase the amount of smaller group meetings and inform everyone what communication medium we will use
5	Excessive workload from other courses makes continuous work difficult	Project	Possible	Moderate	Decide whether a sprint is possible to make up for the loss of time in the upcoming team meeting
6	Someone is unable to attend a meeting and misses important information	Project	Almost certain	Minor	Always write down what was said during meetings and post in an appropriate communication medium to be accessed at a later time
7	Someone does not understand the code or other parts of the project and is unable to continue development	Project	Unlikely	Major	Ask other team members who might be able to help. If they cannot, and if it is possible, ask Mattias, David or Anoud. Otherwise, alter the timeline or project so that it can still be delivered.
8	The code is not scalable and must be rewritten	Result	Possible	Major	If the portion of code is smaller, reprioritize our timeline to accommodate new tasks solving the issue. If the portion of code is significant and its impact is large, documentation of the faults and possible solutions needs to be made. Before the next scrum sprint, collectively plan a strategy for rewriting the code based on the documentation and execute it during the next sprint
9	Not all tasks are completed in a sprint	Project	Possible	Moderate	Plan the next sprint with this in mind and assign additional team members to help finish the remaining tasks if required

10	A previously agreed upon tool or framework is undesirable to use when developing	Result	Rare	Major	Inform all the members of what is undesirable and propose a solution. For instance, switching to a different tool or framework. If it is agreed upon that there will be a change, set up a task to handle the change.
11	The necessary features are not implemented in time for the final product to support all planned functionality	Result	Possible	Minor	The time required for implementing each feature needs to be estimated and features that are dependent on each other needs to be taken into account when planning. However, the features added after the MVP are only a bonus and completing them is not a strict requirement, but they should be planned accordingly to ensure that there are as many product functionalities as possible.

To illustrate the severity of the risks included in the analysis, Figure 8 contains a gradient of severity obtained from combining the impact and probability of each risk. Our current risks are split between medium and high severity risks with no risks being low or very high severity. This indicates the risks in this project might take up a fair amount of resources but with no single risk being detrimental to the continuation of the project. However, a danger lies in the possibility of having several risks coincide during the project which could create a heavy load for the team. Because of this danger, it is necessary for individual issues to be solved effectively and the planned measures to be acted upon quickly to avoid overlapping risks.

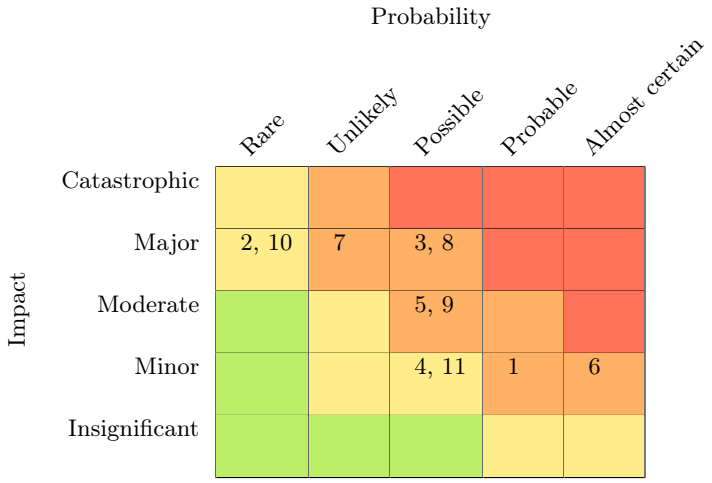


Figure 8: All identified risks laid out in a matrix. Each number refers to the risk described in Table 4. Green indicates low severity, yellow indicates medium severity, orange indicates high severity and red indicates very high severity.

12 Change of management

In order improve the efficiency of our workflow we have adjusted out methodology slightly during the project timeline. The biggest change was that we started utilizing pair programming, where the responsibility to complete a task was assigned to two team members who could collaborate. This had the advantage of more direct communication and created a natural way of asking for help

and learning from each other. With regards to the design process we also realized the advantage of having more people involved early on in the design process. When developing the Alpha version, we therefore decided to have all group members create their own paper prototype. This lead to having more creative and different ideas that could then be combined into the final version, which was a big improvement from the design of the MVP. This directly lead to the Alpha version having a more consistent and thought-out design.

