МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет економіки і підприємництва
Кафедра інформаційних технологій

Допущено до захисту
«_____» _____ 2026 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

Кваліфікаційна Робота Бакалавра

РОЗРОБКА ДОДАТКУ ДЛЯ ПЛАНУВАННЯ РОБОЧИХ ЗМІН ПРАЦІВНИКІВ

Виконала: студентка 4 курсу, 41-кн групи
Освітньо-професійної програми
«Комп'ютерні науки»
спеціальності 122 «Комп'ютерні науки»
Савранська Анна Олегівна

Науковий керівник:
зав. кафедри, к.т.н., доцент
Ліщук Роман Ігорович

Умань – 2026

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
UMAN NATIONAL UNIVERSITY

Faculty of economics and entrepreneurship
Department of information technology

Approved for defense
«_____» _____ 2026 y.
Head of Department, c.t.s., docent
_____ **Lishchuk R.I.**

Bachelor's Qualification Paper

DEVELOPMENT OF AN APPLICATION FOR EMPLOYEE SHIFT SCHEDULING

Done by: student of 4th year, 41-кн group
Of educational-professional program
«Computer science»
specialty 122 «Computer Science»
Savranska Anna Olehivna

Academic Supervisor:
Head of Department, c.t.s., docent
Lishchuk Roman Igorovych

Uman – 2026

TABLE OF CONTENTS

INTRODUCTION

CHAPTER 1 ANALYSIS OF EXISTING APPLICATIONS FOR EMPLOYEE SCHEDULING AND DEFINITION OF PROJECT EXPECTATIONS

1.1 Overview of expectations of a scheduling system

1.2 Comparison analysis of existing solutions

1.3 Specification of the main requirements

CHAPTER 2 SYSTEM DESIGN AND MODELING OF EXPECTED BEHAVIOUR

2.1 Specification of inputs and outputs of the system

2.2 System design and architecture

2.3 User interface and interaction planning

2.4 Analysis and justification of used technologies

CHAPTER 3 DEVELOPMENT OF THE APPLICATION

3.1 Design and development of the server of the application

3.2 Design and development of the client side of the application

3.3 System testing

CONCLUSIONS

LIST OF USED SOURCES

APPENDICES

INTRODUCTION

Relevance of the research. Starting a business is not easy – managing it is even harder. Numerous businesses use employee scheduling applications for their workforce management. From a business perspective, there is interest in finding a solution that will be easy to use, will provide enough functionality, and will meet the budget of the company. From the developer's perspective, there is an opportunity to build an application that makes the lives of thousands of people run more smoothly and allows the customer processes to be controlled and organized. Although the pool of solutions is growing, there are a few popular choices in the market of scheduling applications. Most of them, however, either stop evolving and fixing problems that exist in their applications or set price policies that are not suitable for small or medium-sized customers.

The goal of this research is to develop an application that will provide customers with an easy-to-use employee shift management system and will allow them to track, analyze, and schedule shifts corresponding to the needs of their business.

The object of study is employee management at small and medium-sized businesses with a focus on work scheduling.

The subject of study is the process of relaying information about scheduled shifts and using the most efficient technologies for creating schedules for the company's employees.

The research tasks include:

- Specification of the expectations for the existing applications and the one that is being developed;
- Comparison analysis with the existing systems in the market;
- System expectations and system architecture design using the MVVM software design pattern;
- Plan of the user interface and user interactions;
- Design and development of the server-side of the application that includes MongoDB database setup;

- Development of the React Native client-side of the application that provides authorization, shift viewing, and scheduling;
- Manual and unit testing for the application.

The methods that were used during the research include analytical comparison of existing technologies for the web applications, system design and analysis that follow the modern standards of software development, and building the application using modern programming languages and frameworks for the best outcome of the product.

Thus, this research was done in order to study and develop a software solution that exceeds the expectations of possible customers and provides a reliable application for their business needs while adhering to existing software standards and practices.

CHAPTER 1 ANALYSIS OF EXISTING APPLICATIONS FOR EMPLOYEE SCHEDULING AND DEFINITION OF PROJECT EXPECTATIONS

1.1 Overview of expectations of a scheduling system

In modern society, there is a substantial growth of businesses that employ part-time workers and have a need for flexible scheduling for their workforce. With the development of technologies, it became possible for this managerial task to be automated and completed from anywhere and at any time.

This demand created an opportunity for many enterprise software development companies to start deploying their own solutions. One of the most popular choices is a cross-platform application that provides both shift scheduling and viewing for the employee and the manager, or separates those functions based on the platform. If there is a separation of those functions, the usual model is to allow manager tasks (shift scheduling, viewing, etc.) to be done in a web application, while allowing employee tasks (shift viewing, setting of availability, etc.) to be done on any of the platforms.

A development of a scheduling system would also entail an understanding of the limitations and expectations that customers would have for the system. For example, does a business expect to have a constant flow of employees or a stable set of them? Is there a need for collecting data about worked hours and a necessity for calculating expected compensation? What is the expected process and allowed actions for the managers and employees with regard to shift management? Besides that, it is important to ensure that the services that an application provides are always available and are secure against any infiltrations, as there could be sensitive data at risk, such as personal earnings, the location and time of work, and personal information of employees.

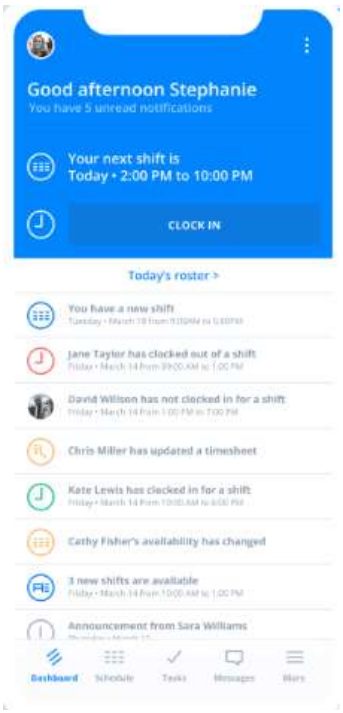
All these examples provide deeper context to the research that must be done and the communication that has to happen between a customer and the development team prior to the beginning of the development cycle. Meeting expectations of the customer and building a secure application greatly improves the productivity of managerial staff

as well as provides employees with fast access to data that allows them to plan around their work hours and get an ability to have a healthier work-life balance.

1.2 Comparison analysis of existing solutions

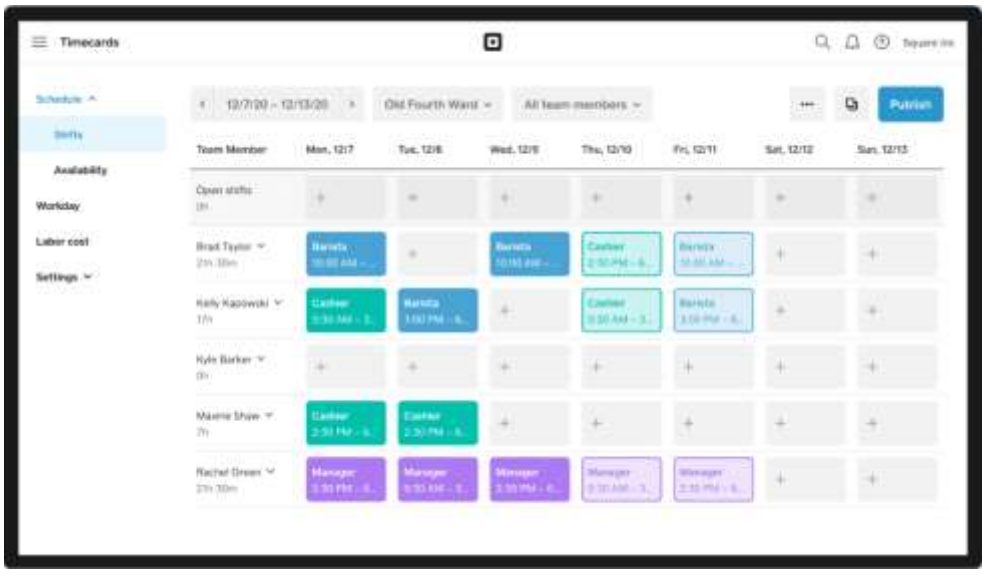
A need for shift scheduling solutions did not emerge recently. As the technology evolved, the company managers were able to move on from pen and paper and start using more advanced systems such as magnetic tapes and punch cards. Although this allowed tracking employee arrivals, there was still a hardship in scheduling. Later, the Microsoft Spreadsheets allowed the managers to have an easier way to save their planning data, but it still required skill and time in order to form the schedules. Nowadays, there are several noteworthy software applications that provide solutions for shift scheduling with the use of Cloud-based software [1]. The ones that are the most popular in the market are Sling, Square, and Homebase. Each of them has its own different implementation of the same concept of shift scheduling.

For example, *Sling* (img. 1.1), developed by a company called Toast, has a wide range of functionality that, apart from regular scheduling, also covers reports of productivity, time-off management, in-app team communication, and labor optimization by analyzing previous data and making a prediction for a future workload or expenses. Additionally, this is a cross-platform application that is available on the Web, Android, and iOS. This solution is heavily employed in small to medium-sized businesses across various industries that have a need for flexible scheduling [2].



