OULUN AMMATTIKORKEAKOULU

Nabeel Hussain

Full Stack Hybrid Web Application for an Auto Repair Company

Full Stack Hybrid Web Application for an Auto Repair Company

Nabeel Hussain
Bachelor's Thesis
Spring 2023
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Engineering

Author(s): Nabeel Hussain
Title of the thesis: Full Stack Hybrid Web Application for an Auto Repair Company
Thesis examiner(s): Janne Kumpuoja
Term and year of thesis completion: Autumn 2023 Pages: 35

The thesis presents the development of a Full stack hybrid web application for an auto repair company, empowering its users to create, update, and manage jobs, log hours, and access job history. The research involved the careful selection of Frontend, Backend, and Database frameworks to suit the specific use case. Ionic with TypeScript React was chosen for the frontend, while Node.js with ExpressJS served as the backend technology, and PostgreSQL with Prisma for storing data.

The selected frameworks were evaluated against criteria, ensuring the application's responsiveness across devices and operating systems. Extensive documentation and debugging resources facilitated smooth development and future maintenance. The hybrid approach allowed for the possibility of converting the application into a native version if needed.

This work delves into the planning and implementation of functional requirements, and the achieved results showcase improved performance and user experience. Through a comprehensive study of various documentations and Internet resources, this thesis makes a valuable contribution to the hybrid web application development domain.

Keywords: Web Development, Hybrid Web Application, FullStack Application, Cross-Platform

CONTENTS

Vocabulary

Fullstack – Development of an application that includes the Frontend, Backend, and Database.

GUI – Graphical User Interface

Framework – A solution that provides premade components.

Frontend – Development of User Interface and its logic.

Backend – Development of Server-side logic.

IDE – Integrated Development Environment

Pull Request – An event which means that a developer is ready to integrate their changes to the project.

API – Application Programming Interface

REST – Representational State Transfer

CRUD – Create, Read, Update, Delete

HTTP – Hyper Text Transfer Protocol

JSON – JavaScript Object Notation

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

Pseudocode – An informal method used to describe an application's code.

SDK – Software Development Kit

SQL – Structured Query Language

Hash – Event of changing a string to another value.

Props - Method of passing data from one component to another.

Cache – Entity in a software or hardware that temporarily stores data.

JWT – JSON Web Token

Schema – Structure of Database

Middleware – Software that provides extra capabilities to applications.

ORM – Object-relational Mapping

NAS – Network-attached Storage

INTRODUCTION

Businesses around the world began incorporating technology into their domain to maximize efficiency and quality of their products and services for their customers. This is true, especially in today's world where technology has managed to infiltrate almost every aspect of our lives. The case is no different when it comes to the world of automotive repair businesses where multiple instances of jobs and their scheduling becomes a hassle, and a modern solution is required to refine the flow of work.

A vast majority of automotive repair businesses utilize conventional methods such as using a notepad to keep track of the daily jobs or an employee's working hours. This method has proven to be inefficient and frustrating as it is limited to access, edits, and updates by its users. Furthermore, it has a large room for errors in reporting hours of work by the employees which contributes to false reporting and or mismanagement of the payment of salaries. As a result, countless hours are lost which could have been spent on productivity instead of investigating mismanagement.

To meet the demand of an efficient method to control the jobs and their scheduling, a Full stack hybrid application was created, in which the users can manage their jobs, working hours and their history of jobs in a systematic manner so that resources may be allocated efficiently in the future. The idea of hybrid applications is to create an application that would run on numerous devices with different operating systems using the same codebase.

FIGURE 1: Hybrid vs Native Apps (1)

This eliminates the concept of rebuilding the same application with same functionalities from scratch while performing as any other native application, hence saving precious time and money of developers (2). Ionic is such a framework where applications that are developed with it are hybrid as standard due to its UI structure and reusable components. It is for this reason that Ionic was chosen as the frontend framework for this project. For the backend Node.JS with ExpressJS was used as it has extensive documentation and guides and is the most used backend framework in the web application development domain. Database is handled by PostgreSQL connected by Prisma which is a database-connecting library for Node.JS. Lastly, TanStack Query which is a library for data-fetching in the frontend from the backend.

This thesis aids in the study and development of hybrid applications through the extensive research and demonstration of the tools and methods used in developing this application for the specific use case of an automotive repair business, while maintaining a responsive UI for many of the devices.

1 FOUNDATION

The following sections discuss various concepts and methods applied by software developers around the world, as well as their significance. These include Version Control, The use of APIs, OOP and Modular Code, Cross-Platform Compatibility, as well as Database and Security. These concepts are the cornerstone of the modern software development world and set the stage for the development of Car Repair App:

1.1 Version Control

In the world of software development, the ability to control the progress of work, i.e. (making changes to the work and going back to a previous version to update/remove something) has shown its benefits when used and demonstrated the apparent disadvantages when not used. As such, the use of a version control system became not just a supportive system but a necessity of developers when building a software. Git has become the standard version control system of the software development world, thanks to its robust features which allow developers to collaborate on their work, track their progress, create separate branches for purposes such as working on a separate feature of the application. Along with that, keep the versions of their work safe by having a local copy on their machine as well as on a remote server. The developers can use Git in various forms such as using their operating systems' Terminal window, GUI software such as Sourcetree (FIGURE 2) which allows graphical representation of branches and commits, or as an extension in their favorite IDE.