In regards to our business requirements we made some adjustments during the project in order to maintain realistic goals. The first business requirements for the MVP had to be scaled down since they were not specific enough and too extensive. We removed the requirement that the system should be able to scale, both since it is useless to scale a product if it is not useful in the first place, and also since it can be difficult to measure scalability. We also removed the requirement that tickets should be delivered as e-mails in the MVP so we could focus on the core functionality of a single transaction flow. In the beginning of March we had fulfilled our initial requirements as part of the MVP and had to expand them by adding new requirements. This method was really productive since we were now comfortable with our tools and frameworks, and aware of what level of ambition we should aim towards. We could define new requirements that were realistic in terms of the time-frame of the course. We were also more specific when defining the requirements for the Alpha version which greatly contributed to being able to fulfill them within the project timeline.

During the project we had to handle some of the scenarios we predicted in the Risk Analysis. For example, during a week of exams in other courses, the team had less time to focus on development. Fortunately this coincided with the finalization of the MVP, which was positive since we could use this time to research and design the new version of our system. We assigned less tasks and focused only on sketching features for the Alpha version which alleviated the amount of stress on the team. The other predicted scenario that actually happened was that we had to adjust some of the tools and frameworks used to develop the system. Quite early in the project we switched from writing the backend in C# to NodeJS, in order to use Javascript both in the frontend and backend. Since this change was implemented early on in the project we did not have to rewrite a lot of code and it went quite smoothly. The other big change was when we stopped writing our own SQL and instead made use of an ORM called Prisma. This change was a bit more difficult since it was made later in the project, but it was worth it since we had to update the database quite a few times when adding features for the Alpha version.

13 Verification and validation

13.1 Verification

To ensure that the product works as intended, several forms of verification and validation have been performed throughout the project. Verification is often an internal process to check that the system meets the requirements through reviews, walk-throughs and inspection. One method which has been used for verification is the process of reviewing during the design phases for the user interface. When the UI for the product was initially designed as well as when it was redesigned for the Alpha version, several team members first created their own versions of the UI as paper prototypes. These various paper prototypes were then collectively reviewed and the group decided which features or design choices seemed best. This method of verification ensured that the most important features were included and that the business requirements were met.

13.2 Code maintainability

Another way to ensure that the code works as intended and is maintainable is the review system for Github issues which has been used throughout the production process. As described in the software methodology section, each task or issue is assigned to one or more people and once the issue has been completed it is put up for review. An issue is not considered complete until it has been reviewed by a team member who has not worked on the task themselves. The reviewer will run the code to see that everything works as it should and that the new features don't break any other part of the product. This review system ensures that the product is tested thoroughly throughout the coding process and makes finding bugs and other issues easier.

13.3 Validation

Validation on the other hand has the purpose of determining whether the product meets the needs of the users. In order to validate our system we have performed user testing. The tests were performed by 3-4 members of the project group and consisted of us both asking the user questions and asking them to perform tasks which were timed. One example is that we asked the tester to attempt to buy tickets for an event and then we timed the task and asked them how intuitive the process felt from 1-10. Our method for user testing was inspired by Nielsens usability testing which focuses a lot on making sure that the system is intuitive and coherent in design choices (Moran 2019).

13.4 Test Cases

In order to both verify and validate our system we have created test cases, as seen in Table 5. Each test case is related to either a functional or non-functional requirement from subsection 3.2. The functional requirements are related to the system functionality and can be performed by the team in order to verify our system. The tests related to non-functional requirements focus more on the usability of the product and therefore need to be tested on users in order to validate our system. This is to ensure that we can accurately and objectively determine if the interface feels intuitive and is easy to navigate.

