**INTRODUCTION**

This project focuses on developing a task management application using React Native with Expo. The app will allow users to add new tasks, view their tasks, update, or delete them, helping them stay organized and on top of their daily activities. React Native is a great choice because it lets developers build apps that work on both Android, iOS and WEB using the same code. With Expo, the process becomes even easier, providing tools to quickly test, debug, and deploy the app. React Native gives an opportunity to write an application using React and JavaScript while making it ideal for quickly building working products, especially when speed and simplicity are priorities. Since I already have experience with React, using React Native felt like a natural choice. It combines the performance of native apps with the flexibility of JavaScript and web technologies, making it ideal for rapid development. Moreover, it lets the developers provide the styles for each WEB, Android as well as IOS separately if necessary. Expo eliminates the need to install Android or IOS emulator since we can be connected to the same network i.e. WI-FI to see the application’s behavior from own smartphone.

**Executive summary.** The project involves the usage of React Native with Expo which is the one of the most popular technologies in modern mobile development for building the user interface and taking care about cross-platform compatibility. React Native is used for managing navigation between different screens, while React hooks (not React Native) deal with the app’s dynamic state changes. Expo simplifies the testing and debugging as we don’t need to install the emulator but just install the application on your own smartphone.

**Project aim.** The project’s aim is to create a simple task management application that allows users to create, retrieve and view, update as well as delete their own tasks. The focus is to develop a project quickly, implement a user-friendly interface to work with the app.

**Project outcome.** The project contains several outcomes such as a working task management app where users can easily manage their tasks (adding a new task, viewing existing tasks, editing them along with deleting them). The integration of Expo into the projects gives a possibility to an application performs perfectly across multiple platforms. As we use React based approach the application can reuse plenty of similar blocks of designed templates that make the code clean and readable.

**Project objectives.** The primary objective of the project is to develop a cross-platform task management application using React Native with Expo which includes:

1. Building a functional UI where users can add, view, update and delete the tasks.

2. Using React Navigation we can smoothly navigate between pages.

3. Creating reusable components to maintain our code clean, beautiful and readable for other developers.

It means that the project demonstrates powerful and flexible approaches by using just one library framework for mobile applications for cross-platform apps.

**1. Overview of React Native development and TypeScript**

Before the beginning of the lab’s process pay attention that username ‘nekofetishist’ belongs to Maratuly Temirbolat (ID 23MD0409). Figure 1.1 illustrates

who is the owner of the account which will be used to cover the work.

