UNIVERSITY OF TARTU

FACULTY OF SOCIAL SCIENCES

NARVA COLLEGE

INFORMATION TECHNOLOGY SYSTEMS DEVELOPMENT

Daria Kozhevnikova

DEVELOPMENT OF A WEB APPLICATION FOR BOOKS EXCHANGE

Diploma thesis

Supervisor: M.Sc. Andre Sääsk

NARVA 2023

Olen koostanud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, põhimõttelised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

……/töö autori allkiri/

Non-exclusive license to reproduce the thesis:

I, Daria Kozhevnikova (date of birth: 27.08.1992),

1. herewith grant the University of Tartu a free permit (non-exclusive license) to reproduce, for the purpose of preservation, including for addition to the DSpace digital archives until the expiry of the term of validity of the copyright “developing an application for participation in local events”, supervised by A. Sääsk,

2. I am aware that the author retains the right referred to in point 1.

3. This is to certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Narva, 05.05.2023

CONTENTS

Terms and abbreviations

AXIOS – is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests. (Axios…2023)

JSON (JavaScript Object Notation) – is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page. (W3Schools…2023)

Web app (web application) – an application (= a computer program designed for a particular purpose) that you can use on the internet rather than having to download it. (Cambridge…2023a)

Access token – contains the security credentials for a login session and identifies the user, the user's groups, the user's privileges, and, in some cases, a particular application. (Wikipedia…2023a)

API (Application Programming Interface) – a way of communicating with a particular computer program or internet service (Cambridge…2023b)

IDE (Integrated Development Environment) – is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of at least a source code editor, build automation tools, and a debugger. (Wikipedia…2023b)

HTTP (Hyper Text Transfer Protocol) – a set of instructions made by a computer program that allows your computer to connect to an internet document. (Cambridge…2023c)

HTML (Hyper Text Markup Language) – a way of marking text so that it can be seen on the internet. (Cambridge…2023d)

CSS (Cascading Style Sheets) – a piece of computer software that allows you to describe and control how a web page or a series of web pages should appear to the user. (Cambridge…2023e)

SSR (Server-side rendering) – is the ability of an application to contribute by displaying the web page on the server instead of rendering it in the browser. Server-

side sends a fully rendered page to the client; the client's JavaScript bundle takes over and allows the SPA framework to operate. There is also client-side rendering which slows down the procedure of viewing and interacting with the web page. (EducativeIO…2023)

TS (TypeScript) – is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale. (TypeScript…2023)

JS (Java Script) – is a scripting or programming language that allows you to implement complex features on web pages. (MDNWebDocs…2023)

UI (User Interface) – the way in which the information on a computer, phone, etc. and instructions on how to use it are arranged on the screen and shown to the user. (Cambridge…2023f)

UX (User Experience) – the experience of someone using a product, system, or service, for example whether they find it enjoyable and easy to use. (Cambridge…2023g)

SEO (Search engine optimization) – methods of making sure that the address of a website is shown near the top of the list of results of an internet search. (Cambridge…2023h)

JWT (JSON Web Token) – is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. (JWT…2023)

NPM (Node Package Manager) – is a package manager for the JavaScript programming language maintained by npm, Inc. (Wikipedia…2023c)

SQL (Structured Query Language) – is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables. (Wikipedia…2023d)

FE (Front end) – the parts of a computer, piece of software, or website that are seen and directly used by the user. (Cambridge…2023i)

BE (Back end) – the part of a computer system, piece of software, etc., where data is stored or processed rather than the parts that are seen and directly used by the user. (Cambridge…2023j)

Debugging – to look for and remove bugs (= hidden listening or recording devices) from a place. (Cambridge…2023k)

ORM (Object-Relational Mapping) – is a programming technique in which a metadata descriptor is used to connect object code to a relational database. (Rouse M., 2011)

SWOT (Strength, Weakness, Opportunity, Threat) – a way of considering all the good and bad features of a business situation or a company. (Cambridge…2023l)

MVP (Minimal Viable Product) – is a version of a product with just enough features to be usable by early customers who can then provide feedback for future product development. (Wikipedia…2023e)

MVC (Model-View-Controller) – is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements. (Wikipedia…2023f)

PK (Primary key) – is a specific choice of a minimal set of attributes (columns) that uniquely specify a tuple (row) in a relation (table). (Wikipedia…2023g)

FK (Foreign key) – is a set of attributes in a table that refers to the primary key of another table. (Wikipedia…2023h)

GUI (Graphical User Interface) – s a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicator, such as primary notation. (Wikipedia…2023i)

HOC (Higher Order Component) – is a function that takes a component and returns a new component. (ReactJS…2023)

RDBMS (Relational database management system) – is a program used to create, update, and manage relational databases. (GoogleCloud…2023)

Introduction

In today's fast-paced world, we often find ourselves accumulating an abundance of possessions, including books. Many of these books may be read once or twice and then forgotten, collecting dust on shelves, or taking up valuable space in our homes. Meanwhile, there are others who may not have the means to purchase or access these books, and so miss out on the joy of reading and learning.

To address this issue, this project aims to create a web application that will allow individuals to easily swap their books with others who are interested in reading them. This application aims to solve several problems related to book ownership, including the high cost of purchasing books, the overconsumption of paper, and the limited accessibility of books to certain populations.

Goal

The goal of this thesis is to develop a web application MVP version and a new experience for book exchange, based on a comparative analysis of applications that are currently available on the market, and their pros and cons, and to promote the sharing and circulation of books among individuals in a sustainable and accessible way.

Main problems

One of the main problems this project aims to solve with BookEx is the high cost of purchasing books. By providing a platform for individuals to swap books they no longer need, I hope to reduce the financial burden of buying new books and make reading more accessible to everyone.

Another issue I aim to address is the overconsumption of resources. By promoting the reuse and sharing of books, I hope to reduce the environmental impact of book production and contribute to a more sustainable future.

Finally, this application aims to address the limited accessibility of books to certain populations, such as those who may not have access to a library, the means to purchase books or need some specific books that are not available in their place of residence. By providing a platform for book exchange, I hope to create a more equitable distribution of knowledge and resources.

Tasks

To reach this goal, the author plans to create a web application that will allow users to create a profile, add their books, browse available books in a feed, like or dislike books, and initiate book exchanges with other users. The application will also enable users to share their contact information with each other, such as phone numbers or a link to a Telegram account. I will need to ensure the security of user data, by avoiding unauthorized access to data. The application will also need to have a user-friendly interface to encourage widespread use.

To achieve described features of this thesis, the following tasks are needed to be accomplished:

- Create a scalable and secure database to store user data, book information, and exchange notifications.
- Develop the backend logic, APIs, and server-side functionality using Node.js.
- Develop the frontend user interface accessible through a web browser using such modern technologies as Next.js framework, React library, and Ant Design components library.
- Implement user authentication, including account creation and login.
- Create a book management system for users to add, view and delete books.
- Implement a matching algorithm that matches users based on their likes.
- Implement various security measures to prevent unauthorized access.
- Test the application rigorously to ensure it is functioning correctly, is user-friendly, and meets the project requirements.

Outline

Section 1 provides descriptions of comparable solutions, as well as selected tools and technologies.

Section 2 describes the application development process.

Section 3 presents the development results.

