UNIVERSITY OF TARTU

FACULTY OF SOCIAL SCIENCES

NARVA COLLEGE

INFORMATION TECHNOLOGY SYSTEMS DEVELOPMENT

Artur Pushkov

DEVELOPING A SOCIALIZATION APPLICATION FOR INTERNATIONAL STUDENTS OF NARVA COLLEGE

Diploma thesis

Supervisor: Assistant 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/

CONTENTS

TERMS AND NOTATIONS

React Native — A JavaScript framework designed for building cross-platform apps, including applications for iOS, Android, as well as web applications.

Babel — It is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

Go programming language — Go is a statically typed, compiled high-level programming language designed at Google.

PostgreSQL — PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads.

REST — Representational State Transfer is an architectural style that defines a set of constraints and properties based on HTTP. (REST ... 2018)

JSON — JavaScript Object Notation is a text syntax that facilitates structured data inter-change between all programming languages. (JavaScript Object Notation ... 2017)

Mobile application — Computer program or software application designed to run on a mobile device such as a phone, tablet, or watch.

HTTP — Hypertext Transfer Protocol

API — Application Programming Interface

CSS — Cascading Style Sheets

Tailwind — Tailwind CSS is a utility-first CSS framework for rapidly building modern websites without ever leaving your HTML.

Front-end — The parts of a computer, piece of software, or website that are seen and directly used by the user. (Cambridge... 2018a)

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... 2018b)

Access token — In computer systems, an 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. (Access token ... 2017)

Cross-platform development — Cross-platform development refers to the creation of software applications that can run on multiple platforms or operating systems, such as Windows, Mac, Linux, Android, and iOS.

INTRODUCTION

Higher education internationalization has resulted in an increasing number of international students studying in different countries. In addition to providing numerous opportunities for personal growth, studying abroad also poses challenges for international students, particularly in terms of socialization. In order to develop cultural competence and connect with peers, socialization is a vital aspect of the international student experience.

The challenge that international students at Narva College face in socializing is addressed in this diploma. Due to cultural and linguistic barriers, international students often have difficulty socializing in a foreign country. Feeling isolated and lonely can negatively impact their overall student experience as a result of these barriers.

As Narva College attracts a high number of international students, it is important to provide an environment that fosters socialization and cultural exchange. As a result, a solution is needed that addresses these challenges. International students can benefit from this solution by connecting with one another, sharing experiences, and learning about other cultures.

This bachelor diploma proposes to design and develop a socialization application. It caters to the specific needs of international students. International students can use the application to connect with each other, establish support networks, and exchange cultural information. The experience of socializing will be enhanced.

The aim is to design and develop a socialization application that addresses international students' socialization challenges.

This diploma has the following objectives:

- Identify and prioritize the key features of the application based on audience insights;
- Create wireframes and prototypes of the application's user interface using Figma design to visualize the layout, navigation, and functionality of the application;
- Develop the application's front-end using React Native;

- Develop the back-end of the application using Go and PostgresSQL, implementing the application's logic and functionality, such as user authentication, socialization features, and data storage;

- Test and debug manually the application to ensure its reliability, functionality, and user experience, using testing frameworks and tools for React Native and Go;

The work consists of an introduction; the first chapter, which describes the author's tools and technologies.; the second chapter, which describes the research and design process of the application; and the third chapter, which demonstrates the development process. The full source code of the scripts and applications created in this work is presented in Appendix 1.

1 USED TECHNOLOGIES

There are several technologies employed in this project, as well as an explanation of why these technologies were chosen by the author for the project.

1.1 React Native

React Native is an open-source mobile application framework developed by Facebook. By using JavaScript and React, developers can create native mobile apps for iOS, Android, and other platforms. It improves performance and creates a consistent look and feel across multiple platforms by translating JavaScript code into native UI components. This technology was chosen for the development of the application's frontend because it is a popular and robust framework that allows the app to be developed in a single codebase for both iOS and Android, thus reducing development time and effort.

1.2 React Native Navigation

React Native Navigation is a comprehensive library for handling navigation and routing in React Native applications. It provides a native navigation experience by leveraging platform-specific navigation components and integrating them seamlessly into the React Native ecosystem. Among the features offered by this library are screen transitions, deep linking, and tab navigation. For this project, it was chosen because of its native navigation experience, customizable UI elements, and a wide range of features that enhance the user experience.

1.3 Expo

Expo is a set of tools and services based around React Native that simplifies app development. It is a platform-independent solution that provides a managed workflow for developing, building, and deploying React Native applications, without requiring any platform-specific tools or configurations. The Expo platform offers a wide range of pre-built components, libraries, and APIs that developers can use to reduce the amount of boilerplate code and accelerate the development process. To streamline the development process of the React Native app, it was chosen as the best option. As a result, the author was able to access a

wide variety of pre-built components and modules, which simplified the process of developing an app and reduced the amount of manual configuration required.

1.4 Go programming language

Go, also known as Golang, is an open-source programming language developed by Google. It features strong typing, garbage collection, and built-in concurrency features and is designed for systems programming. Due to its efficient compilation and execution model, Go is ideal for building high-performance server-side applications. Due to its performance, simplicity, and ease of use, the author chose this language for the backend of the networking application.

1.5 Go Fiber

Go Fiber is a lightweight and flexible web framework for Go that was selected to build the backend API for this networking application. Using Fiber simplifies the building of RESTful APIs, as it is inspired by the Node.js framework Express. As a result of its performance, middleware support, high-performance, flexibility, minimalistic design, extensibility, and ease of use, it was chosen by the author for this project.