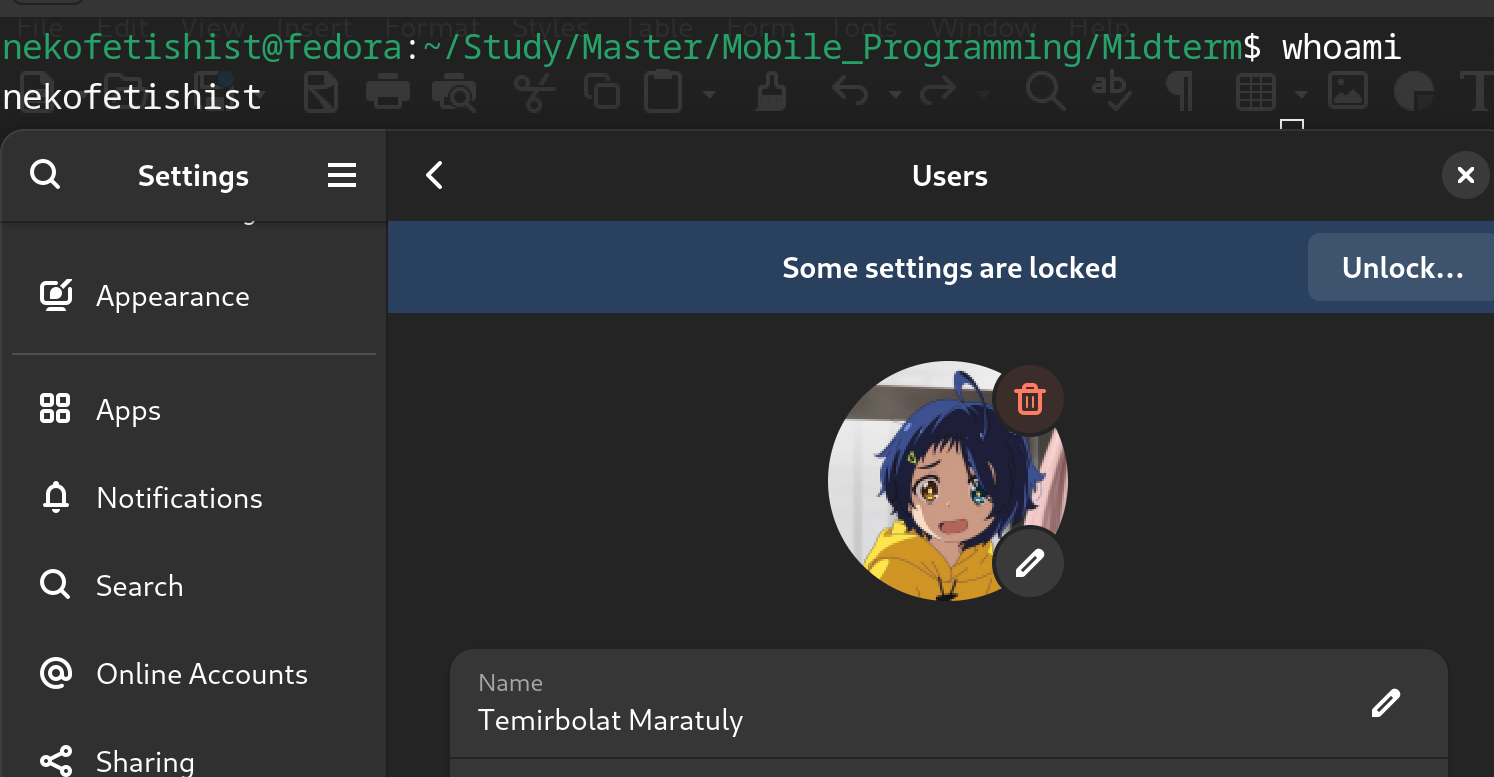


Figure 1.1 - Username and account’s owner credentials information

Thus, according to a figure, the command ‘whoami’ from the Ubuntu command line shows the current user, moreover, this username is used in the whole path (marked with green letters).

Now, let’s look at the project’s setup. The project was installed using expo. Expo is a production-grade React Native Framework. It provides developer tooling that makes developing apps easier, such as file-based routing, a standard library of native modules and much more. Another advantage that it is free and open source, it lets the developer to run an application in a terminal, then if a developer does not have an installed emulator for android or iOS, the generated QR code could be scanned and been connected to the same network we can open the started project in expo mobile application.

To create a sample application using export run the command “npx create-expo-app@latest”. The crucial factor that by default expo or standard react native use TypeScript to generate a code which is also a great approach as we use data types implicitly. Additionally, to use emulators, we need to install Android Studio for android devices and Xcode for iOS devices respectively. Let’s now examine the generated working tree of a project which is illustrated in Figure 1.2.