1 Chosen technologies and similar solutions

1.1 Chosen technologies and instruments

1.1.1 Frontend technologies

1.1.1.1 ReactJS

ReactJS is an open-source JS library that simplifies building dynamic and interactive user interfaces by breaking the UI into small, reusable components. It also offers features like state management and event handling, making it easy to build complex UIs. (ReactDev…2023)

It was chosen by the author because she has experience working with this library and considers it a modern quick and lightweight way to create FE interfaces and business logic.

1.1.1.2 Next.js

Next.js is an open-source framework on top of React.js. It is designed for server-side rendering and static site generation and offers benefits such as simplified routing, optimized rendering, automatic code splitting, and built-in support for static site generation. (NextOrg…2023)

It was chosen by the author because she has experience working with this framework and considers it a perfect tool for improving React library by using SSR and simplified routing.

1.1.1.3 Typescript

TS adds additional syntax to JS to support a tighter integration with your editor. It understands JS and uses type inference to give you great tooling without additional code. (TypeScriptLang…2023)

It was chosen by the author because she has experience working with this technology and prefers strong typing opportunities to prevent a possible amount of errors and unexpected behavior, as well as simplify the debugging process.

1.1.1.4 Ant design

Ant Design is a UI library for React.js that provides a variety of reusable components for creating user-friendly interfaces. Its focus on accessibility and user experience has made it popular within the React.js community. (AntDesign…2023)

The author selected this components library for its extensive selection of pre-built components that meet the application's functional requirements. It also offers a straightforward theme to easily adjust the visual style and layout appearance.

1.1.1.5 Axios

This technology was used by the author to simplify the process of fetching data from created API, to easily pass parameters, which improves performance.

1.1.2 Backend technologies

1.1.2.1 Node.js

Node.js is an open-source JavaScript runtime environment for server-side development. Node.js is event-driven and non-blocking, allowing it to handle many connections and requests simultaneously without blocking other tasks. (NodeJs…2023)

The author's experience with Node.js and preference for using JavaScript on both client and server-side were the reasons for its selection.

1.1.2.2 Express

Express is framework for Mode.js, that speeds up and simplifies a development process.

The author selected this framework as a modern and flexible way to create and handle Node.js code. Its middleware architecture simplifies adding functionality, including request and response handling, authentication, and error handling, making it a versatile option.

1.1.2.3 MySQL

MySQL is an open-source RDBMS that uses SQL to store and manage data. It is ideal for web applications and publishing.

MySQL was selected due to the author's familiarity with its reliable and performant database management capabilities.

1.1.2.4 Sequelize

Sequelize is a modern ORM for Oracle, Postgres, MySQL, MariaDB, SQLite and SQL Server, and more.

The author's familiarity with Sequelize and its powerful querying capabilities were the reasons for its selection. Additionally, the library's convenient tools for managing database schemas, such as creating and modifying tables, columns, and indexes, make it a fast and efficient option for database building.

1.1.3 Instruments

1.1.3.1 WebStorm IDE

WebStorm is a powerful and flexible IDE developed by JetBrains for web development. It provides features for code completion, refactoring, debugging, testing, and version control integration.

This IDE was chosen by the author because it is used by her in everyday life at work and is considered by her a convenient tool to develop FE and BE parts of the application.

1.1.3.2 MySQL Workbench

MySQL Workbench is a unified visual tool for database architects and developers. It provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. (MySql…2023)

This tool was chosen by the author to get simplified access to a created database, for testing SQL queries and checking table data after requests performance.

1.1.3.3 Postman

Postman is an API platform for building and using APIs. It simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster. (Postman…2023)

This tool was chosen by the author since she has experience working with it and considers it a convenient tool for testing API requests without the necessity to create some FE interface, which speeds up the development and debugging process.

1.2 Similar solutions on the market

1.2.1 Raamatuvahetus

This is a book exchange website based in Estonia. Users can list books they want to give away and browse books that other users have listed. The platform uses a point system to facilitate trades. Users can earn points by trading books and use those points to request books from other users.

This website is accessible from desktop, as well as on mobile devices, using responsive design (**Figure 1**). Also, Raamatuvahetus has mobile applications for iOS and Android devices.

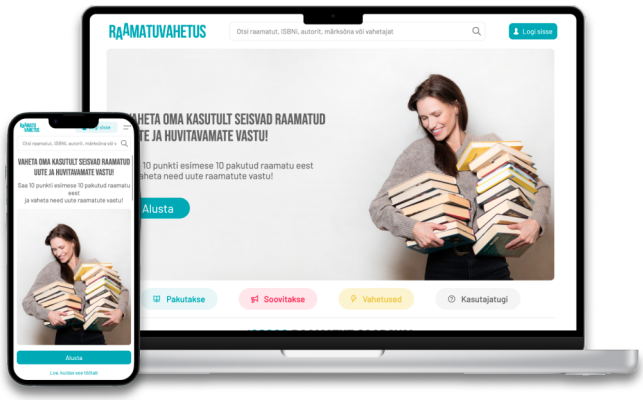


Figure 1. Raamatuvahetus mobile interface (left) and Raamatuvahetus desktop interface (right). (Source: author)

Following an extensive research endeavor, a comprehensive SWOT analysis of the platform has been documented in Table 1.

Table 1. SWOT analysis of Raamatuvahetus application. (Source: author)

Strength	**Weakness**
• User-friendly interface and easy to use. • Wide selection of books available for exchange. • Point system incentivizes users to trade books. • Dedicated mobile app available for iOS and Android devices.	• Limited to users in Estonia, which may limit the pool of available books for exchange. • Shipping costs can be high if exchanging books with users in different parts of Estonia. • A limited selection of foreign language books may make it difficult for non-Estonian speakers to find books of interest.
Opportunities	**Threats**
• Expand to other Baltic countries to increase the pool of available books for exchange. • Partner with local libraries and bookstores to offer more books for exchange. • Offer a premium membership with additional benefits such as free shipping or priority book requests.	• Competition from other book exchange platforms in Estonia and beyond. • Changes in postal regulations or pricing could increase shipping costs for users. • A lack of interest from users in Estonia could limit the growth potential.

1.2.2 Bookcrossing

Bookcrossing is a global book exchange platform that operates in Estonia. Users can register books they want to release into the wild, and other users can find and claim those books. Users can also trade books directly with each other on the platform.

This website is accessible from a desktop, as well as on mobile devices, using responsive design (**Figure 2**).

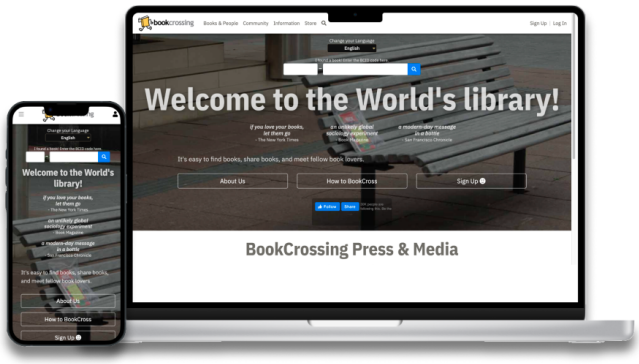


Figure 2. Bookcrossing mobile interface (left) and Bookcrossing desktop interface (right). (Source: author)

Following an extensive research endeavor, a comprehensive SWOT analysis of the platform has been documented in Table 2.