1.6 PostgreSQL

PostgreSQL is a powerful, enterprise-class, open-source relational database management system (RDBMS). It supports advanced data types, such as arrays, hstores, and JSON, as well as extensible indexing and full-text search. In addition to robust transaction management, PostgreSQL delivers high concurrency control, strong consistency guarantees, making it an ideal platform for mission-critical applications and large-scale data processing. In this project, PostgreSQL was chosen for its robust feature set, high performance, and compatibility with a variety of data types because it is a powerful and reliable open-source relational database management system. In this networking application, it is ideal for storing and managing data because it can handle complex queries and transactions.

1.7 Railway

Railway is a fully managed platform-as-a-service (PaaS) that simplifies the deployment, scaling, and maintenance of cloud-based applications. It offers a streamlined deployment

process with continuous integration and delivery (CI/CD) built-in, automatic scaling, and a global content delivery network (CDN). The Railway dashboard provides a convenient way to manage infrastructure, monitor application performance, and access logs and metrics. Due to its ease of use and simplicity, Railway was chosen by the author as a deployment platform. As well as being cost-effective, Railway offers powerful features that allow developers to easily deploy and manage applications. It is also secure and reliable, allowing developers to focus on their applications instead of worrying about infrastructure.

1.8 Tailwind

Tailwind is a utility-first CSS framework for rapidly building custom user interfaces. An extensive set of utility classes make it easy to create responsive, consistent, and maintainable designs. Tailwind CSS encourages a component-based design approach, allowing developers to create reusable UI components and promoting consistency across the application. Also, it can be configured easily through configuration files and integrates well with modern build tools. The author chose Tailwind for styling the networking application because it is a utility-first CSS framework that makes it easy to create responsive and modern designs. With Tailwind, the author quickly prototyped and customized user interfaces while maintaining a consistent design system.

2 RESEARCH AND DESIGN

2.1 Audience analysis

Students from diverse cultural backgrounds pursuing higher education at Narva College can take advantage of a socialization application developed specifically for international students. The college offers a wide range of courses and programs for students of all academic levels.

The application is designed to help international students adjust to the foreign academic and cultural environment by addressing their socialization needs. It may be difficult for these students to establish relationships with their peers and integrate into the local community due to isolation and culture shock.

It facilitates socialization by providing a platform for international students to engage in social and cultural activities with their peers. As a result, cross-cultural exchanges will be promoted as well as a sense of belonging will be fostered. There will be an extensive range of features offered by the application, including event chat rooms, social networking tools, and a range of other features. Interaction and connections between students will be made possible by these features.

Narva College's socialization application is intended to be an invaluable resource for international students. As well as providing them with a supportive and inclusive environment, it also allows them to develop their social and cultural competencies and thrive academically and socially.

In terms of interests, the audience may be interested in a range of social and cultural activities. In addition to connecting with their peers, they will be able to engage in cross-cultural exchanges.

Among the audience, it is likely to be predominantly young adults around the age of 18 to 30. It reflects the typical age range of students enrolled in higher education programs. Depending on the specific courses and programs at Narva College, there may be some variation in age.

Individuals with different backgrounds and experiences tend to hold a variety of cultural and personal values. Among the values this audience shares is a desire for social connection and a

sense of belonging. A commitment to academic success, a respect for diversity and inclusion, as well as an openness to cross-cultural exchange are also included.

Narva College's socialization application is designed for international students who share a common desire to connect with others and engage in social and cultural activities that are beneficial to academic and personal development. It is aimed at a diverse and dynamic group of individuals.

2.2 Functional requirements

Functions of an application are determined by functional requirements. Therefore, they comply with the request of the user and the task being performed.

- User can access from mobile client application;
- User can log in with unique credentials;
- User can sign up with unique credentials;
- Unique token can be created for registered user;
- User can connect and communicate with other users through chat;
- User can customize their own profile settings;
- User can log out from their account;
- User can delete chat with other users from own chat history;

2.3 Non-functional requirements

A socialization application requires the following non-functional requirements:

- Performance: The application should provide fast response times and low latency to ensure a seamless user experience;
- Scalability: The application should be able to handle a growing number of users and connections without degradation in performance or user experience;
- Reliability: The application should be fault-tolerant and able to recover from failures quickly, ensuring uninterrupted service for users;

- Security: The application should provide secure user authentication and protect sensitive user data (e.g., passwords and personal information). It should also prevent unauthorized access to user data and system resources;

- Application maintainability: The application should be easy to maintain and update, with a modular and clean codebase, clear documentation, and adherence to best practices;

- Usability: The application should have an intuitive user interface that is easy to navigate and understand, ensuring a smooth user experience;

- Extensibility: The application should be designed to accommodate future changes and enhancements, such as adding new features or modifying existing ones;

- Monitoring and Logging: The application should include monitoring and logging capabilities to track its performance, detect issues, and facilitate troubleshooting;

- Data Integrity: The application should ensure data consistency and integrity, preventing data corruption or loss during storage, retrieval, and transmission;

- Resource Efficiency: The application should efficiently utilize system resources like CPU, memory, and storage, optimizing its performance and minimizing its footprint on users' devices or servers;

- Cross-Platform Compatibility: The application should be compatible with different platforms and devices, ensuring a consistent user experience across various operating systems, browsers, and screen sizes;

- Real-time Communication: The application should support real-time communication features, like chat, notifications, or updates, providing a seamless and interactive user experience;

- Version Control: The application's source code should be managed using a version control system, enabling tracking of changes, collaboration between developers, and efficient release management.