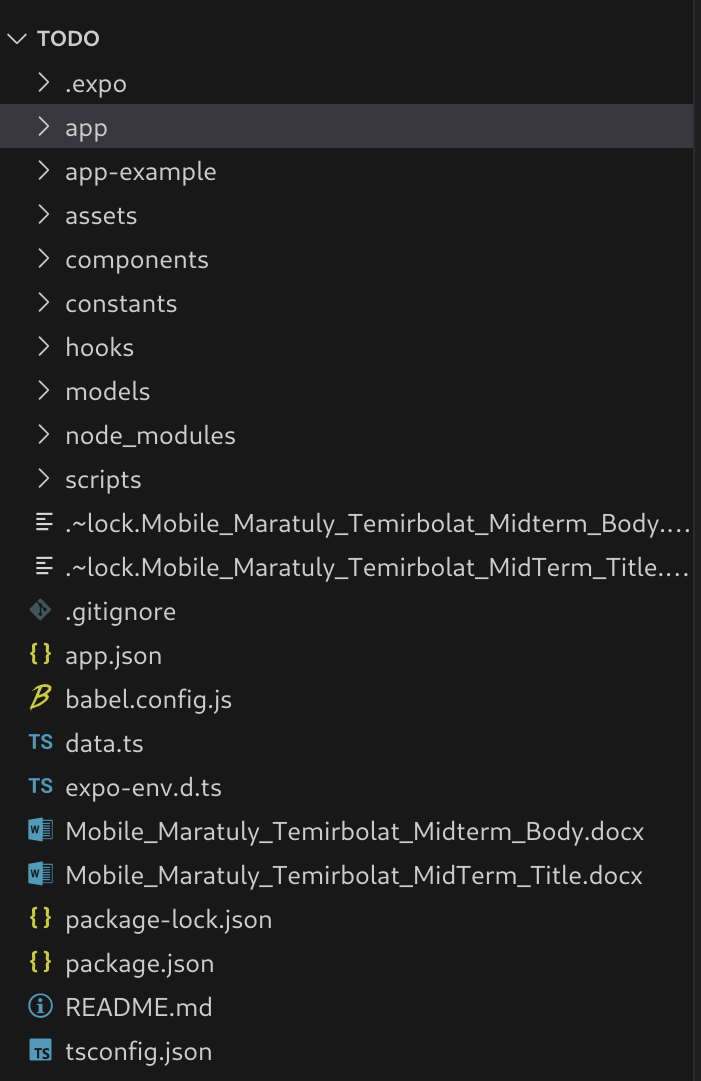


Figure 1.2 - Working tree of the TODO project

This React Native project, built with Expo, contains several important folders. The "app/" directory holds the code for screens and navigation, while "app-example/" is generated by default with Expo and provides example implementations. Reusable UI components, such as buttons or headers, are located in \*\*"components/". The "assets/" folder stores static files like images and fonts. "hooks/" contains custom React logic for managing state and behavior; it is also generated by default with Expo as a starting point for developers. "models/" defines data structures used across the app, "utils/" organizes common utility functions, and "scripts/" holds automation tools or setup scripts, another default folder provided by Expo for demonstration purposes. React native, by default, is currently written completely with TypeScript, so, all the files here with “.ts” or “.tsx” extension. TypeScript is not a different programming language, it is just a compiler or a wrapper for a JavaScript to add data typing. Data typing is a great approach for developing applications since you, at least, a code editor like VS Code helps a developer by annotating available fields, methods, functions, which parameters are expected and so on.

Several key configuration files are included in the project. "app.json" defines the app’s configuration, and "package.json" lists its dependencies. "babel.config.js" allows the app to use modern JavaScript, and "tsconfig.json" provides settings for TypeScript. ".eslintrc.js" enforces code style rules. These elements, combined with Expo, make the development process streamlined and manageable. Then, each folder will be unwrapped to see its content along with the code, but not in this section as it would be unclear. To run an application in browser use the command: “npm run web”. You will obtain about the same results in Figure 1.3.



Figure 1.3 – Result of running web version of an application

The provided QR code is used here to open the application from your smartphones (but you must be in the same network). Additionally, web version is visible on the localhost with specified port (here 19002).

**2. Functions and Lambdas in TypeScript**

This section discuss functions and anonymous functions in TypeScript based on the project. Functions are commonly used in any programming language and TypeScript is not an exception. Functions are blocks of code that perform a specific task and can be **reused**. Let’s take a look at “data.ts” file which is located in the root of the project. TypeScript has opportunity to implement functions in several ways: named and anonymous. Anonymous will be discussed after names one. Figure 2.1 illustrates the project’s code related for updating, retrieving, creating tasks in the way of using named functions. We can set optional parameters for a function just by adding a question mark and TypeScript will validate it with a default value (null) but internally, so, if you provide something as a value it will substitute your value, otherwise null.