Table 2. SWOT analysis of Bookcrossing application. (Source: author)

Strength	**Weakness**
• Free to use and open to users around the world. • Large community of users who are passionate about sharing books. • No shipping costs involved, as books are left in public places for others to find and claim.	• Limited control over where books end up after being released into the wild. • Finding books in Estonia may be difficult if there are few active Bookcrossing users in the area. • Limited selection of foreign language books may make it difficult for non-Estonian speakers to find books of interest.
Opportunities	**Threats**

• Encourage more users in Estonia to participate in Bookcrossing to increase the availability of books in the area. • Partner with local libraries and bookstores to promote Bookcrossing and increase participation. • Offer additional incentives for users to release books into the wild, such as discount codes for online bookstores.	• Competition from other book exchange platforms in Estonia and beyond. • Changes in postal regulations or pricing could increase shipping costs for users who prefer to send books through the mail. • A lack of interest from users in Estonia could limit the growth potential.

1.2.3 Conclusion

Conducted on the foundation of a SWOT analysis of potential competitors, an evaluation of the prospects of BookEx has been undertaken. This analysis is introduced in a comprehensive SWOT assessment of the application, which is recorded in Table 3.

Table 3. SWOT analysis of BookEx application. (Source: author)

Strength	**Weakness**
• A unique concept of matching book lovers, making it easier to exchange books. • User-friendly interface and easy to use. • One platform for all devices and the web makes it easily accessible to everyone. • Potential to create a strong community of book enthusiasts in Estonia. • Can encourage users to read more and discover new books.	• Limited to users in Estonia, which may limit the pool of available matches and books for exchange. • Dependence on user-generated content may lead to a limited selection of books and low engagement from users. • Lack of financial resources to market and promote the app may limit the growth potential.

Opportunities	Threats
• Partner with local libraries and bookstores to offer more books for exchange and promote the app. • Offer additional features, such as book recommendations and reviews, to increase user engagement and retention. • Expand to other countries to increase the pool of available matches and books for exchange. • Integrating functionality for sending books right in the app, with the help of existing infrastructure (e.g., Omniva postamats)	• Competition from other book exchange platforms in Estonia and beyond. • Lack of interest from users in Estonia or a low adoption rate of the app may limit the growth potential.

Following a thorough analysis, it can be concluded that the initial MVP iteration of BookEx presents significant potential for future improvement, encompassing the introduction of novel features. Additionally, it is imperative that the author devotes ample attention to promoting and establishing partnerships for the application to occupy a distinct niche in the market.

2 Application development

In this chapter a description of a development process is presented: the architecture of BE and FE parts, database tables, project layers, and other aspects.

2.1 Development process

Given that the application was created by a single developer within a condensed timeframe, it was of utmost importance to select an appropriate development strategy that would satisfy all the essential MVP functional requirements and enable timely delivery of the initial iteration of the project.

After an analysis of existing strategies, an Agile methodology was chosen. It has many advantages that influenced the choice of the author. Among them, there are such:

- Flexibility: Agile methodology is highly flexible and can easily adapt to changes in project requirements or priorities. This is especially important in dynamic or uncertain environments.
- Faster delivery: Agile development is characterized by short iterations or sprints, resulting in faster delivery of working software.
- Improved quality: Agile methodology promotes continuous testing and integration, leading to higher-quality software.
- Team empowerment: Agile development encourages self-organizing teams and fosters a culture of collaboration and accountability. This results in higher job satisfaction and motivation.
- Risk reduction: Agile methodology involves frequent reviews and adaptations, which can help identify and mitigate risks early in the project lifecycle.

Given all the advantages of agile methodology, it was determined that this approach is highly conducive to enabling an author to achieve her desired outcomes.

2.2 A description of the backend part of the application

2.2.1 Architecture description

The chosen approach for developing the backend architecture was to adopt the Model-View-Controller (MVC) architecture pattern, which has been adapted to include a middleware component while delegating the «view» aspect to a distinct frontend application.

In MVC architecture, controllers manage user requests, models handle database interactions, and views present data to users. Middleware, which is situated between the application and the server, can perform a range of tasks before or after the controller handles requests and before the response is sent to the client.

The backend of the application manages logic and data management, serving an API for the frontend to consume. Controllers handle client requests and return JSON data. Models interact with databases and define table structures. Middleware performs functions such as logging, authentication, image handling, and route guarding.

2.2.2 Routes description

Book-related routes:

- POST /api/v1/books/create-new – to create a new book
- GET /api/v1/books/my-books – to retrieve books that belong to a current user
- DELETE /api/v1/books/delete – to delete a book
- GET /api/v1/books/feed – to get a feed of books for a user
- POST /api/v1/books/like – to add a book to bookmarks
- GET /api/v1/books/bookmarks – to retrieve bookmarks of a current user
- DELETE /api/v1/books/bookmarks – to delete a book from bookmarks
- GET /api/v1/books/bookmarks/liked – to retrieve notifications about a match

Auth-related routes:

- POST /api/v1/auth/signup – to create a user
- POST /api/v1/auth/signin – to sign in with existing credentials

- GET /api/v1/auth/signout – to sign out

Notification-related routes:

- GET /api/v1/notifications/feed – to get all notifications
- POST /api/v1/notifications/dismiss – to dismiss notification

Images-related routes:

- POST /api/v1/images/upload – to upload an image for a book

2.2.3 API flow description

The BookEx frontend communicates with the backend through API requests using HTTP protocols and exchanges data in JSON format. However, this approach can lead to the possibility of SQL injections, which can corrupt or destroy the database. To minimize this risk, Sequelize was selected, which uses parameterized queries to sanitize user inputs and prevent malicious SQL injections. This eliminates the need for manual input sanitization and reduces the risk of human error.

In addition, route guards were implemented to authenticate users and validate their access to specific routes. JSON Web Tokens (JWT) were chosen as a secure solution, with a token containing user data verified by the server upon each request. The **verifyToken** function (**Figure 3**) checks the token received in the request headers and grants access to the routes as needed. Middleware integration into each route is demonstrated in Figure 4.

```
const verifyToken = async (req, res, next) => {
  console.log('headers', req.headers);
  const token = req.headers.authorization?.replace( searchValue: 'Bearer ', replaceValue: '');

  if (!token) {
    return res.status(401).send({
      message: "No token provided!"
    });
  }

  const result = await InvalidatedToken.findOne({ where: { token: token } });
  if (result) {
    return res.status(401).send('Unauthorized: Invalid token');
  }

  jwt.verify(token, config.secret, options: (err, decoded) => {
    if (err) {
      return res.status(401).send({
        message: "Unauthorized!"
      });
    }
    req.userId = decoded.id;
    next();
  });
};
```

Figure 3. Example of a verifyToken() middleware code. (Source: author)

```
// /books/my-books => GET
router.get( path: '/my-books',
    handlers: [authJwt.verifyToken],
    controller.getBooksForUser)
```

Figure 4. Example of implementation of verifyToken() middleware to the logic of the route. (Source: author)

2.2.4 Matching algorithm

The core functionality of the application involves the implementation of a matching algorithm, designed to facilitate the discovery of mutually liked books between users. This algorithm is comprised of two main components:

Identifying potential matches based on whether a user has liked a book that belongs to another user who, reciprocally, has liked a book in the first user's collection (**Appendix 1**. Matching algorithm SQL-query).

Facilitating communication between matched users, by sending out notifications containing relevant contact information and details about the liked books (**Appendix 2**. Checking and sending notifications).