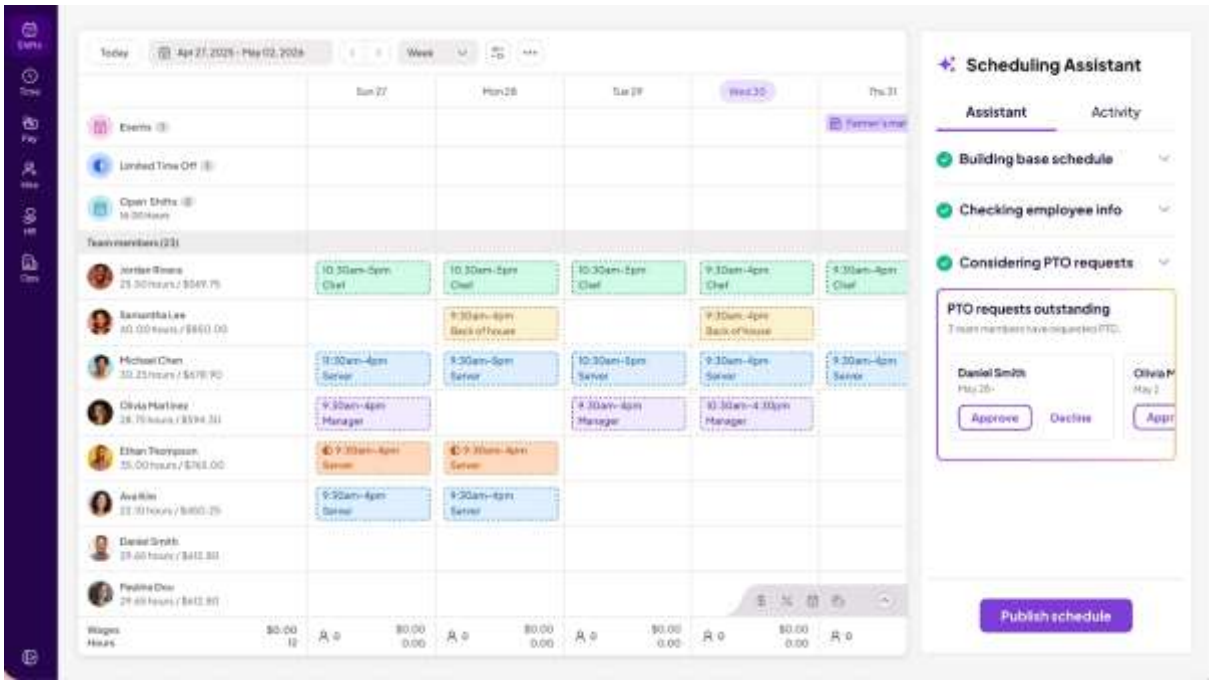
Img. 1.1 Example of Sling application dashboard page

Another instance of a shift scheduling application is *Square* (img. 1.2), which is owned by Block, Inc. *Square*'s advantage is that one of its main products is also a Point of Sale (PoS) system that allows businesses to make credit card transactions from their customers. Combined with their PoS systems and hardware, they also provide shift scheduling, payroll management, and marketing opportunities, which makes it easier for the business to pay for one package and have access to everything they could need [3].



Img. 1.2 Example of Square Schedule page usage

A third example of a research application is *Homebase* (img. 1.3), made by the company of the same name. They provide a similar set of base features as Sling and Square, such as payroll and shift management; however, they also have additional services in Human Resources and hiring new employees through an embedded application that pulls data from job websites, and they have an AI Assistant that allows a manager to automate their tasks [4].



Img 1.3 Example of Homebase user interface

As seen with these examples, in order to stay competitive with these products, there needs to be something more than just a scheduling application. Whether it is with PoS systems, Human Resources features, or AI assistants, these systems try to grab their part of the market and stay relevant by any means. At the same time, the difference in provided features allows the customer to find a system that is suitable for their needs. Such a variety and abundance positively affect the managers in the customer companies and allow them to focus on problem-solving instead of dealing with mundane tasks.

While the product that will be developed as a part of this study will not add all the features that the other products have, it will focus on the shift scheduling system and will intend to improve it and make a version of it that will be better than analogs in the market.

1.3 Specification of the main requirements

The demand for this type of product shows the relevance of its development and the importance of providing features of shift scheduling. This type of application will especially be useful to smaller businesses that do not want to pay for too many features that they might not use, and just want to focus on building and accessing workload management.

Create a Web application that will automate scheduling and allow managers and employees to access their schedules.

These are the requirements for the application:

1. Authentication
    a) A user can create a new account.
    b) A user can log in with an existing account.
2. Showing the shifts
    a) A user has 3 different ways of viewing their shifts: shifts scheduled just for them, a weekly schedule for all the employees, and a schedule for each employee.
3. Scheduling shifts
    a) A manager can create shifts for an employee with a selected role and for a selected date.
    b) A manager can create new roles with an assigned salary.

Technical requirements for this application include:

- Web application that could also be transformed into a cross-platform application.
- Easy to use and learn interface that responds fast and smoothly.
- Security on each level: safe data transfer and retrieval.

- Not relational database for saving data about the shifts and employees.

- Scalability and code maintainability for possible future improvements and enhancements.
- A modern set of languages and frameworks used for programming a Web, native, or reactive application.

Assumptions and limitations:

- It is assumed that more functionality is expected to be added later.
- This application is expected to be used by a small business (less than 100 employees).

The expected results of the product:

- An improved workflow for the managers of the business.
- Intuitive user interface that leads the user to anywhere they need to go.
- Clear showcasing of shifts and the schedule.

CHAPTER 2 SYSTEM DESIGN AND MODELING OF EXPECTED BEHAVIOUR

2.1 Specification of inputs and outputs of the system

In order to make a proper design, there is a need to analyze the inputs and outputs that are a part of the expected behavior of the system. With this analysis comes an understanding of the kind of database needed to be used for data storage, what kind of modules are needed for relaying information to the user, and how to build the architecture of the whole system.

Inputs are all the data that the user provides for the system. In the case of this application, it includes personal information about the user and data about the schedule. All the data is expected to be manually entered and stored on the server side of the system.

One of the requirements is the authorization system that includes creating an account, signing in with credentials, and signing out. The user should be able to share their name, email, and create a password for making an account. Afterwards, the user can input their email and password in order to sign in to their account.

A user with manager permissions should be able to create new shifts and roles for those shifts. When creating a shift, the manager provides data about the employee for the shift, their assigned role, the date of the shift, start time, and end time of the shift, and can add notes about the shift. When creating a new role, the manager should be able to provide the name of the new role and the expected hourly salary for that position.

Additionally, there are inputs that are not directly entered by the user but are created in the system. An example of this occurrence would be creating a token for the user every time either an account gets created or a person signs into the system.

The output information of the system is the data that gets generated by the system and is relayed to the user. The best example of outputs in this application is displaying shift data based on the application page. The user should be able to get the output of their personal shifts, shifts of the entire team for a given week, and shifts for a chosen employee.

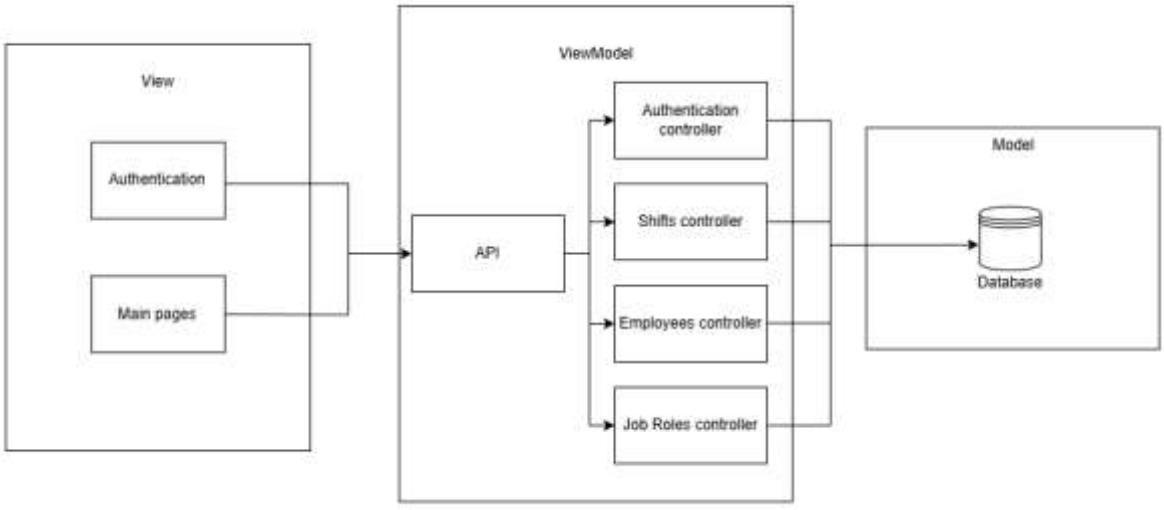
When creating shifts, the manager user can also see the list of existing employees and existing roles, which is computed on the backend and shown to them as an output of the system.