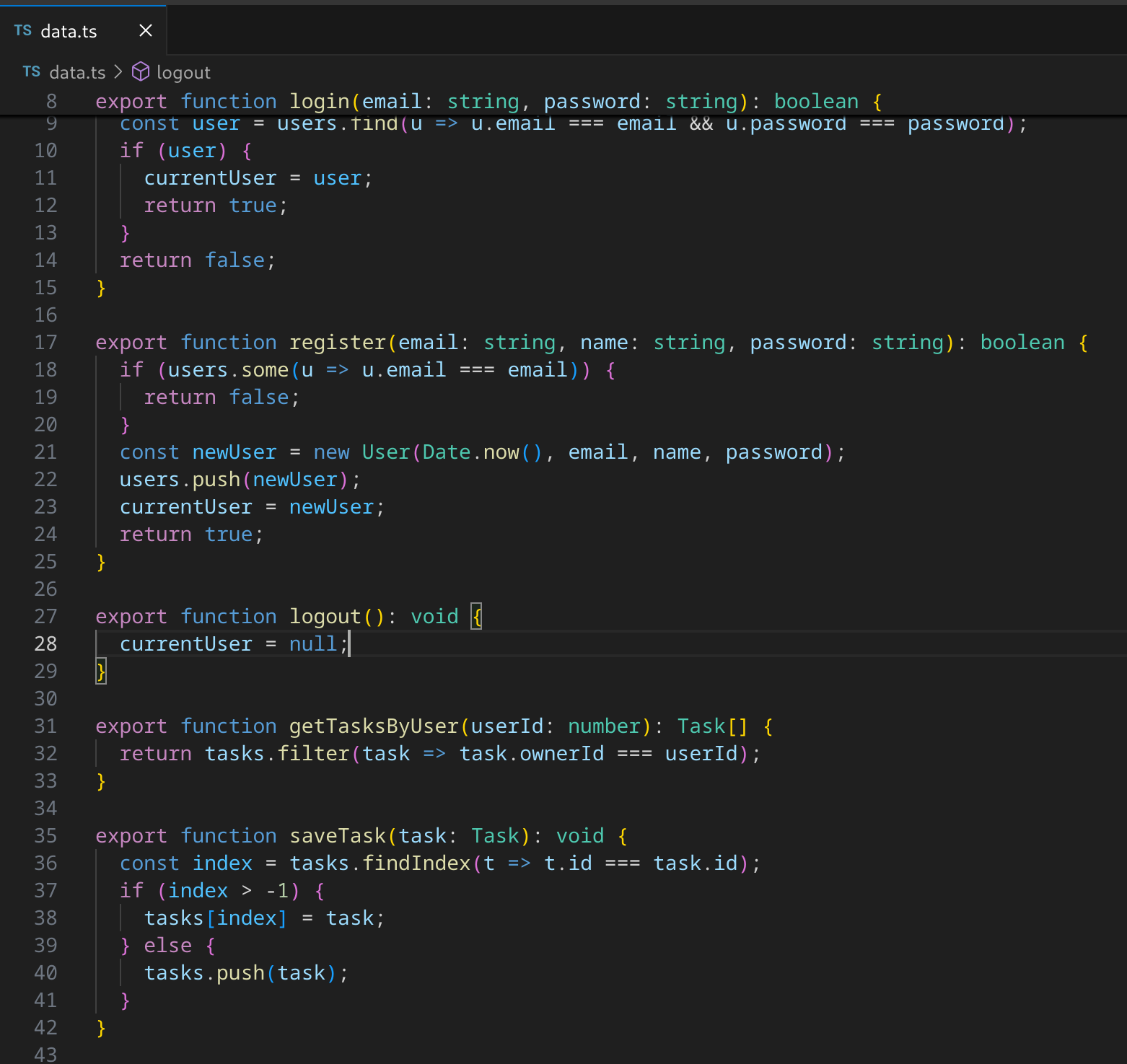


Figure 2.1 – Named functions related for updating, retrieving and creating tasks

According to a figure, all the functions has “export” as the beginning which means that we can import those functions in other files (reuse these functions). The next is a also a built-in syntax to initialize the function, followed it, the name of the function is present. TypeScript, JavaScript uses a camel case syntax for naming, so, each new word in a variables starts with a capital letter. Any function in TypeScript and JavaScript can take parameters and return something or not (void functions). However, TypeScript functions gives an opportunity to assign or tell a developer what are the data types of each input parameter and what is a data type a returned value from this function. For example, function “getTasksByUser” accepts “userId” which is a number and returns the array of tasks where a user with specified ID is the owner of a task.