2.4 User flow

When first meeting a user, an onboarding screen appears where the new user is briefly told about the application. Next, the login screen displays. If the user enters the correct login and password, opens the main screen. The user can also click on the "Register" button if he/she does not yet have an account. Then a registration form will open. After successful registration the main screen opens.

In the main menu a user can see the list of people from Narva College who have already registered and are looking for friends of interest. The main screen consists of user cards and a navigation menu. A user's card has all the important information, such as their first and last names, their photo, their age, their interests, and a brief description of themselves. There is a "Start a chat" button that opens a chat with that user when you click on it.

In the bottom navigation menu there are three buttons: home, connections, settings. "Home" button leads to the application's main page. Pressing this button refreshes the page each time.

"Settings" button leads to his personal account settings, where the user can change information about himself: name and surname, interests, text about himself, age, photo. And he can also log out of his account, returning to the login page. The user can delete their account completely by clicking on the "Delete Account" button.

"Connections" button opens a screen with a list of all chats with other users. This section stores chat history. Chats can be opened and deleted. When a user clicks on a chat, it opens. If the user clicks on the "Delete" icon, the chat is deleted. The list of chats consists of significant information for the user: name and surname of the interlocutor, photo. Separately, the opened chat stores message history. Also the user will see the name and surname of the interlocutor, photo and messages .

3 APPLICATION DEVELOPMENT

3.1 Context level 0 diagram

This context diagram illustrates how data is processed between the actors in an information system (**Figure 1**).

A diagram illustrating the format and action of the process is shown by arrows. The client enters the required data into the client application form. Data is converted into JSON format and sent to the backend server via AJAX requests by the client application. API servers prepare database statements and send queries to database servers. Query is executed by the database management system and/or the results of the query are returned to the API if a select statement was executed. Data is returned to the client application in JSON format by the API server. A frontend application receives that data and displays it to the user.

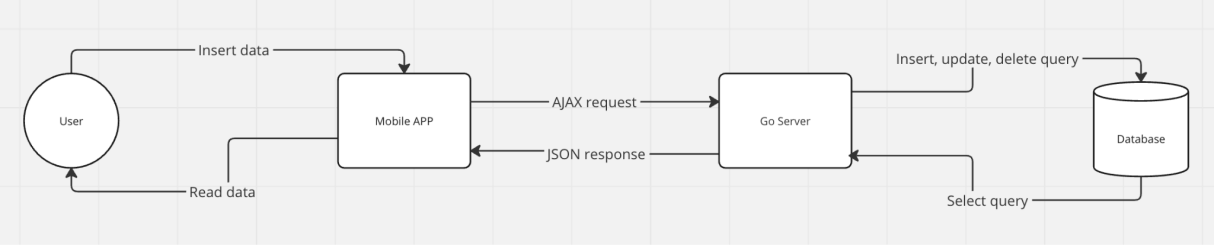

***Figure 1**.Context level 0 diagram. (Source: author)*

3.2 Database development

PostgreSQL has been chosen for storing all required data because it has a wide range of advanced features, including support for JSON, and because the author has experience working with it.

Currently the database contains several entities and 5 tables, which are designed for future API features. The USERS table stores all user information such as name, email, password, bio, and profile picture. The INTERESTS table stores all the interests users can choose from. The USER_INTERESTS table is a many-to-many relationship table between USERS and INTERESTS. It allows each user to have multiple interests, and each interest to be associated with multiple users. The CONNECTIONS table stores all connections between users. It

includes the user IDs of the two connected users and the timestamp of when the connection was created. The structure of the database is shown on **Figure 2**.

Database key entities are:

1. users
2. connections
3. messages
4. interests
5. user_interests

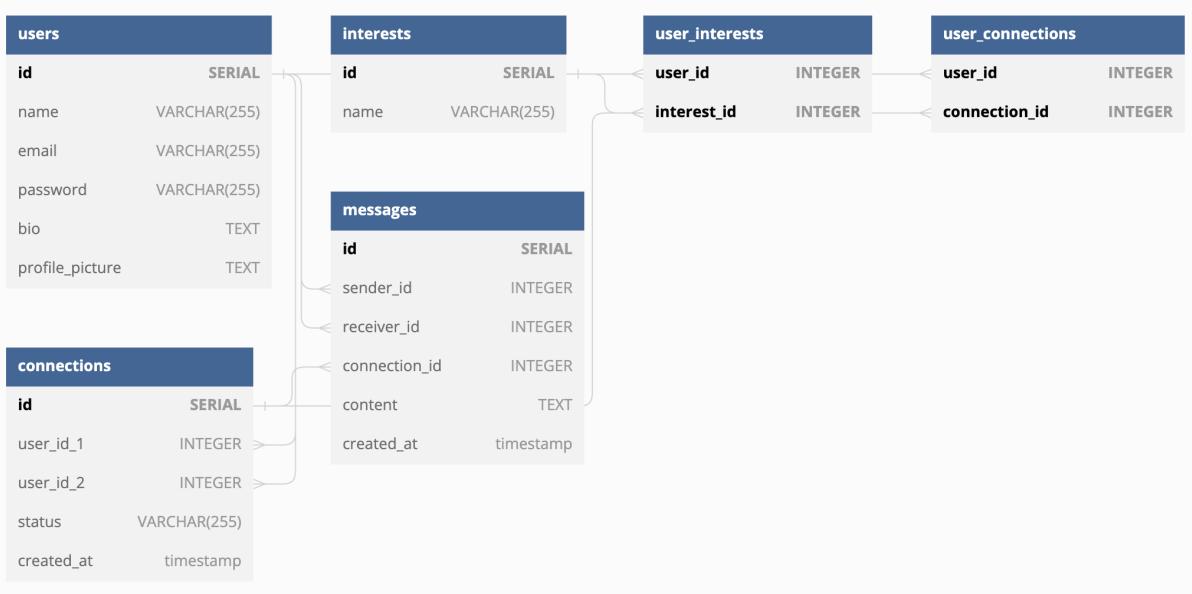

Figure 2*.The structure of the database. (Source: author)*

3.3 REST API development

3.3.1 Application architecture setup

Networking application is designed using a layered architecture that separates the presentation, application, and data layers. The presentation layer is responsible for handling the user interface and user interactions, the application layer is responsible for processing data