ID	Description
TC001	View an event, should display the relevant event information
TC002	User test: Ask if users are able to determine if an event is interesting or not
TC003	Buy one ticket
TC004	Buy multiple tickets
TC005	Attempt to buy negative tickets, should not work
TC006	User test: Ask if buying a ticket feels intuitive
TC007	User test: Ask if buying multiple tickets feels intuitive
TC008	User test: Ask if the amount to pay is clear
TC009	Check that all pages are reachable from a smart phone
TC010	Check that all pages are reachable from a desktop
TC011	User test: Ask if users perceive the design to be coherent
TC012	Log in using Google
TC013	Attempt to log in with incorrect Google authentication, should fail
TC014	User test: Ask if login process is intuitive
TC015	Open explore page and see events
TC016	Click one event on explore page
TC017	User test: Ask if they can determine if they are interested in an event from the explore page
TC018	View "my tickets", should see a list of different bought tickets
TC019	View an event with more than one price category, tickets with different prices should be visible
TC020	User test: Ask if information about different tickets and prices is clear
TC021	View event with future ticket release, should see exactly when tickets are available
TC022	Try to buy ticket before release, should not be possible
TC023	Try to buy ticket after release, should work
TC024	User test: Ask if users understand when a ticket is unavailable
TC025	User test: Ask if users understand when a ticket is available
TC026	Buy tickets, check that an email is delivered
TC027	Check that a delivered email contains an order number, payment, and event info
TC028	User test: Ask if users find the emails viewable and clear across multiple mail clients
TC029	User test: Ask if users think that emails look professional and not like spam

Table 5: Test Cases

13.5 Traceability matrix

In order to see the progress of the project a traceability matrix has been created. All tests for the functional requirements performed by the project team are considered complete, as can be seen in Table 6. The test cases for the non-functional requirements either consisted of asking the user a yes-or-no-question or asking them to answer on a scale from 1-10. The test is considered OK if a majority of the testers answered yes or if the average value was above 7/10. All tests for the non-functional requirements were considered OK except one, that e-mails should look professional.

BR ID	FR ID	Tests	Test-status
BR1	FR5	TC009	OK
	FR6	TC010	OK
	NFR5	TC011	OK (9.8/10)
BR2	FR1	TC001	OK
	NFR1	TC002	OK (9/10)
BR3	FR2	TC003	OK
	FR3	TC004	OK
	FR4	TC005	OK
	NFR2	TC006	OK (8.7/10)
	NFR3	TC007	OK (9.7/10)
	NFR4	TC008	OK
BR4	FR7	TC012	OK
	FR8	TC013	OK
	NFR6	TC014	OK (7.5/10)
BR5	FR9	TC015	OK
	FR10	TC016	OK
	NFR7	TC017	OK
BR6	FR11	TC018	OK
	FR12	TC018	OK
BR7	FR13	TC019	OK
	NFR8	TC020	OK (10/10)
BR8	FR14	TC021	OK
	FR15	TC022	OK
	FR16	TC023	OK
	NFR9	TC024	OK
	NFR10	TC025	OK
BR9	FR17	TC026	OK
	FR18	TC027	OK
	NFR11	TC028	OK
	NFR12	TC029	Incomplete (3.7/10)

Table 6: Traceability matrix.

13.6 Results from User Testing

User testing was performed with six different testers on the Alpha version, and gave a lot of interesting findings. Overall, the testing and evaluation of the system proved very successful in the sense that users found the website coherent in design, found the interface generally intuitive and were able to perform all of the major tasks that were asked of them. They particularly pointed out that there was no unnecessary information which made it easy to navigate the different pages. Compiled interesting results can be seen in Figure 9.

As can be seen in the figure, the testers found the design to be coherent, found that it was intuitive

