A picture containing text

Description automatically generated

|  |  |
| --- | --- |
| Module code and name: | Semester Two 5CSOSC017-n – Web Applications Development |
| Assessment: | Report |
| Student ID: | 00013712 |
| Submission deadline: | 3/20/2024 |
| Wordcount | 2375 |
| Results date and type of feedback |  |
| Result (Pass or Fail) |  |
| **Feedback** | |
|  | |

Table of Contents

[Student ID calculation 3](#_Toc161930521)

[Development Plan 3](#_Toc161930522)

[MoSCoW 3](#_Toc161930523)

[Design Shema 5](#_Toc161930524)

[GitHub 8](#_Toc161930525)

[Public Github Repository link: https://github.com/00013712/WebApplicationCW 8](#_Toc161930526)

[Github files 8](#_Toc161930527)

[System Architecture 10](#_Toc161930528)

[Software Pattern: 10](#_Toc161930529)

[API (back end) 13](#_Toc161930531)

[SPA (front end) 19](#_Toc161930536)

[KeyStoreAppItems: 19](#_Toc161930537)

[Key-store-app-service: 20](#_Toc161930538)

[Home.component: 22](#_Toc161930540)

[Details.component: 24](#_Toc161930542)

[Delete.component: 25](#_Toc161930543)

[Create.component: 26](#_Toc161930545)

[“Link to the Demo video” (YouTube): 27](#_Toc161930546)

[Reference list: 27](#_Toc161930547)

# Student ID calculation

I followed the rules for the homework to figure out what topic to write about for my application.

My student Id number is 00013712, so I divided my student Id number by 20 and got 685.6 or 685 12/20, since we are interested in the remainder, which is 12 in my case. The calculation showed that the remainder was 12. This has to do with the Key Store Application topic, as shown in the table. For this reason, I've decided to make the Key Store Application as my portfolio project. Key Store Application Overview: The Key Store Application is made to keep different kinds of keys, passwords, and private data safe and easy to access. The key store has items that users can add, view, change, and remove.

# Development Plan

## MoSCoW

**Must Have:**

1. Set up a method for authentication and permission.
2. Make CRUD features for handling things in the key store.
3. Make an interface that is easy for people to use to add, view, change, and delete things.
4. Make sure data is safe by adding encryption and decoding functions.
5. Add a search tool so that users can quickly find the things they've saved.

**Should Have:**

1. Use flexible design to make sure that your site works on all devices.
2. To improve the accuracy of the data, add validation checks to the input areas.
3. Sorting and filtering choices will help you organize your key store things better.
4. Add ways to handle errors so that users can get notice when something goes wrong.
5. Use a password strength indicator to help you make strong passwords.  
     
   **Could Have:**
6. For extra protection, use two-factor verification.
7. Integrate with external password managers to make importing and exporting passwords easy.
8. Set up user accounts so that the application can be customized and made more personal.
9. Support more than one language to serve a wide range of users.
10. Set up ways to back up and recover files automatically.