All the input and output data should be correctly formatted and unified in order to provide the most relevant information and to not cause confusion. All the data should be securely stored, and it should be ensured that there would not be any data corruption during either loading or accessing of the datastores.

Following these expectations for the inputs and outputs would make the application work properly and securely and would ensure that the system will be an easy-to-use solution that the customers can trust.

2.2 System design and architecture

System design is the backbone of the application development cycle, as it includes planning out the architecture and structure of the whole product. This application will follow the Model-View-ViewModel architecture that separates the application into 3 layers. The Model layer is the backend layer that stores and manipulates data. The View layer is the frontend that the user sees and interacts with. It also sends inputs provided by the user into the ViewModel. The ViewModel layer is a middle ground between View and Model. It converts the data it receives from View into the format that is expected by the Model [5]. The structure of modules and services that follow this architecture is seen on img. 2.1.



Img. 2.1 MVVM architecture diagram for the application

Using the suggested architecture will allow a standardized workflow with separation of concerns between the View and the Model layers. This will allow for the system to be modular, scalable, and maintainable, which enables further development of the system and makes integration of new features less complicated.

Explanation of each component of this system:

1. The Authentication View Component is a frontend component that provides features for creating an account and signing in. It collects user inputs and makes an HTTP call to the API to pass the collected data.

2. The Main pages are a collection of frontend components that will show data to the user depending on the current page. The Dashboard page will show shifts for a given week for the logged-in user. The Weekly Schedule page shows the schedule for all the employees during each day of the week. It is also the page where the manager can create shifts and new roles. The Employee page will show a list of existing employees, and upon choosing the employee, it will show their shifts. All the data retrieval is done by sending HTTP requests to the API component and showing the response.

3. The API component is a router that, after getting a request from the View, routes that request to the corresponding controller.

4. The Authentication controller holds methods for signing in and signing up, and interacts with the Model. This component is responsible for creating Users and Employees and getting the data about Users when needed.

5. The Shift controller includes methods for reading and creating Shifts with passed-in data from the request.

6. The Employee controller includes methods for reading and creating Employees on the Model side with the given information from the requests.

7. The Role controller is responsible for creating and reading Job Roles from the Model and provides methods for that.

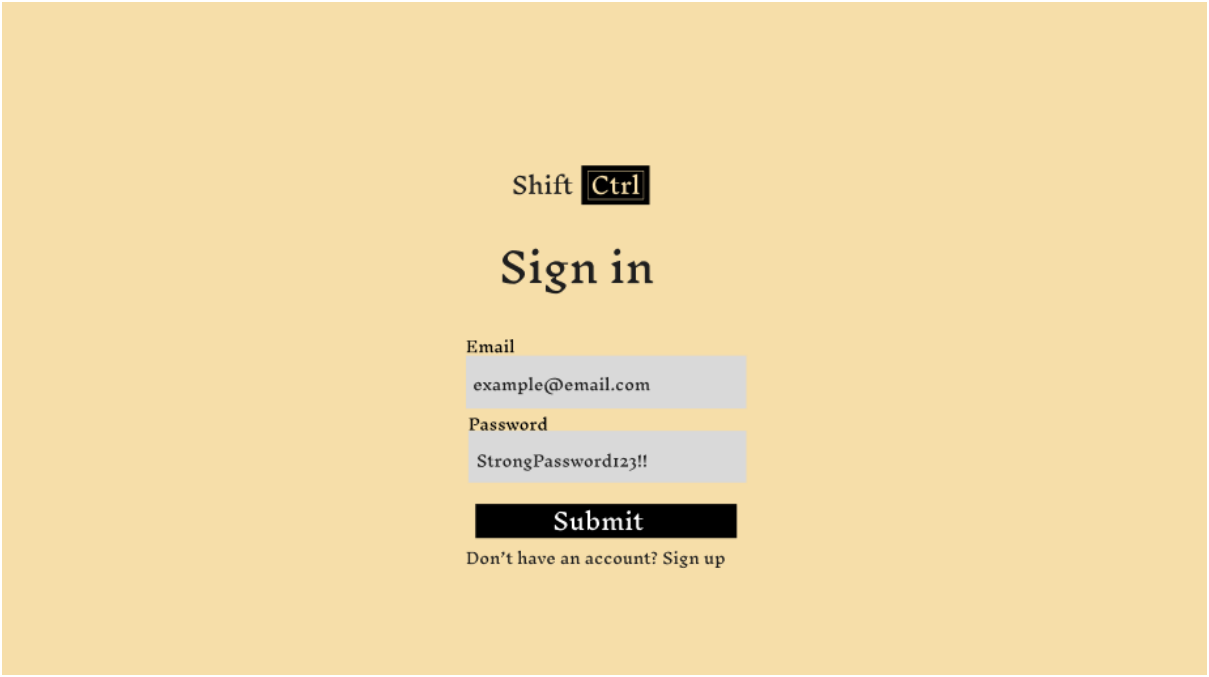
8. Database takes the role of the Model in the system. It stores all the data that includes Users, Employees, Shifts, and Job Roles.

Together, these components form a system that is secure, prone to alterations, and easily modifiable in case there are changes in requirements. This design also shows that following the MVVM pattern will be beneficial, as it will integrate well into the chosen technology stack for the development of the product.

2.3 User interface and interaction planning

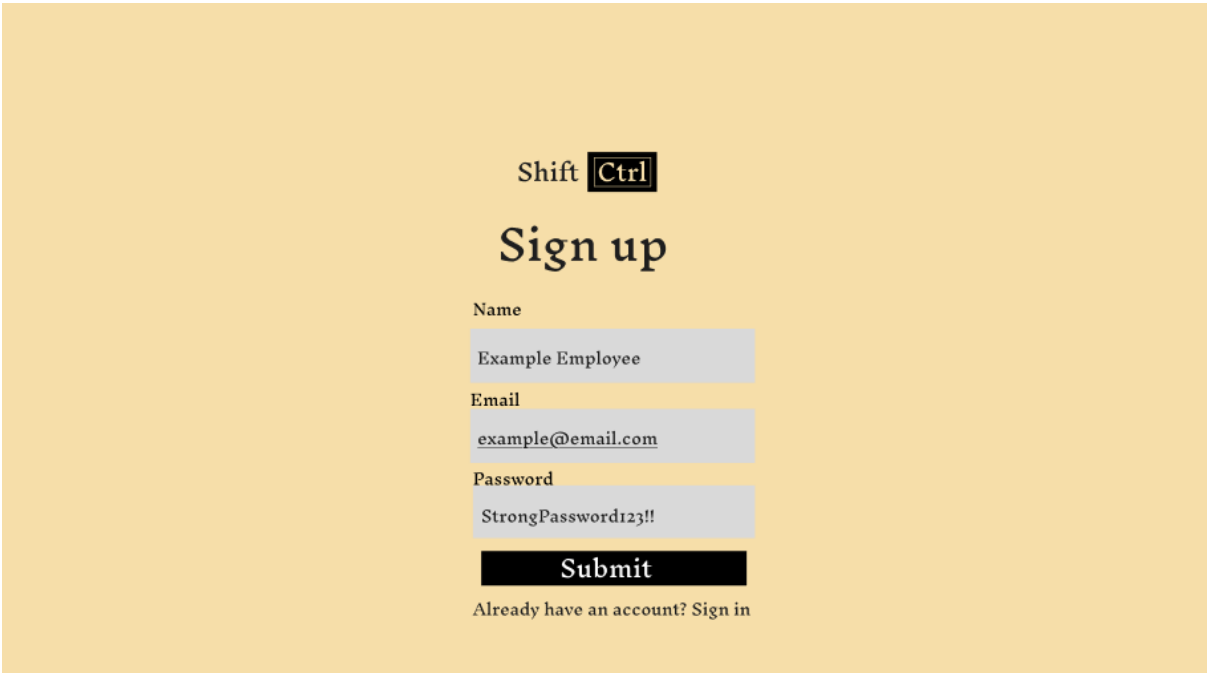
Planning the user interface and designing expected interactions of the user with the system is a crucial phase in the software creation cycle. It makes the development faster, as during this step, a template of what is expected to be built is designed and is later implemented during the execution phase of the project.

The graphical design was made using Figma, which is a popular tool for collaboration and designing prototypes and templates for different products [6]. The next images show the expected look of the user interface.



Img. 2.2 A design for Sign in page

As seen in image 2.2, a Sign in page is expected to have input fields for email and password, and a link that can take the user to the Sign up page if needed. There is also the Submit button that should send the collected data to the API and lead the user to Dashboard page.