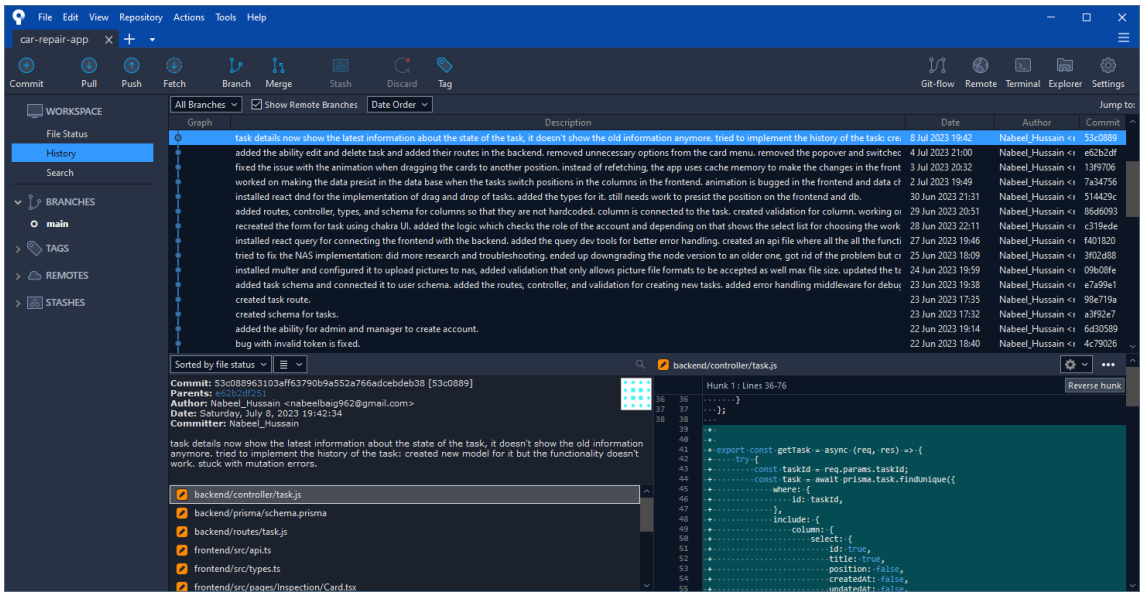


FIGURE 2: Sourcetree: A Git GUI

Progress can be tracked separately if the developer wishes when working on a separate component of the application. When working in a team the other team members can review a commit that was made by a team member and can amend changes or leave a comment for the said team member. The project can also be set up in such a way that an administrator would be able to control which commit can be merged to the main branch by reviewing the team members' pull request (3). This method of version control prevents bugs, breaking of the code, and unintentional mergers which can create frustration and loss of time amongst the team. When it comes to the development of a hybrid web application which conventionally contains multiple components such as the frontend and a backend, the significance of the use of a version control system, and Git is no exception.

1.2 The use of APIs

The APIs are used by the developers to establish a connection between a variety of software that allows them to work alongside each other. An example of this would be of a hybrid web application where the frontend displays the data which the user requests by sending a request to the backend. In between the frontend and backend, is an API that serves as the connection between the two and allows them to communicate this request. The backend then sends the requested data as a response to the frontend by answering the API call. Furthermore, the backend using another API call can make changes to a database which is connected to the hybrid web application (4). The relationship of each component is illustrated below (FIGURE 3):

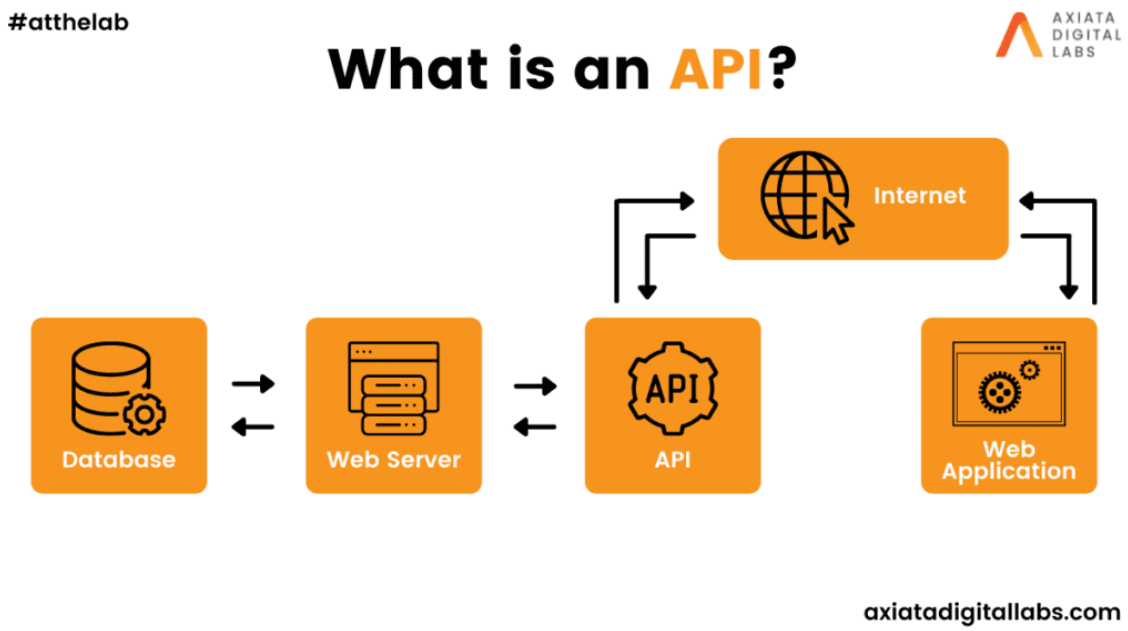


FIGURE 3: Procedure of an API (5)

A commonly used API in web application development industry is known as the REST API. The REST API provides the developer with HTTP functions with which they can perform CRUD operations with their application. Operations such as retrieving, adding, modifying, and deleting data are naturally done within any application, especially when one requires a user of that application to fill in a form, i.e. A user registration form for a web application. The equivalent of these operations in REST API are GET, POST, PUT, and DELETE requests. The request header contains API key or user authentication information, the operation that the request wants to perform i.e., a POST request, an endpoint which describes the destination for the request, and finally the body where the data is contained. The response received back from the backend via the API comes back in JSON format. If there is an error within the request or response, the API provides HTTP status codes to help the developer debug the application. A well-known example of a HTTP status code is '404: Not Found' which means that the requested data/page does not exist. The benefit of REST API is that the information is readable by both the developers and the machine, which streamlines the development of the application and its operations (6).