and performing operations, and the data layer is responsible for managing data storage and retrieval.

With Go and GoFiber as the web framework, the author built the server-side of the application layer, along with a connection to the PostgreSQL database in the database layer.

The server-side folder structure includes controllers, handlers, services, models, middlewares, and configuration, forming the different layers of the application. The controllers handle HTTP requests and responses, and the handlers execute the endpoint logic. The services contain the business logic of the application, and the models represent the data objects used in the application. Middlewares handle HTTP requests and responses interception and modification.

React Native was used by the author to build the presentation layer. Using HTTP requests and responses, the client communicates with the server, allowing seamless integration between the two.

3.3.2 Configuration setup

In order to configure the networking application, the author used a configuration file, which contained all the necessary settings and parameters for the server-side of the application, in order to configure it. In the configuration file, all the configuration variables that are required by the application are stored. The database variables (connection URL which is used to connect to the database), the server variables (the port number that is used by the server), the environment variables (development or production environments), and the jwt secret that serves as the key to the secret key for JSON Web Tokens.

Init is the function that is used for initializing the configuration variables in the configuration file. Using the os.Getenv() function, the configuration variables are set using data from the environment variables, which is read from the environment variables. By using environment variables, the author provided a secure storage method for sensitive information and allowed configuration changes to be made without the need for changing the code.

3.3.3 Database connection

The author connects to the data using a database driver for PostgreSQL called pgx. It provides a low-level, high-performance interface for interfacing with PostgreSQL databases.

The author created a database package, which contains a struct that holds a database connection. In the db.go file, the NewDb() function initializes a new database connection and returns a new Db struct. A new database connection is established by reading the database URL from the configuration file l and using the pgx.Connect() function. When the connection is successful, the NewDb() function returns a new Db struct containing the connection information in the Conn field. The function returns nil if the connection is unsuccessful.

After the database connection is established, the author can utilize the pgx driver to interact with the database.

The author uses SQL queries to interact with the database, which are executed using the pgx driver. In order to prevent SQL injection attacks, SQL queries are written as parameterized statements.

3.3.4 Fiber application initialisation

In order to create a new instance of the Fiber application, the author uses the New() function provided by the Fiber framework. In the initialization of the app, an error handler function is provided that returns a JSON error response in case of an error during a request. The error handler function reads the error code and message from the error object and returns a JSON response containing the error code and message.

Using the Use() function provided by Fiber, the author then adds middleware to the application. The middleware consists of a CORS middleware that permits cross-origin requests only in development mode, as well as a request ID middleware that generates a unique ID for each incoming request.

Additionally, the author adds a logger middleware that logs details of each incoming request. In order to customize the log format, the logger middleware uses the logger.Config[] struct.

Finally, the author launched the Fiber app by calling the Listen() function provided by the Fiber framework. Listen() begins the application and listens for incoming requests on the port specified.

3.3.5 Users implementation

The author implemented the user logic using the users model in the models layer, the users controller in the controllers layer, and the users handler in the handlers layer. To communicate with the database, the author used the services layer.

A new user is created with the provided data and inserted into a database during the registration process. The implementation involves the use of SQL queries and parameterized statements to prevent SQL injection attacks. Before calling the create user method in the users service layer, the users controller must verify that the required data is present and valid. The service layer generates a new JWT using the secret key and user ID, and returns the new user data along with the JWT to the controller if the registration is successful. If registration is unsuccessful, the service layer returns an error message without a JWT.

Data validation is implemented on the handlers layer, where the incoming data is validated to ensure that it meets the expected format and content. If the data is invalid, the handler returns an error message without processing the request.

User authentication is accomplished by verifying the email address and password provided by the client and returning a JWT for authentication. In order to retrieve the user data from the database, SQL queries and parameterized statements are used. Before calling the login method in the users service layer, the user controller must verify that the required data is present and valid. Upon successful login, the service layer generates a new JWT using the secret key and the user ID, and returns the JWT to the controller. The service layer returns an error message without a JWT if the login is unsuccessful.

User updates are implemented by updating the user with the provided data and returning the updated user data to the user. In order to update user data in the database, SQL queries and parameterized statements are used. The user controller is responsible for ensuring that the

required data is present and valid before calling the update user method in the users service layer. Data updated by the service layer is returned to the controller by the service layer.

The user retrieval process is implemented by retrieving the user with the provided ID obtained from the JWT token provided in the authorization header, retrieving the user data using the provided ID from the database and returning it. The implementation involves the use of SQL queries and parameterized statements to retrieve the user data from the database. The user handler is responsible for checking that the ID is present and valid before calling the get user method in the users service layer. The handler returns a JSON response with the user data if the retrieval is successful, and an error message if the retrieval is unsuccessful.

