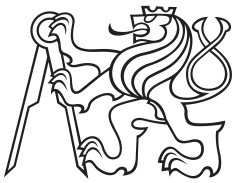
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Potka.me - Collaboration web platform

Kirill Kazakov

Supervisor: Ing. Pavel Náplava, Ph.D.
Field of study: Software engineering and technologies
May 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kazakov** Jméno: **Kirill** Osobní číslo: **495590**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Potka.me - Webová platforma pro spolupráci

Název bakalářské práce anglicky:

Potka.me - Collaboration web platform

Pokyny pro vypracování:

Navrhněte, a v první verzi realizujte platformu spolupráce více uživatelů formou společných akcí. Uživatelé budou moci přidávat přátele, zobrazovat seznamy, účastnit se aktuálních akcí nebo vytvořit/upravit akci vlastní. Postupujte následovně:
1) Prozkoumejte problém osamělosti a sociální izolace, zejména v České republice a Evropě. Porovnejte stavy před a během COVID-19.
2) Analyzujte možnosti boje proti osamělosti.
3) Definujte pojem a problematiku "spolupráce".
4) Analyzujte existující prostředky podpory spolupráce, dostupné v České republice. Vyhodnoťte jejich podporu organizace společných akcí.
5) Proveďte průzkum použitelnosti stávajících prostředků mezi studenty, případně organizátory akcí. Především v kontextu omezených možností osobní komunikace.
6) Definujte požadavky na platformu, která bude vycházet z provedené analýzy.
7) Navrženou platformu implementujte v podobě Proof-Of-Conceptu.
8) Vytvořenou platformu otestujte pomocí UAT.

Seznam doporučené literatury:

1. D'HOMBRES, Béatrice; SYLKE, Schnepf; BARJAKOVÁ, Martina;MENDONCA, Francisco.Loneliness in the EU. Insights from surveysand online media data[online]. JRC125873. Luxembourg: PublicationsOffice of the European Union, 2021 [visited on 2021-12-31].isbn978-92-76-40246-6. Available fromdoi:10.2760/28343.
2. ARLOW, Jim a Ila NEUSTADT. UML 2 and the unified process: practical object-oriented analysis and design. Boston: Addison-Wesley, 2005. ISBN 9780321321275.
3. ROLDAN, CARLOS SANTANA. React 17 Design Patterns and Best Practicesthird Edition: Design, Build, and Deploy... Production-Ready Web Applications Using Industry-S. Birmingham: PACKT PUBLISHING LIMITED, 2021.
4. "Next.js by Vercel - the REACT Framework." by Vercel - The React Framework. Vercel, Inc., 2021. https://nextjs.org/.
5. "React – a JavaScript Library for Building User Interfaces." – A JavaScript library for building user interfaces. Facebook Inc., 2021. https://reactjs.org/.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Náplava, Ph.D. Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Pavel Náplava, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I want to thank my supervisor, Ing. Pavel Naplava Ph.D., for his valuable advice, support, and time spent throughout the process of writing the Bachelor thesis. Also, I want to thank all the students who participated in a survey and in user acceptance testing.

Declaration

I declare that I have prepared the submitted work independently and referenced all the information sources used.

Prague, May , 2022

Abstract

The Bachelor thesis discloses the problems of loneliness and social isolation in Europe and especially in the Czech Republic. After analyzing dedicated researches, it turned out that many people face loneliness and the best way to deal with it is to communicate with other people. The main goal was to design and implement the first version of the web platform that will support cooperation between people throughout the organization of joint events. After the implementation process, the web platform was tested by a group of university students.

Keywords: loneliness, social isolation, software analysis, web development

Supervisor: Ing. Pavel Náplava, Ph.D. Praha, Technická 2, B2-39d

Abstrakt

Bakalářská práce odhaluje problematiku osamělosti a sociální izolace v Evropě a zejména v České republice. Po analýze souvisejících výzkumů se ukázalo, že mnoho lidí se cítí osaměle a nejlepším způsobem, jak se s tím vypořádat, je komunikovat s lidmi. Hlavním cílem bylo navrhnout a realizovat v první verzi webovou platformu, která bude podporovat spolupráci mezi lidmi v rámci pořádání společných akcí. Po procesu implementace byla webová platforma testována skupinou studentů z univerzity.

Klíčová slova: osamělost, sociální izolace, softwarová analýza, implementace webových stránek

Překlad názvu: Potka.me - Webová platforma pro spolupráci

Contents

1 Introduction 1

2 Issue overview 3
2.1 Term definitions . . . 3
2.2 European research on loneliness . 3
2.3 Chapter summary . . . 6

3 Ways of combating loneliness 7
3.1 Research on online articles . . . 7
3.2 Cooperation definition . . . 7
3.3 Cooperation platforms definition . 7
3.4 Survey . . . 8
3.4.1 Participants . . . 8
3.4.2 Questions . . . 8
3.4.3 Results . . . 8
3.5 Evaluation of cooperation platforms . . . 9
3.5.1 Social networks and messengers . . . 10
3.5.2 Event-oriented applications . . 15
3.6 Chapter summary . . . 16

4 Software analysis 19
4.1 Overview . . . 19
4.2 System requirements . . . 20
4.2.1 Functional requirements 20
4.2.2 Non-functional requirements . 23
4.3 User roles . . . 23
4.4 Use Case diagram . . . 23
4.4.1 Actors . . . 24
4.4.2 Time actors . . . 24
4.4.3 Unauthenticated user . . . 25
4.4.4 Authenticated user . . . 26
4.4.5 Host . . . 26
4.5 Process diagram . . . 27
4.6 Class diagram . . . 29
4.7 Chapter summary . . . 29

5 Technology stack specification 31
5.1 Back-end . . . 31
5.1.1 TypeScript . . . 31
5.1.2 Back-end framework . . . 32
5.1.3 Database . . . 33
5.2 Front-end . . . 35
5.2.1 Choosing a front-end framework . . . 36
5.3 Client-Server communication . . . 38
5.4 Deployment . . . 38
5.4.1 Back-end . . . 38
5.4.2 Front-end . . . 38
5.4.3 Deployment diagram . . . 39
5.5 Chapter summary . . . 40

6 Implementation process 41
6.1 Back-end . . . 41
6.1.1 Database setup . . . 41
6.1.2 Database schema definition . . 42
6.1.3 Implementation of the software requirements . . . 43
6.2 Front-end . . . 46
6.2.1 Used libraries and APIs 46
6.2.2 Responsiveness . . . 49
6.3 Deployment . . . 49
6.4 Chapter summary . . . 50

7 User Acceptance Testing 51
7.1 Participants . . . 51
7.2 Testing format . . . 51
7.3 Testing period . . . 52
7.4 Testing scenarios . . . 52
7.4.1 Test Scenario 1 - Browse events . . . 52
7.4.2 Test Scenario 2 - Local authentication . . . 53
7.4.3 Test Scenario 3 - Sign out . . . 53
7.4.4 Test Scenario 4 - Local registration . . . 54
7.4.5 Test Scenario 5 - Event creation . . . 54
7.4.6 Test Scenario 5 - Event registration . . . 55
7.5 User impressions . . . 56
7.6 Detected bugs . . . 56
7.7 Possible improvements . . . 56
7.8 Chapter summary . . . 57

8 The future state of the web platform 59

9 Conclusion 61

Bibliography 63

A Survey questions 65

B Abbreviations 67

C Folder structure of the attached CD 69

Figures

2.1 Loneliness by macro-region [2] . . . 4
2.2 Loneliness in the EU [2] 4
2.3 Impact on the different age groups experienced loneliness [2] 5

3.1 Facebook - Browse events. 10
3.2 Facebook - Create event. 11
3.3 Instagram - Create story 12
3.4 Telegram - Create poll 14
3.5 Eventbrite - All events 15

4.1 Actors UC diagram [author] 24
4.2 Time actors UC diagram [author] 24
4.3 Unauthenticated user UC diagram [author] 25
4.4 Authenticated user UC diagram [author] 26
4.5 Host UC diagram [author] 27
4.6 Enrollment request process diagram [author] 28
4.7 Class diagram [author] 29

5.1 Angular architecture [14] 33
5.2 Redis example 35
5.3 JSX Example [author] 37
5.4 Rendering comparison [author] . 37
5.5 Deployment diagram in the development environment [author] 39
5.6 Deployment diagram in the production environment [author] . . 39

6.1 Setup the Docker PostgreSQL environment [author] 41
6.2 Prisma model example [author] . 42
6.3 Prisma model example [author] . 43
6.4 NestJS resolver example [author] 44
6.5 NestJS service example [author] 44
6.6 Override the getRequest() functionality [author] 45
6.7 Pub/sub module [author] 45
6.8 PubSub use example [author] . . . 46
6.9 Application dashboard [author] . 47
6.10 Handle responsiveness with ChakraUI [author] 49

Tables

3.1 Evaluation of the cooperation platforms 17

7.1 Detected bugs 56

Chapter 1

Introduction

The motivation for choosing this topic was the gradual increase of lonely and socially isolated people in Europe and the Czech Republic. These mental health problems are very close to my heart because I have experienced them myself. Studying software engineering led me to the idea of implementing an application that would make it easier to find new acquaintances through the organization of joint events. It can help some people to cope with loneliness.

The main goal of the Bachelor thesis is to design and implement the first version of a web platform that will support the cooperation of multiple users through the organization of social events. An essential part of the assignment is testing the developed application on potential users and verifying that it satisfies their needs.

The second chapter is devoted to disclosing the problem of loneliness and social isolation based on several articles by the Joint Research Centre. The third chapter analyzes the methods of combating loneliness by gathering information from the Internet and interviewing students. The next chapter introduces system requirements and then visualizes and organizes them using UML and BPMN diagrams. Based on the system requirements, the fifth chapter defines specific technologies that will be used during the development and further maintenance of the application. The sixth chapter describes the application development itself, starting from the database configuration and ending with deployment. After the application became available online, I described the process of user testing, bugs found, and suggested improvements to the application. The eighth chapter describes the future of the application, and the ninth final chapter summarizes the entire work on the Bachelor thesis.

Chapter 2

Issue overview

Recently, there has been significant information technology progress in various spheres of human life. Thanks to social networks' proliferation, communication with other people has become easier than ever before. However, social isolation and loneliness remain common mental problems among Europeans. Since these terms will be used frequently in this section, it is necessary to understand their definition and differences.

2.1 Term definitions

According to the dictionary [1], *social isolation* can be defined as an absence of touch with people, either voluntarily or involuntarily. As a result, it may lead to aberrant behavioral and physiological changes. *Loneliness* is an emotional discomfort caused by being or perceiving oneself to be alone.

2.2 European research on loneliness

The problem of loneliness is becoming more and more acute in Europe. A study by the Joint Research Centre, European Commission's science and knowledge service, has proved this point.

In 2021, the center published a report [2] about loneliness in Europe and how the COVID-19 pandemic had affected the number of lonely people. The study is based on the European Social Survey's data that included two ways of measuring loneliness. The first one lies in asking individuals directly about their subjective feelings. The other includes studying specific determinants of loneliness, such as the frequency of meetings with relatives and acquaintances.

The report says that different regions of Europe have different percentages of socially isolated people and people who frequently feel lonely. The impact of the pandemic is most pronounced in northern Europe: the percentage has almost quadrupled. The other regions are at about the same level (Figure 2.1).

Moreover, the report provides more detailed country-specific statistics (Figure 2.2). In 2016, more than 10% of the population in the Czech Republic

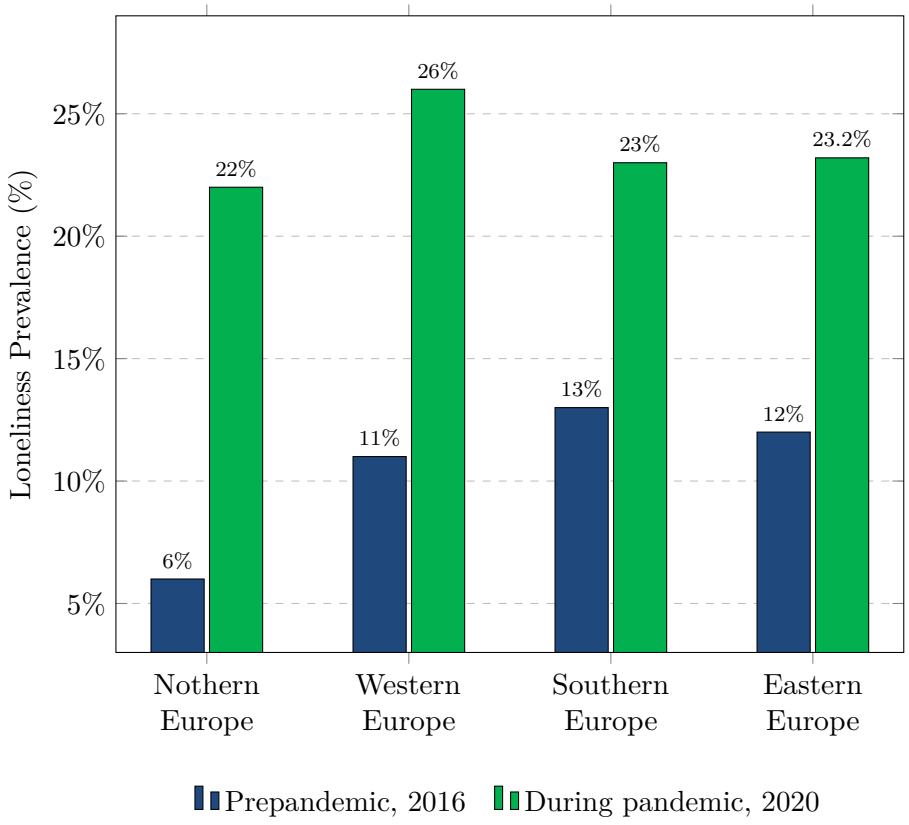


Figure 2.1: Loneliness by macro-region [2]

experienced loneliness; with the coronavirus pandemic, this figure increased by about 8-10%. In some countries, the rate increased even more.