1.3 Object-Oriented Programming and Modular Code

Object-Oriented Programming is already widely utilized by the developers developing a software, and the case is no different when it comes to the development of modern websites, web apps, as

well as hybrid web applications which are mostly developed using JavaScript and its many frameworks available today that complement the backbone of modern web, which are HTML and CSS. The concept of OOP is to have the code of the software written in such a way that it concentrates on modularity and reusability. This is achieved when the focal point of the development is on classes and using those classes to generate multiple instances of objects with various methods and attributes that make them unique from each other. A class itself is a blueprint for generating objects that inherit properties and methods of that class. An example of this can be of a class of countries. The class can have properties such as Name, Language, and Currency, they can have methods that retrieve the value and modify the properties (7). The following pseudocode will illustrate this (FIGURE 4):

```
4 references
class Country;
//Properties
0 references
private string name;
0 references
private string language;
0 references
private string currency;

//Constructor
public Country(name, language, currency):
this.name = name
this.language = language
this.currency = currency

//Methods
public method getName():
return this.name

public method getLanguage():
return this.language

public method getCurrency():
return this.currency

//Instances of Countries
Country finland = new Country("Finland", "Finnish", "Euro")
Country mexico = new Country("Mexico", "Spanish", "Pesos")
```

FIGURE 4: OOP class pseudocode

The class is embodied with instances that constitute Finland and Mexico with their respective properties. This approach of writing code eliminates the chances of repeating the same code and enables modularity reusability since the code is divided into smaller parts that can easily be worked on separately. As a result, the code also becomes easier to test and debug. The developers can also

maintain their focus on different parts of the code without having to worry about breaking anything. Hence, applying updates and other modifications becomes significantly less troublesome.

1.4 Cross-Platform Compatibility

When a software is planned for development, one of the most important factors that is discussed by the developers is if their software will be developed using native technologies i.e., Android or iOS, or will a hybrid technology which can run on both platforms using the same codebase be more suitable? A caveat with software developed with native technologies is that while their performance is great and takes advantage of the devices' many features and have access to the native UI elements, they cannot run on each other's devices. The reason for this is that the native technologies use their own programming languages, SDKs, and tools that are specific for their platform. This however is not the case when a hybrid technology is chosen as the software is made using web technology frameworks which can run on all modern platforms, whether it be Android, iOS, or even a Windows machine. There are vast amounts of UI libraries which can help developers style their software to match with the native ones. This is possible due to the reusability of the codebase by default which saves the developers countless hours and effort. The choice of technologies mainly depends on the software's non-functional requirements (8).

1.5 Database and Security

A database for software is a key component that handles the data in an organized fashion for the purpose of streamlined and easier access upon request. Depending on the requirement of the software, a database usually manages sensitive information such as an email and password of a user, their address and payment methods, or less sensitive information such as the number of apples remaining in a grocery store. A GUI software solution known as the Database Management System such as pgAdmin for PostgreSQL is used to control, create, update, delete the records which are stored digitally on a different machine.

The data is structured in a table with rows and columns configuration which allows trouble-free readability by both humans and machine. This is known as a relational database. A descriptive title is given to the column and more importantly a data type as well as character limit is also assigned for the purposes of organization, validation of erroneous data, and security. An example of this

would be the row of phone numbers. When a user fills a form that contains a field for a phone number, if not for the datatype and character limit such as ‘varchar(50)’ the user can enter erroneous data such as phone number in an invalid format or even letters and symbols, which clearly do not belong in a phone number field. This type of validation is of great significance and is enabled on all columns of data.

As databases often contain sensitive information, their security is also of great importance. In the case of storing a password of a user, it is not stored as one might expect it to be. That is considered highly unsecure as a data breach by bad actors would compromise the sensitive information resulting in damages. Instead as a convention, passwords are stored as hashes which are considered secure as even if they land into the hands of bad actors, it can take years to crack them to retrieve the password which is not feasible. On top of this, the database itself is often encrypted and transfer of data from one machine to another is encrypted. This prevents bad actors from hijacking the data traffic while it is on its way to its destination. All the data fields have validation so attacks such as an SQL Injection are prevented. Finally, there is an off-site backup process that keeps the data safe in case of data corruption or events beyond human control (9).

2 FRONTEND DEVELOPMENT

In the development of an application whether it is a web app or a native app, the development of their frontend has an important contribution in making the user experience well ordered. The design of UI elements and their interactivity with the user is one of the main parts of the frontend development. As such, with the Car Repair App the UI elements were created in a modern user-centric way that would make the usage of its functions streamlined and on par with the current UI design in hybrid web application domain. This enables the application to performance to be highly efficient and use as minimum resources as possible. Apart from this, the frontend provides the user with the connection to the backend from where they can make queries to access the desired data. An example of a query made by the user (administrator in this case) can be of the list of vehicles that are currently entered in the system, what their status is, and which employee is currently working on them.

2.1 Framework Selection

For the development of a Car Repair App, the most significant non-functional requirement was that it must be able to run on different devices as one might expect that not everyone will have a device with the same operating system. Keeping this in mind, it was decided that a certain kind of framework be chosen from which a hybrid web application could be developed. Many frameworks such as React Native, and Xamarin were considered but in the end, Ionic was chosen for its robust UI libraries and React support. It boasts the conversion from a web app to a native app using the Capacitor framework whenever desired which can grant access to native functionalities of the device such as Bluetooth and Camera. Along with the classic web technologies, Ionic can be used with either Angular, React TypeScript, or Vue, which are plenty of options for the developers. Having a smooth prior experience with this framework from using it in previous projects, it was chosen as the frontend framework for this project (10).

The application was developed using React TypeScript. The benefit of React is it allows developers to develop the application in a ‘React’ way. That is, the application development process is component-based, and the codebase is modular. This allows for the code to be cleaner and easier to read

by the developers and helps debugging and code reusability. Each tab in the application is a component that is loaded into the main 'App.js' file i.e., the Inspection tab. Within the components, UI elements are also added as separate components. An example of this is the column component which loads cards that display the car entry information. Data is passed from one component to another using the 'props' mechanism as is standard with any application made with React. This allows the components to communicate with each other during runtime, essentially making the application work.