The implementation of the user logic ensures the security and reliability of the application by using parameterized statements and preventing SQL injection attacks. The implementation also includes error handling and validation to ensure that the required data is present and valid. The use of the services layer to connect to the database helps to separate the database logic from the controller and handler logic, and makes the application easier to maintain and modify in the future.

3.3.6 Interests implementation

Using the interest model, service, controller, and handler layers, the author implemented the interests logic. Through the handler layer, the interests are retrieved using SQL queries and parameterized statements.

Using the user_interests table in the database, the application also supports retrieving a user's interests to establish a many-to-many relationship between users and interests. By retrieving the user's ID from the request, we can query the user_interests table. As a result of the retrieval, the interests table is queried, and the resulting interest data is returned in JSON format.

Adding user interests is also implemented using the user_interests table. The controller layer validates the request data, and the service layer inserts the data into the user_interests table. The handler layer returns a JSON response with updated user interest data.

Deleting a user's interest is implemented by using the user's ID and the interest's ID to query the user_interests table. This will remove the corresponding entry. The controller layer checks that the required data is present and valid before calling the delete method in the service layer. The handler layer returns a JSON response with a success message if deletion is successful.

The author also established a many-to-many relationship between users and interests using the user_interests table in the database. This table allows for the mapping of interests to multiple users and users to multiple interests. The application uses this relationship to enable user interests retrieval, addition, and deletion.

As a whole, the author implemented the interest logic using various methods and functions provided by the different layers of the application. To ensure that the required data is present and valid, the implementation includes SQL queries, parameterized statements, error handling, and validation. The application supports retrieving all interests, retrieving a user's interests, adding interests, and deleting interests. The user_interests table establishes a many-to-many relationship between users and interests.

3.3.7 Connections implementation

The author implemented the connections logic using the connections model, service, controller, and handler layers, and also added a new table user_connection to the database schema to establish a many-to-many relationship between users.

Connections are created by inserting a new record into the connections table with the user IDs of the two users and their status values. Additionally, the user_connection table is updated with the ID of the newly created connection and the user ID of the user who created the connection. Validating the request data is handled by the controller layer, and inserting the data into the user_connections and connections tables is handled by the service layer. A JSON response with the newly created connection data is returned by the handler layer.

When deleting a connection between two users, both the connections table and the user_connection table are deleted. Prior to calling the delete method in the service layer, the controller layer checks that the required data is present and valid. If the deletion is successful, the handler layer returns a JSON response.

To retrieve a user's connections, SQL queries are used to query the user_connection table for all records matching the user ID of the user making the request. Querying the connections table for the corresponding connection data is then done with the resulting connection IDs. Through the handler layer, the resulting data is returned in JSON format.

As a result, the author implemented the connections logic using a variety of methods and functions provided by the various layers of the application and also created a table called user_connection in the database schema to establish a many-to-many relationship between users. To ensure that the required data is present and valid, SQL queries, parameterized statements, error handling, and validation are used. By inserting, deleting, and querying records in the connections and user_connection tables, the application can create, delete, and retrieve connections between users.

3.3.8 Chat implementation

A chat feature was implemented using the messages model, services, controllers, and handlers in the networking application. By using Fiber's WebSocket package, the author was able to establish real-time connections between each user and enable real-time chat.

A message is sent by inserting a record into the messages table with the IDs of the sender and receiver users, the connection ID, the message content, and the time it was sent. The controller layer validates the request data, and the service layer inserts the data into the messages table. Upon creating a message, the handler layer returns a JSON response.

A SQL query is used to retrieve messages between two users by querying the messages table for records matching the sender and receiver's user IDs and connection IDs. Through the handler layer, the resultant message data is returned in JSON format. To allow clients to access chat history, the application stores messages in the database.

The messages table in the database schema has an id column as the primary key, a sender_id column as a foreign key referencing the id column in the users table, a receiver_id column as a foreign key referencing the id column in the users table, a connection_id column as a foreign key referencing the id column in the connections table, a content column to store the message content, and a created_at column to store the time the message was created.

The author implemented the chat feature by utilizing a variety of methods and functions provided by the different layers of the application. In order to ensure that the required data is present and valid, the implementation includes SQL queries, parameterized statements, error handling, and validation. With Fiber's WebSocket package, the application provides real-time chat functionality and stores messages in the database to allow clients to access chat history.

3.4 Mobile application development

3.4.1 Application setup

As part of the development process for the networking application, the author used Expo, React Native, React Native Navigation, Tailwind, and TypeScript to create the mobile application.

TypeScript is a statically typed superset of JavaScript that adds optional static typing, interfaces, and other features to improve code quality and developer productivity.

The structure of mobile applications is as follows: screens, components, hooks, and utils. Screens folder contains the main screens for the application, including the login screen, registration screen, home screen, chat screen, profile screen and connections screen. A component folder contains reusable UI components that can be used across multiple screens.

The utils folder includes utilities functions reused across the app.

The hooks folder contains custom hooks.

The mobile application uses Expo for building and deploying to iOS and Android platforms. Expo provides a set of tools and services that make it easy to develop and test the application. These tools and services include a development server, a mobile app for testing, and a cloud-based build service for building and deploying the application.

The application's configuration files include the app.json file. This file contains general configuration options for the application, such as the name, description, and icon, as well as options for building and deploying the application. The tsconfig.json file configures TypeScript, including compiling options and generating type definitions.

The author also configured the application's navigation using the react-navigation and react-native-gesture-handler libraries, which provide navigation components and gesture handling for React Native applications.