To ensure that the user does not receive redundant notifications, a system has been implemented to include two boolean flags within each notification. The first flag serves to indicate whether a notification has already been presented to a user. The second flag provides the ability for users to dismiss a notification, preventing it from being displayed again.

2.3 A description of the frontend part of the application

2.3.1 Architecture description

To create the frontend part of the BookEx application, React was chosen with the addition of Next.js for server-side rendering. This approach improves performance and search engine optimization.

The application includes 7 pages that use the same components and layout, allowing for code reuse and faster development. The frontend logic also includes helper functions for token storage and services for API requests and response handling.

2.3.2 Description of the process of adding a book with ISBN

To satisfy the functional requirement of adding a book along with its unique ISBN, the author has opted to utilize Google Books APIs. This approach allows the application to efficiently fetch key book data by simply entering its ISBN. The process of user interaction with FE is graphically depicted in a process diagram in Figure 5.

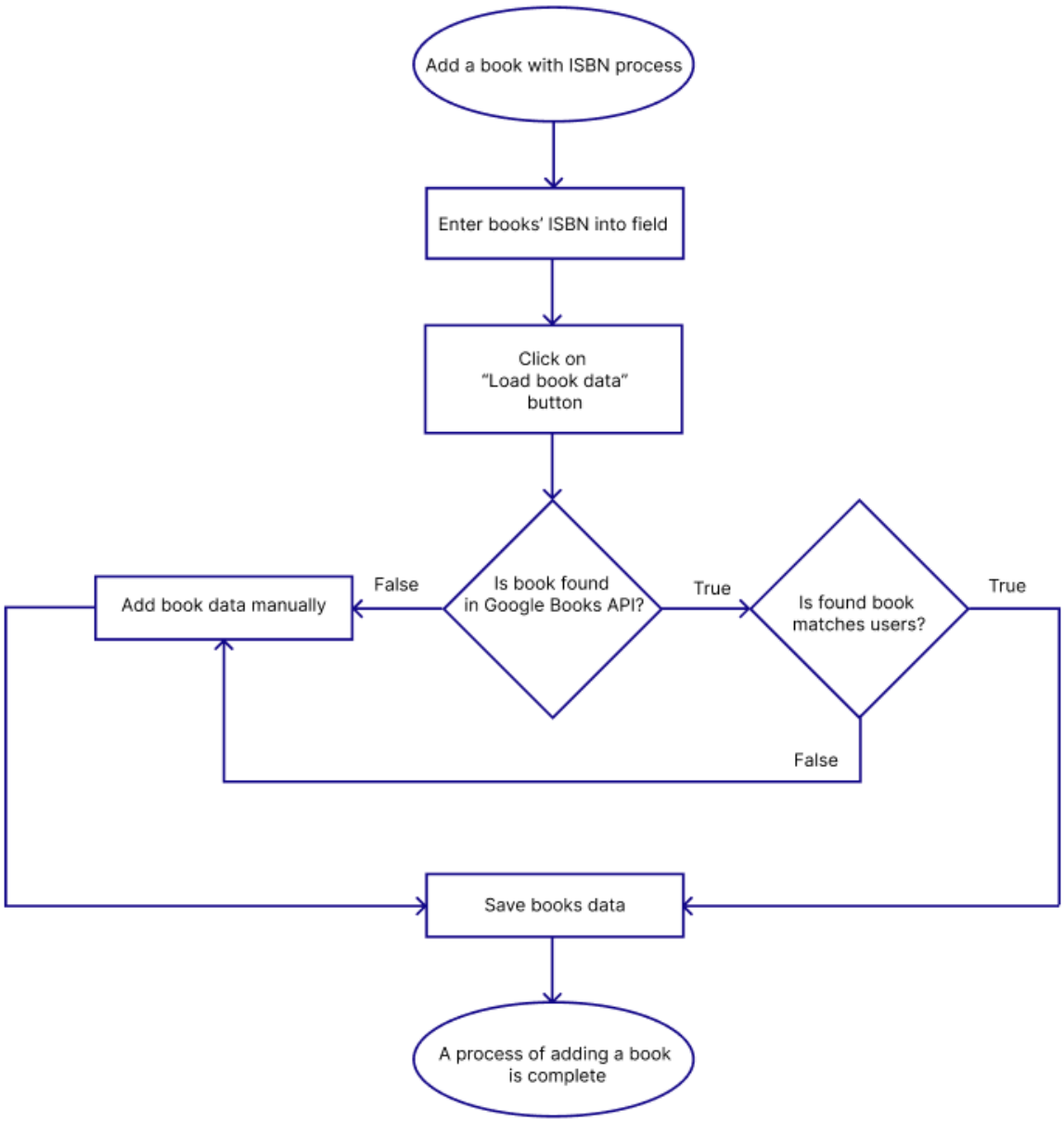


Figure 5. The process of user interaction with FE for adding a book with Google Books API. (Source: author)

The code snippet for handling the ISBN form submission and sending a request to Google Books API to retrieve book data are presented in Figure 6 and Figure 7.

```
const onISBNSubmit = async (values: any) => {
  setIsLoading( value: true);
  await fetchBookDataByISBN(values.bookISBN) ...
    .then(res => {
      console.log(res);
      !!res.data.items?.length && setBookData(res.data.items[0]);
      !res.data?.totalItems  && message.error( content: 'Oops, nothing was found! Try to add a book manually.')
    }
    ) Promise<void>
    .catch(err => {
      message.error( content: err.message || 'Oops, something went wrong!')
    }) Promise<void>
    .finally( onfinally: () => setIsLoading( value: false))
}
```

Figure 6. Handling ISBN form submit code snippet. (Source: author)

```
export const fetchBookDataByISBN = async (isbn: number) => {
  return await axios.get( url: `https://www.googleapis.com/books/v1/volumes?q=
  ${isbn}+isbn&maxResults=1&key=${process.env.NEXT_PUBLIC_GOOGLE_BOOKS_API_KEY}`)
    .then(res => res)
}
```

Figure 7. A request to Google Books API to retrieve book data code snippet. (Source: author)

2.3.3 Description of the polling process to receive notifications

In order to provide timely and pertinent information to users regarding new notifications, a polling system has been devised. This system operates by requesting the latest notifications on a ten-second interval and subsequently displaying the results via a Notification component, as provided by the Ant Design components system (**Appendix 3**. A polling system to show fresh notifications). Polling begins automatically upon application mounting, and is accessible from any page.

On the backend, a notification delivery system has been developed to ensure that only new notifications which the user has not yet seen or dismissed are sent.

All notifications are accumulated on a separate page titled «News», which allows users to peruse their notifications, obtain contact information for matched users, or dismiss notifications to prevent future display.

To further enhance the notification system, plans have been made to implement WebSocket technology on both the frontend and backend.

2.4 Database

2.4.1 Tables description

The database consists of 8 tables. It is in the first normal form. Each table also is also using indexes to make data retrieval more performant. Since I'm using Sequelize, it also always adds two fields with the creation and updating dates to each table.

ER diagram of created tables is presented in Appendix 4. ER-diagram.