For connecting the frontend to the backend, TanStack Query was used. This is a data-fetching library like axios where the server state can also be managed. In the Car Repair App, TanStack Query is responsible for fetching, mutating, and updating the car entries, employees and their working hours, as well as the column data. All operations are communicated with the backend. Keeping the data up to date is essential in the app as both columns and cards have data that changes continuously. This is achieved by using the 'invalidateQueries' method with the 'queryClient' function (11). For example, in the application it is used in a function (FIGURE 5) which adds tasks to the database. In this function, after a successful mutation of the task data, the queries of 'columns' and 'workers' (employees) are invalidated in the cache memory. Therefore, the updated data is displayed to the user on the frontend.

```
66  const addTaskToDb = (e: FormEvent<HTMLFormElement>) => {
67    e.preventDefault();
68    if (!columns || columns.length < 0) {
69      toast({
70        title: 'Create a column before creating a task',
71        status: 'error',
72        duration: 5000,
73        isClosable: true,
74      })
75    } else {
76      const task = new FormData()
77      const assigned = (user && user.role === Role.EMPLOYEE) ? user.id : values.assigned
78      task.append('vehReg', values.vehReg.toUpperCase())
79      task.append('note', values.note)
80      values.images.length > 0 && Array.from(values.images).forEach((file) => {
81        task.append('images', file, file.name)
82      })
83      task.append('assigned', assigned)
84      task.append('column', columns[0].id.toString())
85      mutate(task, {
86        onSuccess: () => {
87          queryClient.invalidateQueries({ queryKey: ['columns'] })
88          queryClient.invalidateQueries({ queryKey: ['workers'] })
89        }
90      })
91    }
92    onClose();
93  };
```

FIGURE 5: addTaskToDb's use of TanQuery functions

2.2 User Interface Design

Since the Car Repair App is a hybrid web application, the UI design and layout of the application capable of running on all types of devices and screen sizes is of utmost significance. Various UI elements available from Ionic itself and another UI library in addition called Chakra UI were used to design a modern and streamlined user experience. The Ionic framework provides templates for different types of layouts for an application. For example, the developers can use a tab view for the layout of their application. The benefits of having a tab view layout in an application are flexibility, consistency in design and layout, as well as having different sections of the application accessible by essentially switching to a different tab. As a result, the user can quickly navigate various parts of the application swiftly. It was for this reason that the tab view was chosen as the layout for the Car Repair App. Additionally, the tab view works flawlessly on all screen sizes, whether it be on a smartphone, tablet, laptop, or a PC (12). In the Car Repair App, each tab has its own layout which depends on the type of content displayed to the user. These are described in the following subsections:

2.2.1 Inspection Tab

The Inspection Tab (FIGURE 6) illustrates the view in which the user can see the cars that are currently entered into the system, as well as their currently assigned employee and status. To make the view as user-friendly as possible while maintaining flexibility, each status was made into a column. In these columns, entries made by the user would appear as a card view with their respective details and be color coded at the same time depending on which column the card is currently in or transferred to. The car entries are added in the system by clicking/touching the 'plus' button on the bottom right-hand corner. A popup modal with a form will appear where the user can enter the details for making a car entry into the system such as the vehicle registration number, employee the vehicle is assigned to (admins and managers can select the employees), task description i.e., Oil Change, and any pictures associated with the job. There is a possibility to delete or edit the cards. Making the cards with different colors helps them become more perceptive to the user so they can easily extrapolate the entries and their details. The columns themselves are customizable, they can be added, renamed, removed, and be moved around to different places. In addition, they are also responsive as their height depends on the number of cards.

FIGURE 6: Car Repair App: Inspection Tab

2.2.2 Shifts Tab

The Shifts Tab (FIGURE 7) exhibits the number of working hours on a specific day for the user that is currently logged in. When the green 'Check In' button in the center is pressed, it allows the user to essentially 'start' their working hours from that point onwards. After it is pressed, it changes to the red 'Check Out' button which when pressed allows the users to mark that they are done for the day and the information is stored for the managers and admins to view. The clipboard icon on the top right-hand side of the screen serves as a button which opens a page where the user can see the history of their working hours in table with row and columns. The last time of check out is displayed on the top left-hand side of the screen. The view differs slightly depending on the permissions that the user has, i.e., the admin can view the working hours of all employees while the employee themselves can only view their own working hours.

FIGURE 7: Car Repair App: Shifts Tab

2.2.3 Summary Tab

The Summary Tab (FIGURE 8) displays a table where the admin or manager can view the number of jobs assigned to an employee in their status. The table has 5 columns which describe the name of the employee, 'In Progress', 'On Hold', 'Car Wash', and 'Done' statuses. The number in each row represents the number of jobs the employee has had in those statuses. The data is updated in real time so the admin or manager can see when the numbers change in those statuses. It is a simple solution developed for the purpose of easier tracking of employees' jobs.

FIGURE 8: Car Repair App: Summary Tab

2.3 Responsive Design

When a hybrid web application is developed, one of the core features of its development is that the app functions well on different types of devices and different types of screen sizes. The idea behind it is consistency in UI design throughout the development. As such, the Car Repair App is developed using hybrid web application framework, so the responsive design methods were used extensively during the development. For this reason, the tab view layout was chosen as it remains consistent on all different screen sizes, as previously mentioned. Due to some UI elements such as the text and image (custom or premade) not being responsive by default, CSS media queries were used so that the layout and UI elements themselves do not fall apart when the screen size is manipulated. These queries allow the developers to write styles which are specific to different screen sizes using breakpoints. Hence, when the screen size changes by width, height, or both from the breakpoint; the elements change their size to the specified sizes (13). An example of this can be seen below:

```
52  @media only screen and (max-width: 780px) {
53      .column-item {
54          padding: .1em;
55      }
56
57      .column-container {
58          overflow-x: scroll;
59      }
60
61  }
```

FIGURE 9: CSS Media Query

Likewise, other elements such as columns and cards in the application change or rearrange according to the sizes specified in the screen size breakpoints.

3 BACKEND DEVELOPMENT

A backend in an application (web application or stand-alone) handles the server-side logic. As the name suggests, the logic is mostly *behind* the given application and out of reach of a regular user. It acts as a bridge between the frontend (which the user interacts with) to the database and other APIs. With regards to any modern web application, the user interacts with the frontend and generates as well as requests data. These requests from the frontend are sent to the backend. The backend then responds to the frontend with the result. An example of this can be of a library web application where the user searches for books that belong to the Sci-Fi category. The backend responds to the user with the information by sending it to the frontend, which then updates the page with the result. Similarly, the logic of user authentication is handled by the backend as well. 3rd Party APIs are also connected to the backend that make the availability of services such as cloud services possible (14). The backend in the Car Repair App provides similar functions.