To style the mobile application, Tailwind is used, which provides a set of utility classes that can be used to quickly style components and layouts. To enable Tailwind on React Native, the author installed the nativewind library, which includes a Babel plugin that converts Tailwind classes into native styles.

Babel.config.js is modified to include the nativewind/babel plugin. This enables the plugin to process the Tailwind classes and generate the necessary styles for the application.

3.4.2 Screens implementation

The author created several screens to make up the user interface of the application:

- Login screen: This screen allows users to enter their email and password to log in to the application. The screen contains a TextInput component for the email and password fields and a Button component to submit the form. If the user enters invalid credentials, an error message is displayed.
- Registration screen: This screen allows users to create a new account by entering their name, email, password, and selecting interests. The screen contains TextInput components for each field and a Button component to submit the form. If the user enters invalid data, an error message is displayed.
- Home screen: This screen serves as the main screen of the application and displays a list of user cards. Each card contains the user's name, profile picture, bio and interests.
- Profile screen: This screen displays the user's profile information, including their name, email, bio, profile picture, and interests. The user can also edit their profile information by clicking an Edit button
- Connections screen: This screen displays a list of the user's connections. This screen displays the user's name with who the current user is connected to. Clicking on each connection will navigate to the Chat screen.
- Chat screen: This screen allows the user to send and receive real-time messages with another user. The screen contains a list of messages that are displayed in chronological order, with the user's messages on the right and the other user's messages on the left. The user can also send new messages using a TextInput component and a Send button. The example of LoginScreen.tsx implementation is shown on **Figure 3**.

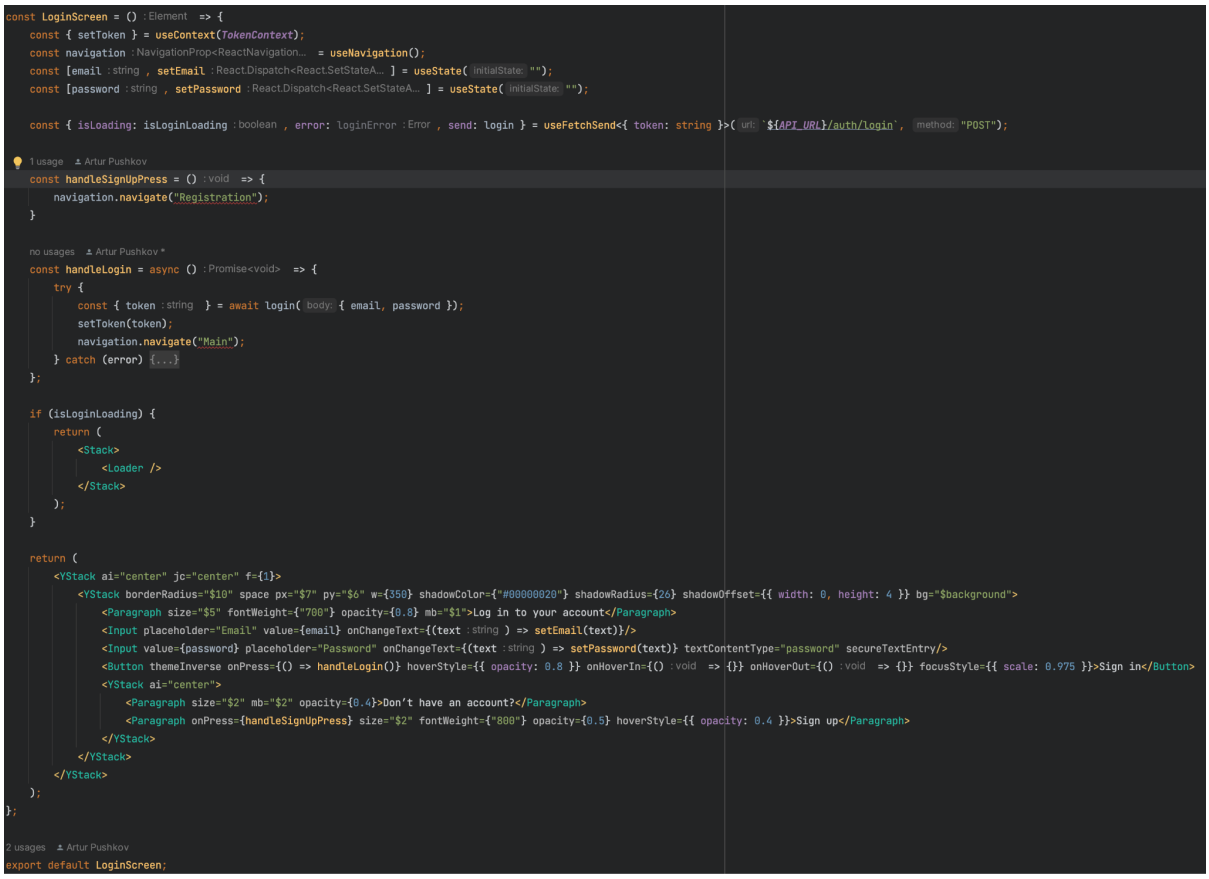

Figure 3*.The example of LoginScreen.tsx implementation. (Source: author)*

3.4.3 Components implementation

Several components were created so they could be reused throughout the app. This allows for a more efficient development process and reduces costs. Furthermore, it allows developers to quickly make changes or additions to an existing component without having to start from scratch.