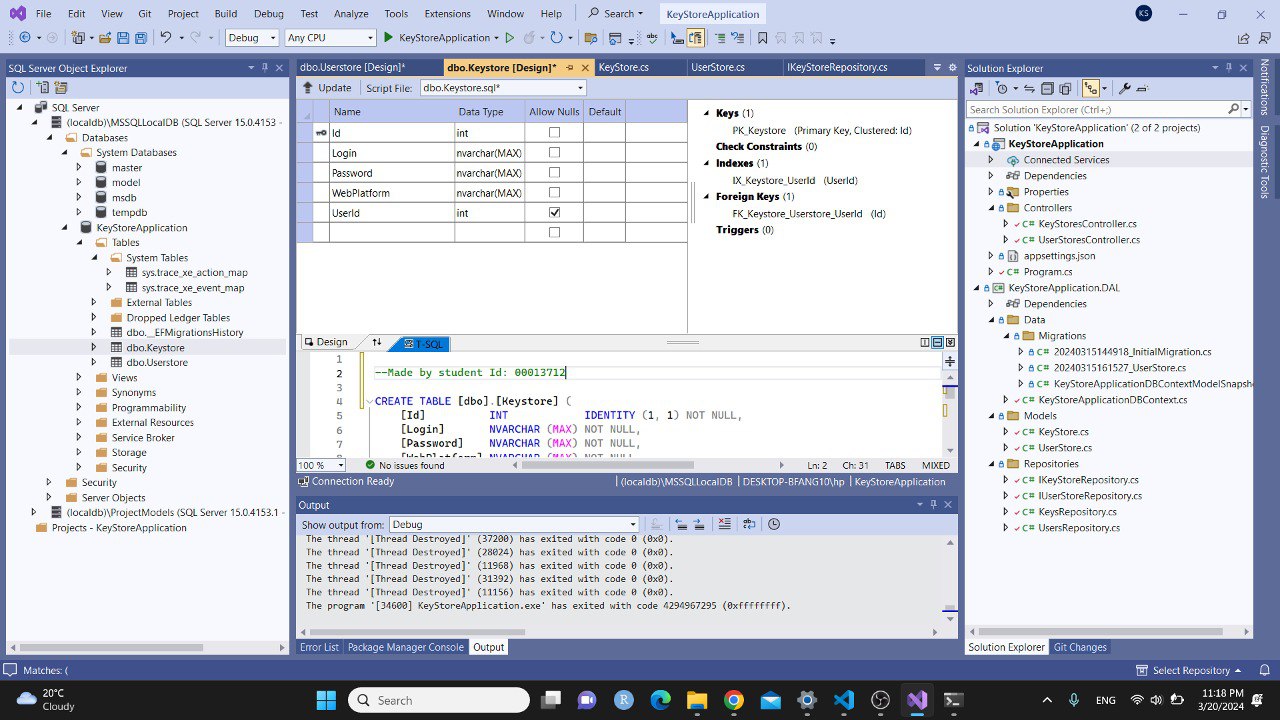
**Won’t Have:**

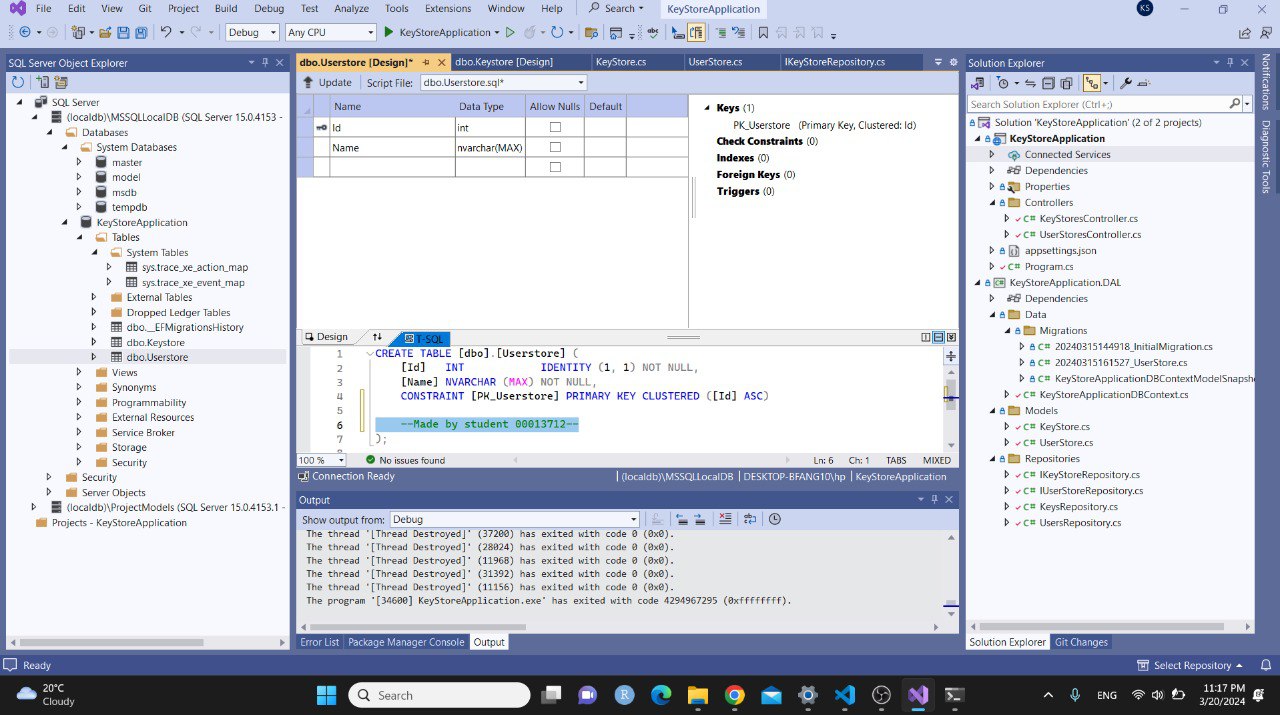
1. For more advanced protection, hardware security units can be added.
2. For extra protection, integration with biometric authentication devices is recommended.
3. Integration with cloud storage companies lets data be automatically backed up and synced.
4. Integration with browser add-ons lets you get to saved things quickly.
5. Connecting to reporting and logging tools lets you keep track of what users do.

## Design Shema



id (Primary Key): A unique number that helps you find each thing in the key store.  
login: This field stores the username or email address that you use to log in to the web platform.  
password: This setting saves the login's password.  
webplatform: This field stores the web platform's name or URL.  
Each key store item is linked to a single person by its userId (a foreign key that references the User Store table.  
id (Primary Key): A unique number that is assigned to each person.  
username: This is the username that the person decided to use for registration.





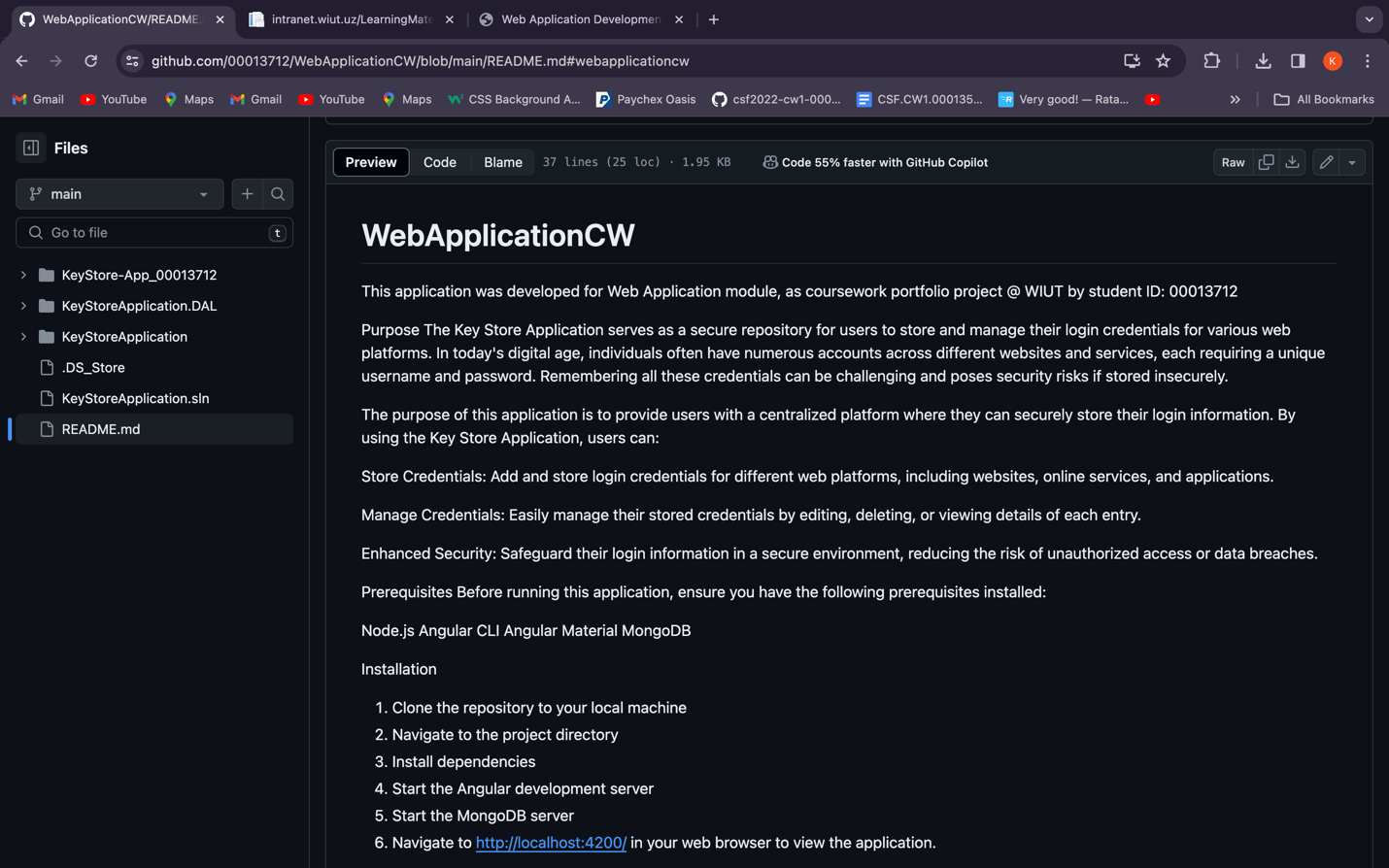
The userId foreign key in the KeyStoreItems table identifies a specific user and each row in the table stores a login password for a certain web site. The Users table holds information about users, such as their chosen username and a unique number (id). User can connect the saved things to the people who own them by using a foreign key link between the KeyStoreItems and people tables. This lets data be properly organized and retrieved based on who owns it. Overall, this schema design gives you a basic framework for your key store app, which lets users safely store and handle their login information for different websites.