2.4.2 Table relations

Based on the model described in Appendix 5. Relational model description, the following relations can be identified:

- Many-to-many relationship between the «role» and «user» tables: both tables have a «belongsToMany» association with each other, using the «user_roles» table as a join table. This means that a user can have multiple roles, and a role can be assigned to multiple users.
- One-to-many relationship between the «user» and «book» tables: a user can have multiple books, but a book can only belong to one user.
- One-to-many relationship between the «user» and «like» tables: a user can have multiple likes, but a like can only belong to one user.
- One-to-many relationship between the «book» and «like» tables: a book can have multiple likes, but a like can only belong to one book.
- Many-to-one relationship between the «like» and «book» tables: a like belongs to one book.
- Many-to-one relationship between the «like» and «user» tables: a like belongs to one user.
- One-to-many relationship between the «user» and «notification» tables: a user can have multiple notifications. The «notification» table has a foreign key that references the «user» table.
- One-to-one association between the book table and the image table, indicating that each book can have at most one associated image. This relationship is established

by adding a foreign key to the image table that references the primary key of the books table.

2.5 Functional requirements

Functional requirements describe specifications of what the BookEx application should do, and what are the key features, capabilities, and behaviors. All main requirements are represented below.

- A user can access the application from a desktop web browser, as well as from a mobile web browser.
- The user can sign in with his credentials.
- The user can sign up by filling out the signup form.
- A registered user can create new books for exchange by searching book ISBN.
- The registered user can create new books for exchange by adding data manually.
- The registered user can look through books that were added by him/her.
- The registered user can delete books created by him/her.
- The registered user can see a feed of books from other users for exchange.
- The registered user can like or dislike some books from the feed.
- Liked book is stored in bookmarks.
- Disliked book is not shown anymore for a user.
- The registered user can look through his/her bookmarks and remove books from there.
- The registered user can look through notifications about book match.
- Each notification should contain data about which book was liked, by whom, and which book that user was liked by the current one.
- Each notification should have a “Dismiss” button to not be shown anymore to the current user.
- On a book match, the registered user should receive a notification in live mode, without a page refresh.

2.6 Non-functional requirements

These are criteria that define how a system will behave, as well as performance, usability, security, etc. All main requirements are represented below.

- BookEx is required to be compatible with the most recent versions of modern web browsers, such as Google Chrome, Mozilla Firefox, and Safari, while also accommodating backward compatibility with three previous versions.
- Access to BookEx application pages should be restricted to authorized users only, with the exception of sign-in and sign-up pages which should be available to all.
- The user interface of the application should be intuitive and easy to navigate, providing a positive user experience.
- The application should exhibit a highly responsive behavior to user inputs and interactions, ensuring a seamless user experience.
- A reliable and stable internet connection is a prerequisite for seamlessly utilizing the application.

2.7 GUI

Upon the user's initial launch of the application in their browser, a login page is immediately presented (**Figure 8**). This page provides the user with the ability to input their credentials to sign in or register for a new account if they have not done so already.

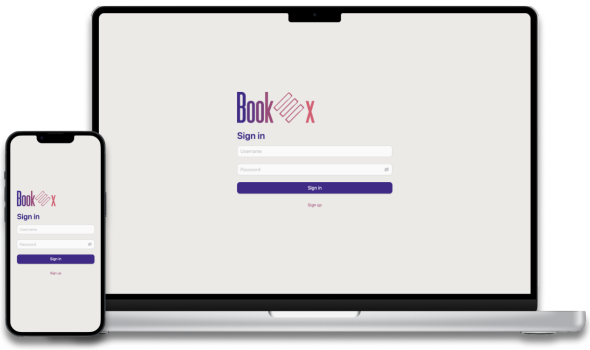


Figure 8. Login page of the application. (Source: author)

During the registration process (**Figure 9**), the user is afforded the opportunity to provide pertinent information including their username, password, name, and nearest

city. Additionally, users may elect to include their preferred method of contact, such as their Telegram or phone number.

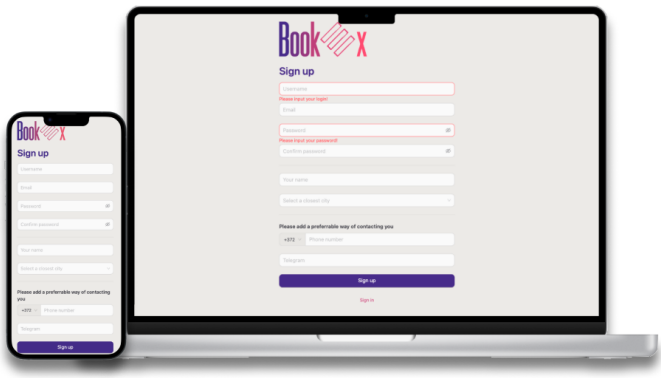


Figure 9. Sign up page of the application. (Source: author)

The application's city selection feature is powered by the GeoDB API, accessible through RapidAPI. The top five most populated cities in Estonia are presently available for user selection. This API enables the development team to modify the number of available cities and expand the application's reach to other countries by adding the option for users to choose their country of residence.

Upon successful login or registration, the user is seamlessly redirected to the application's feed page (**Figure 10**). The page showcases a personalized book selection based on the user's city of residence. Book cards are presented, and the user can either like or dislike them. Liking a book adds it to the user's bookmarks for later reference, while disliking a book removes it from the user's feed.

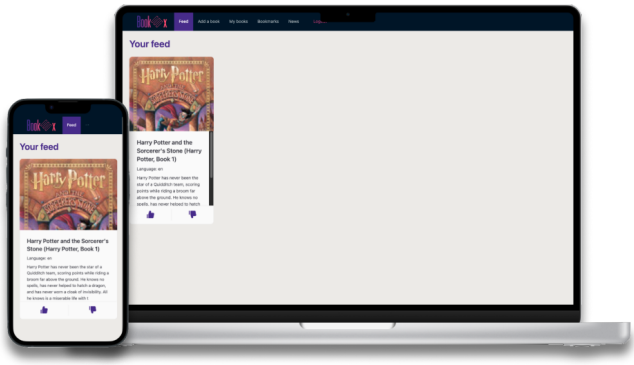


Figure 10. Feed page of the application. (Source: author)

To enhance user experience, a filtering system is being developed to allow users to filter their book feed based on categories, authors, titles, and cities.

Additionally, the application has two dedicated pages, «Bookmarks» (**Figure 11**) and «My Books,» which enable users to review and manage their saved items, including likes and uploads. These features empower users to tailor their experience to their preferences and interests.

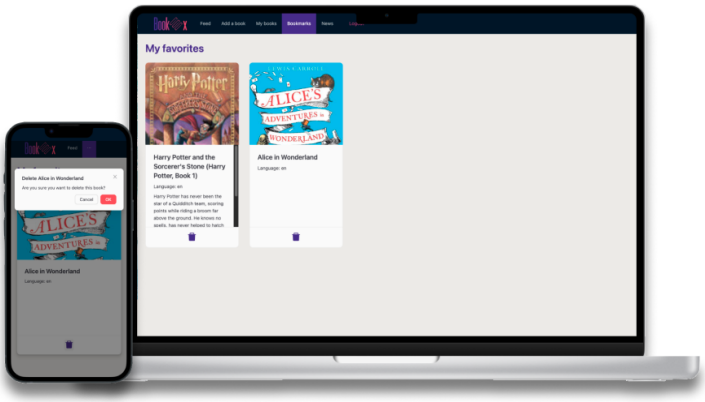


Figure 11. "Bookmarks" page of the application. (Source: author)

While «My Books» includes all of the features of «Bookmarks,» it is worth noting that there are additional functionalities that are going to be implemented in the nearest future. It will soon have the option for book editing, including modifying fields such as categories or descriptions. However, to ensure the integrity of the content, there will be restrictions on the extent to which users can modify an already uploaded book.