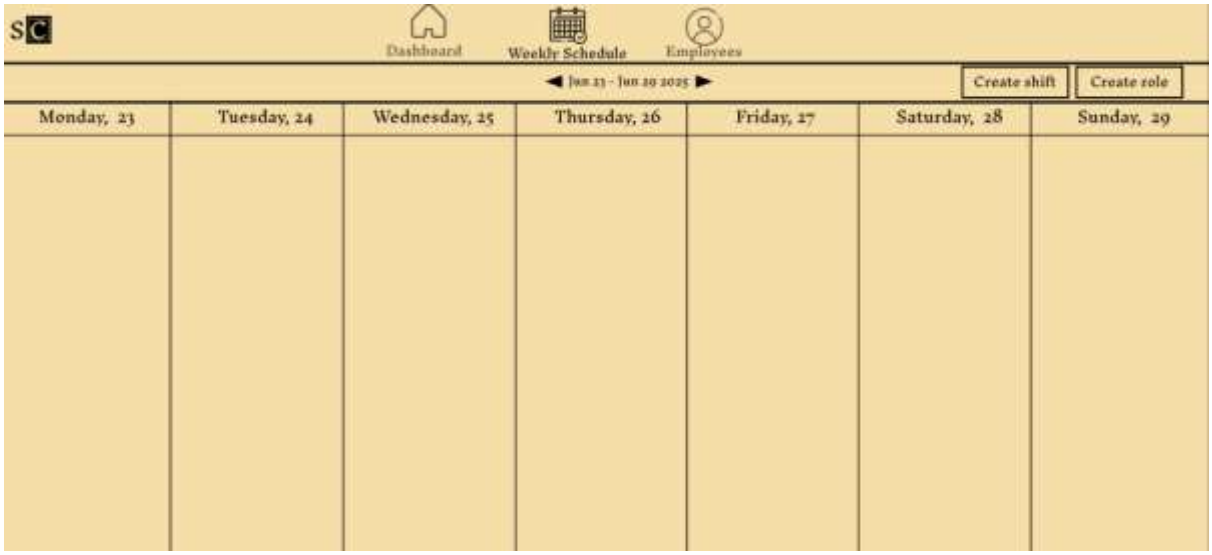
Img. 2.3 A design for Sign up page

The Sign up page (img. 2.3) should provide input fields for name, email, and password of the employee, and a link to the Sign in page. Clicking on the Submit button will validate the entered data and, if it is correct, lead the user to the Dashboard page.

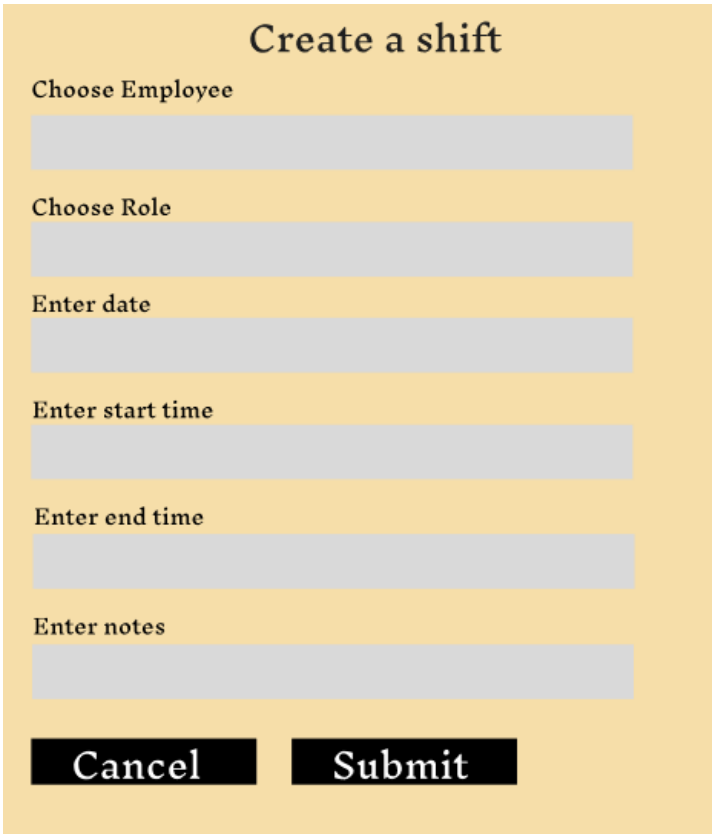


Img. 2.4 A design for Dashboard page

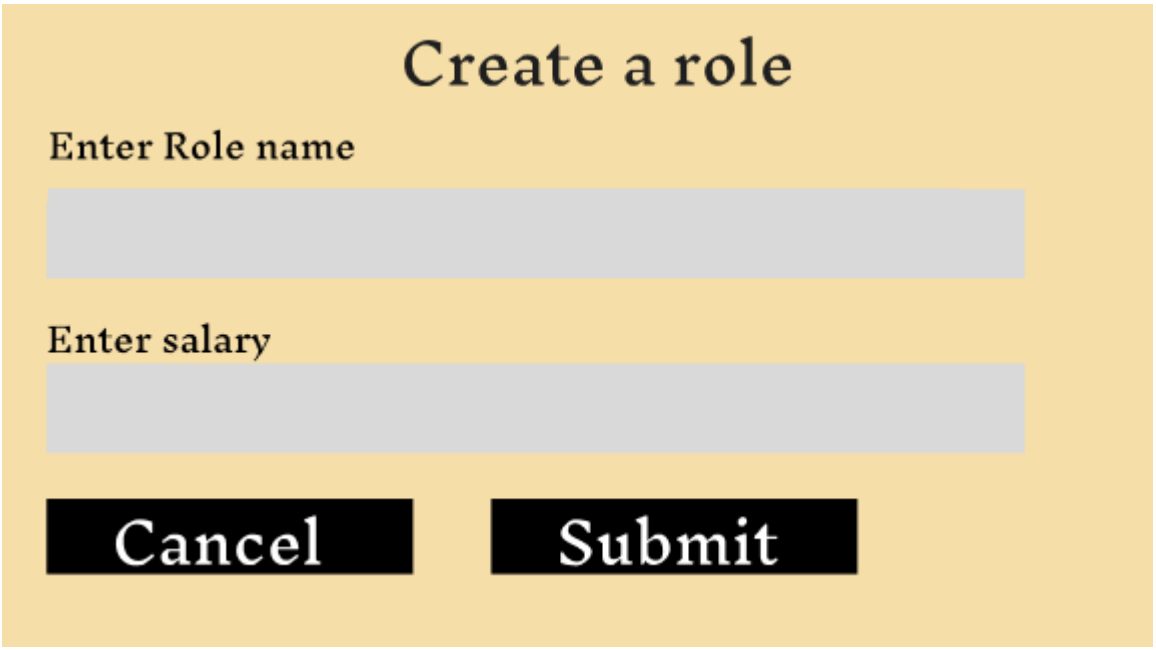
The Dashboard page (img. 2.4) shows a display of all the shifts for the currently logged-in employee and has a navigation bar that allows users to navigate to the other parts of the program, and the week picker, which helps with changing the displayed week.



Img. 2.5 A design for Weekly Schedule page



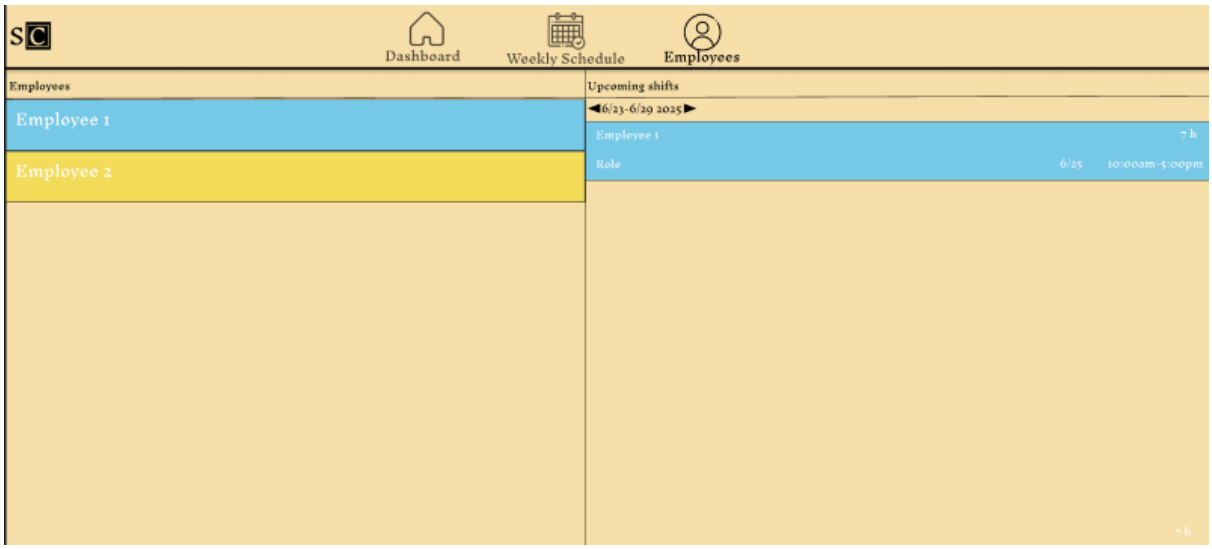
Img. 2.6 A design for creating a shift dialog



Img. 2.7 A design for creating a role dialog

The Weekly Schedule page (img. 2.5) should display all the shifts for a given week. There is a Navigation Bar for navigating to other pages, a week picker, and buttons for creating a shift and creating a role. Images 2.6 and 2.7 also show the creation of the shift and a role, respectively. Each of them provides input fields for all

the information needed for the creation of a shift and a role. They also have a Cancel button that closes the dialogue window and should clear all the entered data. At the same time, a submit button should send a request to the API with all the collected data and close a dialogue window as well.