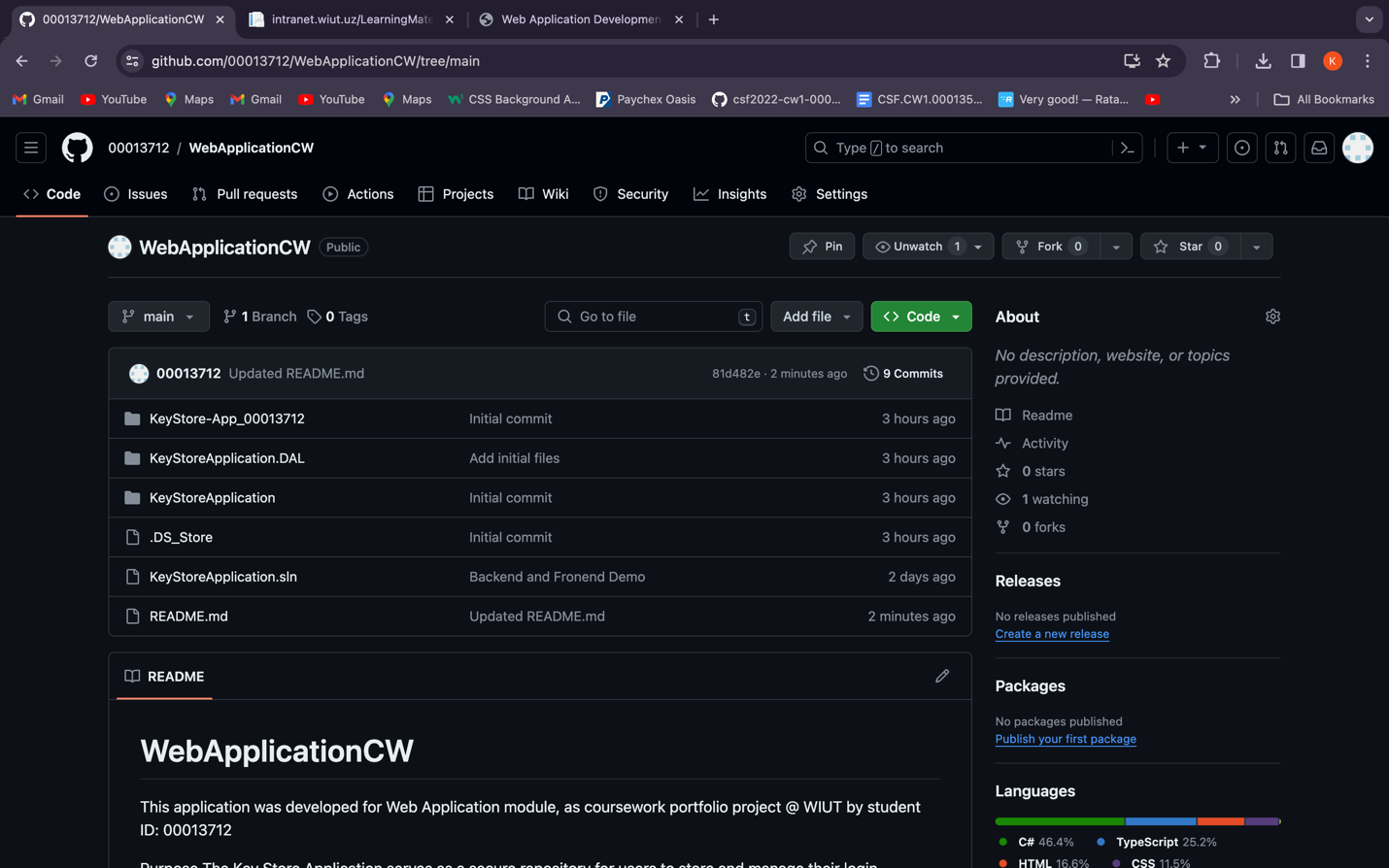
# GitHub

## Public Github Repository link: <https://github.com/00013712/WebApplicationCW>

## Github files

Screenshots:



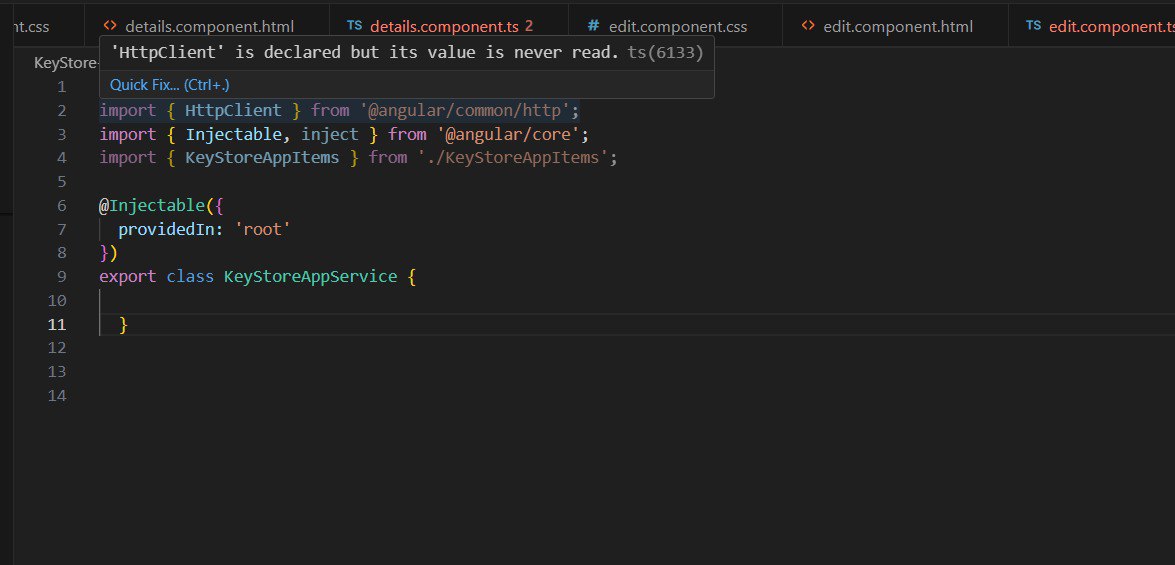


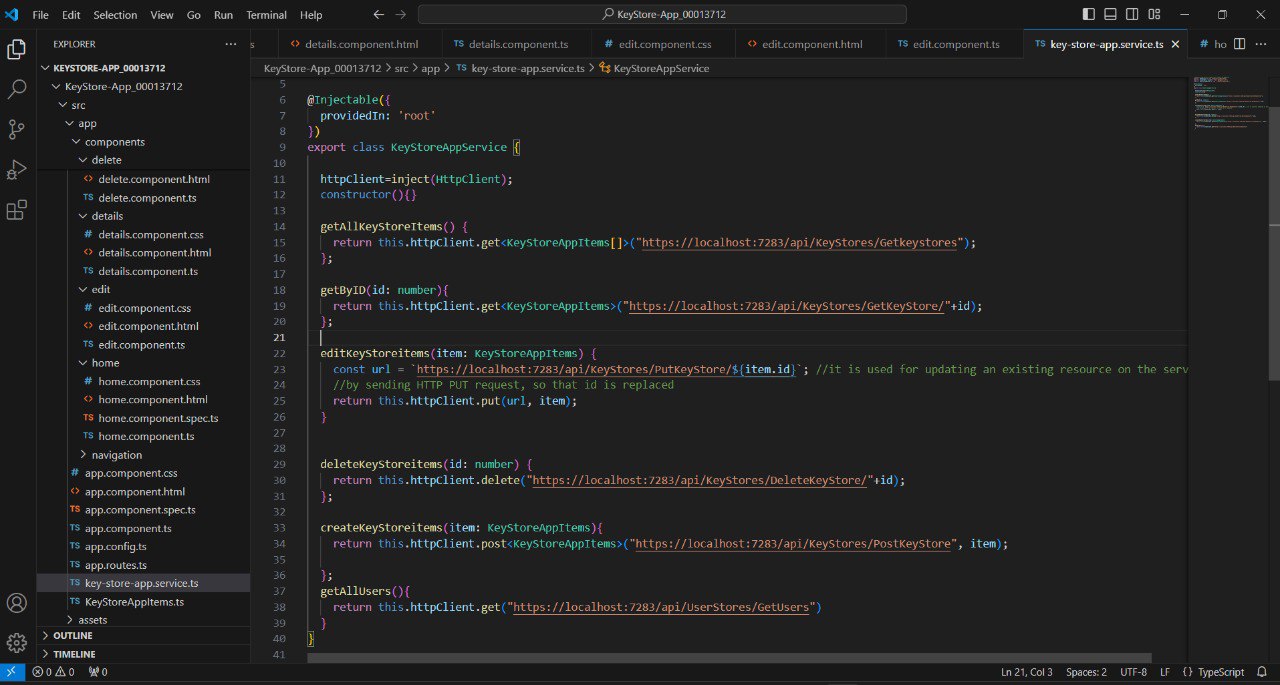
KeyStore-App\_00013712:  
The root directory of the project typically contains Angular frontend application files, including components, services, and HTML templates. The directory's "\_00013712" suffix identifies the student's ID. KeyStoreApplication.DAL: This directory may include Key Store Application Data Access Layer (DAL) files. It may include database models, repositories, or C# backend data access functionality. KeyStoreApplication: The Key Store Application server-side implementation files may be in this backend application directory. This may include backend logic, API controllers, and configuration files.DS\_Store: This macOS-specific file holds folder information like icon position and view settings. It's usually generated by the OS and not necessary for the project.  
KeyStoreApplication.sln:  
  
This is probably the backend solution file. Visual Studio uses.sln files to manage software solution projects like libraries and executables. README.md: The README markdown file describes the project's aim, installation, and use. Project users and contributors utilize it as documentation.

# System Architecture

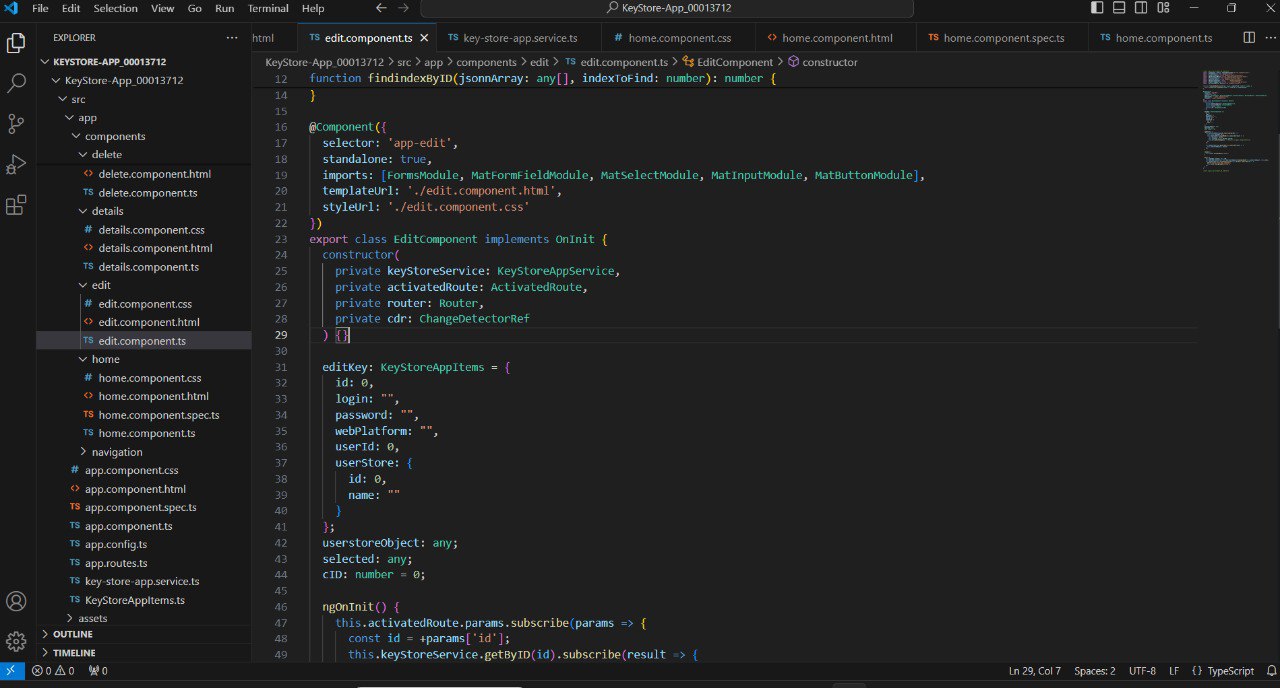
## Software Pattern:

Key Store Application uses the Singleton design pattern to manage the KeyStoreAppService instance. Singleton guarantees that just one instance of the service is generated during the application's lifespan, enabling centralized access to critical store elements. Effect on Project Overall: Code Organization: The Singleton approach makes the KeyStoreAppService instance universally available across components, encouraging code organization and decreasing service instantiation redundancy. Resource Efficiency: Singleton blocks superfluous service object creation, optimizing memory allocation and performance. Maintains data consistency and prevents discrepancies from different service instances by ensuring all components interface with the same service





# 



# API (back end)

# 

# 

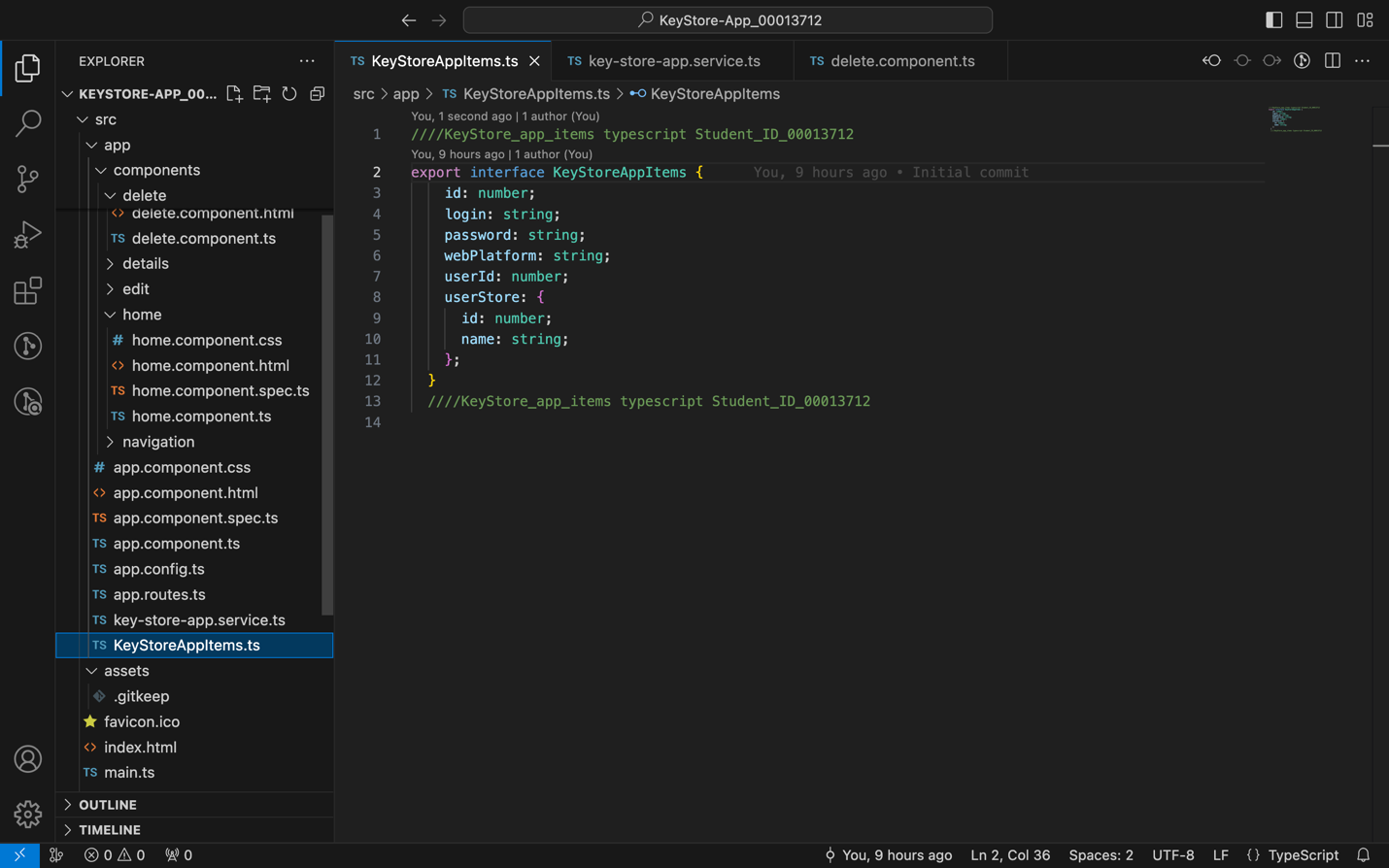
# 

## 

# SPA (front end)

KeyStoreAppItems: What is an interface? The front end KeyStoreAppItems tells the program how to organize key store things. It tells each item object what traits it should have.