To define a functional component using React.FC, the author created a new file with a .tsx extension. Components can be named according to their purpose, such as LoginForm or Button. In order to define the component's props, the author used the interface keyword, which specifies the shape of the props object. The props interface can be included as a parameter in the declaration of a functional component. The component's behavior and appearance is implemented by defining JSX markup in the return statement of the component This may include using built-in React Native components such as View, Text, Image, and

TextInput, as well as custom components defined elsewhere in the application. In order to style the component, the author used tailwind utilities classes, which were enabled by nativewind library in the app. To ensure a consistent user interface, the component can be imported and passed any necessary props throughout the application when it has been defined and styled. The example of UserCard.tsx implementation is shown on **Figure 4**.

```tsx
interface UserCardProps {
    user: UserWithInterests
}
3 usages  Artur Pushkov *
const UserCard = ({user}: UserCardProps) => {
    const navigation : NavigationProp<ReactNavigation...  = useNavigation();
    const {token} = useContext(TokenContext);

    const { send: createConnection } = useFetchSend<UserConnection>( url: `${API_URL}/connections`,  method: "POST", token);

    1 usage  Artur Pushkov *
    const handleConnect = async (id: number) : Promise<void>  => {
        const connection : UserConnection  = await createConnection( body: { connect_with_user: id });
        navigation.navigate("Chat", { connection: connection.id, userId: connection.user_id_2 });
    };

  return (
      <Card key={user.id} elevate size="$4" mb="$5" bordered>
          <Card.Header padded>
              <YStack>
                  {user.profile_picture && (
                    <Stack mb="$2" overflow="hidden" borderRadius="$2"> <Image borderRadius="$2" source={{ uri: user.profile_picture, width: 400, height: 200 }}/></Stack>
                  )}
                  <H2>{user.name}</H2>
              </YStack>
          </Card.Header>
          <YStack px="$4">
              <Paragraph mb="$2" mt={2}>Interests:</Paragraph>
              <XStack space="$2" mb="$4">
                  {user.interests?.map((interest : Interest ) => (
                      <Stack key={interest.id + "-" + user.id} borderColor="$blue8" backgroundColor="$blue5" borderWidth="$1" borderRadius="$4" px="$3" py="$1">
                          <Paragraph size="$3" fontWeight="bold">
                              {interest.name}
                          </Paragraph>
                      </Stack>
                  ))}
              </XStack>
              <Paragraph>About me: {user.bio}</Paragraph>
          </YStack>
          <Card.Footer>
              <Button onPress={() => handleConnect(user.id)} f={1} fontSize="$5" themeInverse borderRadius="$4">Connect</Button>
          </Card.Footer>
      </Card>
  );
}

2 usages  Artur Pushkov
export default UserCard
```

Figure 4*.The example of UserCard.tsx implementation. (Source: author)*

CONCLUSION

During this diploma project, an application was developed to provide socialization assistance to international students at Narva College. This application was specifically developed to allow international students to be able to meet the socialization needs they have.

Using the developed application, international students can register, create profiles, and browse other users' profiles to find friends who share their interests or interests similar to their own. Furthermore, the application also provides a chat feature that enables users to communicate with their fellow users within the application. It was decided to use Go and PostgreSQL in the back end of the application in order to enable high traffic volumes, user authentication, and secure data storage in the back end. The chat feature was also designed to be fast, reliable, and secure. The application was tested on multiple platforms to ensure that it performs well under different conditions. The user authentication system was also designed to be secure and reliable.

It is possible to improve the application in the future by incorporating new features that will enhance the user experience of the application. In addition to the features already present in the application, it could also be expanded in order to enable users to explore the city of Narva, its tourist attractions, as well as events that are held in the area. There is also the possibility that the application can be integrated with social media platforms such as Facebook and Twitter so that users will be able to share their experiences with their friends, followers, and family. Furthermore, there are facilities that enable video chatting and virtual events to be incorporated in order to increase socialization and communication between the users.

In general, with the development of this socialization application, we have been able to make significant strides forward in our attempts to solve the socialization problems of international students. As the application develops, it will become even more useful and user-friendly.

RESÜMEE

Selle diplomiprojekti käigus töötati välja rakendus, mille eesmärk on pakkuda rahvusvahelistele üliõpilastele Narva kolledžis sotsialiseerumisabi. See rakendus töötati välja spetsiaalselt selleks, et rahvusvahelised üliõpilased saaksid rahuldada oma sotsialiseerumisvajadusi.

Välja töötatud rakenduse abil saavad välisüliõpilased registreeruda, luua profiile ja sirvida teiste kasutajate profiile, et leida sõpru, kes jagavad nende huvisid või nendega sarnaseid huvisid. Lisaks pakub rakendus ka vestlusfunktsiooni, mis võimaldab kasutajatel rakenduse sees oma kaaslastega suhelda. Rakenduse tagaküljel otsustati kasutada Go ja PostgreSQL-i, et võimaldada suurt andmemahtu, kasutajate autentimist ja turvalist andmete säilitamist tagaküljel. Samuti kujundati vestlusfunktsioon nii, et see oleks kiire, usaldusväärne ja turvaline. Rakendust testiti mitmel platvormil, et tagada selle hea toimimine erinevates tingimustes. Kasutajate autentimissüsteem kujundati samuti turvaliseks ja usaldusväärseks.