3.1 Selection of Technology and Middlewares

For a hybrid web application like the Car Repair App, the selection of a backend technology which is not only industry-standard but also robust, was necessary. Therefore, ExpressJS was chosen due to being familiar with previous projects, and the vast number of resources available to assist with the implementation in the development of this project. In the backend, there are routes that provide the following with unique URLs for user registration and authentication, retrieving, deleting, editing, and creating columns as well as car entries. Along with that, is the route for employees and their check-in status. All these routes are controlled by a middleware from ExpressJS itself called the 'Router'. The Router connects the backend with routes which then connects them to the callback functions. These contain all the asynchronous functions and logic behind each request as well as responses and error messages (15). Each part of the backend has a dedicated file in a directory called 'controller' which contains the logic mentioned previously. The routes handle various HTTP requests associated with them. To support functions such as user authentication and authorization, server-side form validation, control of multipart form data, and generation of unique IDs, multiple middlewares were used. They are discussed in the following subsections:

3.1.1 User Authentication and Authorization

The feature of User Registration and Authentication is the cornerstone of any modern web application that enables its users to access their accounts which allows them to interact with its features. There are countless examples present today such as Facebook, YouTube, and Amazon. All of these require its user to register and authenticate. It proves to the website that it is a particular user interacting with it so that the website or webapp can provide the expected content. The user information is secured and stored in a database. Passwords are stored as hashes instead of the actual password strings for security purposes. The routes for registration and authentication are all protected. With the Car Repair App, the situation is no different as multiple types of users will register and authenticate with it to use it for their daily work.

To allow the process of user authentication and authorization as well as access control to be systematic, JSON Web Tokens middleware (also known as JWT) was used. JWT is a method which provides a common standard for sending and receiving data as JSON. It is mostly used in authorization of a user, from where they can access the routes and requests which are designated for them. JWT is sent in the Authorization header and uses the Bearer for the format to the API in the frontend. The token is generated by a Secret string which is created upon the authentication of a user with the application. The string has 3 parts which are Header, Payload, and Signature. The Header contains the information about the type of token and the algorithm which signs it i.e., SHA256. The Payload contains information about the user and any other relevant information. The signature combines the Header, Payload, and the Secret, all of which are encoded using Base64. In addition, the algorithm is also included, and the entire thing is then signed. The result of this is an encoded string which becomes the JWT. JWTs are only valid for a limited time and upon expiration the session will also expire, resulting in the user logging off (16).

As mentioned previously, the passwords of an account are stored as hashes. In the development of Car Repair App, this was kept in mind and hence passwords were hashed using a middleware called ‘bcrypt’. ‘bcrypt’ allows the developers to implement a function that secures the passwords of a user account by hashing them. It is done so by providing the bcrypt function’s ‘.hash’ method by providing the password from the request body and using the defined number of salt rounds asynchronously. Both the passwords and PINs are hashed by ‘bcrypt’ in the Car Repair App.

3.1.2 Server-Side Form Validation

Validation means to verify the information and whether it fits the expected format. This is typically implemented in a form to minimize false or absurd data. Although commonly implemented on the frontend of the application, nowadays it is also implemented in the backend for an additional layer of protection. With this reason in mind, it was also implemented in the Car Repair App to keep it updated with the current standards of form validation. To achieve this validation in the backend, 'zod' was installed. Zod is a validation middleware that enables the developers to implement a schema, through which the data is validated and then passed onto the database. A validator file is created that contains schemas for individual form fields i.e., e-mail and password, and for car entries vehicle registration number. The validator file is connected to the route files where the validator functions are imported and used in the route functions themselves (17). An example of a zod schema can be seen below (FIGURE: 10)

```
57  const vehReg = z
58      .string({
59          required_error: "vehicle registration is required",
60          invalid_type_error: "vehicle registration must be a string",
61      })
62      .min(7, { message: "vehicle registration must be 6 characters long" })
63      .max(7, { message: "vehicle registration must be 6 characters long" })
64      .trim()
65      .regex(/^[a-zA-Z0-9-]+$/, {
66          message: "vehicle registration must be alphanumeric",
67      });
68
```

FIGURE 10: Zod Schema for Car Registration Number

In the schema there are conditions for the data that will be passed through it. '.string()' method will check that the data entered is in a string format. If it is not or if it is empty, it will pass error messages respectively. The '.min()' and '.max' method will validate the length of the data and pass an error message if it is too long or too short. '.trim()' will remove any whitespaces upon entry and lastly '.regex()' will validate the format of data that is expected if there are invalid symbols in the data it will pass an error message.

3.1.3 Control of Multi-Part Form Data

In many different types of websites/webapps, users often send pictures or files when sending form data or making some other kind of upload. For example, a user on Amazon may upload pictures of their defective item as evidence when trying to claim a refund. This type of form data is called Multi-

Part Form Data. In the Car Repair App, this was achieved by using a middleware called Multer. Another middleware to allow interaction with the device's filesystem called 'fs' was installed. One of the features in the app for the employees is the ability to upload pictures of their work. For example, an employee may want to make a record of the car's mileage so they can take a picture of the odometer. To do this, the app would need to access the employee's device's file system so the pictures can be uploaded. Multer provides a function in which the developer can add conditions for the filename of the image, the type of files that it will accept, and the file size (18). A unique identifier is attached to the file name by the 'uuidv4' middleware for easier identification. To make sure that only pictures are uploaded with the car entry, a validator is added to the function which checks the file extension. If the file extension does not match the type for pictures, an error message will be displayed to the employee. A limit of file size is also added to the function to make sure that the payload is not too large for the backend, which can cause performance issues.

4 DATABASE MANAGEMENT

A database for a hybrid web application is a key component that handles the data in an organized fashion for the purpose of streamlined and easier access upon request. Depending on the requirement of the software, a database usually manages sensitive information such as an email and password of a user, their address and payment methods, or less sensitive information such as the number of apples remaining in a grocery store. A GUI software solution known as the Database Management System such as pgAdmin for PostgreSQL (FIGURE 11) is used to control, create, update, delete the records (19).