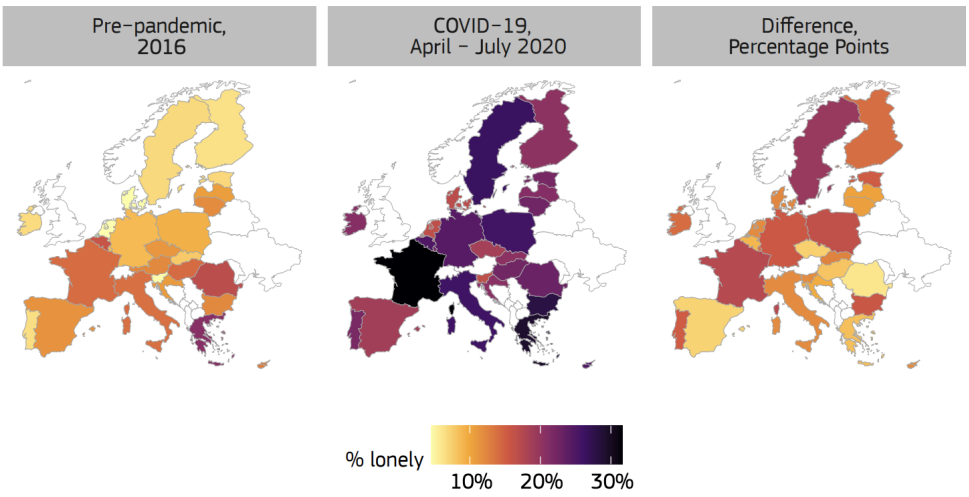


Figure 2.2: Loneliness in the EU [2]

COVID-19 has significantly affected not only the world economy, the work of states, various organizations and firms, but also the way people live and

their mental health. The most significant impact was observed in the 18-25 age group (Figure 2.3). Before the pandemic, about 9% of young adults frequently felt lonely. However, in 2020 it has increased by 26%. In the ages 24-46, 46-64, 65 and older, there is a lower increase in the number of people suffering from loneliness: 15, 11, and 8 percentage points respectively.

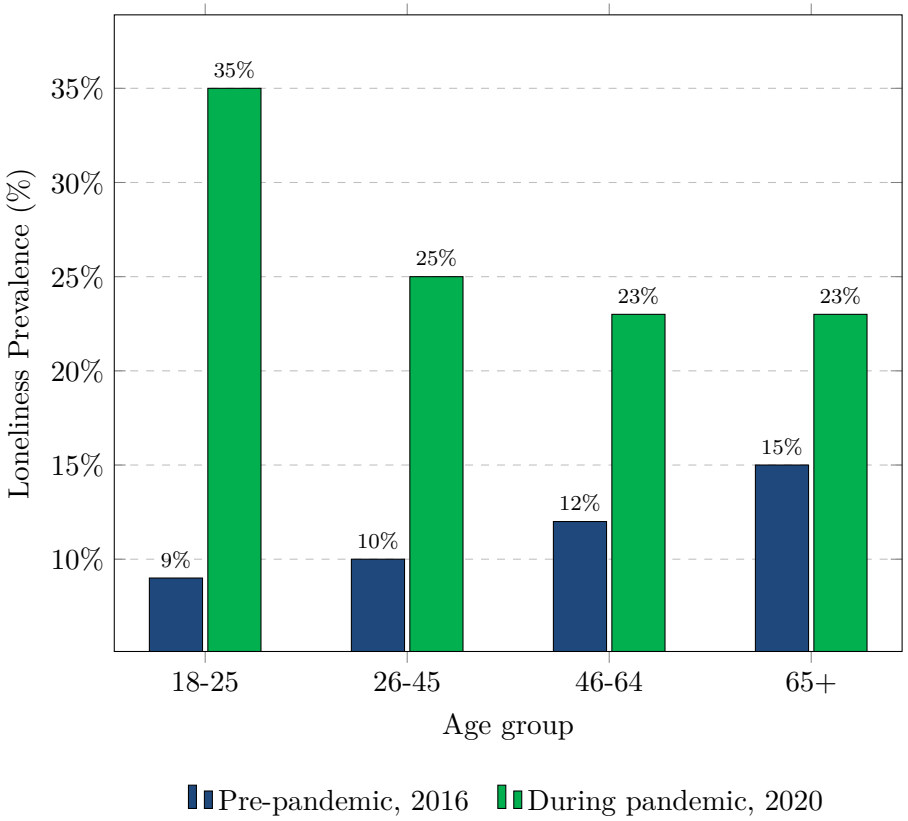


Figure 2.3: Impact on the different age groups experienced loneliness [2]

In addition, JRC researchers found that the increase in the number of lonely people was not directly related to their gender. Before and during the pandemic, the percentage points were nearly equal in all age categories. The slight difference was due to factors other than COVID-19: longer life expectancy and earlier age at marriage.

The last chapter of the report is devoted entirely to analyzing online media reports on loneliness and social isolation between 2018 and 2021 across the whole EU. Many articles were processed for the general sentiments and emotions contained in the articles. Researchers also took into account the examples of policy actions to combat loneliness. The quantitative analysis revealed that the volume of media reporting differs depending on the state. According to the report, the Czech Republic and Slovakia were countries where the volume of media reporting on the issue of loneliness and social isolation was meager. Most of the articles were focused on young people and women since the pandemic affected these groups the most [2].

2.3 Chapter summary

The first chapter described the problem of loneliness and social isolation in Europe. It was stated that loneliness is the feeling of being alone despite the number of people around, while social isolation is the lack of contact with society. These mental health problems have become widely discussed across Europe, and the JRC report proves this point. The following conclusions can also be drawn from this report.

1. The problem of loneliness is often perceived as a public concern.
2. The Czech Republic has one of the highest numbers of lonely people among the countries of the European Union. However, the measures taken by the government and the publicity in the online media leave much to be desired.
3. The most significant impact of the pandemic on the number of lonely people was seen in the 18-25 age group.

Chapter 3

Ways of combating loneliness

3.1 Research on online articles

A service from Google was used to get information on ways to combat loneliness. The search engine produced more than 29 million results for the query "ways to combat loneliness." By analyzing articles on the subject, it is possible to list the most common tips.

- Find a hobby to your liking - hobbies distract people from their daily problems and can set them in the right mood.
- Start doing sports - physical activity strengthens health and reduces depressive symptoms [3].
- Participate in charity work - the feeling of being needed and valuable improves mental state.

Interestingly, it was also observed that by mentioning these tips, the article authors emphasize that a much more significant benefit for the human's mental health is received when an individual cooperates with other people.

3.2 Cooperation definition

The term cooperation will be consequently used within the Bachelor thesis. Therefore, it is essential to understand its meaning. The Cambridge dictionary explains cooperation [4] as "the act of working together with someone or doing what they ask you." This is, however, a general definition that needs to be specified. Throughout the paper, the term cooperation will refer to the act of working together with someone to organize a joint activity where each participant can find new acquaintances, discover common interests, and develop friendships.

3.3 Cooperation platforms definition

Currently, many applications enable real-time communication between people via the Internet. The survey in the next section aims to gather a list of

platforms that allow users to create and register for events in one form or another. Such applications may have polls, group chats, stories, or other helpful functionalities for the event enrollment process. For this Bachelor thesis, such applications will be called *cooperation platforms*.

3.4 Survey

3.4.1 Participants

The analysis conducted in the section 2.2 showed that the most significant impact of the pandemic on the number of lonely people was seen in the 18-25 age group. For this reason, I decided to choose a group of 20 students in this age category.

3.4.2 Questions

The questions were divided into three groups according to their context. The first group includes questions regarding using social networks and messengers for organizing and participating in events. The second group has similar questions regarding web applications. In the final series of questions, students had to rate from one(the worst) to five(the best) the importance of having specific functionality in the developed cooperation platform.

The full survey with all the questions is available in the appendix A.

3.4.3 Results

The results of the survey are presented in the table *Survey on cooperation platforms* that can be found in the attached CD. According to the obtained information, most participants use social networks as their tools for organizing or registering for events. The popularity of individual platforms is illustrated in the following list.

1. Facebook - 18 (85.7%)
2. Instagram - 9 (42.9%)
3. Telegram - 8 (38.1%)
4. Whatsapp - 5 (23.8%)
5. VKontakte - 5 (23.8%)
6. Twitter - 0 (0%)

Respondents highlighted Facebook's event page with its optimized search engine, filtering, and sorting options. In addition, some people noted a good experience with personal event recommendations. On the other hand, some lacked the automatic import of events into the calendar and could not see events on a map.

Instagram users follow different accounts that post events through stories or posts. The application allows people to set notifications for a specific Instagram profile. After posting some content, a person will get a push notification. However, the search engine can be improved by providing filters, sorting, and searching by content.

Good optimization and privacy are the reasons why the respondents use Telegram. Polls make the process of registering for events more convenient because participants can also see those who registered, who have not decided yet, and who will not go.

Regardless of the platforms, participants ranked the given set of functionalities in order of importance. The results are listed from the most to the least desirable feature.

1. Searching, filtering, categorizing.
2. Push notifications about event updates.
3. Reactions (likes, comments, shares).
4. Map integration (view events on a map).
5. Followers/following lists.
6. In-app chat.
7. Text formatting in the event description.
8. Availability to add an event to a calendar.

3.5 Evaluation of cooperation platforms

The current section provides an overview of cooperation platforms. It evaluates their relevance to the organization of social events based on the functionality ranking from the previous section. Several platforms from the survey were taken into account: three most popular social networks among the students and two event-oriented applications.

3.5.1 Social networks and messengers

Social networks are very popular among people as they allow them to stay in contact with family, relatives and friends. The top three most popular social networks among the survey respondents are Facebook, Instagram, and Telegram.

Facebook

Available at: https://www.facebook.com/

Description
Facebook is the most popular social network in the world. Like other social networks, Facebook allows making new acquaintances, adding users to friends, and chatting with them in an internal chat room. Users can also publish stories and posts with their thoughts, emotions, and experiences.

However, the functionality of Facebook has long gone beyond the typical social network: the platform allows its users to play games, watch short and long videos and even create and participate in events.

A separate page (Figure 3.1) is dedicated to events where users can browse, search and filter events based on various criteria. In addition, there is a section that includes enrolled events, saved events, and invites for participation.

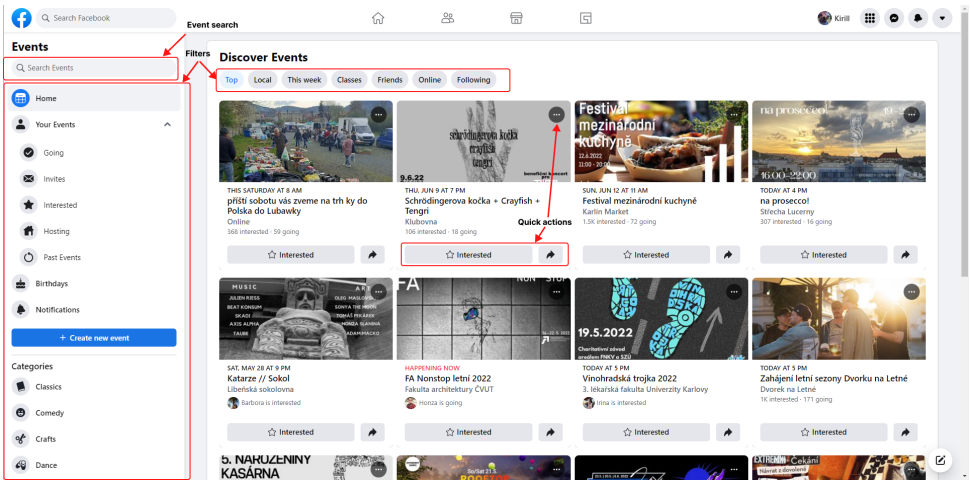


Figure 3.1: Facebook - Browse events.

It is also worth noting the event creation page (Figure 3.2). There is a multi-step form on the left side and a real-time event preview on the right side. The preview lets a host see how the event would look on desktop computers and mobile devices.

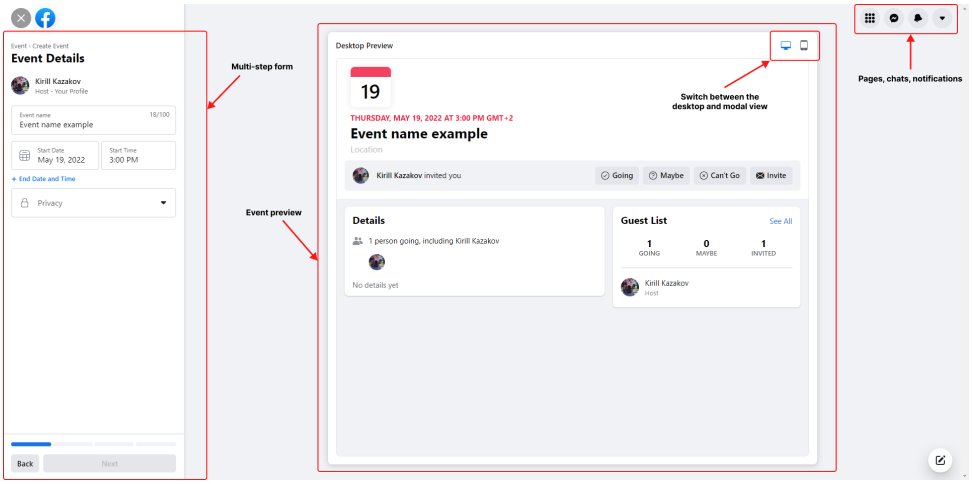


Figure 3.2: Facebook - Create event.