Img. 2.8 A design of Employees page

The Employees page (img. 2.8) is expected to show the employee list and allow for the employee to be clicked on. After the click, the shifts for that employee should get displayed on the right side of the screen, together with a week picker. This design allows a manager and an employee to track the availability and workload of any given employee.

When implemented, this user interface will provide the user with an easy-to-follow workflow through the application. Additionally, it will satisfy their needs for viewing and creating a schedule. Having this design also improves the development process, as during the implementation phase, there will be a template to base the work on.

2.4 Analysis and justification of used technologies

To choose the best technology stack, it is worth investing time in an analysis of existing frameworks and code editors, and comparing them. Since this application will be on a Web platform but will need to be transferable into a mobile version in the future, it is a good choice to decide between native frameworks that allow building for Web, iOS, and Android.

One of the first choices for this case is using JavaScript with React Native. JavaScript is a programming language for Web development that is one of the most used languages for this type of development [7]. At the same time, React Native is a mobile development framework that allows using the reactive paradigm in the development of the application. Using React Native Expo framework allows implementing existing and creating your own components that get translated into the native code of the needed platform without the need to write different code [8].

For a backend solution in that case, it would be advised to use Express.js, which is a framework for Node.js created for Web applications and API development. It is simple to set up, and in combination with a chosen database, it creates a good backend server for storing and accessing the data.

Another possible solution for this case is using Flutter, which is an open-source framework for building multi-platform applications. Flutter uses Dart as its programming language, which is an object-oriented language that needs to be compiled [9].

Flutter shows better benchmarks in performance and security since it is compiled. At the same time, React Native provides a real native feel to the applications, has a smaller application size, and allows for the over-the-air code pushes, which help speed up updates to the product once it gets released to the mobile application stores. Below is the comparison table that shows the results of the analysis.

Table 2.1 Comparison of Flutter and React Native

Framework Name	Flutter	React Native
Multiplatform support	**1**	**1**
Security	**1**	**1**
Stability	**1**	**1**
Performance	**1**	**-**
Over-the-air code pushes	**-**	**1**
Smaller application size	**-**	**1**
Total score	**4**	**5**

Based on the results from table 2.1, it was decided to use React Native for the development of this application.

Another choice to make was which database to use. There are relational and non-relational databases that have different structures of their schema. Relational databases use tables and contain strict relationships between data. At the same time, non-relational databases are more flexible and use key-pair storage or document-based storage. Based on the data that we expect, which is shifts, notes for the shifts, and employee data, it was assumed that a non-relational database would be a better fit.

There are several choices among popular NoSQL databases. One of them is Cassandra, which is a wide-column datastore. It uses tables to store data, but each row of data can have differently named and formatted columns [10].

Another solution is using MongoDB, which is a document-based database. It consists of collections that have documents inside them. Each document is like a row in a traditional relational database, but it does not have to adhere to a strict schema.

Table 2.2 Comparison of Cassandra and MongoDB

Database name	Cassandra	MongoDB
Wide functionality of indexing	**-**	**1**
High scalability	**-**	**1**
Cloud support	**1**	**1**
Documented	**1**	**1**
Security	**1**	**1**
Total score	**3**	**5**

Based on research and comparison results, it was decided to use MongoDB and its mongoose package for the development of the application.

CHAPTER 3 DEVELOPMENT OF THE APPLICATION

3.1 Design and development of the server of the application

The server side of this application includes API methods for getting data from the frontend and a setup that directly interacts with the database. Since the chosen database is MongoDB, mongoose package was used in order to develop the database setup. There were 4 collection schemas developed: Employee, Job Role, Shift, and User. Their structure can be seen in images 3.1-3.4.

```
import mongoose from "mongoose";
const { Schema } = mongoose;

const employeeSchema = new Schema(
    {
        name: {
            type: String,
            trim: true,
            required: true,
        },
        assignedRoles: [{
            type: Schema.Types.ObjectId,
            ref: 'JobRole',
        }],
        user: {
            type: Schema.Types.ObjectId,
            ref: 'User',
        }
    },
);
export default mongoose.model("Employee", employeeSchema);
```

Img. 3.1 Employee schema

```
import mongoose from "mongoose";
const { Schema } = mongoose;

const jobRoleSchema = new Schema(
    {
        roleName: String,
        salary: String,
    },
);
export default mongoose.model("JobRole", jobRoleSchema);
```

Img. 3.2 Job Role schema

```
import mongoose from "mongoose";
const { Schema } = mongoose;

const shiftSchema = new Schema(
    {
        startTime: Date,
        endTime: Date,
        employee: {
            type: Schema.Types.ObjectId,
            ref: 'Employee',
        },
        jobRole: {
            type: Schema.Types.ObjectId,
            ref: 'JobRole',
        },
        notes: String,
    },
);
export default mongoose.model("Shift", shiftSchema);
```

Img. 3.3 Shift schema

```
import mongoose from "mongoose";
const { Schema } = mongoose;

const userSchema = new Schema(
    {
        email: {
            type: String,
            trim: true,
            required: true,
            unique: true,
        },
        password: {
            type: String,
            required: true,
            min: 6,
            max: 64,
        },
        role: {
            type: String,
            default: "Subscriber",
        },
    },
    { timestamps: true }
);
export default mongoose.model("User", userSchema);
```

Img. 3.4 User schema

Another part of the server is an Express.js router that is responsible for routing requests from an API call to the corresponding method of a controller.

The main file of the server side is an index file that creates a connection with the database, instantiates the Express.js application, the router, and opens a port listening for the application.

To handle all the requests from the frontend, controllers were created. Authentication controller handles all operations needed for signing in and signing up for an account for the user. The signup function takes in email and password for the newly created account, does the validation of input by checking that both fields are not null, checking the size of the password, and making a query to the database to see if the user with this email already exists. After that, the password gets hashed using the bcrypt package. If the user is successfully created with the given email and password, a JWT token is created and returned together with the newly created user document instance. The signin method takes in email and password as well, tries to find the user using the email, in case it is successful, it compares the provided and hashed passwords using bcypt's compare function, and if the passwords match, it creates and returns a token with the user instance.

The employees controller is responsible for all of the interactions with the employee collection. CreateEmployee method takes in a name and a user ObjectId and tries to create a new Employee document with the given fields. If that operation succeeds, the new employee document is returned. GetAllEmployees function finds all the employees in the database, populates their assigned role field, and returns all the employees.

The roles controller helps with getting and creating Job Roles. GetAllJobRoles finds all the existing job roles in the database and returns them if the operation is successful. CreateJobRole function takes in newRoleName and newRoleSalary as parameters, tries to create a new document with those values, and returns the new instance.

The shift controller has responsibilities for all requests that concern shifts. GetUpcomingShiftsForEmployeeWithUserId method takes in a userId as a parameter.

It then finds the employee document with this user id. It also gets today's date and uses it together with the employee id to find shifts that are happening today or later than today for this employee. I then returns the array of shifts. GetUpcomingShiftsForEmployee expects to get the employee id as an argument. It also gets today's date and searches for the shifts with the employee id and later or on the day of today, and then returns the search results. GetAllShiftsForDate expects to take in a date and the next date for which the shifts are needed to be retrieved. It creates local variables for both dates and returns the shifts whose start date is after or on the needed date and before the next date. GetShiftsEmployeeUserIdForGivenWeek receives the userId, the first date of the week, and the last date of the week. It finds the user with the given userId, and then uses the id of the found employee, the week start date, and the week end date to find all the shifts for the employee in a given week. GetShiftsEmployeeForGivenWeek gets employeeId, the first day of the week, and the last day of the week. It searches for all of the shifts for this employee with shifts before the end of the week date and on or after the first date of the week, and returns the results. CreateShift method receives employeeId, roleId, chosenStartTime, chosenEndTime, and notes for the shift, and creates and returns the new shift document.

With these elements, the backend is able to support calls to the database and create and return data to the frontend. Using Express.js allows for this server to work fast and smoothly, and support uninterrupted performance of such packages as mongoose.

3.2 Design and development of the client side of the application

The frontend is the client side of the application. It is created by React Native components that form the pages of the Web application. The pages include: SignIn, SignUp, Dashboard, WeeklySchedule, and Employees.

The SignIn page follows the initial interface design and is depicted in image 3.5. It shows text input fields for collecting data of the user's email and password, a submit button, and a link that takes the user to the SignUp page if needed. To handle a click of the submit button, handleSubmit method was developed. It checks that the email and password are not null and makes a POST request to the api/signin route using the axios package. The email and password are passed in as parameters. Additionally, the data from a response is stored in AsyncStorage with the help of the react-native-async-storage/async-storage package.