Next, we will examine anonymous functions in TypeScript. In TypeScript, anonymous functions are functions that do not have a name associated with them. Anonymous functions are commonly used only as callback parameters which takes some parameters and be executed after some successful actions like getting a response after making a request, render some components for the UI and so on. The most popular approach it to use anonymous functions as arrow functions. Arrow functions are a shorthand way of writing anonymous functions. Figure 2.2 illustrates one of usages of anonymous (arrow type) functions in the project.

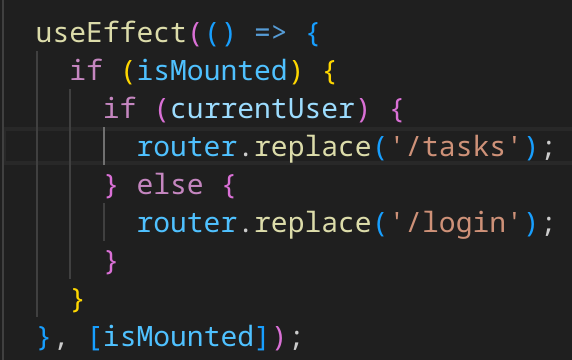


Figure 2.2 – Anonymous function (arrow type) example

The code above uses on of the React hooks “useEffect” where as the first parameter it takes a callback function would be executed as one of the variables in the list has been changed (array of parameters is present as a second parameter). So, This tells React to run the function inside the “useEffect” when the “isMounted” variable changes. If “isMounted” is true, it checks if current user exists and if the current user exists, it redirects the user to the “/tasks” page using “router.replace('/tasks')”. If the current user is missing (means null), it redirects the user to the “/login” page using “router.replace('/login')” command. Another example of using anonymous functions in the project is shown in Figure 2.3

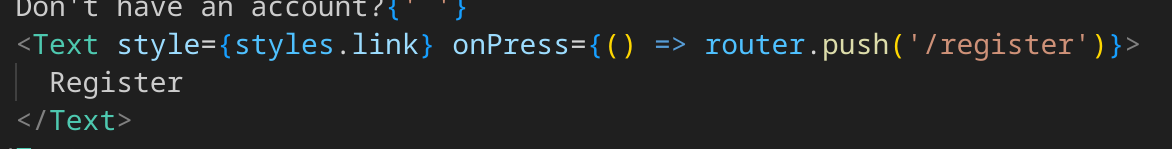


Figure 2.3 – Usage of an anonymous function as event handler

Here we used “Text” component to show “Register” in the layout. The usage of of callback functions here that we process clicking on this text. As we click (press) on this text the callback function triggers (here we navigate to a register page).

**3. Object-Oriented Programming in TypeScript**

This section is responsible for describing generally Object-Oriented Programming, how it is implemented in TypeScript and used in the project. Object-Oriented Programming or shortly OOP is just a programming paradigm that aims to implement real-world entities using classes. OOP has 4 main principles: inheritance, abstraction, polymorphism and encapsulation. We will shortly describe each of them but only as we use in the project. All the related to an OOP concept classes are located in the folder “models” which is in the root of the project (Figure 3.1)

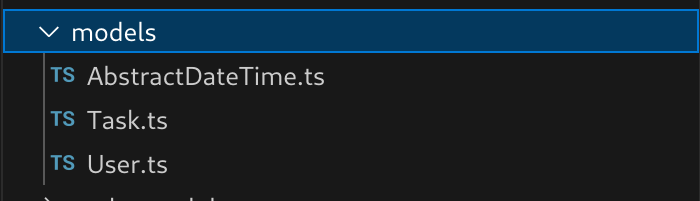


Figure 3.1 – Location of the models in the project

We can clearly see that we have only 3 files related to the models. There are: Task, User and Abstract base models. Let’s take a look at each of them. Figure 3.2 demonstrates “AbstractDateTime” model.