Supported functionality from the survey

- Searching, filtering, sorting.
  Platform provides an event-oriented search, categories and filters.
- Push notifications about event updates.
  Important event updates may be lost among friend invitations, recommended posts and other notifications.
- Post reactions.
  Comment, Interested, Going, Invite, Save, Share, Report.
- Followers/following lists.
- In-app chat.
  With one or multiple users
- Text formatting in the event description.
  Only in stories.

Unsupported functionality from the survey

- Map integration (view events on a map).
- Ability to add an event to a calendar.

Instagram

Available at: https://www.instagram.com/

Description
Instagram is the second most popular social network among the survey respondents. It is owned by the same company that created Facebook - Meta and is designed for posting photos and videos in the form of a post or story.

Instagram does not provide a specific section dedicated to events. However, it supports other valuable tools for a cooperation platform. One of them is story creation.

Figure 3.3: Instagram - Create story

Several types of attachments can be added to a story: customized text, links, drawings, polls, and many others. They can be used to emphasize important lines of text, provide additional information with the help of links, or even create a poll where users can register for an event (Figure 3.3).

Supported functionality from the survey

- Push notifications about event updates.
  Important event updates may be lost among new likes posts, recommendations, and other notifications.

- Post reactions
  Like, comment, save, repost.

- Followers/following lists.

- In-app chat.
  With one or multiple users.

- Text formatting in the event description.
  Only in stories.

Unsupported functionality from the survey

- Searching, filtering, sorting
  Platform provides only a general search by tags or accounts. It is impossible to search for a specific post by its title. Filtering and sorting are unavailable.

- Map integration (view events on a map).

- Ability to add an event to a calendar.

Telegram

Available at: https://www.telegram.org/

Description
Telegram is a popular messenger which enables its users to send messages in text, audio, and video formats. Users can create channels (only a few people can send messages) and groups (all people can send messages).

Similar to Instagram, Telegram does not support event management directly; however, users can attach polls to let others register for events. (Figure 3.4).

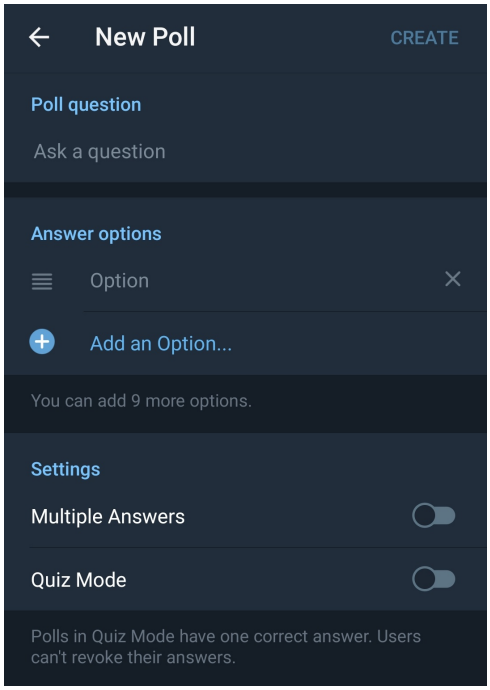


Figure 3.4: Telegram - Create poll

Supported functionality from the survey

- Push notifications about event updates.
  Platform provides with only message notifications, for this reason event updates may be lost among other messages.
- Message reactions
  Like, resend, reply.
- In-app chat.
- Text formatting in the event description.
  Platform does support text formatting, namely: bold, italic, underline, strikethrough, spoiler, link, monospace.

Unsupported functionality from the survey

- Searching, filtering, sorting
  Platform provides a general search only by groups. Filtering and sorting are unavailable.
- Followers/following lists.
- Map integration (view events on a map).
- Ability to add an event to a calendar.

3.5.2 Event-oriented applications

Event-oriented applications were created to enable people to participate in events and manage created ones. Such tools provide almost everything people need to browse, search, and register for events comfortably.

Eventbrite

Description
Eventbrite is the event management and ticketing platform that allows its users to view existing events or create their own.

Supported functionality from the survey

- Searching, filtering, sorting.
  Platform provides an event-oriented search, categories and filters.
- Push notifications about event updates.
- Post reactions
  Like, comment, share (to other media).
- Followers/following lists.
- Text formatting in the event description.
- Ability to add an event to a calendar.
- Map integration (view events on a map).
  Map is integrated in a search page (Figure 3.5).

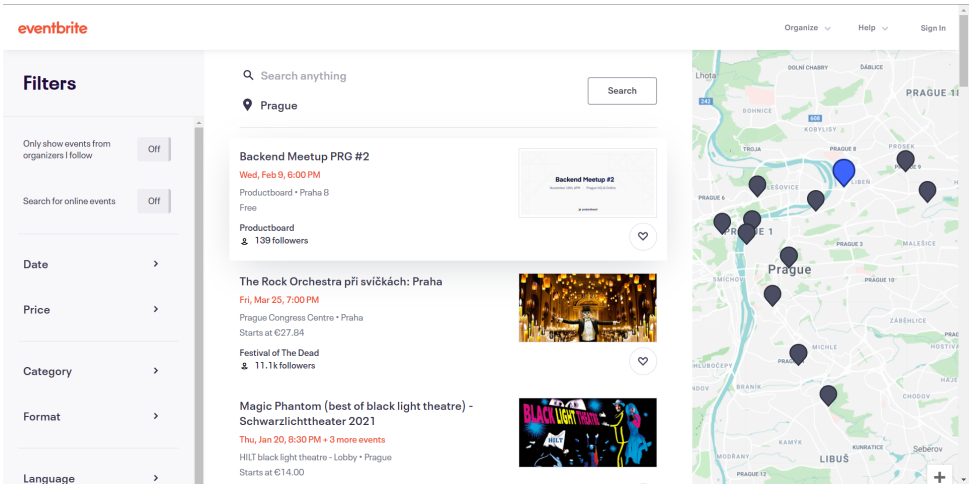


Figure 3.5: Eventbrite - All events

Unsupported functionality from the survey

- In-app chat.
  Participants are unable to contact their host through direct messages.

Meetup

Description
Meetup is the web platform designed for organizing social activities that can be online or in-person.

Supported functionality from the survey

- Searching, filtering, sorting.
  Platform provides an event-oriented search, categories and filters.
- Push notifications about event updates.
- Post reactions.
  Like, comment, share (to other media).
- Followers/following lists. A user can join a group and it will be visible in his profile. As a paid option, he can create his own group.
- Text formatting in the event description.
- Ability to add an event to a calendar.
- In-app chat.

Unsupported functionality from the survey

- Map integration (view events on a map).

3.6 Chapter summary

The main question that this chapter was supposed to answer is how to deal with loneliness. Communication with other people turned out to be the most effective way. Many different applications facilitate communication and organization of meetings between people in one way or another. Therefore, I surveyed students on their favorite communication channels in the context of organizing events. Based on the results obtained, I compared the three most popular social networks (they included one messenger) and two platforms directly focused on events. Table 3.1 summarizes the functionalities that are supported or not supported by the applications mentioned above.

Table 3.1: Evaluation of the cooperation platforms

	Social networks and messengers			Event-oriented	
	Facebook	Instagram	Telegram	Meetup	Eventbrite
Searching, filtering, sorting	✓	✗	✗	✓	✓
Push notifications	✓*	✓*	✓*	✓	✓
Reactions (likes, comments, shares)	✓	✓	✓	✓	✓
Map integration (view events on a map)	✗	✗	✗	✗	✓
Followers/following lists	✓	✓	✗	✓	✓
In-app chat.	✓	✓	✓	✓	✗
Text formatting in the event description	✓*	✓*	✓	✓	✓
Adding an event to a calendar	✗	✗	✗	✓	✓

✓- supported
✓* - partially supported
✗- not supported

As expected, the event-oriented platforms support most functionalities from the survey, while social networks are more limited in functionality.

Chapter 4

Software analysis

The previous chapter compared the functionality of existing cooperation platforms. It concluded that there is not yet a platform that fulfills all the requirements of the respondents from the survey. Therefore, I decided to implement my application to simplify the organization and registration for events. Having an easy option of meeting new acquaintances will help people start socializing and thereby fight loneliness.

The application will aim at simplicity and ease of use. All the application functions will focus on the context of organizing or participating in events. The platform will not be a social network and encourage people to communicate more in real life than on the Internet.

4.1 Overview

The chapter aims to create a solid base for the cooperation platform. At first, it will only have basic functionality that will be enough to engage potential users and satisfy their basic needs for the application. This functionality will be improved and extended in future releases after gathering first feedback. Frank Robenson came up with this development technique in 2001 and called it *Minimum Viable Product* [5].

The software analysis starts with a system requirements specification followed by several UML diagrams, namely:

- **Use Case diagram** demonstrates how the system will interact with specific actors without specifying implementation details.
- **Process diagram** helps to clarify complex processes within the system.
- **Class diagram** illustrates entities across the system and relations between them.

During the creation of the above mentioned diagrams I was guided by the book "Destilované UML" by Martin Fowler[6].

4.2 System requirements

A set of system requirements specifies what should be implemented in the final product. There are two types of requirements:

- **Functional requirements (FR)** describe the system behavior or, another words, what the system should do.
- **Non-functional requirements (NFR)** define a list of system properties and constraints.

Each requirement may also have metadata to describe some extra details. This information may include complexity, deadlines, resources etc. I decided to add a *priority* attribute to every requirement, because it will help me to organize the implementation process in future. I have prioritized requirements according to the "MoSCoW" set of criteria [7].

- Must have - M
  Tasks and requirements with the highest priority. They must be delivered and applicable to the product, otherwise the product will not be released.
- Should have - S
  Essential requirements that are not of the highest priority but still must be delivered.
- Could have - C
  Desirable requirements will be satisfied only if there are resources and time.
- Want to have - W
  The least critical requirements can be ignored or postponed until the next releases.

4.2.1 Functional requirements

The functional requirements were compiled based on the survey results described in the section 3.4. For convenience, the functional requirements were grouped into several sets: User, Event, Notifications, and Follows.

User

FR-1 Local sign up (M)
The web application will enable users to sign up with name, username and password.

FR-2 Sign up with SSO (C)
The web application will enable users to sign up with SSO.

FR-3 Local sign in (M)
The web application will enable users to sign in with username and password.

FR-4 Sign in with SSO (C)
The web application will enable users to sign in with SSO.

FR-5 Sign out (M)
The web application will enable users to sign out.

FR-6 View profile of a specific user (C)
The web application will enable users to view users‘s profiles.

FR-7 Block a user (W)
The web application will enable users to block other users.

FR-8 View account information (C)
The web application will enable users to view their account information.

FR-9 Update account information (C)
The web application will enable users to update their account information.

FR-10 Send direct messages (W)
The web application will enable users to send direct messages to others.

Event

FR-11 Display all ongoing events (M)
The web application will display events in a grid and on a map.

FR-12 Filter events (S)
The web application will enable users to filter events by starting date, category and city.

FR-13 Detailed information about events (M)
The web application will display detailed information of an event on the dedicated page. Information will include title, description, starting date, end date, host contacts and an image.

FR-14 Search events (M)
The web application will enable users to search for an event by its title.

FR-15 Create events (M)
The web application will enable users to create their own events by providing a title, a starting date and a location. Optionally, users may also add a description, end date and an image.

FR-16 Delete / modify of events (C)
The web application will enable users to delete and update their own events.

FR-17 Save events to bookmarks (M)
The web application will enable users to save events to bookmarks.

FR-18 View bookmarks (M)
The web application will enable users to view their bookmarks.

FR-19 View created events (M)
The web application will enable users to view their created events.

FR-20 Comment on events (W)
The web application will enable users to comment on events.

FR-21 Reply to comments (W)
The web application will enable users to reply to comments.

FR-22 Like comments (W)
The web application will enable users to like comments.

FR-23 Sharing events on social media (W)
The web application will enable users share events on social media.

FR-24 Request for a participation (M)
The web application will enable users to send participation requests.

FR-25 Accept / decline a participation request (M)
The web application will enable users to accept and decline participation requests.

Notifications

FR-26 View a list of notifications (S)
The web application will enable users to view a list of notifications.

FR-27 Notify on a new participation request (S)
The web application will notify users when they receive a participation request.

FR-28 Notify on a new participation response notification (S)
The web application will notify users when they receive a participation response.

Follows

FR-29 Follow a user (W)
The web application will enable users to follow other users.

FR-30 Unfollow a user (W)
The web application will enable users to unfollow other followed users.

FR-31 Show following users participating in a specific event (W)
The web application will enable users to view a list of following users participating in a specific event.

4.2.2 Non-functional requirements

The non-functional requirements were compiled based on the previous experience in software development.

NFR-1 Security (M)
The web application shall provide secured information only to authorized users.

NFR-2 Internationalization (C)
The web application is available in multiple languages.

NFR-3 Usability (M)
The web application shall provide an intuitive interface to all people despite of their age and education level.

NFR-4 Responsiveness (M)
The web application shall provide a convenient use from desktop and mobile devices.

NFR-5 Archiving past events (W)
The web application shall archive past events every month.

NFR-6 Censorship of profanity (W)
The web application shall search for profanity and remove explicit content every day.

NFR-7 Search engine optimization (S)
The web application shall be optimized for search engines.