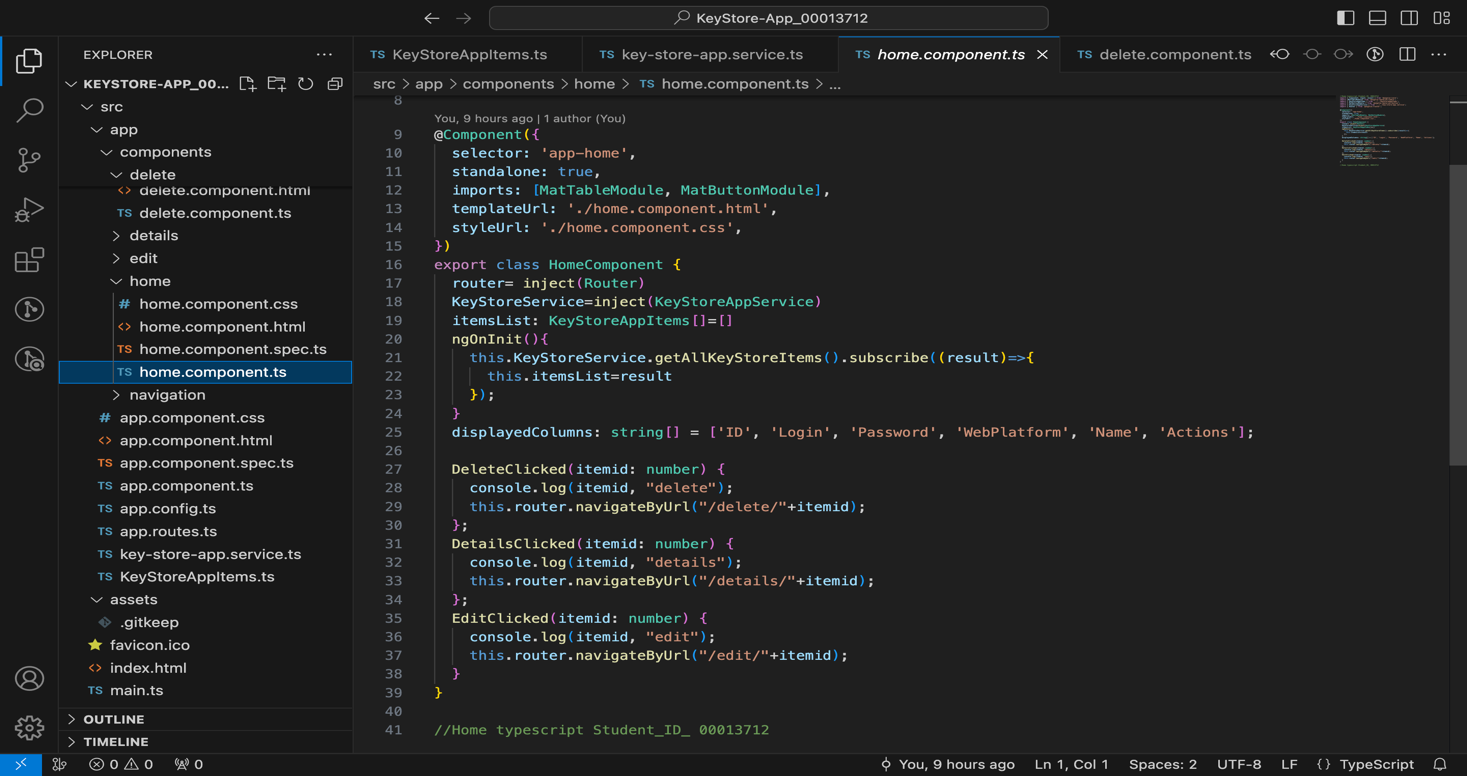
**Properties:** The following properties are part of the interface: **id:** This is the key store item's unique identity. **login:** This field holds the login information for the item. **Password:** This is the password that is linked to the thing. **webPlatform:** This is the name of the web platform or service that is linked to the thing. **userId:** This is the unique number of the person who owns the thing.  
**userStore:** This is more information about the person who owns the thing. It has features like name and id. By setting up this interface, the code makes sure that working with key store things across the whole program is consistent and type safe. It makes it possible for TypeScript to check types, offer auto-completion, and support type inference in the source. It also helps keep the code clean and easy to understand by making the layout of key store items clear.



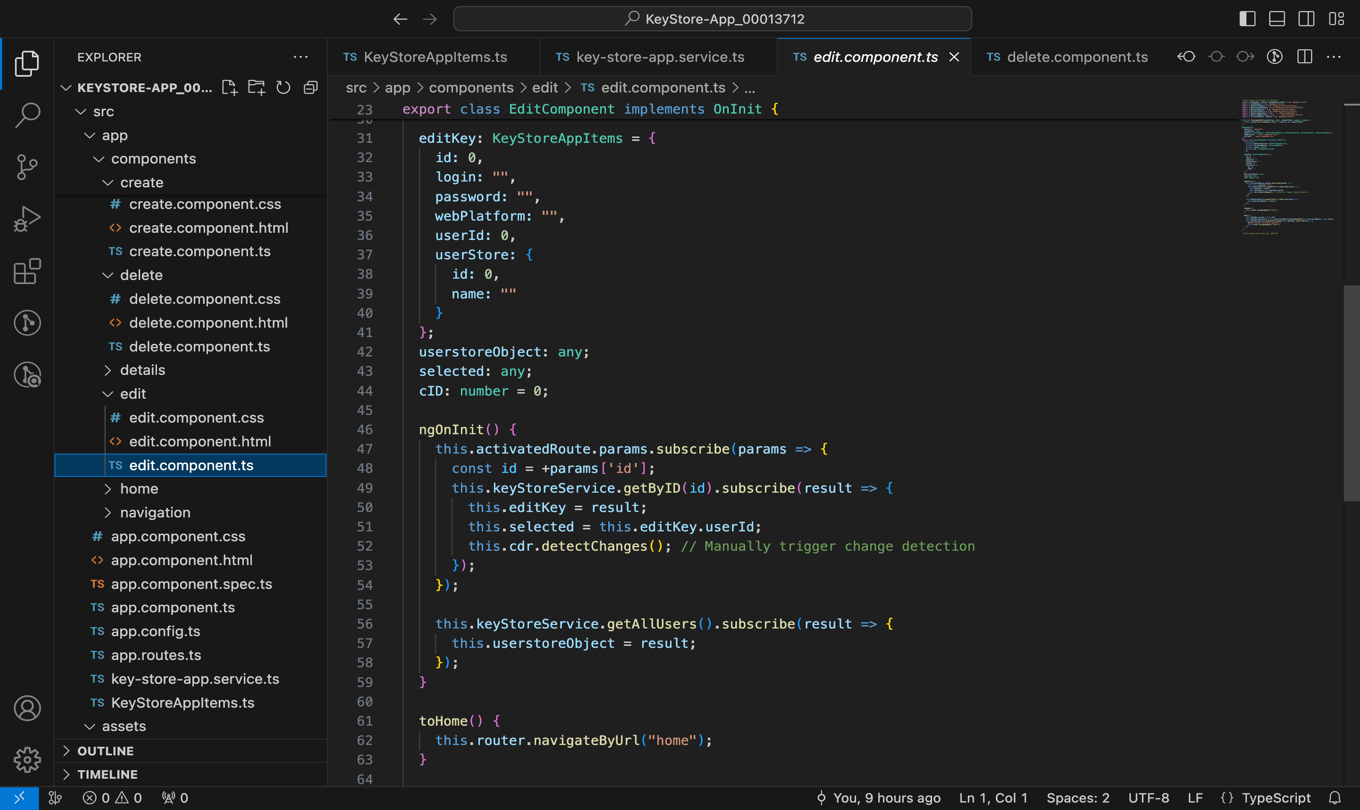
Key-store-app-service: The getAllKeyStoreItems() method makes a GET call to the server to get all the things in the key store. It wants a collection of KeyStoreAppItems as an answer, which is a list of all the key store things in the database. **GetByID(),** If you give this method a number as an argument, it will send an HTTP GET request to the server to get a specific key store item based on its ID. To use it, you need to give it the item's ID and expect a single KeyStoreAppItems object in return. **EditKeyStoreAppItems(),** is the thing that you want to change. This method makes an HTTP PUT request to the server to change a key store item that is already there. It takes as a parameter a KeyStoreAppItems object that represents the thing that has been changed. Based on the ID of the item that was sent, the server-side address should handle the change process. If you call **deleteKeyStoreitems(id: number),** it will send an HTTP DELETE request to the server to get rid of a key store item with the given ID. It needs to know the ID of the thing that needs to be deleted. The item will be deleted from the database by the server-side link. When you call **createKeyStoreitems(item: KeyStoreAppItems)**: The HTTP POST request to make a new key store item on the server is sent by this code. It takes a KeyStoreAppItems object as an input, which is the new item. The server-side address should oversee adding the item to the database. Use **getAllUsers():** To get a list of all people from the server, this method makes an HTTP GET request. In the app, it's used to fill in the choice or list that lets the user choose. It is assumed that the server's answer will have a list of user items in it. Ultimately, these methods give the user the tools you need to manage key store items and people within the app by interacting with the backend API URLs. They hold the code for CRUD tasks and make it easier for the frontend and backend of the program to talk to each other.