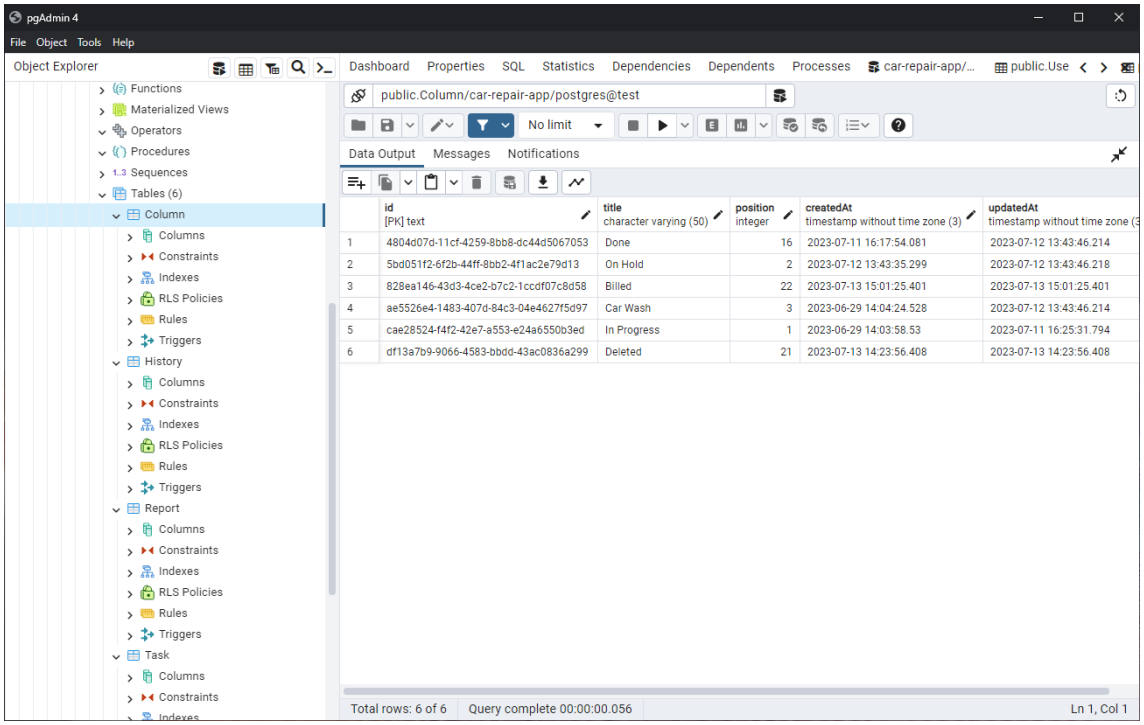


FIGURE 11: pgAdmin for PostgreSQL

The data is structured in a table with rows and columns configuration which allows trouble-free readability by both humans and machine. This is known as a relational database. A descriptive title is given to the column and more importantly a data type as well as character limit is also assigned for the purposes of organization, validation of erroneous data, and security. An example of this would be the row of phone numbers. When a user fills out a form with a phone number field that lacks data type and character limits, they can enter incorrect data, like invalid phone number formats or non-numeric characters. For this reason, this type of validation is of great significance and

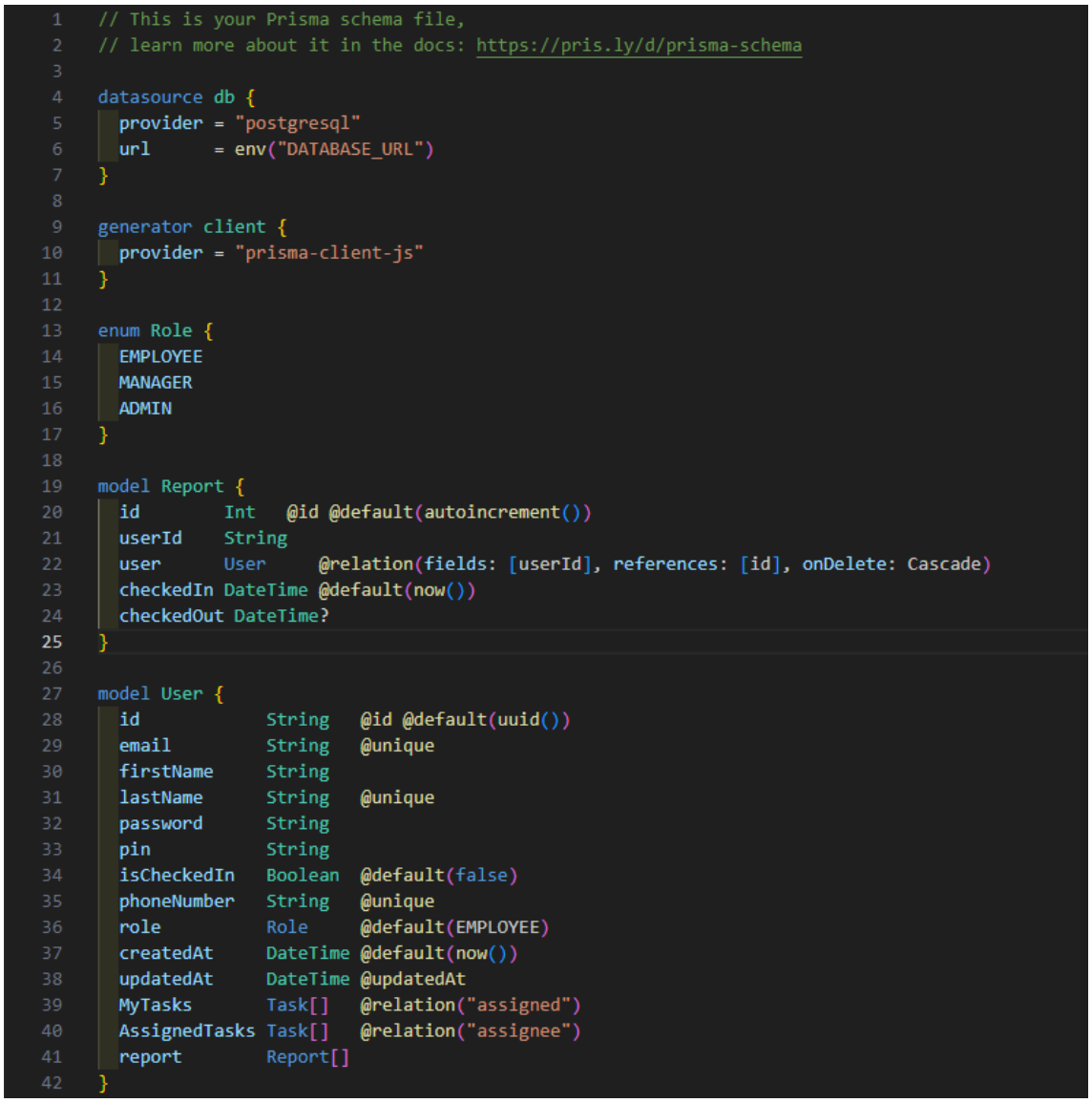
is enabled on all columns of data. A backup of the database is also made to ensure the security and availability of the data in case of system failure or events beyond human control.