4.3 User roles

The system will offer a wide range of different functionalities. However, not all the functions will be available to every user. Below is a list of user roles that will be implemented in the first version of the application.

- Unauthenticated user
  A person who is not logged into the system,
- Authenticated user
  A person who is logged into the system,
- Host
  An authenticated user who has created at least one event.

4.4 Use Case diagram

The section 4.3 listed different roles that will be available across the application. However, it is not yet defined how a user with a specific role can interact

with the system. The following section will cover both the relations between the actors and use cases of each individual actor. The Use Case diagram was also used to visualize and organize the system requirements described in the section 4.2

4.4.1 Actors

The diagram 4.1 shows the main actors in the system: 3 actors represent users with the certain roles, 2 others are related to time. The arrow drawn from the organizer to the user means that the organizer inherits all the functions available to the authenticated user. Moreover, the organizer has access to other functions available only to his role. In order to emphasize the functions available to the organizer, I decided not to add the functions of the authenticated user to the Host diagram.

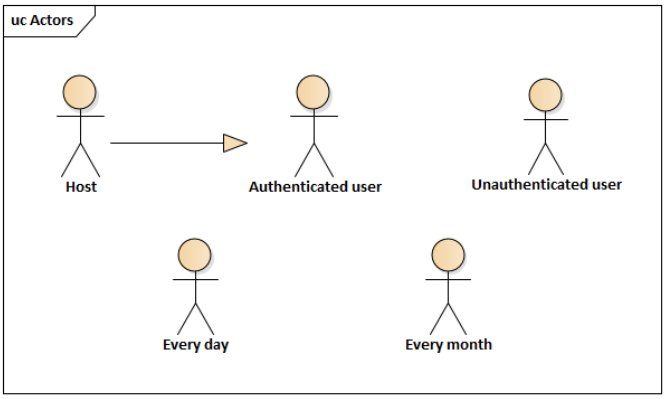
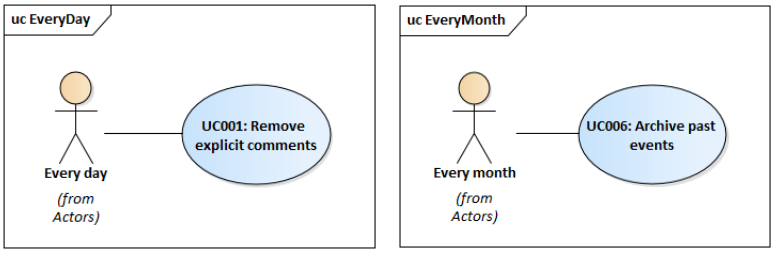


Figure 4.1: Actors UC diagram [author]

4.4.2 Time actors

Figure 4.2 illustrates use cases for time actors. Figure 4.2a relates to *NFR-6*, Figure 4.2b relates to *NFR-5*.



(a) : EveryDay actor **(b) :** EveryMonth actor

Figure 4.2: Time actors UC diagram [author]

4.4.3 Unauthenticated user

At first glance, it might seem that the authenticated user can inherit the use cases of an unauthenticated user. However, as shown in Figure 4.3, some use cases will only be available to this actor, namely registration and authentication. Only the unauthorized user will be able to create a new account or log in to an existing one.

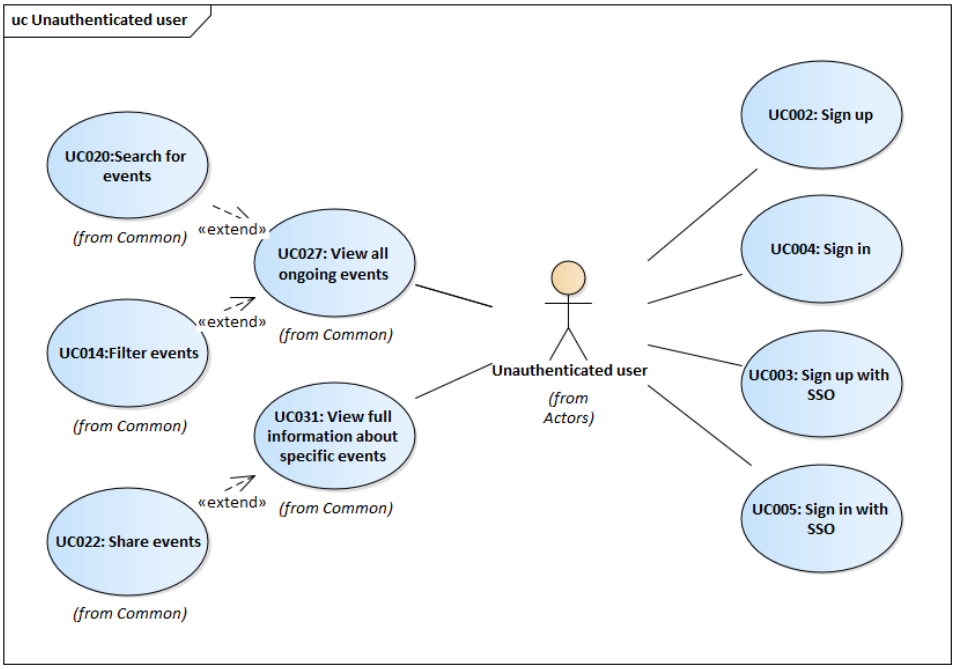


Figure 4.3: Unauthenticated user UC diagram [author]

4.4.4 Authenticated user

The diagram in Figure 4.4 shows the use cases of the central user role, the authenticated user. In the first version of the application, only the host inherits the authenticated user functionality. However, in the future, there will potentially be other roles that can also inherit this functionality, such as administrator or moderator.

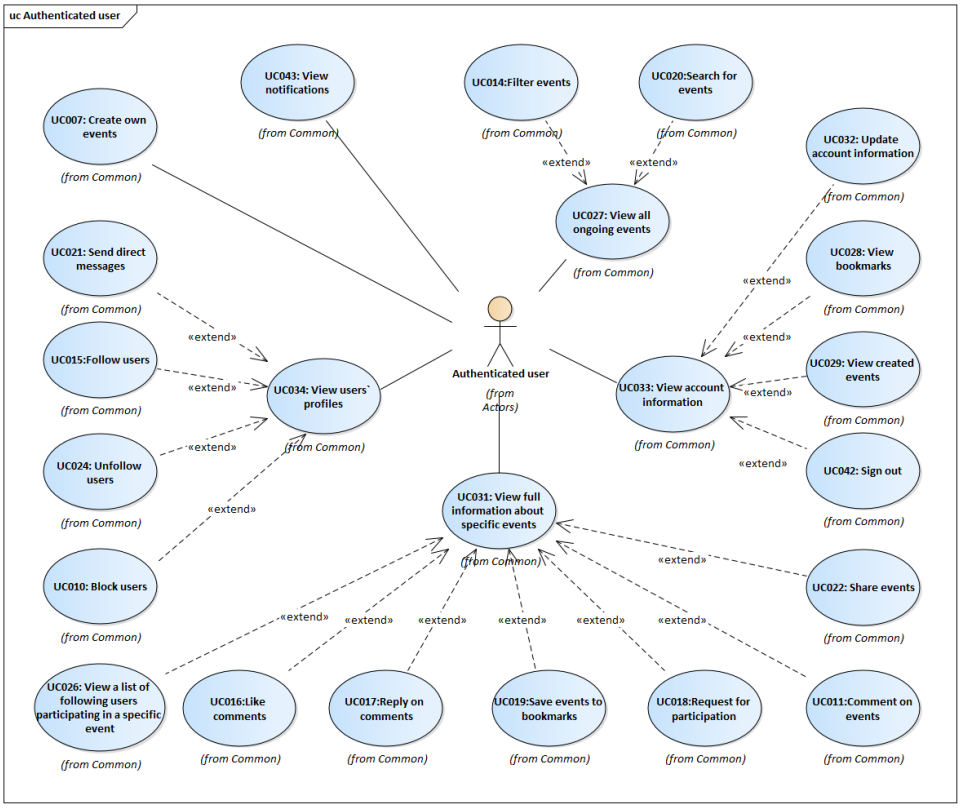


Figure 4.4: Authenticated user UC diagram [author]

4.4.5 Host

Figure 4.5 demonstrates all the use cases for the actor Host. In addition to the functionality available to the ordinary user, the organizer can also delete/change events, accept/reject requests for participation, view the list of participants, block and mute them.

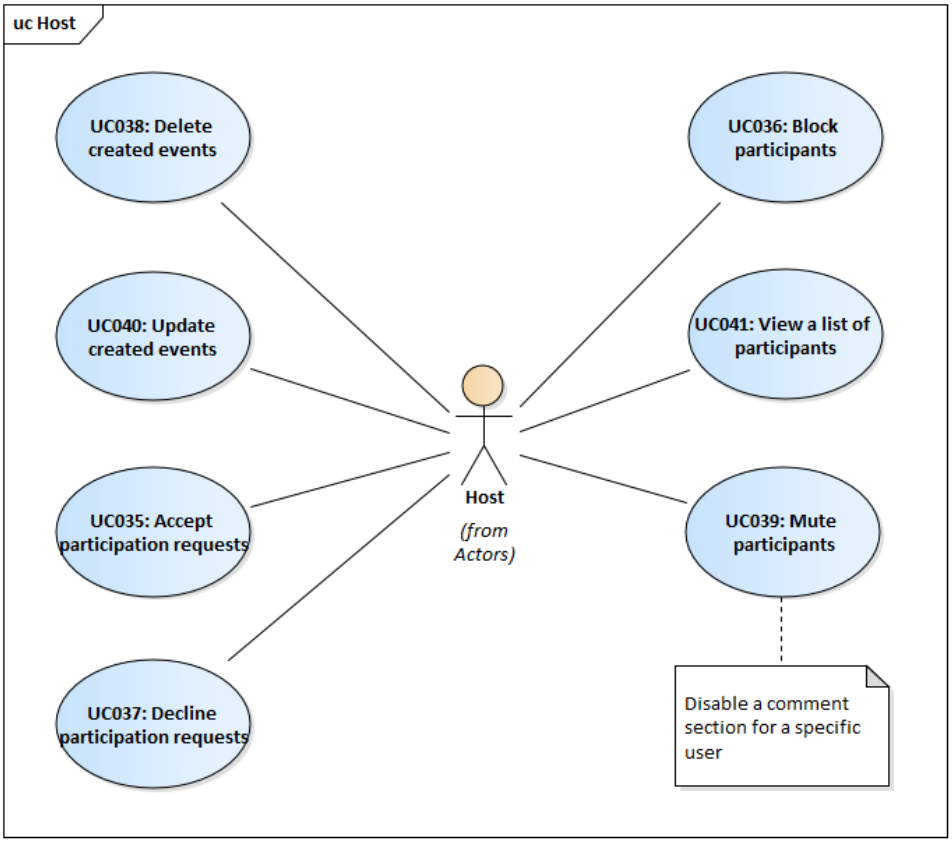


Figure 4.5: Host UC diagram [author]

4.5 Process diagram

To visualize application processes, I have used BPMN[1] 2.0 [8], which stands for a graphical notation of the business process model. This specification has become a standard for BP diagrams and is intended to be used directly by stakeholders [9].

The *Potka.me* web application focuses on simplicity and usability, so the vast majority of functions is trivial and does not require a process diagram representation. However, some key features have more complex logic. One of them is sending a participation request. To have a solid grasp of the entire working process I decided to represent it in the form of the process diagram.

Enrollment request

Figure 4.6 demonstrates the start of the process when a user sends a participation request. A host of the related event receives a notification of a newly

[1] Business Process Model and Notation

created request, reviews it, and sends a response back. Subsequently, the user obtains information on whether he or she is enrolled in the event or not. It is also worth keeping in mind the situation when the host will not check the enrollment request. In this case, the enrollment request will expire when the event starts.

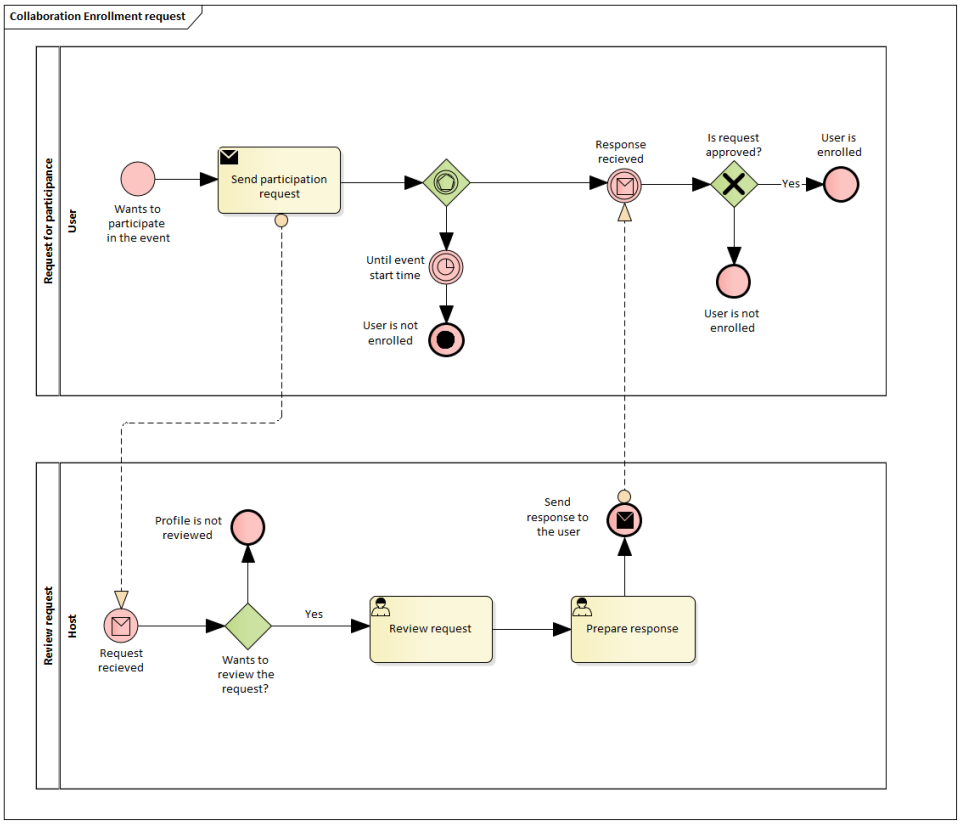


Figure 4.6: Enrollment request process diagram [author]

4.6 Class diagram

Class diagrams demonstrate entities in the system, their attributes, and relations between each other. Figure 4.7 contains the class diagram of the developed web application. There are two main entities: User and Event. According to the requirements FR-7 and FR-29, users will be able to gain followers and block other users. To show this, I used recursive associations. There are five relationships between User and Event. A one-to-many relationship means that a user can create zero to many events. Many-to-many relationships mean that a user can participate, save, comment, or send a participation request to multiple events. At the same time, an event can have different comments from different users, many participation requests, participants, and people who saved this event.