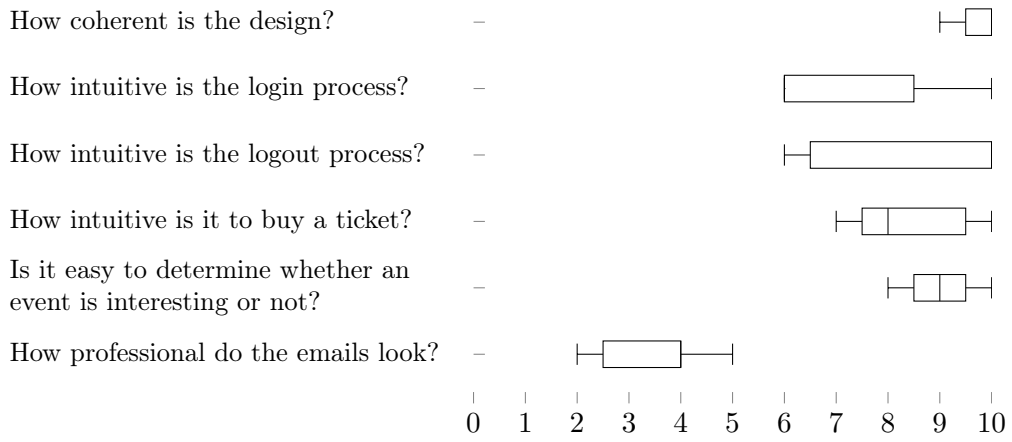


Figure 9: User testing results from the 10-point scale questions

to buy a ticket, and were able to easily determine if an event was interesting or not. In addition to the results presented in the figure, all testers found that it was easy to buy multiple tickets and found that it was easy to distinguish between different types of tickets. All testers rated both of these aspects as 10/10 or 9/10. However, we found two areas in which we could improve our design to make it more usable.

The biggest flaw in our system was confusion regarding the log in/log out functionality. When a user would attempt buy a ticket from an event page, they would be asked to log in and then be redirected back to the same page as before. This was regarded as counter-intuitive by our testers, instead, they expected to be redirected to the ticket purchasing page after signing in. They also found that symbol for logging out and logging in could be clearer. Our final finding was that the e-mails looked unprofessional with regards to their formatting. All of these issues were taken into account and improved upon before shipping the final product.

14 Discussion

In conclusion we achieved all of the business requirements given by our stakeholders, which is a great achievement given our previous experience and the project timeline. This owes to the fact that we used an iterative process and set realistic goals that were expanded upon after experience using our tools were gathered. The result from the user testing showed that the system is intuitive to use and consistent in the design. The user interface had a few usability issues that were remedied before the final version was completed.

The cooperation and communication with our stakeholders worked very smoothly. We seemed to have a mutual understanding of what level of ambition could be achieved given our resources. We were able to ask for guidance while still having freedom to prioritize which parts of the system we wanted to focus on. The reason behind this is that the company had realistic expectations of us from the start and were not dependent on the result from our project. They wanted this to be a learning experience for us. This allowed them to let us work in our own pace with a fair amount of freedom.

Our implementation of the scrum methodology worked really well for our development and collaboration within the group. The scrum meetings allowed us to be updated of each others work while

also providing a space to ask for help and cooperate on issues. The small tasks made it possible to work continually and gave a deeper understanding of the project since we always reviewed each others work and worked on all parts of the system. In addition, it was really helpful to have a structured workflow where everyone is always aware of what is expected of them. A downside to this is that it leaves limited space for taking it slower when there is less time to work on the project. If you always have responsibility for several tasks you can get stressed out if the workload is temporarily increased in other courses. The scrum methodology is also intended to be used when working in a professional environment. We on the other hand are students with different schedules which means that team members finish their tasks at unpredictable times. This can sometimes mean that reviews need to be completed last minute.

During the project we had to handle some of the scenarios we predicted in the Risk Analysis. During periods of more workflow in other courses we adjusted the sprint tasks to reflect the time each team member could work without straining their mental health and stress levels. We also changed some of the frameworks we had intended to use in the beginning of the project. The biggest change was when we stopped writing our own SQL and instead made use of an ORM, Prisma. This required a bit of extra work in the transition phase, but was worth it when we updated the database in later tasks.

References

Deshpandé, Rohit and Hubert Gatingon (1994). "Competitive Analysis". In: *Marketing Letters* 5.3, pp. 271–287.

Moran, Kate (Dec. 2019). *Usability Testing 101*. URL: https://www.nngroup.com/articles/usability-testing-101/.

Schwaber, Ken and Jeff Sutherland (Nov. 2020). *Scrum Guide*. URL: https://scrumguides.org/scrum-guide.html.

Seidl, Martina et al. (2015). *UML@ classroom*. Springer.