## 

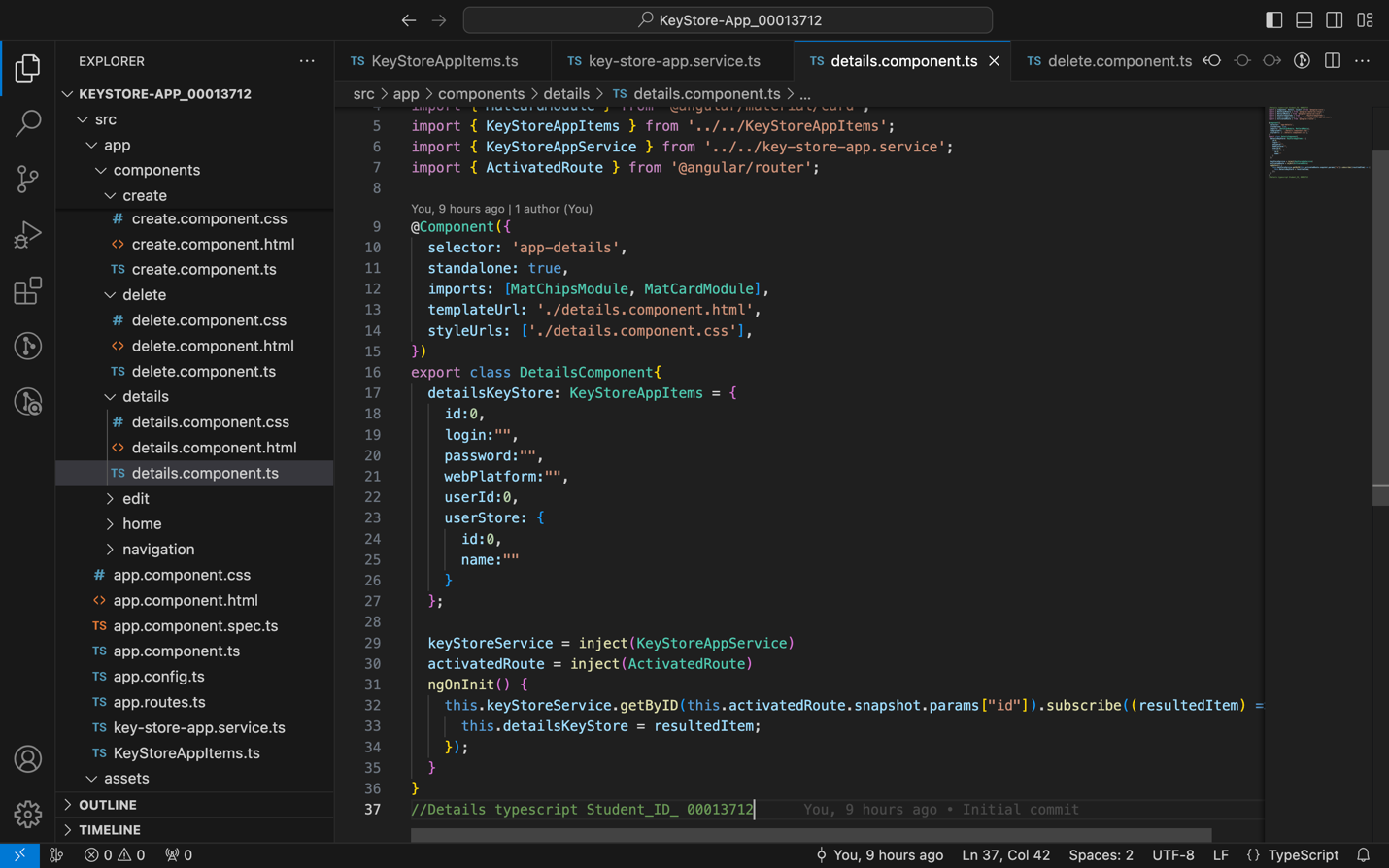
Home.component: It is the function of the HomeComponent to be the main view for the key store app. It gives people an easy-to-use tool for seeing a list of database-stored key store things. Users can view more information about an item, change its features, or delete it from the list, among other things. Using the **ngOnInit()** lifecycle hook, the component gets a list of key store items from the server service when it is first set up. It shows the things from the key store that were received in the UI as a table. The component lets you navigate for several different actions: When **DeleteClicked(itemid: number)** is called, the page to delete the chosen key store item identified by itemid is shown. **DetailsClicked(itemid: number):** Goes to the details page for the itemid of the chosen key store item. **EditClicked(itemid: number):** Goes to the change page for the itemid of the key store item that was clicked on. The HomeComponent is the main way that users interact with the key store application. It shows key store things and lets users move between them to manage them. It does what it's supposed to do by getting data from a server service, giving navigation routes, and using Angular's dependency injection to get to the services it needs.



## Edit.component: **Initialization (ngOnInit()):** The component subscribes to route parameters to extract the key store item ID to modify when it initializes. It then uses keyStoreService's **getByID()** function to get the item's data. It also obtains all user store objects using KeyStore Service’s **getAllUsers()** function. It fills the e**ditKey** object with item information from the specified key store item and all user store objects. It also assigns the modified item's user ID to the variable. Navigation: **ToHome()** uses the router to return the user to the application's home page. Method navigateByUrl. Submitting the modified key store item calls edit(). It assigns the specified user ID (**cID**) to edit Key’s userId attribute. The editKey's userStore field receives the associated user store object. Finally, it updates the backend database with the changed item using keyStoreService's editKeyStoreitems() function. An alert confirms changes and returns the user to the home page after editing. Basically, this code section edits a crucial store object in the program. It obtains item details and user information, lets users modify items, and updates the backend database.