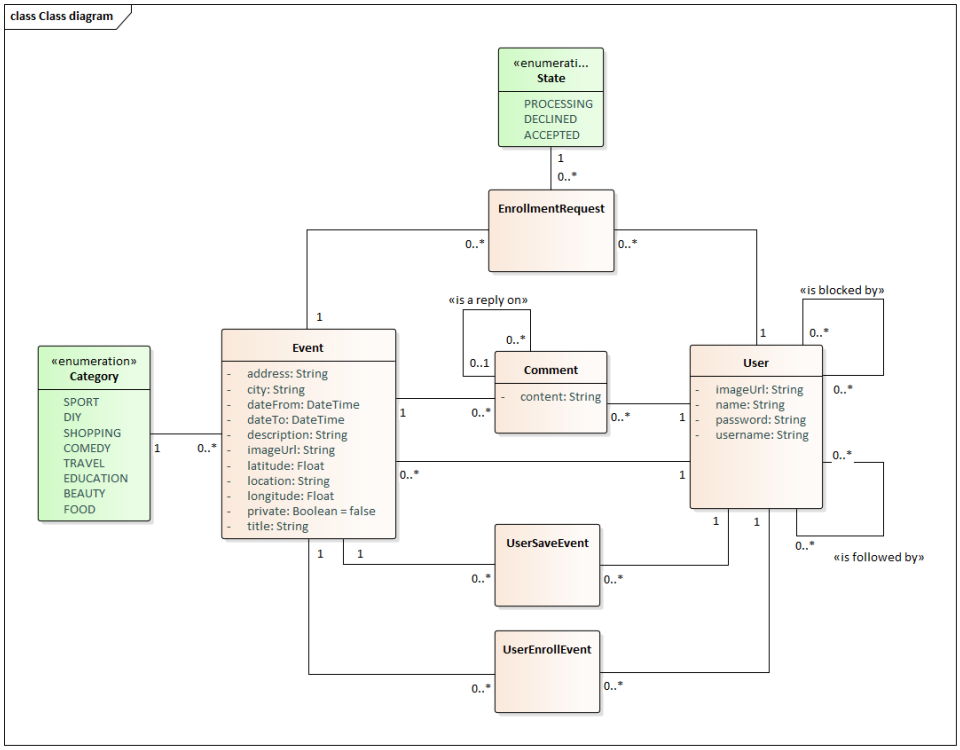


Figure 4.7: Class diagram [author]

4.7 Chapter summary

The first version of the developed platform will support the cooperation of multiple users through the organization of joint events. The main focus will be on improving the user experience in searching and organizing events. This can be achieved by satisfying the software requirements with a priority of

Must Have and Should Have. Other requirements for the first release are optional and will be implemented if there is time and resources.

The next chapter will focus on choosing the most suitable technologies that will be used during the development process.

Chapter 5

Technology stack specification

This chapter contains information about the choice of technologies used during the development and further maintenance of the project. The application's main functionality is divided into back-end and front-end.

The back-end includes a server (a machine receiving requests), an application (run on a server, processes requests, optionally operates with a database, and sends responses), and a database (an organized collection of data) [10]. The front-end is the layer where a user interacts with a system. It sends requests to the server, receives, processes, and depicts server responses.

5.1 Back-end

This section is devoted to the choice of the main technologies that will be used on the back-end side of the application. At the beginning, I would like to refer to the Stack Overflow survey [11], since it covers numerous topics in the tech world, including the top of programming, scripting, and markup languages.

What is noticable is that the JavaScript language remains the most preferred and leading programming language nine years in a row. The reason is probably that this language is universal: it can be used to write both the server side of the web application and the client side. Unfortunately, during the previous years at the university, I did not have enough time to learn modern back-end technologies using javascript. The Bachelor thesis gave me a great opportunity to learn new things, and I decided to stick to technologies that use the power of JavaScript.

5.1.1 TypeScript

JavaScript is a dynamically typed language and it simply means that it will convert a type of a variable based on its value. From one point of view, it may save some time while programming, from another - a programmer may face runtime errors caused by *coercion* [12]. To eliminate most of the related problems, I prefer to use TypeScript over JavaScript.

TypeScript is a superset of JavaScript and, as its name indicates, provides an optional type system. Consequently, .js files can be safely renamed to .ts

and the code will still be compiled into valid JavaScript.

Along with the elimination of coercion problems, there are other reasons to use Typescrpt that are worth mentioning.

- Programmers can use lambda functions, spread operator, object destructuring, and other modern JavaScript features remaining compatible with older browser versions;
- TypeScript gives a great developer tooling. Mistakes are catched while writing a code in a text editor. This improves code quality and its reliability.

With all this said, the TypeScript programming language will be used across the developed web application.

5.1.2 Back-end framework

The programming language is defined, and the next step is to choose the right JavaScript back-end framework. Having no experience working with any of them, I conducted a research on existing solutions according to the following criteria:

1. number of GitHub starts,
2. number of GitHub issues,
3. documentation quality and existing learning courses,
4. TypeScript support,
5. scalability and,
6. modularity.

For this application, I chose NestJs. Nest is a framework for building efficient and scalable server-side applications with TypeScript using Express[1] under the hood [13].

It provides both the REST API out-of-the-box and contains numerous modules to work with databases, handle security, caching, websockets, and many others.

Nest is shipped with a predefined architecture that is heavily inspired by the Angular framework (Figure 5.1). It helps to build maintainable, testable, and modular applications. As a project grows, each module can be developed at the same time by a separate team of developers and can be updated or scaled as needed without impacting the other parts of the application.

The above mentioned reasons inspired me to use this framework on the back-end side of the application.

[1] Express is a robust HTTP Server framework that provides a set of features for mobile and web development

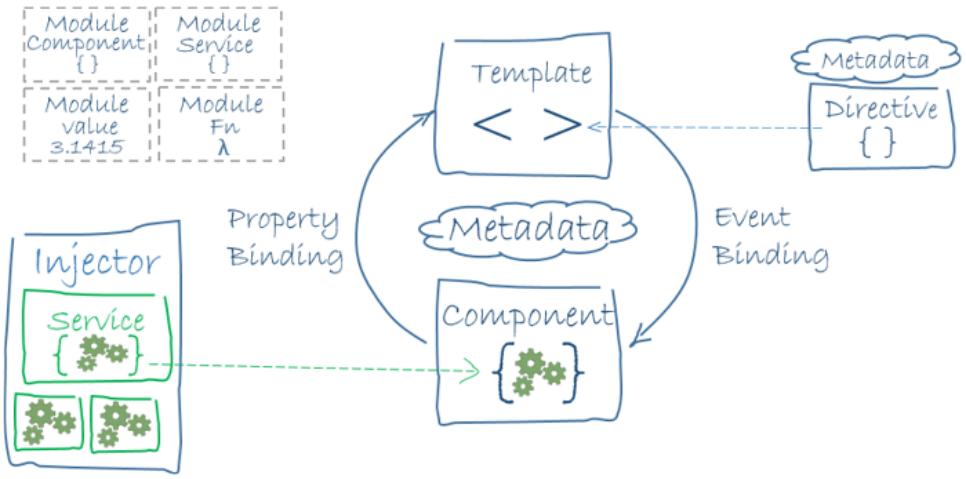


Figure 5.1: Angular architecture [14]

5.1.3 Database

Choosing a suitable database is vital while building software applications. There are many types of databases that are suitable for different use cases. To choose the best option for the currently developed web application, it is better to compare existing types and their popular solutions. According to the most recent surveys on databases held by Stackoverflow [11], the two most common types database types are relational and non-relational.

Relational databases

During the definition of relational databases, I was inspired by a certain amount of books and articles, specifically "The Relational Model for Database Management" [15] and "A Relational Model of Data for Large Shared Data Banks" [16] by an English computer scientist and inventor of the relational model for DBMS Edgar F. Codd.

Such databases are based on the relational model - an intuitive way of storing information in the form of tables (relations). Each table represents an unordered set of rows (tuples, records) of the same type. These rows include data of objects or entities that are stored in a database along with a unique identifier - the primary key. Rows can also be referenced in other tables using foreign keys - specific fields that link to the related primary keys. In addition to rows, the table also contains columns (attributes) that define a certain kind of data and fields - the actual values of these columns.

The Structured Query Language (SQL) is used to access and write data to the database. Therefore, such databases are also called SQL databases. The relational model is solidly based on two parts of mathematics: predicate logic and the theory of relations. Relational algebra provides various mathematical operations, including different types of joins that are commonly used while writing SQL queries. One or more SQL statements are also called *transactions*.

They must be ACID compliant or, in other words, be Atomic, Consistent, Isolated, and Durable to ensure data integrity. Imagine there is a movie theatre information system, and we need to retrieve information about customers to perform some other actions in the future. For this example, the inner join can be used as follows:

```
SELECT Orders.Email, Customers.*
FROM Orders
INNER JOIN Customers ON Orders.CustomerEmail = Customers.Email
```

This type of database is widely used in production nowadays. It is recommended for almost all applications with pre-defined database schema and where it is supposed to store structured data.

Examples of relational databases are Oracle, PostgreSQL, MySQL, MariaDB, and others.

For this particular project, the PostgreSQL database will be used because I had a practical experience with it during the development of various side projects and during some university courses.

Non-relational databases

The title itself tells that such databases store data not only in tabular form. Therefore, they can also be called NoSQL or Not Only SQL databases.

We can further divide non-relational databases into the following types:

1. Key-value
2. Document-oriented
3. Wide-column
4. Graph
5. Full Text Search Engines
6. Multi model

I have not used any other non-relational database during the study course except the key-value one. Thus, I chose it for the current application and described its best use cases below.

Key-value storages

The simplest type of database holds its data similar to objects in the JavaScript programming language. A storage element contains a key and a value that can be accessed using this particular key. The two most common use cases for this type of databases are caching and pub/sub.

A widely used example of key-value storage is Redis. The code in Figure 5.2 illustrates the basic usage of this storage.

The *Potka.me* website will use the Redis power for two reasons:

```
redis 127.0.0.1:6379> set mykey somevalue
OK
redis 127.0.0.1:6379> get mykey
"somevalue"
```

Figure 5.2: Redis example

- According to the requirements, the system will notify application users about specific event updates. Implementation will include the Pub/Sub module that is responsible for managing real-time communication between a publisher and its observers.

- It is also necessary to control if the user is authorized to enter a particular page. For that, it is required to send a request to the database to check if a user is logged in and if a user is obliged to access a page. Undoubtedly, the back-end can query the PostgreSQL database, but the use of Redis will ensure better performance.

Summary on databases

I will use PostgreSQL as a primary database because it is based on the relational model, and there are available relational algebra operations. PostgreSQL is widely used and is almost the most popular solution among developers who have participated in the Stackoverflow.
Redis storage will also be used in the project for the pub/sub module and for retrieving currently authenticated users. The main reasons are performance improvements and scalability.

5.2 Front-end

Even though I have chosen TypeScript on the back-end side, many other languages can be used: Python (Flask, Django, Fastify), Java (Spring Boot), C++, Rust, C#, and others. However, things are different on the client-side. The first programming language that comes to mind while speaking about front-end development is JavaScript.

The tech world has examples of frameworks and technologies like WebAssembly enabling programmers to write code in their preferred language. However, none of them except JavaScript has a direct access to the Document Object Model. This leads to code workarounds and spending hours on solving issues related to the inability of manipulation with DOM. Therefore, for this particular moment, JavaScript provides the most convenient experience for writing the client-side of a web application.

5.2.1 Choosing a front-end framework

Frameworks speed up any development, and it is mostly recommended to use one. Thus, the client-side of the application will also be based on a particular technology. A list of the most used frameworks and libraries includes React.js, Angular, and Vue.js. However, I only have experience writing a code in React, and to deliver the MVP in time, I decided to stick with it.

React.js

React.js is technically a library and is considered the most popular technology for building the front-end. It is built and maintained by developers from the Meta team. React is designed to be minimalistic. That is why a programmer will need to consider the following:

1. React does not support features like routing, complex state management, animations, and others out-of-the-box. Modules like react-router, react-redux, and react-spring can compensate for missing components;

2. It is also worth noticing that the library does not enforce a strict folder structure. Therefore, a developer should be aware of keeping the application scalable, testable, and modular.