Img. 3.5 SignIn page

The SignUp page (img. 3.6) allows the user to create a new account by entering their preferred name, email, and password. It also has a link for a SignIn page in case the user already has an account and a submit button. This button works similarly to the SignIn page's submit button, except for the request. For the SignUp page, the request is made to the api/signup. The response is also stored in AsyncStorage, and a new employee is created by passing the name and the new userId to the post request to the api/createEmployee route.

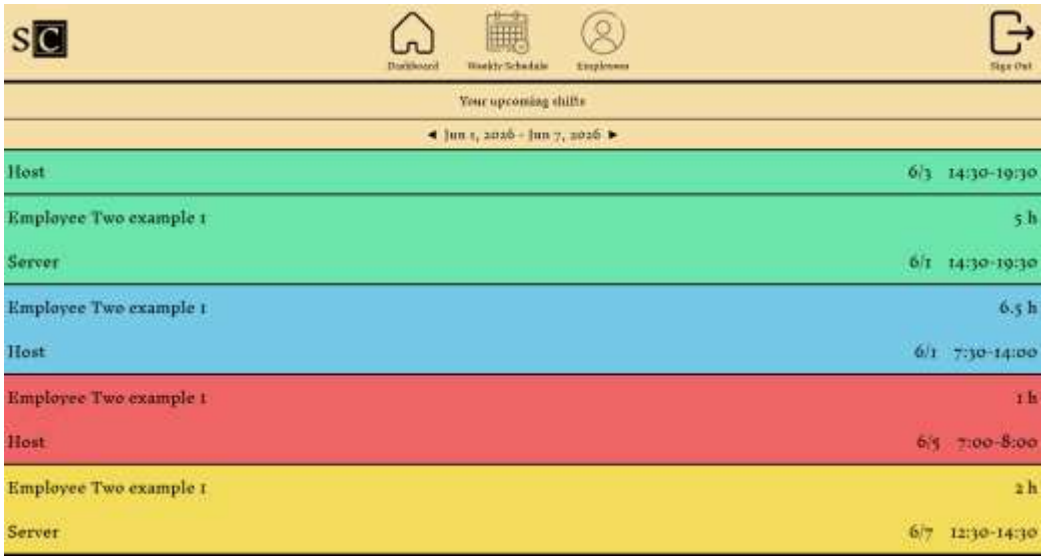


Img. 3.6 SignUp page

After signing in or signing up, the user is taken to the Dashboard page (img. 3.7) where the scheduled shifts for this employee are shown. There is a week picker that allows to manipulate which week's data is showing. For example, in image 3.7, the current week is shown, in image 3.8, a previous week with more shifts is depicted, and in image 3.9, a week ahead of the current one is chosen and has no shifts scheduled yet. The shift data is provided by fetchUpcomingShiftsForWeek. FetchUpcomingShiftsForWeek makes a request to api/shiftsEmployeeUserIdForGivenWeek with the current's user id, the first day of the week, and the last day of the week. The response contains the shifts for a given week, and they get displayed.



Img. 3.7 Dashboard page displaying current week shifts

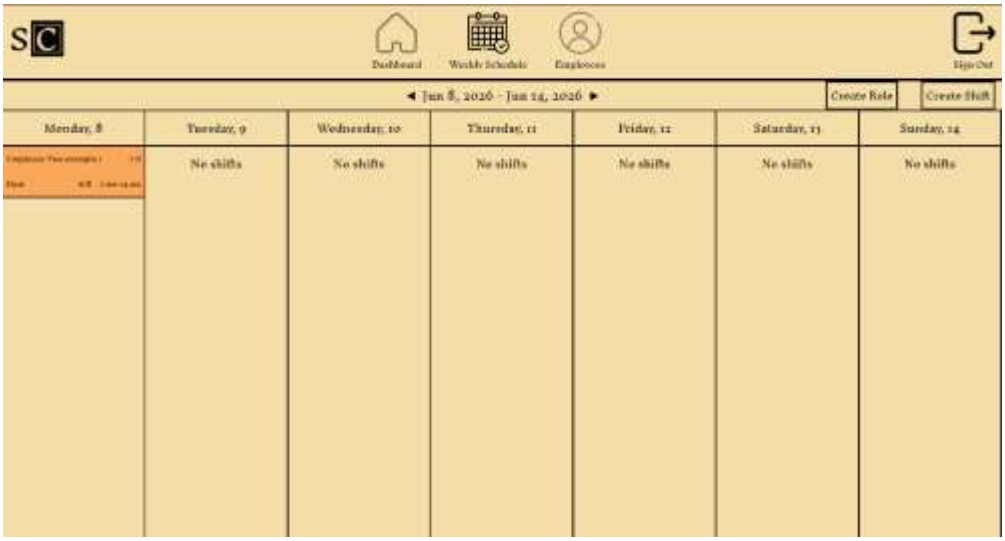




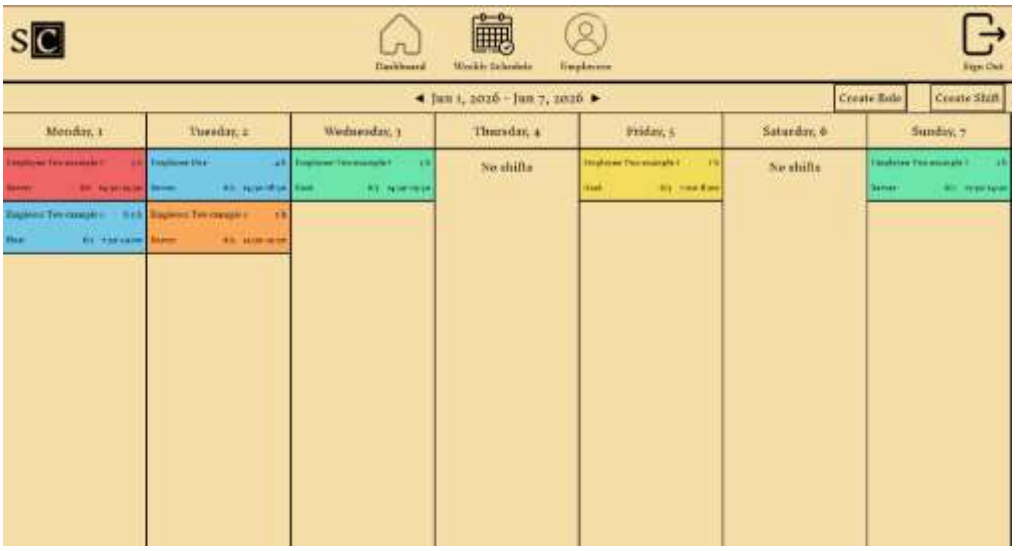
Img. 3.8 Dashboard page displaying previous week shifts

Img. 3.9 Dashboard page displaying next week shifts

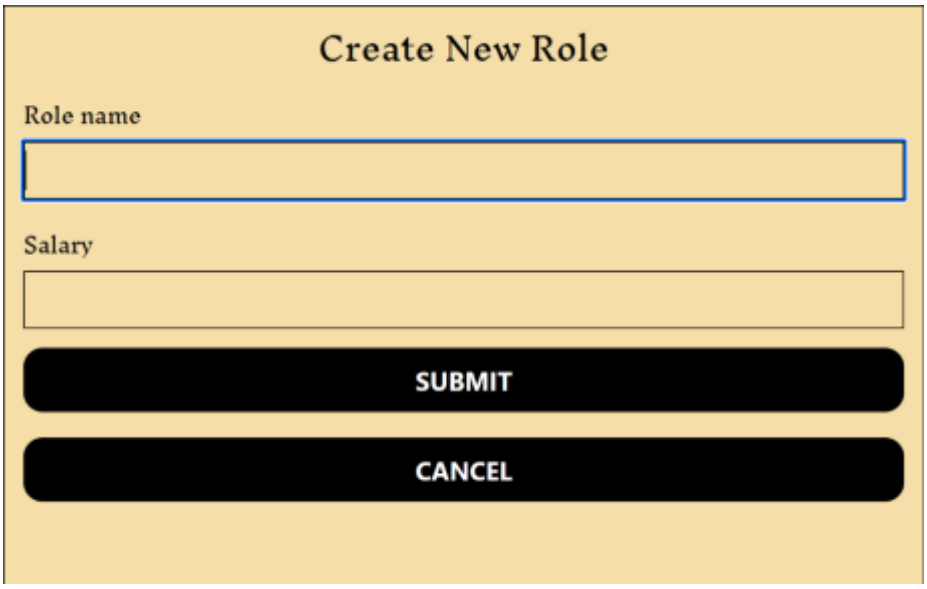
The Weekly Schedule (img. 3.10) displays a full schedule of all employees for a given week. It also has a week picker, which allows the user to choose the displayed week. Image 3.11 shows a previous week's schedule. This page also has buttons for creating a role and creating a shift. Creating a role button opens a dialogue (img. 3.12) that prompts a user to enter a new role name and a new role salary. When a submit button is clicked, submitAndCreateRole function is called, which makes a post request to api/createJobRole route and closes the dialogue. Create a shift button that opens a modal (img. 3.13), which has all the fields needed for creating a shift. Employee and Role fields are dropdowns that have all of the employees and roles listed using a Dropdown component from the react-native-element-dropdown package. It also supports searching for items. Clicking a submit button calls submitAndCreateShift function, which prepares data for being sent in a POST request to api/createShift route. Cancel buttons in both dialogues close the modals and return to the regular WeeklySchedule page.