The «News» page is a centralized location for all notifications related to the user's matches with other users. Each notification is displayed as an individual card that provides essential information about the matched books and the users. These cards feature two buttons, one for dismissing the notification and the other for accessing the contact information of the matched user.

The «Add a Book» page provides users with a comprehensive set of options to add new books to the application's repository. The ISBN input field (**Figure 12**) allows for dynamic data retrieval from Google's Books API, generating a pre-populated card that can be modified to suit the user's specific needs. Upon completion, the user can save the card.

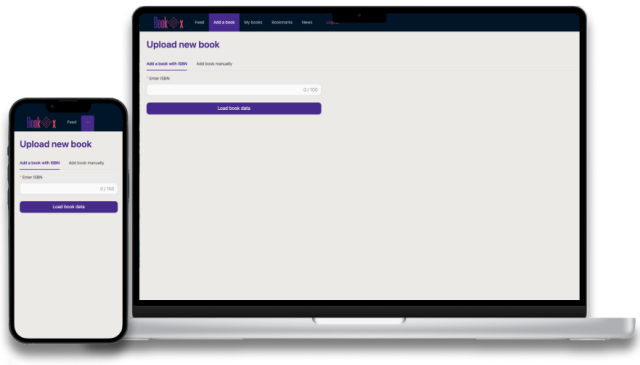


Figure 12. Uploading new book with ISBN. (Source: author)

For those who prefer to add books manually, there is an option to upload or take a photo, as well as enter the book's title, author, language, description, and categories (**Figure 13**).

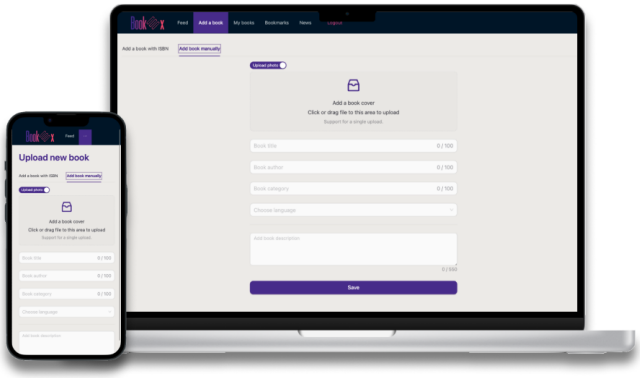


Figure 13. Uploading new book manually. (Source: author)

Conclusion

The primary objective of the thesis was to develop a web application that offers a modern and optimized experience for book exchange. This was accomplished by conducting a thorough comparative analysis of currently available applications on the market and their respective advantages and limitations. The overarching goal was to promote sustainable and accessible book-sharing practices among individuals.

The development process was divided into two primary components: server-side development using Node.js and client-side development using React and Next.js. During server-side development, the following objectives were completed: the creation and establishment of a MySQL database to store and manage data, the configuration of a server to process API requests, the development of a model system for existing tables to facilitate future scalability, and the implementation of business logic to handle data retrieval and processing, as well as the transmission of data to the client. Furthermore, the JWT token was integrated to ensure that only authenticated users could access system routes.

During client-side development, the following goals were accomplished: the creation and configuration of a React application, the creation of an architecture to separate views (pages and components) and business logic (services, utils, HOCs), the creation of reusable components to facilitate scalability and refactoring, the development of a user-friendly UX/UI, support for responsive design, and the implementation of authentication guards to restrict unauthorized access to the system.

All set goals were achieved, meeting the intended objectives of the thesis. However, there is an ample room for improvement in terms of both functionality and scalability. Future plans include the incorporation of the web socket technology, updating the application design, adding the ability to filter data, and enabling book reviews and user ratings. Additionally, the author aims to begin beta-testing and expand the application's reach to additional cities to maximize accessibility for users.

Resümee

Lõputöö esmane eesmärk oli arendada veebirakendus, mis pakub kaasaegset ja optimeeritud kogemust raamatute vahetamiseks. Selleks viidi läbi põhjalik võrdlev analüüs praegu turul olevate rakenduste ja nende vastavate eeliste ning piirangute kohta. Üldeesmärk oli edendada jätkusuutlikke ja kättesaadavaid raamatute vahetamise tavasid üksikisikute vahel.

Arendusprotsess jagunes kaheks põhikomponendiks: serveripoolne arendus Node.js-i abil ja kliendipoolne arendus Reacti ja Next.js-i abil. Serveripoolse arenduse käigus viidi lõpule järgmised eesmärgid: MySQL-andmebaasi loomine ja loomine andmete säilitamiseks ja haldamiseks, API-päringute töötlemiseks vajaliku serveri konfigureerimine, olemasolevate tabelite mudelisüsteemi arendamine, et hõlbustada tulevast skaleeritavust, ning äriloogika rakendamine andmete päringu ja töötlemise ning kliendile andmete edastamise jaoks. Lisaks sellele integreeriti JWT-token, et tagada, et ainult autenditud kasutajad saaksid juurdepääsu süsteemi marsruutidele.

Kliendipoolse arenduse käigus saavutati järgmised eesmärgid: Reacti rakenduse loomine ja konfigureerimine, arhitektuuri loomine, et eraldada vaated (leheküljed ja komponendid) ja äriloogika (teenused, utiliidid, HOCid), taaskasutatavate komponentide loomine, et hõlbustada skaleeritavust ja refaktooringut, kasutajasõbraliku UX/UI arendamine, responsive disaini toetamine ja autentimiskaitsmete rakendamine, et piirata volitamata juurdepääsu süsteemile.

Kõik püstitatud eesmärgid saavutati, mis vastab lõputöö eesmärkidele. Siiski on nii funktsionaalsuse kui ka skaleeritavuse osas veel palju arenguruumi. Tulevikus on kavas lisada veebisokettide tehnoloogia, ajakohastada rakenduse disaini, lisada võimalus andmeid filtreerida ning võimaldada raamatute ülevaated ja kasutajate hinnangud. Lisaks kavatseb autor alustada beetatestimist ja laiendada rakenduse ulatust täiendavatesse linnadesse, et maksimeerida kasutajate jaoks ligipääsetavust.

References

Axios, 2023. *Getting started. What is Axios?*. Available at https://axios-http.com/docs/intro, accessed March 14, 2023.

W3Schools, 2023. *What is JSON?*. Available at https://www.w3schools.com/whatis/whatis_json.asp, accessed April 17, 2023.

Cambridge Dictionary, 2023a. *Web app*. Available at https://dictionary.cambridge.org/dictionary/english/web-app, accessed April 16, 2023.

Wikipedia, 2023a. *Access Token*. Available at https://en.wikipedia.org/wiki/Access_token, accessed April 16, 2023.

Cambridge Dictionary, 2023b. *API*. Available at https://dictionary.cambridge.org/dictionary/english/api?q=API, accessed April 16, 2023.

Wikipedia, 2023b. *IDE*. Available at https://en.wikipedia.org/wiki/Integrated_development_environment, accessed April 16, 2023.