3. There is only one type of page rendering available - client-side rendering (CSR). In other words, page content will be rendered when a user enters the page. To access other ways of site generation, such as server-side rendering (SSR), static site generation (SSG), and on-demand site generation (ISR or DSG), it will be better to take a look at Next.js or Gatsby. These frameworks are built on React and provide a wider range of developer tools.

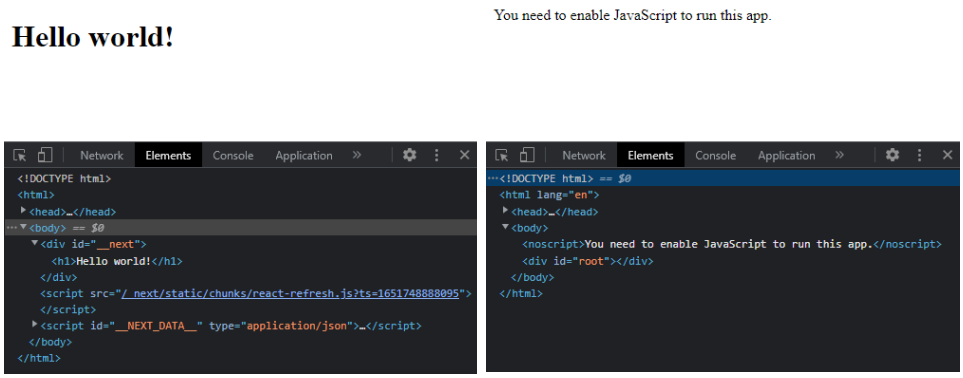
React also provides JSX or JavaScript Syntax Extension (Figure 5.3). It can be understood as HTML that has been extended by an additional syntax allowing to embed a JavaScript code inside HTML. A code that is written in JSX is located in a file with a specific extension - .jsx for JavaScript and .tsx for TypeScript.

```jsx
import * as React from 'react';

export default function App() {
  const emojis = ['❤️', '✨', '🔥'];
  return (
    <div>
      <h1>Emojis!</h1>
      <ul>
        {emojis.map((emoji, i) => (
          // {emoji} will be replaced with a value thanks to the JSX
          <li key={i}>{emoji}</li>
        ))}
      </ul>
    </div>
  );
}

```

Figure 5.3: JSX Example [author]

As I have already mentioned above, React provides only one type of site generation. It is also not optimized for the search engine optimization (SEO), which is crucial for the cooperation web platform (NFR-7). Therefore, my choice fell on Next.js.

Next.js, by default, pre-renders every page, meaning that HTML is generated in advance instead of inserting it with JavaScript when a user has entered a page. Thanks to it, web crawlers, including some from Google, can analyze a page content and index a website in the search [17]. To demonstrate the difference between content pre-rendering and client-side rendering, I disabled JavaScript in my browser and took the two screenshots from the Chrome DevTools.



(a) : NextJS app with disabled JavaScript **(b) :** React.js app with disabled JavaScript

Figure 5.4: Rendering comparison [author]

The picture 5.4a represents the application built with NextJS. It is noticeable that this page displays the content despite disabled JavaScript. Some styles were not loaded, but the main information on the site is still visible to a user. The web application on the picture 5.4b, however, asked a user to

enable JavaScript in order to reach the content. This example clarifies the difference between rendering with NextJs and React.

5.3 Client-Server communication

According to the system requirements defined in the section 4.2, website users will be able to search, save or create their own events. For this reason, it is necessary to support client-server communication. Specifically, in my project, the term *client* means the Next.js web application, and the *server* is the NestJS application with an exposed API.

Modern servers expose two kinds of APIs: REST API and GraphQL API. For this particular project, I will build a GraphQL API, mostly because of the type safety and auto-generated documentation. The disadvantage of this solution may be a lack of caching mechanism by default. However, as it was described in the section 5.1.3, Redis is also used for caching and can compensate for it.

Web development has many examples of awesome GraphQL tools that can be used within the project. Two of them are Apollo Client and GraphQL Code Generator. The synergy of these libraries will speed up the development process by generating TypeScript types and special React hooks[18].

5.4 Deployment

Developing an application on a local machine is standard practice. However, to make the website accessible to other people, there is an option to deploy it to a hosting provider. When choosing a suitable hosting provider, there are a few requirements to consider:

1. the developed application uses two rendering strategies: SSG and SSR;
2. a hosting provider must support PostgreSQL and Redis technologies;
3. the website is a proof of concept, so a hosting provider must offer a free plan.

5.4.1 Back-end

Based on my previous experience hosting server-side applications, I still believe that Heroku remains a leader among other providers, including Amazon AWS, Microsoft Azure, Digital Ocean, and others. It offers a free plan with up to 1000 hours of hosting and various add-ons, including a free PostgreSQL database and Redis storage.

5.4.2 Front-end

Next.js is the primary framework for developing the client-side of the project. It is built and maintained by Vercel, which provides a hosting service for

static sites. I had a great experience using it because it provides a convenient and user-friendly interface, offers a straightforward deployment workflow, has powerful development tooling, and has the best integration with Next.js.

5.4.3 Deployment diagram

The deployment diagram shows the hardware and software components and their relationships. Figure 5.5 illustrates the deployment diagram in the development environment, Figure 5.6 - in the production environment. The communication between nodes goes as follows:

1. A user device (smartphone, laptop, desktop, and others) sends a request to the server through a web browser.
2. The server handles the request and, if necessary, sends the request further to the database.
3. The server sends a response to the client.

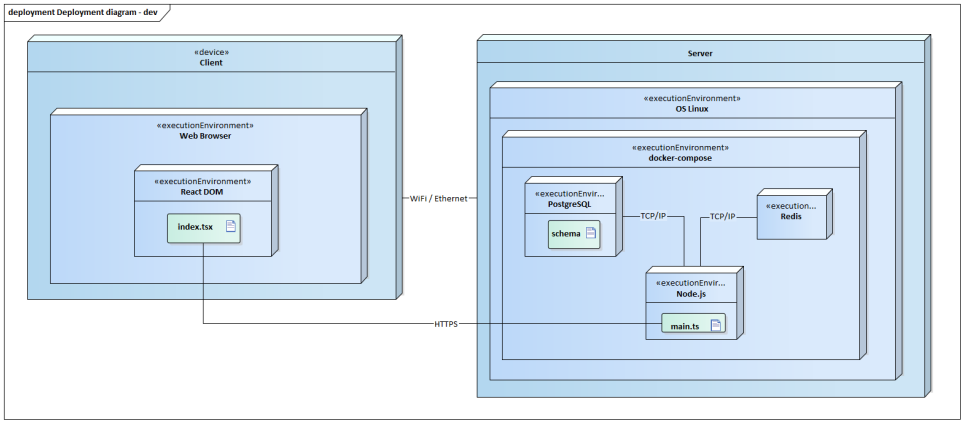


Figure 5.5: Deployment diagram in the development environment [author]

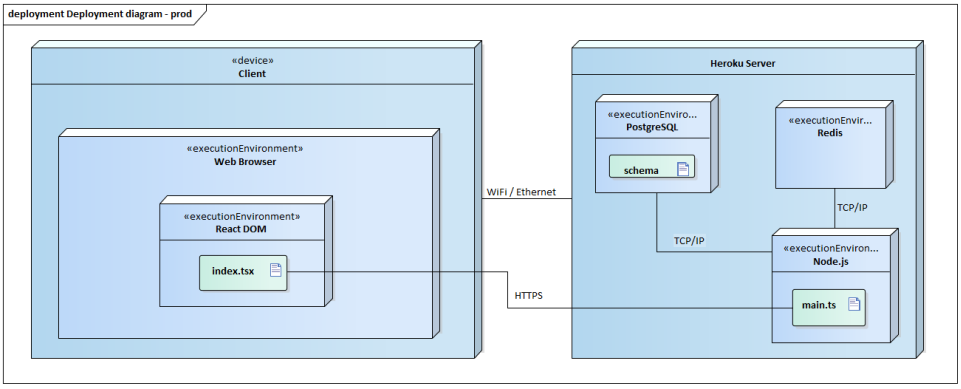


Figure 5.6: Deployment diagram in the production environment [author]

5.5 Chapter summary

The chapter described all the technologies and programming languages that will be used during the web application development.

Front-end

- Programming language: TypeScript (type safety, powerful tooling, TS[2] compiler).
- Framework: Next.js (support of various rendering types, SEO, rich API).

Back-end

- Programming language: TypeScript (type safety, powerful tooling, TS compiler).
- Framework: NestJs (predefined Angular-like architecture, actively maintained).
- Primary database: PostgreSQL (relational database, popularity).
- Additional storage: Redis (performance, popularity, pub/sub, caching).

Client-server communication

- GraphQL - type safety, automatic schema generation, powerful tools, no over/under-fetching.

In addition, it was mentioned that the back-end would be deployed to Heroku as it provides a free plan, supports needed technologies, and has a user-friendly web interface. Vercel was chosen for the front-end deployment because Next.js is built and maintained by the Vercel developers.

[2] TS - TypeScript

Chapter 6

Implementation process

The motivation for the sixth chapter is to describe the whole process of the project development. The source code of the application will be publicly available. To ensure that everyone can build and run the application, I will cover all the development stages, starting from the schema definition and ending with the application deployment to the chosen hosting providers.

6.1 Back-end

This section describes the most important details of developing the server part of the application.

6.1.1 Database setup

The first step is to configure the PostgreSQL database. I chose to setup the Docker[1] PostgreSQL environment. For that I added these lines of code into the `docker-compose.yml`:

```
1   version: '3'
2   services:
3     db:
4       image: postgres:12
5       ports:
6         - 5432:5432
7       environment:
8         POSTGRES_USER: pg_username
9         POSTGRES_PASSWORD: pg_password
10        POSTGRES_DB: pg_db_name
```

Figure 6.1: Setup the Docker PostgreSQL environment [author]

By running `docker-compose up`, Docker will install needed files and start the PostgreSQL database.

[1] Docker is an open platform for developing, shipping, and running applications. Available at: https://www.docker.com/

6.1.2 Database schema definition

After setting up the database, I needed to choose a suitable ORM. According to npm trends [19], the most commonly used tools for working with databases are Sequelize, Knex, TypeORM, and Prisma. Despite the tight deadline, I tried the last two and opted for Prisma, as its developers intentionally added another level of abstraction for easy learning and more comfortable, intuitive use.

After entering the `npx prisma init` command, Prisma generated a specific file called *schema.prisma*. It contains the database connection details along with models. These models define a set of fields and relations between models. Figure 6.2 demonstrates the User model used in the application.

```
14  model User {
15    id        String   @id @default(uuid())
16    createdAt DateTime @default(now())
17    updatedAt DateTime @updatedAt
18
19    username  String  @unique
20    password  String
21    name      String
22    imageUrl  String?
23
24    // events
25    eventCreatedMany   Event[]           @relation("UserCreatesevent")
26    eventSavedMany     UserSaveEvent[]
27    eventEnrolledMany  UserEnrollEvent[]
28
29    eventRequestedMany EnrollmentRequest[]
30
31
32    // Comments
33    commentCreatedMany Comment[] @relation("UserCreatesComment")
34    commentLikedMany   Comment[] @relation("UserLikesComment")
35
36    // Followers
37    followedBy User[] @relation("UserFollows", references: [id])
38    following  User[] @relation("UserFollows", references: [id])
39
40    @@map("users")
41  }
```

Figure 6.2: Prisma model example [author]

By looking at it, we may notice a specific structure of fields:

1. **Field name** represents a column that has a similar name by default. Optionally, the column name can be changed using the `@map()` attribute.
2. **Field type** specifies a type of values that can be inserted.
3. **Set of attributes and functions separated with spaces** change a behavior of related fields and models.

When the database schema was defined, I synced it with an actual database using the `npx prisma db push` command.

GraphQL configuration

NestJS, by default, does not support GraphQL. For this reason, it was imported as a module in the *app.module.ts* file.

6.1.3 Implementation of the software requirements

The application database is now ready to operate with data; however, we need to implement the back-end to communicate with it. NestJS tries to promote and recommends following a specific architecture, also known as the *3 Layered Architecture*. This architecture is well-described in the article *Bulletproof node.js project architecture* by the Node.js developer Sam Quinn [20]. During the development, I tried to stick to this concept.

The implementation of each requirement included the following steps:

- Generate a dedicated resource by entering the `nest g resource-name` command. The resource contains the following files:

```
/resourse-name
├── /dto
│   ├── create-resourse-name.input.ts
│   └── update-resourse-name.input.ts
├── /entities
│   └── resourse-name.entity.ts
├── resourse-name.module.ts
├── resourse-name.resolver.ts
└── resourse-name.service.ts
```

1. Fulfill the *resource-name.entity.ts* file. It should hold the information about a specific entity and its fields that will be accessible via the GraphQL API. Figure 6.3 is used as an example of exposing the Event entity.