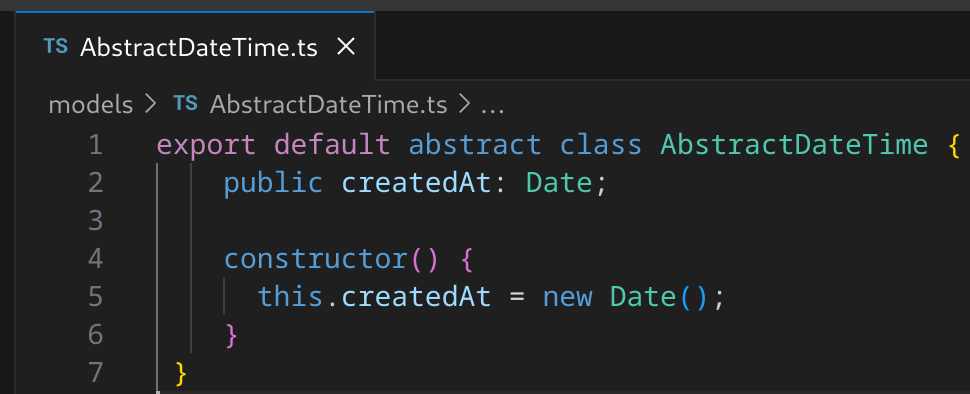


Figure 3.2 – Abstract date time class model

Here we define an abstract class “AbstractDateTime”. This class is exported as the default export, meaning it can be imported without curly braces from another module. Here we discuss abstraction. Abstraction in OOP refers to hiding the internal details and only what the project has as fields and what this object can do or how behaves. Basically, abstract classes can contain both implemented methods and abstract methods (which must be implemented by the child classes). You cannot create an instance of an abstract class directly. Instead, other classes extend it, and those child classes must implement any abstract methods defined in the base class. To make a class abstract, you need to use a key word “abstract”.

Encapsulation in OOP is the practice of restricting access to certain parts of an object to protect the internal state. In TypeScript, access modifiers like “public”, “private”, and “protected” are used to control the visibility of class properties and methods. The modifiers mean the following: “public” (access from anywhere), “protected” (access within the class and its child classes) and “private” (only access within the class where it is defined). Here, “createdAt” property is marked as “public” with Date data type. Let’s now consider inheritance in scope of our project. Figure 3.3 demonstrates the realization of “Task.ts” model.