Rakendust on võimalik tulevikus täiustada, lisades uusi funktsioone, mis parandavad rakenduse kasutajakogemust. Lisaks rakenduses juba olemasolevatele funktsioonidele võiks seda laiendada, et võimaldada kasutajatel tutvuda Narva linna, selle turismiobjektidega ning piirkonnas toimuvate üritustega. Samuti on võimalik, et rakendus saab integreerida sotsiaalmeedia platvormidega, nagu Facebook ja Twitter, et kasutajad saaksid oma kogemusi jagada oma sõprade, jälgijate ja pereliikmetega. Lisaks on olemas võimalused, mis võimaldavad videovestlust ja virtuaalseid üritusi integreerida, et suurendada kasutajate vahelist suhtlemist ja suhtlemist.

Üldiselt oleme selle sotsialiseerumisrakenduse arendamisega suutnud teha olulisi edusamme oma püüdlustes lahendada rahvusvaheliste üliõpilaste sotsialiseerumisprobleeme. Rakenduse arenedes muutub see veelgi kasulikumaks ja kasutajasõbralikumaks.

REFERENCES

Agarwal, S., Kulkarni, P., & Jain, R. (2017). Go mobile: building apps with Golang. Packt Publishing Ltd.

Akinjise, K. (2021). Introduction to Go Fiber: A Fast HTTP Framework for Go. logrocket. Available at https://blog.logrocket.com/introduction-to-go-fiber-a-fast-http-framework-for-go/, accessed April 16, 2023.

Asaduzzaman, M., & Alhadidi, D. (2020). A survey of React Native for mobile app development. Journal of King Saud University-Computer and Information Sciences, 32(1), 68-76. Available at https://www.sciencedirect.com/science/article/pii/S1319157820303263, accessed April 16, 2023.

Atlas, J., & John, J. (2021). React Native in Action. Manning Publications.

Beighley, L. (2016). Head First SQL: Your Brain on SQL--A Learner's Guide. O'Reilly Media Inc.

Bojars, U. (2019). The impact of mobile technology on student achievement. International Journal of Education and Development using Information and Communication Technology, 15(1), 113-127. Available at https://files.eric.ed.gov/fulltext/EJ1211297.pdf, accessed April 22, 2023.

Brown, D., & Mitchell, A. (2019). React Native: Building Mobile Apps with JavaScript. Addison Wesley Professional.

Cambridge University Press 2018a. front end. available at https://dictionary.cambridge.org/dictionary/english/front-end, accessed April 18, 2023.

Cambridge University Press 2018b. back end. Available at https://dictionary.cambridge.org/dictionary/english/back-end, accessed April 18, 2023.

Cambridge University Press 2018c. API. Available at https://dictionary.cambridge.org/dictionary/english/api, accessed April 18, 2023.

Cambridge University Press 2018d. Android. Available at https://dictionary.cambridge.org/dictionary/english/android, accessedApril 18, 2023.

Ekanayake, P. (2019). Building a RESTful Web Service with Golang. medium. Available at https://medium.com/@prasanna.ekanayake/building-a-restful-web-service-with-golang-5e7c2b1387e1, accessed April 17, 2023.

Elshenawy, M., & Rabie, A. (2018). React Native for Cross-Platform Mobile App Development. In 2018 4th International Conference on Computing Sciences (ICCS) (pp. 44-48). IEEE. Available at https://ieeexplore.ieee.org/abstract/document/8676337, accessed April 22, 2023.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). Database Systems: The Complete Book. Pearson Education Inc.

Go fiber. (n.d.). fiber documentation. Available at https://docs.gofiber.io/, accessed April 18, 2023.

Hacker Noon. (2020). React Native vs Flutter: Which one to choose for your next mobile app? Available at https://hackernoon.com/react-native-vs-flutter-which-one-to-choose-for-your-next-mobile-app-wa20323j, accessed April 18, 2023.

JSON 2018. Introducing JSON. Available at https://json.org/, accessed April 18, 2023.

Kim, C., & Lee, M. (2019). Building mobile applications with React Native. Wiley.

Lee, D. (2013). Enhancing international students' socialization via mobile applications. Journal of Educational Technology Development and Exchange, 6(1), 1-14. Available at https://www.editlib.org/p/152317/, accessed April 16, 2023.

PostgreSQL. (2023). PostgreSQL Documentation. Available at https://www.postgresql.org/docs/, accessed April 22, 2023.

React Native Navigation. (2023). React Native Navigation Documentation. Available at https://wix.github.io/react-native-navigation/docs/before-you-start/, accessed April 22, 2023.

React Native. (2023). React Native Documentation. Available at https://reactnative.dev/docs/getting-started, accessed April 22, 2023.

Tailwind CSS. (2023). Tailwind CSS Documentation. Available at https://tailwindcss.com/docs, accessed April 18, 2023.

Wikipedia 2018. REST. Available at https://en.wikipedia.org/wiki/Representa- tional_state_transfer, accessed April 16, 2023.

Wikipedia 2017. Access token. Available at https://en.wikipedia.org/wiki/Access_to- ken, accessed May 5, 2018

Zheng, B., & Veeraraghavan, R. (2016). A mobile application to improve socialization for international students. Journal of Educational Technology Development and Exchange, 9(1), 1-14. Available at https://www.editlib.org/p/185628/, accessed April 16, 2023.

APPENDICES

Appendix 1. Source code downloads

- Back-end – https://github.com/pArtour/networking-server
- Front-end - https://github.com/pArtour/networking-app-mobile

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Artur Pushkov,
(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose
DEVELOPING A SOCIALIZATION APPLICATION FOR INTERNATIONAL STUDENTS OF NARVA COLLEGE,
(lõputöö pealkiri)

mille juhendaja on Andre Sääsk,
(juhendaja nimi)

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

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi 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.

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

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

Artur Pushkov
14.05.2023