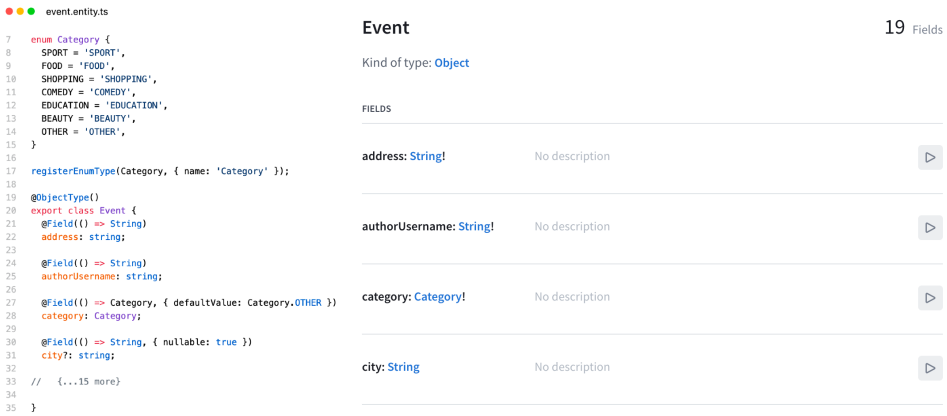


Figure 6.3: Prisma model example [author]

2. The next step is to implement resolvers. A resolver is an endpoint called from the client-side. It receives HTTP or WS requests, processes them,

calls functions from the service layer and returns a response. If a resolver behavior depends on a specific state, it may receive a set of parameters. It is recommended to store these parameters in dedicated files in a *dto* folder. The basic resolver is demonstrated in Figure 6.4. In the GraphQL API it is available under the name `eventMany`, receives arguments of the type `FindManyArgs` defined in the dto and calls the related function from the `eventsService`.

events.resolver.ts

```
@Query(() => EventManyResponse, { name: 'eventMany' })
  async findAll(@Args() args: FindManyArgs) {
    return await this.eventsService.findAll(args);
}
```

Figure 6.4: NestJS resolver example [author]

3. The last step is to update a service layer. As illustrated in Figure 6.10 a service receives arguments from a resolver, calls Prisma functions to operate with the database and optionally returns a result.

events.service.ts

```
@Injectable()
export class EventsService {
  constructor(private readonly prisma: PrismaService) {}

  async findOne(id: string) {
   const event = await this.prisma.event.findUnique({ where: { id } });

   if (!event) {
     throw new UserInputError('Event does not exist');
   }

   return event;
 }

//  {... other functions}
}
```

Figure 6.5: NestJS service example [author]

The code above demonstrates not only the service function but also the Dependency Injection (DI) design pattern being used to inject prisma instance.

Authentication and authorization

The GraphQL endpoints are not secured yet, and every user can mutate and retrieve data. To change this state, I implemented the authentication mechanism using JSON Web Token (JWT) and protected specific resolvers using NestJS guards [21].

As it was mentioned above, NestJS does not support GraphQL by default, therefore some functionalities must be overridden in order to work properly. One of them is retrieving a request from a context for authentication using a username and a password:

```
@Injectable()
export class JwtAuthGuard extends AuthGuard('jwt') {
  getRequest(context: ExecutionContext) {
    // retrieve a request from the context
    const ctx = GqlExecutionContext.create(context);
    return ctx.getContext().req;
  }
}
```

Figure 6.6: Override the getRequest() functionality [author]

Having the AuthGuard overridden, I added `@UseGuards(JwtAuthGuard)` before `@Query`, `@Mutation`, `@Subscription` decorators to protect some endpoints that require user to be authenticated. For example, retrieving of created and saved events.

Redis pub/sub module

The Redis pub/sub module was used to implement GraphQL subscriptions. Subscriptions allow the server to communicate with the client when a specific event happens. Firstly, I updated the *docker-compose.yml* with a Redis image. After that, I created the module itself (Figure 6.7).

pubsub.module.ts

```
export const PUB_SUB = 'PUB_SUB';

@Global()
@Module({
  providers: [
    {
      inject: [ConfigService],
      provide: PUB_SUB,
      useFactory: (configService: ConfigService) =>
        new RedisPubSub({
          connection: configService.get('REDIS_URL'),
        }),
    },
  ],
  exports: [PUB_SUB],
})
export class PubsubModule {}
```

Figure 6.7: Pub/sub module [author]

When the module is ready, it can be injected into other components. An example of use is shown in Figure 6.8. The key lines of code are listed below.

- line 89 - The function emits the event.
- line 99 - The applied filter determines what client will receive a new notification.
- line 103 - The function sends a notification to the client with a provided data (line 90)

enrollment-requests.resolver.ts

```
79    @UseGuards(JwtAuthGuard)
80    @Mutation(() => Boolean, { name: 'enrollmentRequestAcceptOne' })
81    async accept(
82      @Args('username', { type: () => String }) username: string,
83      @Args('eventId', { type: () => String }) eventId: string,
84    ) {
85      const enrollmentRequestAccepted =
86        await this.enrollmentRequestsService.accept(username, eventId);
87
88      if (enrollmentRequestAccepted) {
89        this.pubSub.publish(ENROLLMENT_ACCEPTED_EVENT, {
90          enrollmentRequestAccepted,
91        });
92        return true;
93      }
94
95      return false;
96    }
97
98    @Subscription(() => EnrollmentRequest, {
99      filter: (payload, variables) =>
100       payload.enrollmentRequestAccepted.username === variables.username,
101   })
102   enrollmentRequestAccepted(@Args('username') username: string) {
103     return this.pubSub.asyncIterator(ENROLLMENT_ACCEPTED_EVENT);
104   }
```

Figure 6.8: PubSub use example [author]

6.2 Front-end

6.2.1 Used libraries and APIs

Throughout the front-end development, I have used various npm dependencies that enriched the application functionality, enhanced the user experience, and sped up the implementation process.

Client-server communication

- **Apollo Client** is a comprehensive library for managing local and remote data by using GraphQL. It enables to send GraphQL queries and mutations to the API[2], cache data, mock data and test endpoints.

[2] In my case, to the NestJS server.

Available at: https://www.npmjs.com/package/@apollo/client

- **GraphQL Code Generator** significantly reduced the development time by generating TypeScript types and special React hooks with queries, mutations and subscriptions.

  Available at: https://www.npmjs.com/package/@graphql-codegen/typescript-react-apollo

- **Axios** is a HTTP client used to call external APIs:
  - Mapbox API - dynamic and static maps, reverse geocoding [22].
  - Cloudinary API - storing, modifying media content.

  Available at: https://www.npmjs.com/package/axios

ChakraUI

Available at: https://www.npmjs.com/package/@chakra-ui/react

Writing Cascading Style Sheets manually is time-consuming. Therefore I chose a UI library called ChakraUI. It provides nice-looking components, including containers, buttons, input fields, modals, and others. The other advantage of this library is it has the official Figma template. I have used it to design high-fidelity prototypes. Figure 6.9 demonstrates the application dashboard, other wireframes are located in the attached CD.

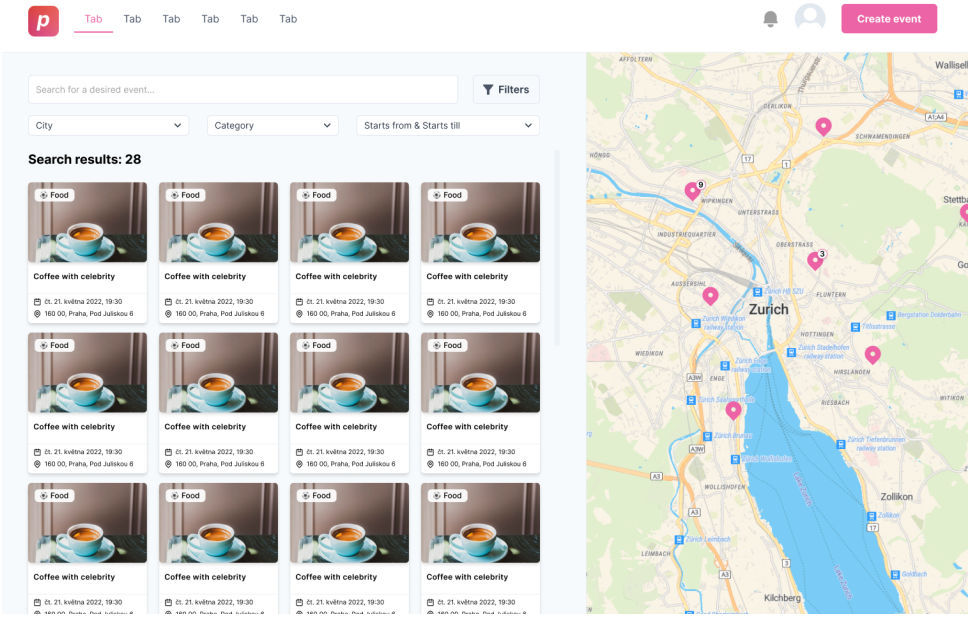


Figure 6.9: Application dashboard [author]

React Hook Form

Available at: https://www.npmjs.com/package/react-hook-form

The React Hook Form library is used on the pages with authentication: login and registration, as well as on the page with the event creation. It manages a form state, validates input fields, and increases performance by isolating component re-renders.

React Map GL

Available at: https://www.npmjs.com/package/react-map-gl

Software requirements also include the implementation of maps. Initially, I have chosen Google Maps as it is the most commonly used option. They are well-maintained, have a beautiful design, and stay up to date. However, Google Maps services are quite expensive for the proof of concept, so I leaned towards another option - Mapbox.

The React Map GL library simplifies the use of the Mapbox API by providing a special React map component.

React Icons

Available at: https://www.npmjs.com/package/react-icons

React icons is a collection of popular icons.

Zustand

Available at: https://www.npmjs.com/package/zustand

Zustand is a state management library that is primarily used on the event creation page to hold the input data from individual steps of a multi-step form.

date-fns

Available at: https://www.npmjs.com/package/date-fns

Date-fns is a simple and consistent toolset for manipulating JavaScript dates.

Next.js API

Available at: https://nextjs.org/docs

- **next/link**
  Next.js provides with its own file-based routing system: each file located in the *pages* folder is meant as a route. To keep the application state while changing routes, it is recommended to wrap links with the next/link component.

- **next/router**
  Next.js router object can be initialized either with `useRouter()` or `withRouter()`. It contains routing information along with useful functions: `push()`, `replace()`, `reload()` etc.

- **next/image**
  Next.js recommends using its own special component `<Image />` for images to optimize them and increase performance.

6.2.2 Responsiveness

To ensure comfortable use of the web application on phones and tablets, its styles should vary depending on the size of the device. Therefore, after creating a base template for a specific page, I used Chrome DevTools to simulate a mobile/tablet device and added more styles applicable only on such screen sizes. The code below demonstrates how to handle responsiveness using ChakraUI. The `mx` prop[3] defines a horizontal margin and equals `0px` on small devices and `20vw` on large devices.

```
<Heading size={'md'} mx={{ base: 0, lg: '20vw' }}>
  Create event
</Heading>
```

Figure 6.10: Handle responsiveness with ChakraUI [author]

6.3 Deployment

The deployment process was fast and easy, both with the back-end and front-end. I deployed the server using the Heroku CLI: staged changes, committed them, and pushed to Heroku master. The client deployment process was even easier: all I needed was to link the GitHub to Vercel and click the *Deploy* button. Once the website is deployed, I also added a custom domain name in Vercel project settings. It is not required to provide a custom domain because there always be one assigned by default.

[3] React components are JavaScript functions. Their parameters are called *props*.

6.4 Chapter summary

The chapter described the process of application development. The back-end section included the database configuration, implementation details of individual system requirements, security, and a Redis Pub/Sub module for notifications. In the front-end section, I described necessary npm dependencies, demonstrated the design of the website's home page created in Figma, and explained how to create adaptive components using ChakraUI. The final part aims at describing the deployment process.

Chapter 7

User Acceptance Testing

After the website is available online, it is ready for User Acceptance Testing (UAT). UAT is the final web development phase which involves potential users testing the application before it is published to the market. The main goal is to evaluate whether the application works according to the defined requirements and whether it does not have any serious GUI problems.

7.1 Participants

Many respondents from the survey described in the section 3.4 could not participate in the user testing. The main reason was the lack of time due to the upcoming final exams. However, five people still agreed to help. To obtain more accurate data, I found five other students. Since they were unfamiliar with the project, it was necessary to provide a brief project description, goals, and main functions at the beginning of testing.

7.2 Testing format

All the testing sessions were held in person because this way I could see first impressions on the website, control if all styles were applied correctly, check how the website looks on each device and answer possible questions. In addition, I was willing to help in case users had problems with the site navigation.

During the whole process of testing, I was writing down all the steps made by the individual student. Having a list of steps reduces the debugging time because a developer will know how to reproduce the bug in the dev environment.

At the end of the test, participants shared their impressions about the application, talked about its usability, how to improve the platform, and if they are ready to use it at the current stage.

7.3 Testing period

Since all the testing sessions were held in person, each tester completed a Doodle[1]. Based on the given answers, it was possible to choose the most convenient time.

7.4 Testing scenarios

At the start of testing, each student got pre-prepared lists of actions - testing scenarios. Scenarios aim to test the most important features in the application with which the user will most often operate, namely:

- 1) browsing, searching, and filtering events,
- 2) logging in,
- 3) logging out,
- 4) creation of a new user account,
- 5) creation of a new event,
- 6) registration for the event.

Each testing scenario included only specific actions but not the site navigation. For this reason, each user had to figure out how to access needed website pages. This was made intentionally to test the ease of use and the intuitiveness of the user interface.

To test both the mobile and desktop versions of the website, the first half of the participants used their phones for testing, and the second half - used a laptop or PC.

7.4.1 Test Scenario 1 - Browse events

Use Case ID: UC027
Actors: system, person.
Goal: to verify that the person can view, search and filter events.

Main scenario:

1. The user requests to browse all events.
2. The system displays the page with all ongoing events (WF - UC027: View all ongoing events).

Alternative scenario:

1. The user requests to search a particular event.

[1]Doodle - Free online meeting scheduling tool, available at https://doodle.com/

2. The system displays the page with all events (WF - UC027: View all ongoing events).
3. The user types *Marathon events* to the search field and clicks on the *Search* button.
4. The system displays all events with the title *Marathon events*.