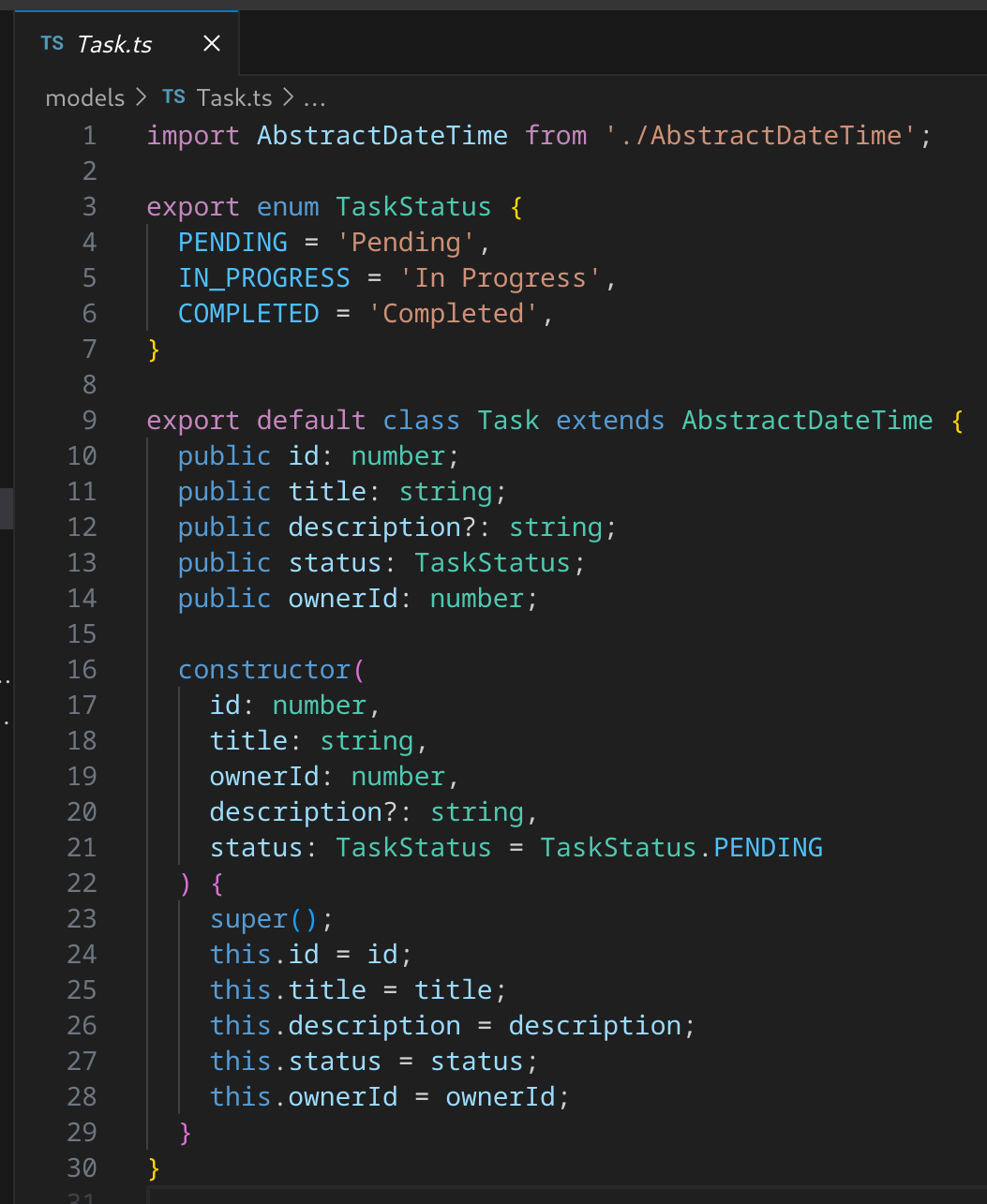


Figure 3.3 – Realization of “Task.ts” file

According to the figure above we can see that define Task class in TypeScript. The Task class represents a task entity with several properties such as an ID (number), title (string), description (optional string), status (enum), and owner (number). The Task class inherits from the “AbstractDateTime” class, meaning every Task object will have a “createdAt” property, which is initialized to the current date and time. This promotes code reuse since the date time logic is abstracted in the parent class. In case of the encapsulation, the properties of the Task class are marked as public, meaning they can be accessed from anywhere.

**4. Working with collections in TypeScript**

**5. Android Layout Design**

**6. User Input Handling**

**7. Components**

**8. Data handling**

Data handling is more about getting, posting and storing data inside of the application or on a remote server. Generally, if we consider the requests to any API we use “fetch”, “axios” or any other third-party library. These mentioned functions are used in React applications or any JavaScript related code since React Native is still a react family solution. In the current project we simulated data uploading and fetching by creating local lists of constants where we store all the data. It doesn’t mean that we cheated somehow, no, since the logic of the code maintains the same. Figure 8.1 represents the location of the “data.ts” file containing all the constants.

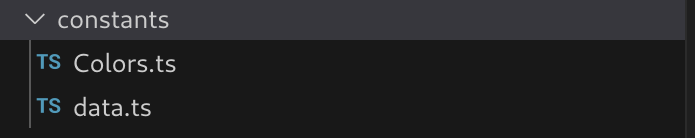


Figure 8.1 – Location of the project constants

If we look at the “data.ts” file we can see that we have no initial data about users, tasks and so on. We add them in the code from the Figure 2.1 by importing and calling corresponded functions. Figure 8.2 proves that there are no initialized users in the project before its building.

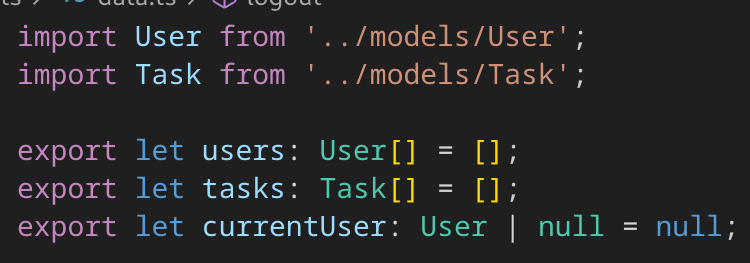


Figure 8.2 – Initial data of users, tasks and current user

So, as it was mentioned we have no initial tasks, users in the project, we create them manually.

**CONCLUSION**

To sum up, we managed to create a task management application successfully using ...