Sommerville, Ian (2011). *Software engineering*. eng. 9. ed., International ed.. Harlow: Addison-Wesley. ISBN: 0-13-705346-0.

A Project status

During each sprint a list of small development tasks is created, and the work is divided between the team members. The following table contains the backlog of completed tasks each sprint. The issue numbers represent the issue number that we have on our sprint board on Github.com. The tasks are assigned different categories that reflect the nature of the tasks. As described in section 10, Adrian and Klara were mainly responsible for creating the tasks before collectively handing out tasks during Spring Planning Meetings.

Task	Assignees	Category
Sprint 1, 2022-01-20 - 2022-02-4, MVP research and design		
#1 Set up a backend dev environment	marcus	backend
#2 Define an API specification	josefina sina	backend
#3 Define the user flow for the MVP	adrian carl erik harry	system design
#4 Design paper prototype	erik harry carl marcus	frontend system design
#5 Set up dummy database environment	klara	database
#6 Dockerize dev environment	marcus sina	backend
#7 Create design components	adrian carl sina	frontend
#8 Design mobile version of event page	erik harry	frontend
#9 Design desktop version of event page	klara carl adrian	frontend
#10 Create database schemas	klara josefina	database
#13 Set up frontend dev environment	marcus	frontend
Sprint 2, 2022-02-05 - 2022-02-18, MVP implementation		
#25 Create API specification for event	josefina sina	system design
#26 Create API specification for tickets	josefina sina	system design
#27 Implement buying a ticket in backend	harry marcus	backend
#28 Implement read event info API	marcus	backend
#29 Implement specific event page	klara adrian	frontend
#30 Implement ticket buying page	carl erik	frontend
#31 Implement redux for state management	carl erik	frontend
#32 Implement purchase complete page	sina	frontend
#33 Add docker-compose for database	marcus	database
Sprint 3, 2022-02-21 - 2022-03-01, MVP integration, evaluation and testing		
#41 Update event schema in database	klara	database
#42 Connect event API with database	harry marcus	backend integration
#43 Fetch data from API on Event page	sina carl	frontend integration
#44 Connect frontend with API for buying a ticket	harry carl	frontend integration
#45 Fetch data from API on Purchase page	klara adrian	frontend integration

#46 Fetch data from API on Purchase complete page	erik josefina	frontend integration
#47 Connect ticket buying with database	marcus	backend integration
#48 Change to node.js backend	marcus sina	backend
#50 Implement swagger in jsdoc	adrian	backend
Sprint 4, 2022-03-02 - 20222-03-23, Alpha research and design		
#57 Define requirements for Alpha	everyone	system design
#58 Design paper prototypes Alpha	everyone	system design
Sprint 5, 2022-03-25 - 2022-04-12, Alpha Implementation		
#63 Enable sign in via Google	marcus harry	backend database
#64 Add top navbar	sina	frontend
#66 Redesign event page	klara adrian	frontend
#68 Redesign purchase page	erik carl	frontend
#69 Add functionality for multiple tickets	josefina	backend database
#70 Redesign purchase complete page	sina	frontend
#71 Add orders	adrian klara	backend database
#72 Evaluate prisma as ORM	adrian	backend database
Sprint 6, 2022-04-15 - 2022-04-26, Alpha Implementation and Integration		
#67 Add optional ticket release times	josefina	backend
#81 Integrate ticket buying with API	carl erik	frontend
#82 Implement Log-In in frontend	harry	frontend
#83 Implement user session when buing tickets	marcus	backend
#84 Add ticket release countdown in frontend	josefina sina	frontend
#86 See own orders in backend	adrian klara	backend
#87 Return order information after successful purchase	adrian klara	backend
#88 Integrate purchase complete page with API	carl erik	frontend
#89 Harmonize pop-ups	sina	frontend
#90 Add email functionality	marcus	backend
Sprint 7, 2022-04-26 - 2022-05-09, Alpha finalization		
#91 Add page for previous orders	adrian klara	frontend
#109 Fix login to redirect to ticket page	sina	frontend backend
#110 Improve readability of logging in and out	carl erik josefina	frontend
#111 Improve e-mail layout	marcus	backend
#112 Look into/try deploying the code	adrian	backend integration