Alternative scenario:

1. The user requests to filter events by category.
2. The system displays the page with all ongoing events (WF - UC027: View all ongoing events).
3. The user chooses a category.
4. The system displays all events with the chosen category,

7.4.2 Test Scenario 2 - Local authentication

Use Case ID: UC004
Actors: system, person.
goal: to verify that the person can login with a username and a password.
Entry conditions: a person has an account in the system.

Main scenario:

1. The user requests to login into the system.
2. The system displays the authentication form (WF - UC002: Sign in).
3. The user fills in a username and a password.
4. The system validates input data.
5. IF the data is valid AND there is identical data in the database THEN the system signs in ELSE the system warns the user of the error and asks to enter the data again.

7.4.3 Test Scenario 3 - Sign out

Use Case ID: UC042
Actors: system, person.
Goal: to verify that the person can sign out.
Entry conditions: the person is signed in.

Main scenario:

1. The user requests to sign out from the system.
2. The system logs the user out.

7.4.4 Test Scenario 4 - Local registration

Use Case ID: UC002
Actors: system, person.
Goal: to verify that the person can register with a name, a username and a password.
Entry conditions: the person is not registered yet.

Main scenario:

1. The user requests to register in the system.
2. The system displays the registration form (WF - UC002: Sign up).
3. The user fills in a name, a username and a password.
4. The system validates input data.
5. IF the data is valid AND there is no identical data in the database THEN the system registers ELSE the system warns the user of the error and asks to enter the data again.

7.4.5 Test Scenario 5 - Event creation

Use Case ID: UC007
Actors: system, person.
Goal: to verify that the person can create events.
Entry conditions:

- the person is signed in,
- the title field is required and its length is in the range from 5 to 100,
- the description field is optional and its length is in the range from 0 to 3000,
- the start date is required.
- the end date is optional and it is later than the start date.
- the location field is required.

Main scenario:

1. The user requests to create a new event.
2. The system displays the event creation form (WF - UC007: Create own events - General).
3. The user fills in all needed fields.

4. The system displays the event preview (WF - UC007: Create own events - Preview).
5. The user submits the form.
6. The system creates a new event and redirects the user to the event page.

Exception scenario: the user enters invalid data (Step 3)

1. The system displays the error message and asks the user to enter the data again.
2. The user enters valid data.
3. The system creates a new event and redirects the user to the event page.

7.4.6 Test Scenario 5 - Event registration

Use Case ID: UC018
Actors: system, host[2], person.
Goal: to verify that the person can register for events.
Entry conditions:

- the person is signed in,
- the host is signed in,

Main scenario:

1. The user requests to view detailed information about the event.
2. The system displays the event detail page (WF - UC007: Create own events - General).
3. The user requests to register for the event.
4. The system displays a warning about new notifications on the host side.
5. The host requests to view notifications.
6. The system displays a new notification about the participation request on the host side.
7. The host requests to accept the participation request.
8. The system removes the warning about new notifications.
9. The system displays a new notification about the accepted participation request on the user side.

[2] a host is a person who has created at least one event

Exception scenario: the host declines the participation request (Step 5)

1. The system displays a new notification about the declined participation request on the user side.
2. The user marks the notification as read.
3. The system removes the warning about new notifications.

7.5 User impressions

All the participants perceived the application quite positively. Many of them noted the user interface as well as the ease of use of the application. All the elements were in the expected place, so the needed information was found quickly.

Testers especially liked the event creation page: splitting the form into several parts allowed them to concentrate on one task; thanks to the created event preview, students checked the entered data and saw how it would look for other users.

Students concurred regarding the representation of events on the map. They found it convenient to have the ability to view the nearest events.

Six out of ten students were ready to start using the application in its current state. Others decided to wait for future releases and the first stable version of the application.

7.6 Detected bugs

There are several bugs that were detected during the testing sessions. All of them are listed in the table 7.1.

Table 7.1: Detected bugs

Description	Complexity[1]	Priority[2]	Current state
When a user allows geolocation, the application fetches events by city only after reload	Low	80	Planned
When a user allows geolocation, the related modal closes too slowly	Low	50	Planned
When a user declines geolocation, the related modal opens again after page reload/return from other page	Low	90	Planned
Login and registration forms have shadow on medium devices	Low	10	Planned
Create event page can be accessible for unauthenticated users	Low	100	Planned
Notifications about accepted/declined participation requests are not displayed after page reload	Medium	70	Planned
Searching without any filters leads to server crush	Medium	100	Done

[1] Approximate complexity to fix a bug. [Low, Medium, High]
[2] Defines the bug fixing order. [Lowest priority - 5, Highest priority - 100]

7.7 Possible improvements

At the end of each testing session, students described some features that would increase the user experience. The most requested functionalities are listed below.

- Update user profile information: name, username, and profile image.
- View profiles of other users.
- Follow other users.
- Change a password.
- Recover a password.
- Register and authenticate with SSO.
- Browse events on the map on mobile devices.
- Event restrictions: age, the total number of participants, and others.

7.8 Chapter summary

The user acceptance testing showed that the basic system requirements were met. Despite several problems with the application, most students were ready to use the application in everyday life. There is a list of various ideas on how to improve the application. Each of them will be analyzed and potentially planned for development.

The next chapter will describe the future state of the application.

Chapter 8

The future state of the web platform

Since the students have positively evaluated the application, I will continue to work on the project. I will fix all the bugs found during user testing in the upcoming releases. The next step will include a more extensive user acceptance testing that will involve a much more significant number of participants (approximately 200). Based on the obtained data, I will understand whether people like the idea and whether they will use it in everyday life. This will determine if the project will be further developed or not.

Chapter 9

Conclusion

The goal of the Bachelor thesis was to implement the platform in the form of a proof of concept (POC). In other words, to implement only fundamental features. All of them have the priority of *Must Have* or *Should Have*. Once the application was ready, the platform was tested using UAT, which revealed several bugs and, more importantly, showed that the system requirements were met. Therefore, the project is considered to be successful.

More and more people who live in Europe, especially in the Czech Republic, face social isolation and loneliness. The Bachelor thesis gave me the opportunity to help people cope with these problems by developing the first version of the website that will facilitate the organization of meetings between people.

The first part of the Bachelor thesis included a comparison of statistics before and during COVID-19. It turned out that the most significant impact of the pandemic on the number of lonely people was seen in the 18-25 age group. One of the effective ways to overcome loneliness is to communicate with other people. Therefore, I surveyed students on their favorite cooperation platforms and collected a list of desired functionalities. Popular tools do not fully satisfy students' needs. For this reason, I decided to implement a new web application based on their requirements. Having experience with software development, I have also defined several requirements for the system.

During the entire development process, I tried as many technologies as possible to clearly understand which frameworks, libraries, and utilities are the most suitable for this project. Despite the hours spent with technologies that were not used in the final application, I am happy with the chosen approach because I learned a lot more than I could have imagined.

Bibliography

[1] *Social isolation – APA Dictionary of Psychology* [online] [visited on 2021-11-28]. Available from: https://dictionary.apa.org/social-isolation.

[2] D'HOMBRES, Béatrice; SYLKE, Schnepf; BARJAKOVÁ, Martina; MENDONCA, Francisco. *Loneliness in the EU. Insights from surveys and online media data* [online]. JRC125873. Luxembourg: Publications Office of the European Union, 2021 [visited on 2021-12-31]. ISBN 9789276402466. Available from DOI: 10.2760/28343.

[3] ROSENBAUM, Simon; TIEDEMANN, Anne; SHERRINGTON, Catherine; CURTIS, Jackie; B. WARD, Philip. Physical Activity Interventions for People With Mental Illness [online]. [N.d.], vol. 75, no. 9 [visited on 2021-12-02]. ISSN 0160-6689. Available from DOI: 10.4088/JCP.13r08765.

[4] *COOPERATION | meaning in the Cambridge English Dictionary* [online] [visited on 2021-12-03]. Available from: https://dictionary.cambridge.org/dictionary/english/cooperation.

[5] *What is a Minimum Viable Product (MVP)? - Definition from Techopedia* [online] [visited on 2022-01-16]. Available from: https://www.techopedia.com/definition/27809/minimum-viable-product-mvp.

[6] FOWLER, Martin. *Destilované UML* [online]. 1. vyd. Boston: Grada, 2005 [visited on 2022-04-03]. ISBN 9788024720623.

[7] *Chapter 10: MoSCoW Prioritisation | The DSDM Handbook | Agile Business Consortium* [online] [visited on 2022-05-15]. Available from: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation.

[8] *BPMN Specification - Business Process Model and Notation* [online] [visited on 2022-04-04]. Available from: https://www.bpmn.org/.

[9] *About the Business Process Model And Notation Specification Version 2.0* [online] [visited on 2022-04-04]. Available from: https://www.omg.org/spec/BPMN/2.0/About-BPMN/.

[10] *Back-End Web Architecture | Codecademy* [online] [visited on 2022-05-15]. Available from: https://www.codecademy.com/article/back-end-architecture.

[11] *Stack Overflow Developer Survey 2021* [online] [visited on 2022-04-16]. Available from: https://insights.stackoverflow.com/survey/2021.

[12] *JavaScript type coercion explained* [online] [visited on 2022-04-16]. Available from: https://www.freecodecamp.org/news/js-type-coercion-explained-27ba3d9a2839/.

[13] *Documentation | NestJS - A progressive Node.js framework* [online] [visited on 2022-04-16]. Available from: https://docs.nestjs.com/.

[14] *Angular - Introduction to Angular concepts* [online] [visited on 2022-05-14]. Available from: https://angular.io/guide/architecture.

[15] CODD, E. F. *The Relational Model for Database Management: Version 2* [online]. USA: Addison-Wesley Longman Publishing Co., Inc., 1990 [visited on 2022-04-15]. ISBN 0201141922. Available from DOI: 10.5555/77708.

[16] CODD, E. F. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* [online]. 1970, vol. 13, no. 6, pp. 377–387 [visited on 2022-04-15]. ISSN 0001-0782. Available from DOI: 10.1145/362384.362685.

[17] *How Google's Site Crawlers Index Your Site - Google Search* [online] [visited on 2022-04-18]. Available from: https://www.google.com/search/howsearchworks/crawling-indexing/.

[18] *Introducing Hooks – React* [online] [visited on 2022-04-19]. Available from: https://reactjs.org/docs/hooks-intro.html.

[19] *knex vs nodejs-mysql vs prisma vs sequelize vs typeorm | npm trends* [online] [visited on 2022-04-20]. Available from: https://www.npmtrends.com/knex-vs-nodejs-mysql-vs-prisma-vs-sequelize-vs-typeorm.

[20] *Bulletproof node.js project architecture - DEV Community* [online] [visited on 2022-04-20]. Available from: https://dev.to/santypk4/bulletproof-node-js-project-architecture-4epf#architecture.

[21] *Guards | NestJS - A progressive Node.js frameworky* [online] [visited on 2022-04-20]. Available from: https://docs.nestjs.com/guards.

[22] *Reverse Geocoding Tool Convert Lat Long to Address* [online] [visited on 2022-04-22]. Available from: https://www.latlong.net/Show-Latitude-Longitude.html.

Appendix A

Survey questions

Social networks and messengers

1. *Do you use social networks or messengers to create or to search for social events?*
   - Yes
   - No
2. *Please, specify what social networks / messengers do you use to create or to search for social events.*
   - Facebook
   - Instagram
   - Twitter
   - Whatsapp
   - Telegram
   - Vkontakte
   - Other
     .....
3. *Why have you chosen these particular social networks / messengers?*
   .....
   .....
   .....
4. *Do you want to change something in the applications you use?*
   .....
   .....
   .....

Social networks and messengers

1. *Do you search for social events on websites other than social networks?*
   - Yes
   - No

2. *Please, specify what websites do you use to create or to search for social events.*

   - eventbrite.cz
   - meetup.cz
   - Other

     .....

3. *Why have you chosen these particular social networks / messengers?*

   .....
   .....
   .....

4. *Do you want to change something in the applications you use?*

   .....
   .....
   .....

Functionality

1. *What app features are the most valuable for you? Rate the following list from 1 (I do not need it) to 5 (I totally need it).*

Event reactions (likes, comments, shares)	1	2	3	4	5
Event searching, filtering, categorizing	1	2	3	4	5
Push notifications about event updates	1	2	3	4	5
Map integration (view events on a map)	1	2	3	4	5
Availability to add event to your calendar	1	2	3	4	5
Text formatting in the event description	1	2	3	4	5
Followers/following lists	1	2	3	4	5
In-app chat	1	2	3	4	5

2. *Can't find other important features above? Describe them below.*

   .....
   .....
   .....

3. *Do you use other tools that are related to social events?*

   .....
   .....
   .....

4. *Do you want to change something in the applications you use?*

   .....
   .....
   .....

Appendix B

Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
AWS	Amazon Web Services
BPMN	Business Process Modeling Notation
CLI	Command-line Interface
CSR	Client-side Rendering
DBMS	Database Management Systems
DI	Dependency Injection
DOM	Document Object Model
DSG	On-demand Static Regeneration
GUI	Graphical User Interface
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
JSX	JavaScript XML
JWT	JSON Web Token
MVP	Minimum Viable Product
(N)FR	(Non)Functional Requirement
NoSQL	Not Only SQL
NPM	Node Package Manager
POC	Proof Of Concept
REST	Representational State Transfer
SEO	Search Engine Optimization
SQL	Structured Query Language

Appendix C

Folder structure of the attached CD

- `wireframes.zip` ..... high fidelity wireframes
- `Survey_on_cooperation_platforms.xlsx` ....student survey results
- `uml_diagrams.eap` ..... all UML diagrams
- `code.zip` ..... code of the application