4.1 Selection of Technology

When planning the Car Repair App, one of the key functional requirements was to have a SQL database. After researching the available solutions for a SQL Database, PostgreSQL was chosen due to having previous experience and its compatibility with many different platforms. For managing PostgreSQL, pgadmin4 was installed to provide graphical representation of the schema and easier modification of the data. pgAdmin4 also allows the database to be backed up and stored safely elsewhere for the protection of the data (20). To connect the database with the application, Prisma ORM was installed in the backend. Rather than writing SQL, Prisma makes the process easier by allowing the developers to exhibit the data in object-oriented code. It also provides its own definitions for schemas which are human-readable. Schemas are transformed into type safe definitions which allows for easier debugging during the development. Based on the schemas, Prisma generates queries which saves time by not having to write SQL manually. Prisma also handles schema migration, when something is modified in the database it will keep records of changes made to it which helps with the maintainability of the database (21).

4.2 Schema

In Prisma, the schema is defined in a file called ‘schema.prisma’. The schema is written in Prisma’s own language, and it contains all the tables, rows, their datatypes, and constraints. Similarly, the schema for Car Repair App is also defined in this file created by Prisma in the backend. Below is a screenshot of the schema used in the Car Repair App (FIGURE 12):



```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

enum Role {
  EMPLOYEE
  MANAGER
  ADMIN
}

model Report {
  id        Int   @id @default(autoincrement())
  userId    String
  user      User     @relation(fields: [userId], references: [id], onDelete: Cascade)
  checkedIn DateTime @default(now())
  checkedOut DateTime?
}

model User {
  id            String   @id @default(uuid())
  email         String   @unique
  firstName     String
  lastName      String   @unique
  password      String
  pin           String
  isCheckedIn   Boolean  @default(false)
  phoneNumber   String   @unique
  role          Role     @default(EMPLOYEE)
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
  MyTasks       Task[]   @relation("assigned")
  AssignedTasks Task[]   @relation("assignee")
  report        Report[]
}
```

FIGURE 12: Car Repair App Schema

For Prisma to communicate with PostgreSQL database, they are connected with the database URL. This can be seen in the ‘database db’ block of the code. Role of the users’ ‘EMPLOYEE’, ‘MANAGER, and ‘ADMIN’ are described as enum. These are the types of users that will be using the application. Below that, are the tables of ‘Report’ and ‘User’ which are described as ‘model’ in the schema. The ‘Report’ model handles data for the employees’ check in and check out time of their shifts. The ‘User’ model handles the main data of the user of the app. In this model, all the basic information of the user i.e., their first and last name, email and passwords, as well as their tasks and UUIDs are stored. Each row has their respective datatypes and constraints to represent their validation, as well as their relationships. For example, in the ‘Report’ model the user row has a relation constraint which expresses the relationship between the ‘Report’ and the 'User’ model. It defines that the ‘userID’ in the ‘Report’ model is connected to the ‘id’ in the ‘User’ model.

5 CHALLENGES

During the frontend development of the Car Repair App, some parts of it were relatively easy due to experience but many challenges were also encountered. Starting with the layout of the application: while the tab view layout promises a consistent style for all devices and screen sizes, the experience of making it consistent proved to be challenging. Each UI element has their own attributes which define their behavior depending on the framework they are imported from. The Popover UI component from Ionic itself proved to be troublesome when trying to set its position. It would often open in different places when clicking on the 'Add Task' button. It took numerous hours to troubleshoot which included setting a fixed position to FIGURE out the strange behavior. Alas after testing Chakra UI and its various UI components, the Modal UI component was implemented from there instead. The issue seemed to have been resolved after that. The same phenomena happened to the task columns where the columns would not remain in their expected places while the user scrolled through task cards, although this bug was fixed with trial and error with the CSS flex box attributes. Along with the CSS bugs, issues were encountered with functionality of features as well.

During the development of the feature that enables the users to edit a task, errors related to updating the data were encountered between communications of the frontend and the backend. The task data would not get updated and upon thorough investigation, the culprit turned out to be the incorrect logic in the API callback function. The most difficult part of all were the mutation errors with the use of TanStack Query's 'useMutation' hook. Development was further complicated due to factors such as unfamiliarity with the framework, as well as time-consuming research in learning to use the library correctly in the Car Repair App. The 'useMutation' hook is used in creating, updating, and deleting columns and tasks. Errors occurred due to the structure of the hook with the component data being incorrect. Many hours were spent fixing the mutation errors by researching the documentation, as well as searching Stack Overflow for possible answers. Each component that utilized the 'useMutation' hook came with its own set of errors.

With the columns, the drag and drop library, which was installed to enhance the experience on devices with touch screens, caused mutation errors due to bugs with the animation as well as the constant fetching of data. This was resolved by making the app use cache data and using 'queryClient.invalidateQueries({ queryKey: ['columns'] })', this also resolved some of the animation issues that occurred when a column or a task were dragged and dropped to a different location. It

would take a split second for the dragged item to update its state, which resulted in a poor user experience.

6 RESULTS

When the Car Repair App was being planned and developed, the main objective was to make the demands of the automotive repair businesses simple and easy by utilizing modern web technologies, from a time-consuming process of manual record-keeping using a notepad and a calculator. This thesis and project described and illustrated every part of the development of the application, which includes how everything was implemented and connected to form a functioning application that a user can use.

The result of my work exemplifies the capabilities of web technologies which deliver a seamless, easy-to-use solution that increases efficiency, saves time, and is cost-effective. The tools and frameworks used in the development have massively contributed to the success of this project, due to available documentation, experience, and other resources. Making the application cross-platform from its very foundation has proven the versatility of the hybrid web application path as one code base can run on various types of devices no matter what the screen size is, compared to the native development path which would require the entire application to be rewritten. Functionality such as the storage of pictures in a NAS system has not been available due to the unavailability of the system itself, as well as the method to connect it with the application. The pictures therefore are stored in the local storage. The application in its current form is deployable and useable to the solutions where the frontend, the backend, and the database can be hosted. The application can currently run on local host and illustrate its functionality to the customer.