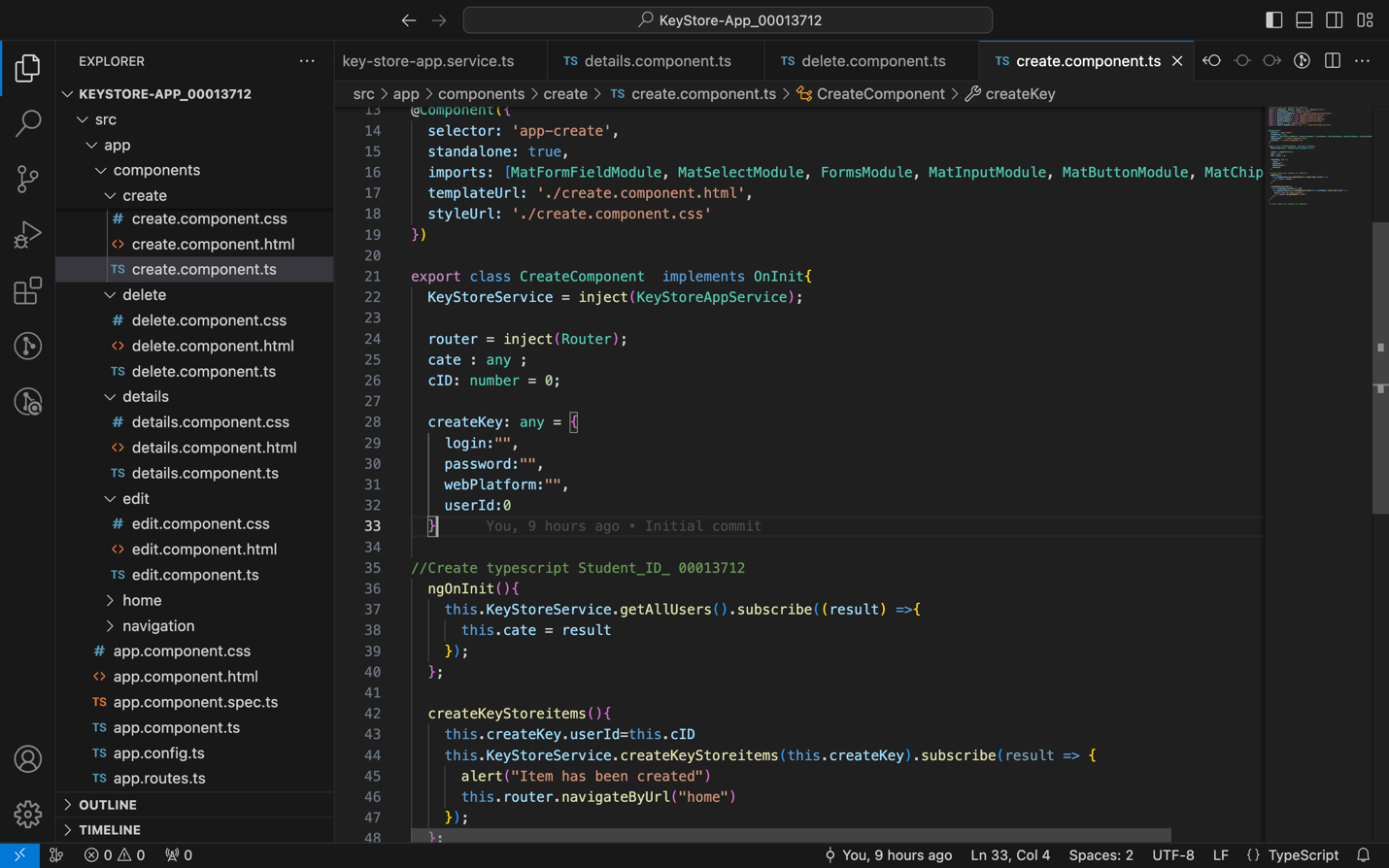
Details.component: **Initialization (ngOnInit):** The component takes the key store item ID from the active route parameters during initialization. It then invokes keyStoreService's getByID() function to get the item's data. Collected item information are stored in the detailsKeyStore object. detailsKeyStore Item: It represents the main store item whose details are shown. It has id, login, password, webPlatform, userId, and userStore. We start with all attributes set to default values (id and userId to 0, login, password, and webPlatform empty strings, and userStore empty object). Dependencies: To get data and route parameters, the component injects KeyStoreAppService and ActivatedRoute objects. In summary, this code creates a component (DetailsComponent) to get and show key store item information from the backend service (keyStoreService) using the route parameters' ID.



Delete.component: Deleting KeyStore: The important storage item to remove is this object. It has id, login, password, webPlatform, userId, and userStore. We start with all attributes set to default values (id and userId to 0, login, password, and webPlatform empty strings, and userStore empty object). KeyStoreAppService, ActivatedRoute, and Router objects are injected to interact with services, retrieve route parameters, and travel routes. Event Managers: onHomeButtonClick(): Returns to home page. Calls the service's deleteKeyStoreitems() function to remove the key store item with the provided ID. It alerts the user and returns to the home page after deletion. In summary, this code creates a component (DeleteComponent) to remove a key store item based on its route parameter ID. It deletes with the backend service (service) and navigates to the home page.

## 

Create.component: Dependencies: The component injects KeyStoreAppService and Router objects to communicate with services and browse routes. Properties: cate: Key store service user list. Selection user ID (cID). object createKey: Create a new key store item using this object. It has login, password, webPlatform, and userId. Initially, userId is 0, and other attributes are empty strings. Event Managers: Creates a new key store item by changing its userId field to the specified user ID (cID) and invoking KeyStoreService's createKeyStoreitems() function. It alerts the user and returns to the home page after creation. This code creates a key store item using CreateComponent. It retrieves the list of users from the backend service, lets the user to provide data for the new item, associates it with a chosen user, and calls the relevant service method to create it.



“Link to the Demo video” (YouTube): <https://youtu.be/mU4M34UoeCE>

Reference list: Hub, G. (2023). Build Software better, Together. *GitHub*. Available from https://github.com/topics/official-website [Accessed 21 March 2024].

Microsoft. (2024). Visual Studio Code. *Visualstudio.com*. Available from https://code.visualstudio.com/ [Accessed 16 March 2024].

microsoft. (2019). Downloads | IDE, Code, & Team Foundation Server | Visual Studio. *Visual Studio*. Available from https://visualstudio.microsoft.com/downloads/ [Accessed 16 March 2024].

OBS Project. (2012). Open Broadcaster Software | OBS. *Obsproject.com*. Available from https://obsproject.com/.

Package, N. (2023). NuGet Gallery | Packages. *www.nuget.org*. Available from https://www.nuget.org/packages.

Stack Overflow. (2008). Stack Overflow - Where Developers Learn, Share, & Build Careers. *Stack Overflow*. Available from https://stackoverflow.com/ [Accessed 19 March 2024].