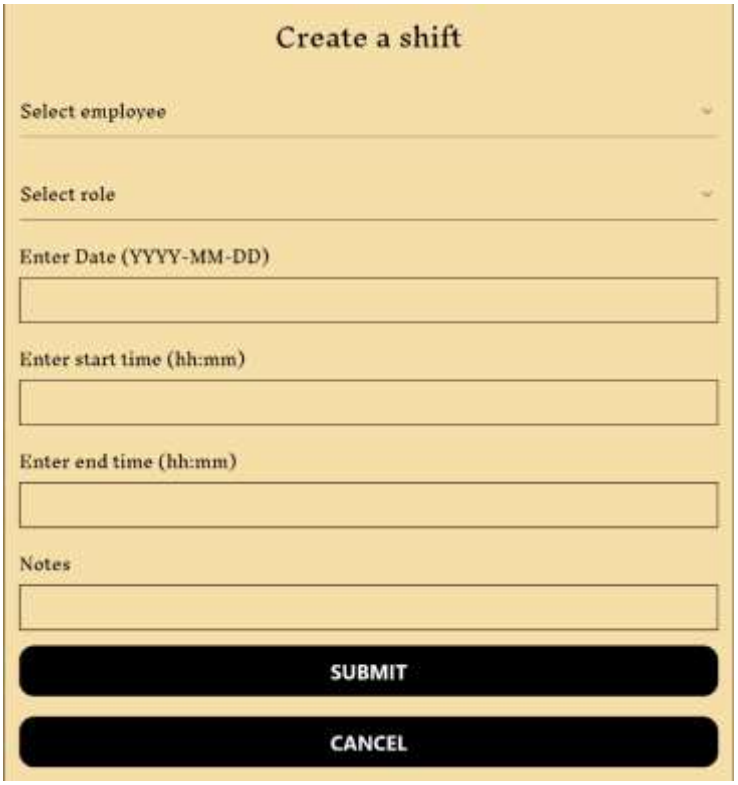
Img. 3.10 WeeklySchedule page for the current week



Img. 3.11 WeeklySchedule page for a previous week



Img. 3.12 Create a role dialogue



Img. 3.13 Create a shift

The Employees page (img. 3.14) shows a list of all employees. Upon clicking on an employee, a list of shifts for a given week for this employee is shown on the right side of the screen. As an example, image 3.15 shows shifts for Employee One, and image 3.16 shows shifts for Employee Two example 1. Shift fetching is happening in a function fetchShiftsEmployeeForWeek that makes a GET request to the api/shiftsEmployeeForWeek route.

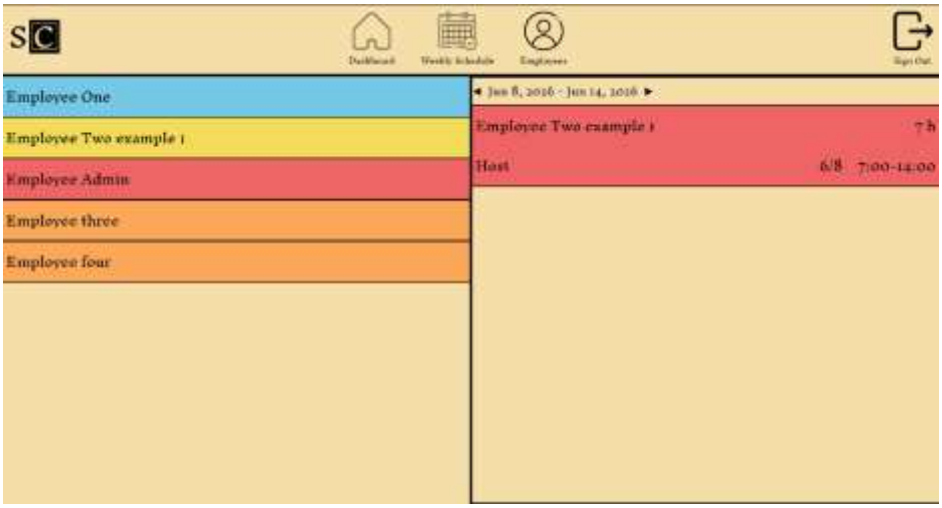


Img. 3.14 Employees page with no chosen employee



Img. 3.15 Employees page with Employee One chosen

Img. 3.16 Employees page with Employee Two chosen



On every page, there is also a Sign Out button that, when clicked, logs off the user, calls a function for clearing the AsyncStorage data, and routes the user to the SignIn page. All these components create an easy-to-follow workflow for the user and allow them to complete their tasks in this application.

3.3 System testing

Testing of the system is a crucial step in the development process, as it allows the development team to see the flaws and bugs in the existing product and fix them before the release. Testing of this system included unit tests for the database. The unit tests were developed with Jest, mongodb-memory-server, and supertest packages, which were used to test all the existing controllers to see that the database interactions are happening as expected. Here is a list of tests and their description:

Test 1. api/signup rejects missing email

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /signup route using the signup controller.
- The user collection is empty.

Steps:

- Send a POST request to /signup route with a body that includes only the password (no email).

Expected result:

- The response contains an error of “Email is required”.

Result:

- The response returns an error, as expected.

Test 2. api/signup rejects a short password

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /signup route using the signup controller.
- The user collection is empty.

Steps:

- Send a POST request to /signup route with a body that includes email and password, but the password is less than 8 characters long.

Expected result:

- The response contains an error of “Password is required and should be at least 8 characters long”.

Result:

- The response returns an error, as expected.

Test 3. api/signup creates a user and returns a token

Preconditions:

- In-memory MongoDB is running and is connected.

- Express.js app is configured with the /signup route using the signup controller.
- The user collection does not have a user with the test email.

Steps:

- Send a POST request to /signup route with a body that includes unique password and unique email

Expected result:

- The response status is 200, and the body contains a token and a user object.
- A user with that email exists in the database.

Result:

- The response returns the token and the user. The user exists in the database as expected.

Test 4. api/signup rejects an existing email

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /signup route using the signup controller.
- A user with the test email exists in the database.

Steps:

- Send a POST request to /signup route with a body that includes the existing email and a password.

Expected result:

- The response contains an error of “Email is taken”.

Result:

- The response returns an error, as expected.

Test 5. api/signin rejects an unknown user

Preconditions:

- In-memory MongoDB is running and is connected.

- Express.js app is configured with the /signin route using the signin controller.
- No user exist with the test email.

Steps:

- Send a POST request to /signin route with a body that includes the test email

Expected result:

- The response contains an error of “No user found”.

Result:

- The response returns an error, as expected.

Test 6. api/singin rejects a wrong password

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /signin route using the signin controller.
- The user collection has a user with the test email but not the test password.

Steps:

- Send a POST request to /signin route with a body that includes the test email and password.

Expected result:

- The response contains an error of “Wrong password”.

Result:

- The response returns an error, as expected.

Test 7. api/signin returns a token for a valid user

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /signin route using the signin controller.
- A user with the test email and test password exists in the user collection.

Steps:

- Send a POST request to /signin route with a body that includes the test email and test password.

Expected result:

- The response status is 200, and the body has the user object.

Result:

- The response returns the correct status and object, as expected.

Test 8. api/employees creates an employee and omits the user field in the response

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /employees route using the employees controller.
- No conflicting employee data exists in the database.

Steps:

- Send a POST request to /employees route with a body that includes a valid name and a valid userId

Expected result:

- The response has status 200 and returns the employee object.

Result:

- The response has the correct status and returns a proper object, as expected.

Test 9. api/employees returns employees with populated assignedRoles

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the / employees route using the employees controller.
- An employee with a test role exists in the database.
- A test role document exists in the database.

Steps:

- Send a GET request to /employees route.

Expected result:

- The response status is 200, and the body contains the test employee with the test role.

Result:

- The response returns the correct status and the correct object, as expected.

Test 10. api/job-roles returns an empty list when no roles exist

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /job-roles route using the jobRoles controller.
- The jobRoles collection is empty.

Steps:

- Send a GET request to /job-roles route.

Expected result:

- The response has a status 200 and returns an empty array.

Result:

- The response contains an empty array and status 200.

Test 11. api/job-roles creates a role

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /job-roles route using the jobRoles controller.
- The jobRoles collection is empty.

Steps:

- Send a POST request to /job-roles route with a body that includes the test jobRole name and test jobRole salary.

Expected result:

- The response contains status 200 and the newly created jobRole object

Result:

- The response returns the correct object and the correct status, as expected.

Test 12. api/job-roles returns a role

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /job-roles route using the jobRoles controller.
- The jobRole collection has a role with the test name.

Steps:

- Send a GET request to /job-roles route with a body that includes the test name for the role.