#114 Implement error handling for unsucessful purchase sina frontend backend

B Prototypes

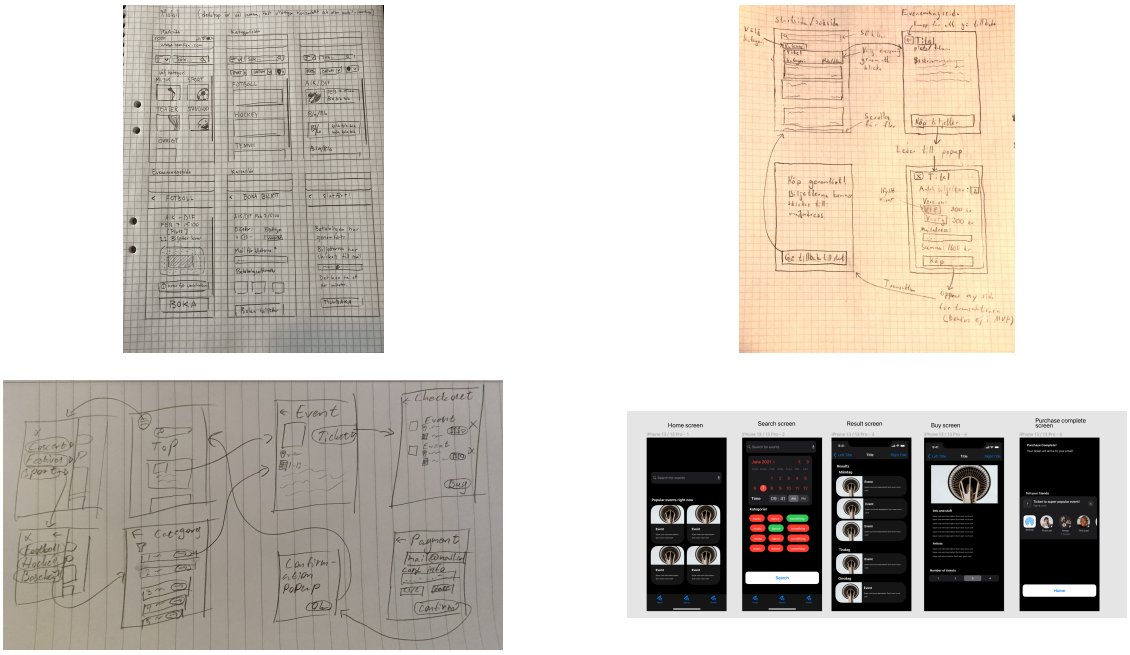


Figure 10: Paper prototypes for the MVP that were done individually

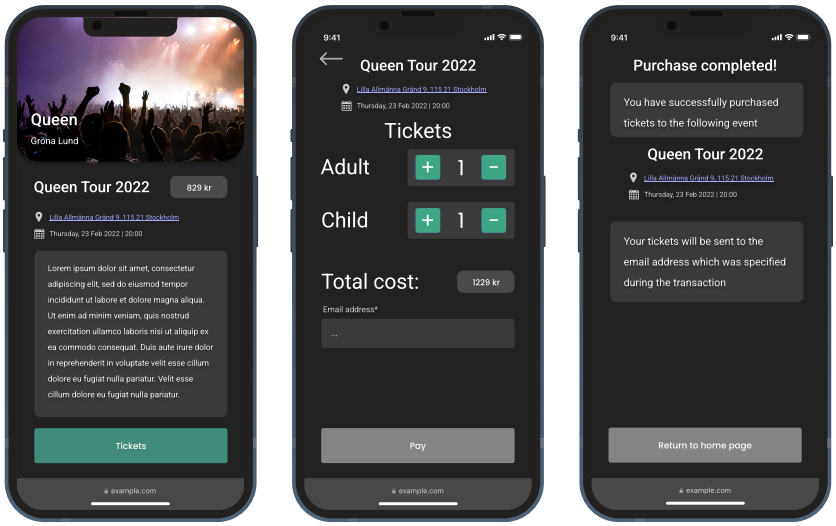


Figure 11: MVP prototype in Figma