7 FURTHER DEVELOPMENT

One of the parts that can be worked on in the future is the implementation of the NAS system. Another part would be the increase in the number of pictures that an employee can upload when making an entry. The current limit is 5 pictures to prevent possible performance issues with the current implementation. The NAS connection may allow this functionality to be implemented in a streamlined fashion. As technologies evolve daily, the application should be maintained with up-to-date libraries which include vulnerability patches for security purposes, as well as improvements in performance. For deploying the application, there are solutions that consist of cloud computing services such as Microsoft Azure and Amazon Web Services (AWS) where these types of applications can be deployed. When deployed, the application and its data are accessible from anywhere in the world, as they are now available on the Internet.

AWS was introduced in 2006 and has been leading the technology and the market share of cloud computing services, providing extensive solutions for application deployments, storage, IoT, security, and many more. Azure was introduced in 2010 and provides many of the same but limited cloud services and has steadily gained interest and demand over the years. AWS has the benefit of having multiple examples of applications and services utilizing it and thus, provides future developers with insight about how to use their cloud services via their extensively detailed guides and documentation.

When it comes to integration with Windows applications, Azure holds the benefits here as they support it from the ground up. Open source models are supported by both AWS and Azure. Databases such as SQL and NoSQL are all supported by AWS and Azure. AWS provides RDS (Relational Database Service) which allows various popular database engines such as PostgreSQL and MariaDB to be deployed. Azure provides its own service for individual databases i.e., PostgreSQL and MariaDB would have their own service. For application deployment, both AWS and Azure provide excellent cloud service. AWS provides its customers with EC2 (Elastic Compute Cloud) which offers many configurations and can be easily utilized by almost any type of application. They can be created as instances or containers. Azure provides its customers with VM (Virtual Machines) to deploy their applications.

Lastly, the pricing model of the cloud services is different when comparing AWS and Azure. AWS charges its users on a per hour basis and pay-as-you-go basis. Similarly, Azure also charges its users on a pay-as-you-go basis but the key difference here is that instead of charging per hour, they charge per minute. Customers who utilize AWS with their products can save more money the more they use their services due to volume-based discounts. Azure on the other hand is a little less adaptable and offers their customers either monthly charges or prepaid charges (22).

8 DISCUSSION

The experience gained from developing this fullstack hybrid web application has been invaluable. The topic of an automotive repair business was an interesting one and a hybrid application as the number requirement is what caught my attention. I got the opportunity to utilize my experience as a software developer, and at the same time also learned new things which include using new frameworks. The skills earned from learning these frameworks will help with my career as a software developer and will be seen as a plus on my resume when seen by the companies. Since I used these new frameworks, it required some time to learn them and implement them in my project. This also caused some bugs and broken features that I fixed using documentations and Internet resources effectively.

The task was challenging but I enjoyed the progress I made in it. It was rewarding to see everything come together and work after finding out that a missing dot caused the frontend of the application to break. My testing skills improved during the development as every new feature that I attempted to integrate somehow caused other features to behave improperly. Testing the code with each change made the development process a little smoother and saved big amounts of frustration. Although not currently discussed, the deployment of the application would most likely be done on a cloud service. Including the storage for pictures of new vehicle entries if NAS system becomes infeasible. Maintaining the NAS system will require a lot of attention due to the data being stored, as well as any other technical issues, which is why a cloud service will be easier to deal with.

REFERENCES

1. DC Kumawat. Hybrid Apps Vs Native Apps for Mobile Devices. Search Date: 23.07.2023. https://www.orioninfosolutions.com/blog/hybrid-apps-vs-native-apps-for-mobile-devices
2. Denis Eleskovic. A comparison of Hybrid and Progressive Web Applications for the Android platform. Search Date: 16.10.2023. https://www.diva-portal.org/smash/get/diva2:1530474/FULLTEXT01.pdf
3. Taryn McMillan. A History of Version Control. Search Date: 23.07.2023. https://blog.taryn-mcmillan.com/a-history-of-version-control
4. AWS. What is an API? (Application Programming Interface). Search Date: 24.07.2023 https://aws.amazon.com/what-is/api/
5. Axita Digital Labs. What are APIs and the applications that utilize it? Search Date: 27.07.2023. https://www.axiatadigitallabs.com/what-are-an-api-and-the-applications-that-utilize-it/
6. Nathan Hekman. IBM Technology. What is a REST API? Search Date: 26.07.2023. https://www.youtube.com/watch?v=lsMQRaeKNDk
7. Hillary Nyakundi. OOP Meaning – What is Object-Oriented Programming? Search Date: 05.08.2023. https://www.freecodecamp.org/news/what-is-object-oriented-programming/
8. Chris Griffith. What is Hybrid Mobile App Development? Search Date: 06.08.2023. https://ionic.io/resources/articles/what-is-hybrid-app-development
9. Oracle. What is a Database? Search Date: 09.08.2023. https://www.oracle.com/database/what-is-database/
10. Ionic. Introduction to Ionic. Search Date: 10.08.2023. https://ionicframework.com/docs
11. TanStack Query^{v4}. Overview. Search Date: 17.08.2023. https://tanstack.com/query/latest/docs/react/overview
12. Codecademy. Tab Views and Labels. Search Date: 13.08.2023. https://www.codecademy.com/article/tabviews-and-labels-swiftui#heading
13. Coursera. What Is Responsive Web Design? And How to Get Started. Search Date: 16.08.2023. https://www.coursera.org/articles/responsive-web-design
14. Roadmaps. What is Backend Development? Search Date: 19.08.2023. https://roadmap.sh/backend
15. ExpressJS. Guide. Search Date: 20.08.2023. https://expressjs.com/en/guide/routing.html

16. Auth0. Introduction to JSON Web Tokens. Search Date: 21.08.2023. https://jwt.io/introduction
17. Zod. Introduction. Search Date: 03.09.2023. https://zod.dev/?id=introduction
18. Multer. Search Date: 04.09.2023
19. Richard Peterson. 13 BEST Free Database Software (SQL Databases List) in 2023. Search Date: 06.09.2023 https://www.guru99.com/free-database-software.html
20. pgAdmin. What is pgAdmin 4? Search Date: 14.09.2023 https://www.pgadmin.org/faq/#:~:text=pgAdmin%20is%20a%20management%20tool,the%20Features%20and%20Screenshots%20pages
21. NestJS. Prisma. Search Date: 17.09.2023. https://docs.nestjs.com/recipes/prisma
22. AWS vs Azure-Who is the big winner in the cloud war? Search Date: 14.10.2023. https://www.projectpro.io/article/aws-vs-azure-who-is-the-big-winner-in-the-cloud-war/401