Expected result:

- The response has status 200 and contains an an object of the jobRole with the corresponding name.

Result:

- The response has the correct status and the correct object, as expected.

Test 13. api/shifts creates a shift and omits notes in the response

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /shifts route using the shifts controller.
- The test employee exists in the database.
- The test jobRole exists in the database.

Steps:

- Send a POST request to /shifts route with a body that includes all of the test data for creating a valid employee except the notes.

Expected result:

- The response contains status 200 and the shifts object without notes.

Result:

- The response has the correct status and returns expected object.

Test 14. api/upcoming-by-user returns employee's shifts

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /shifts route using the shifts controller.
- A shift with test data already exists in the database.
- A user with the test data already exists in the database.

Steps:

- Send a GET request to /shifts route with a userId query parameter.

Expected result:

- The response has status 200 and contains the correct test shift.

Result:

- The response has the correct status and returns the expected object.

Test 15. api/upcoming-by-employee returns the employee's shifts

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /shifts route using the shifts controller.
- A shift with test data already exists in the database.
- An employee with the test data already exists in the database.

Steps:

- Send a GET request to /shift route with an employeeId as a query parameter.

Expected result:

- The response has status 200 and contains the correct test shift.

Result:

- The response has the correct status and returns the expected object.

Test 16. api/all-by-date returns shifts for the requested date range

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /all-by-date route using the shifts controller.
- The shift collection has a shift that happens in the test range of dates.

Steps:

- Send a GET request to /all-by-date route with a test date and the next test date as query parameters.

Expected result:

- The response has status 200 and contains the correct test shift.

Result:

- The response has the correct status and returns the expected object.

Test 17. api/week-by-user returns shifts in the requested week

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /week-by-user route using the shifts controller.
- The shift collection has a shift that happens in the test range of dates.

Steps:

- Send a GET request to /week-by-user route with userId, firstDate, and lastDate as query parameters.

Expected result:

- The response has status 200 and contains the correct test shift.

Result:

- The response has the correct status and returns the expected object.

Test 18. api/week-by-employee returns shifts in the requested week

Preconditions:

- In-memory MongoDB is running and is connected.
- Express.js app is configured with the /week-by-employee route using the shifts controller.
- The shift collection has a shift that happens in the test range of dates.

Steps:

- Send a GET request to /week-by-employee route with employeeId, firstDate, and lastDate as query parameters.

Expected result:

- The response has status 200 and contains the correct test shift.

Result:

- The response has the correct status and returns the expected object.

During this study, the application was properly tested and showed a sufficient quality of the final product. The whole system works as expected, has minimal bugs, and has great performance.

CONCLUSION

This application shows that, as a result, it reached the main goal – to build an easy-to-use application for shift scheduling and workload management. It provides the user with a simple interface and allows them to easily complete their work tasks and manage their time. This application went through the full development cycle: initial research of the topic, creation of the requirements, design of the system architecture, development, and testing of the product.

Several existing solutions in the market were analyzed. Square, Sling, and Homebase were compared, and several main features were highlighted as the ones with high importance. For example, management of the employees and the shift display and creation were the ones that existed in each of the competitors' products, and hence were considered valuable for the application.

The requirements for the product were formed and met as a result. Based on the requirements, the system designs and architecture were constructed in order to be a guide during the development of the application. Those designs included not only user interface planning, but also expectations for the backend support and following of the MVVM architecture.

The most modern frameworks were used for the build of the Web application, such as React Native and JavaScript for the frontend, and Express.js and MongoDB for the backend. These choices were made based on a thorough comparison with other options for this kind of development, and together make a stable, secure, and easy-to-use and maintain application that is very scalable and will thrive in case there are upcoming changes to it.

The system was also greatly tested. There were unit tests written for the backend of the application, and all the existing data controllers were tested on happy and sad paths. The system responds as expected, gives correct data, and shows no corruption or unexpected behavior.

As a final product, this application is an example of a system that is ready to be implemented in a small or medium-sized business in order to make the managerial work less strenuous and allow employees to have better access to their work schedules.

This system would also welcome new features and additions, and is ready for continuous development and support. Overall, this research proves the usefulness of the technologies in the managerial field, especially in the task of shift scheduling.

LIST OF USED SOURCES

1. Connecteam URL: https://connecteam.com/online-employee-scheduling-apps/
2. Sling URL: https://getsling.com/
3. Square URL: https://squareup.com/
4. Homebase URL: https://www.joinhomebase.com/
5. Model-view-viewmodel URL: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
6. Figma URL: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma
7. Wikipedia JavaScript URL: https://en.wikipedia.org/wiki/JavaScript
8. React Native URL: https://reactnative.dev/
9. Flutter URL: https://flutter.dev/
10. Wikipedia Wide-column store URL: https://en.wikipedia.org/wiki/Wide-column_store
11. Sommerville I. Software Engineering. 10th ed. Boston : Pearson Education, 2016. 816 p.
12. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2019. 976 p.
13. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 560 p.
14. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
15. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
16. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Boston : Prentice Hall, 2008. 464 p.
17. Date C. J. An Introduction to Database Systems. 8th ed. Boston : Addison-Wesley, 2004. 1024 p.

18. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2016. 1272 p.
19. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. New York : McGraw-Hill, 2019. 1376 p.
20. Nixon R. Learning PHP, MySQL & JavaScript. 6th ed. Sebastopol : O'Reilly Media, 2021. 832 p.
21. Duckett J. PHP & MySQL: Server-side Web Development. Indianapolis : Wiley, 2022. 928 p.
22. Freeman E., Robson E. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2020. 694 p.
23. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge : MIT Press, 2022. 1312 p.
24. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Hoboken : Pearson, 2021. 1136 p.
25. Ernst A. T., Jiang H., Krishnamoorthy M., Sier D. Staff scheduling and rostering: a review of applications, methods and models // European Journal of Operational Research. 2004. Vol. 153, No. 1. P. 3–27. DOI: 10.1016/S0377-2217(03)00095-X.
26. Van den Bergh J., Beliën J., De Bruecker P., Demeulemeester E., De Boeck L. Personnel scheduling: A literature review // European Journal of Operational Research. 2013. Vol. 226, No. 3. P. 367–385. DOI: 10.1016/j.ejor.2012.11.029.
27. Ernst A. T., Jiang H., Krishnamoorthy M., Owens B., Sier D. An annotated bibliography of personnel scheduling and rostering // Annals of Operations Research. 2004. Vol. 127. P. 21–144. DOI: 10.1023/B.0000019087.46656.e2.
28. Burke E. K., De Causmaecker P., Berghe G. V., Van Landeghem H. The state of the art of nurse rostering // Journal of Scheduling. 2004. Vol. 7. P. 441–499.
29. Baker K. R. Introduction to Sequencing and Scheduling. New York : Wiley, 1974. 305 p.

30. Pinedo M. Scheduling: Theory, Algorithms, and Systems. 6th ed. Cham : Springer, 2022. 673 p.
31. Brucker P. Scheduling Algorithms. 5th ed. Berlin : Springer, 2007. 371 p.
32. Glover F., Kochenberger G. Handbook of Metaheuristics. 3rd ed. Cham : Springer, 2019. 843 p.
33. Hillier F. S., Lieberman G. J. Introduction to Operations Research. 11th ed. New York : McGraw-Hill, 2020. 960 p.
34. Winston W. L. Operations Research: Applications and Algorithms. 4th ed. Belmont : Thomson Learning, 2004. 1416 p.
35. Taha H. A. Operations Research: An Introduction. 10th ed. Boston : Pearson, 2017. 848 p.
36. Park J., Vrettos E. A hybrid optimization approach for employee rostering: Use cases at Swissgrid and lessons learned. URL: https://arxiv.org/abs/2111.10845 (date of access: 10.06.2026).
37. Uhde A., Schlicker N., Wallach D. P., Hassenzahl M. Fairness and Decision-making in Collaborative Shift Scheduling Systems. URL: https://arxiv.org/abs/2001.09755 (date of access: 10.06.2026).
38. Gaikwad H. R., Mahind P. K., Mane S. U. Comparative Analysis of GPGPU based ACO and PSO Algorithm for Employee Scheduling Problems. URL: https://arxiv.org/abs/2203.12239 (date of access: 10.06.2026).
39. Shetty S., Prabhu V. Quantitative Models for Workforce Management in a Large Service Operation // IFIP Advances in Information and Communication Technology. 2024. P. 367–381. DOI: 10.1007/978-3-031-65894-5_26.
40. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (date of access: 10.06.2026).
41. MySQL Documentation. URL: https://dev.mysql.com/doc/ (date of access: 10.06.2026).
42. Microsoft .NET Documentation. URL: https://learn.microsoft.com/dotnet/ (date of access: 10.06.2026).

43. ASP.NET Core Documentation. URL: https://learn.microsoft.com/aspnet/core/ (date of access: 10.06.2026).
44. React Documentation. URL: https://react.dev/ (date of access: 10.06.2026).
45. Mozilla Developer Network (MDN Web Docs). URL: https://developer.mozilla.org/ (date of access: 10.06.2026).

APPENDICES