Cambridge Dictionary, 2023c. *HTTP*. Available at https://dictionary.cambridge.org/dictionary/english/http?q=HTTP, accessed April 16, 2023.

Cambridge Dictionary, 2023d. *HTML*. Available at https://dictionary.cambridge.org/dictionary/english/html?q=HTML, accessed April 16, 2023.

Cambridge Dictionary, 2023e. *CSS*. Available at https://dictionary.cambridge.org/dictionary/english/css?q=CSS, accessed April 16, 2023.

EducativeIO, 2023. *What is server-side rendering?*. Available at https://www.educative.io/answers/what-is-server-side-rendering, accessed March 14, 2023.

TypeScript official documentation, 2023. Available at https://www.typescriptlang.org/, accessed April 16, 2023.

MDNWebDocs, 2023. *JavaScript — Dynamic client-side scripting*. Available at https://developer.mozilla.org/en-US/docs/Learn/JavaScript, accessed April 16, 2023.

Cambridge Dictionary, 2023f. *UI*. Available at https://dictionary.cambridge.org/dictionary/english/ui?q=UI, accessed April 16, 2023.

Cambridge Dictionary, 2023g. *UX*. Available at https://dictionary.cambridge.org/dictionary/english/ux?q=UX, accessed April 16, 2023.

Cambridge Dictionary, 2023h. *SEO*. Available at https://dictionary.cambridge.org/dictionary/english/seo?q=SEO, accessed April 16, 2023.

JWT, 2023. *Introduction to JSON Web Tokens*. Available at https://jwt.io/introduction, accessed April 16, 2023.

Wikipedia, 2023c. *NPM*. Available at https://en.wikipedia.org/wiki/Npm_(software), accessed April 16, 2023.

Wikipedia, 2023d. *SQL*. Available at https://en.wikipedia.org/wiki/SQL, accessed April 16, 2023.

Cambridge Dictionary, 2023i. *Front end*. Available at https://dictionary.cambridge.org/dictionary/english/front-end, accessed April 16, 2023.

Cambridge Dictionary, 2023j. *Back end*. Available at https://dictionary.cambridge.org/dictionary/english/back-end, accessed April 16, 2023.

Cambrigde Dictionary, 2023k. Debugging. Available at https://dictionary.cambridge.org/dictionary/english/debug?q=debugging, accessed April 16, 2023.

Rouse M., 2011. *Object-Relational Mapping*. Available at https://www.techopedia.com/definition/24200/object-relational-mapping--orm, accessed April 16, 2023.

Cambridge Dictionary, 2023l. *SWOT*. Available at https://dictionary.cambridge.org/dictionary/english/swot, accessed April 16, 2023.

Wikipedia, 2023e. *MVP*. Available at https://en.wikipedia.org/wiki/Minimum_viable_product, accessed April 16, 2023.

Wikipedia, 2023f. *MVC*. Available at https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller, accessed April 16, 2023.

Wikipedia, 2023g. *Primary key*. Available at https://en.wikipedia.org/wiki/Primary_key, accessed April 16, 2023.

Wikipedia, 2023h. *Foreign key*.Available at https://en.wikipedia.org/wiki/Foreign_key, accessed April 16, 2023.

Wikipedia, 2023i. GUI. Available at https://en.wikipedia.org/wiki/Graphical_user_interface, accessed on April 16, 2023.

ReactJs Docs, 2023. HOC. Available at https://legacy.reactjs.org/docs/higher-order-components.html, accessed April 16, 2023.

Google Cloud, 2023. *What is a relational database?*. Available at https://cloud.google.com/learn/what-is-a-relational-database, accessed April 16, 2023.

ReactDev, 2023. Available at https://react.dev/, accessed March 14, 2023.

NextOrg, 2023. Available at https://nextjs.org/, accessed March 14, 2023.

TypeScriptLang, 2023. Available at https://www.typescriptlang.org/, accessed March 14, 2023.

AntDesign, 2023. Available at https://ant.design/, accessed March 14, 2023.

NodeJs, 2023. *About Node.js*. Available at https://nodejs.org/en/about, accessed March 14, 2023.

MySql, 2023. *MySql Workbench*. Available at https://www.mysql.com/products/workbench/, accessed March 14, 2023.

Postman, 2023. Available at https://www.postman.com/, accessed March 14, 2023.

Appendices

Appendix 1. Matching algorithm SQL-query listing

```
exports.getFavoritedBooks = async (req, res) => {
 const { userId } = req;
 try {
  const query = `SELECT
   u.id AS user_id,
   u.name AS user_name,
   u.email, u.phone, u.telegram,
   GROUP_CONCAT(DISTINCT b.title ORDER BY b.id ASC SEPARATOR
'||') AS liked_book_title,
   GROUP_CONCAT(DISTINCT b.id ORDER BY b.id ASC SEPARATOR
'||') AS liked_book_ids,
   (SELECT COUNT(*) FROM u880646468_books_exchange.likes l
WHERE l.userId = u.id AND l.bookId = b.id AND l.liked = true)
AS liked_by_current_user,
   (SELECT GROUP_CONCAT(DISTINCT b2.title ORDER BY b2.id ASC
SEPARATOR '||')
   FROM ${process.env.DB_DATABASE}.likes l2
   JOIN ${process.env.DB_DATABASE}.books b2 ON l2.bookId =
b2.id
   WHERE l2.userId = 3 AND l2.liked = true AND b2.userId =
u.id) AS liked_book_by_current_user
   FROM
   ${process.env.DB_DATABASE}.likes l
   JOIN ${process.env.DB_DATABASE}.users u ON l.userId = u.id
   JOIN ${process.env.DB_DATABASE}.books b ON l.bookId = b.id
 WHERE
   l.liked = true
 AND l.bookId IN (SELECT id FROM
u880646468_books_exchange.books WHERE userId = ${userId})
 AND l.userId <> ${userId}
 AND EXISTS (
```

```
 SELECT 1 FROM ${process.env.DB_DATABASE}.likes l2
 JOIN ${process.env.DB_DATABASE}.books b2 ON l2.bookId =
b2.id
 WHERE l2.userId = u.id AND l2.liked = true AND b2.userId =
${userId}
)
AND EXISTS (
 SELECT 1 FROM ${process.env.DB_DATABASE}.likes l3
 JOIN ${process.env.DB_DATABASE}.books b3 ON l3.bookId =
b3.id
 WHERE l3.userId = ${userId} AND l3.liked = true AND
b3.userId = u.id
)
 GROUP BY
u.id
 ORDER BY
u.id;`
const rows = await sequelize.query(query, { type:
Sequelize.QueryTypes.SELECT });
  await getNotifications(rows, userId)
    .then(notifications => {
     res.status(200).json({rows, notifications});
    })
    .catch((err) => {
     res.status(404).json({ message: 'Error finding
notifications!' });
    });
 } catch (err) {
  res.status(404).json({ message: 'Error finding books!' });
 }
}
```

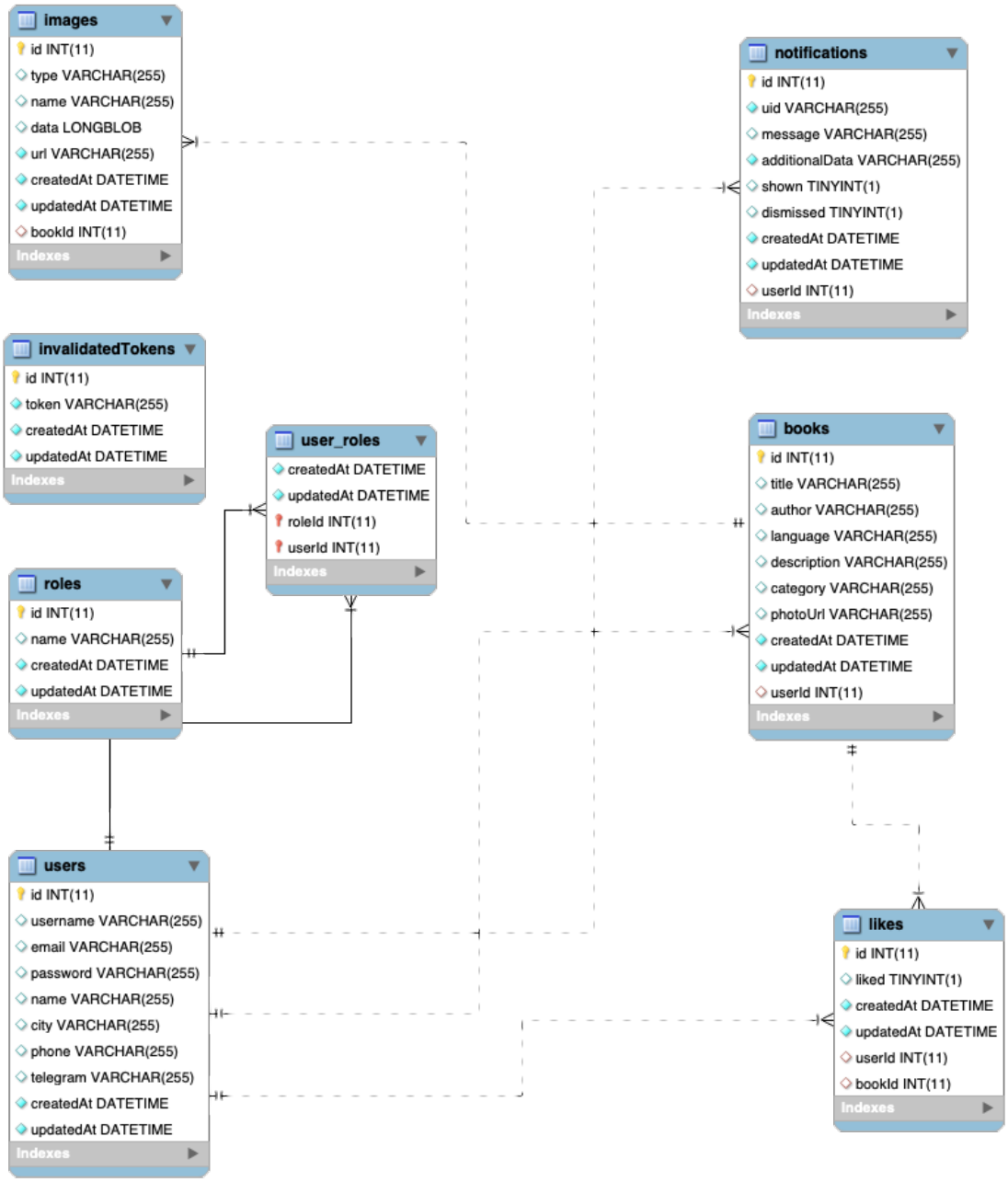
Appendix 2. Checking and sending notifications

```
async function getNotifications(rows, userId) {
  const notifications = [];
  for(const row of rows) {
    const ids = row.liked_book_ids.split("||").join("-");
    const notificationKey = `${row.user_id}-${ids}-${userId}`;
    const existingNotification = await Notification.findOne({ where: { uid: notificationKey } });
      if (!existingNotification) {
        const bookTitles = row.liked_book_title.split("||").join(" & ");
        const likedBookTitled = row.liked_book_by_current_user.split("||").join(" & ");
        const message = `Your book(s)
        "${bookTitles}" was liked by ${row.user_name}.
        You liked his book(s) named "${likedBookTitled}".`;
        let additionalData = "";
        if(!!row.email) {
          additionalData = additionalData + `Email: ${row.email}||`;
        }
        if(!!row.telegram) {
          additionalData = additionalData + `Telegram: ${row.telegram}||`;
        }
        if(!!row.phone) {
          additionalData = additionalData + `Phone number: ${row.phone}||`;
        }

        const notification = {
            uid: notificationKey,
            message: message,
            additionalData: additionalData,
            shown: true,
            dismissed: false,
            userId: userId
        };
        Notification.create(notification);   Kozhevnikova, 26.03.2023, 18:14 • notifications add
        notifications.push(notification);
      }
  }
  return notifications;
}
```

Appendix 3. A polling system to show fresh notifications

```
const showNotification = (data: INotification[]) => {
  if (data && data.length > 0) {
    data.forEach(item => {
      const args = {
        message: item.message,
        duration: 10,
        onClose: () => console.log('Notification closed'),
      };
      notification.open(args);
    });
  }
};

useEffect( effect: () => {   Kozhevnikova, 14.04.2023, 16:42 • updated
  let intervalId: any;

  if (!!authToken) {
    intervalId = setInterval( handler: async () => {
      await getFavorited() ...
        .then(result => {
          console.log(result)
          showNotification(result.data?.notifications)
          setCounter( value: result.data?.rows?.length || 0)
        }) Promise<void>
        .catch(err => setError(err))

    }, timeout: 10000); // Poll every 10 seconds
  }

  return () => clearInterval(intervalId);
}, deps: [!!authToken]);
```

Appendix 4. ER-diagram



Appendix 5. Relational model description

```
db.role.belongsToMany(db.user, options: {
  through: "user_roles",
  foreignKey: "roleId",
  otherKey: "userId"
});
db.user.belongsToMany(db.role, options: {
  through: "user_roles",
  foreignKey: "userId",
  otherKey: "roleId"
});
db.user.hasMany(db.book);
db.book.belongsTo(db.user, options: {
  foreignKey: {
    name: 'userId'
  }
});

db.user.hasMany(db.like, options: {foreignKey: 'userId'})
db.book.hasMany(db.like, options: {foreignKey: 'bookId'})

db.like.belongsTo(db.book, options: { foreignKey: 'bookId' });
db.like.belongsTo(db.user, options: { foreignKey: 'userId' });

db.user.hasMany(db.notification, options: {foreignKey: 'userId'})
db.notification.belongsTo(db.user)
```

Appendix 6. Source code

Gitlab links contain the following items:

Backend application files – https://gitlab.com/Damshka/bookexbackend

Frontend application files – https://gitlab.com/Damshka/bookexfrontend

Both backend and frontend repositories have README files, that contain information about starting up server and client sides. It is necessary to use Node.js version 16.15.0 or higher to make sure all packages are installed correctly.

Also, to set up a database, in the backend repository it is needed to set the correct credentials, depending on where you set your database, and then run the local backend server to create all tables based on the existent model.

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, _____Daria Kozhevnikova_____,

(autori nimi)

annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

__DEVELOPMENT OF A WEB APPLICATION FOR BOOKS EXCHANGE__,

(lõputöö pealkiri)

mille juhendaja on __Andre Sääsk__,

(juhendaja nimi)

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commons'i litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Daria Kozhevnikova

10.05.2023