**SET ASSIGNMENT 5**

Q 1. What is a version control system and why it is important?

A version control system (VCS) is a software tool used to track changes to files and directories over time. It allows developers to manage and keep track of changes made to code, documents, or any other type of file. VCS creates a historical record of changes made to a file or set of files, enabling users to access and restore any previous version of a file at any point in time.

Version control is important for several reasons. First, it allows multiple developers to work on the same codebase simultaneously without fear of overwriting each other's work. Second, it enables users to keep track of changes made to a project and identify who made those changes. Third, VCS enables users to revert to a previous version of a file or project if something goes wrong.

Moreover, version control provides a way to collaborate effectively with other team members or contributors. It facilitates code review, helps manage conflicts, and provides a way to track issues and bugs. Additionally, VCS enables users to experiment with new ideas without the risk of breaking the current version of the codebase.

Overall, version control is an essential tool for any project, big or small, to manage changes to files and collaborate effectively.

Q 2. Illustrate different types of version control systems with examples.

There are two main types of version control systems: centralized and distributed.

Centralized Version Control System (CVCS)

Centralized version control systems (CVCS) store the entire history of a project in a single central server. Developers check out a copy of the code from the central repository, make changes, and then check it back in. Examples of CVCS include Subversion (SVN) and Perforce.

EXAMPLE **- Subversion** (SVN) is a popular centralized version control system that stores the entire history of a project in a central repository. It uses a client-server architecture, where developers check out a copy of the code from the central repository, make changes, and then check it back in. SVN is commonly used in enterprise environments.

Distributed Version Control System (DVCS)

Distributed version control systems (DVCS) do not rely on a central repository. Each developer has a local repository on their machine, and changes can be made and committed locally. These changes can be shared with other developers by pushing them to a remote repository. Examples of DVCS include Git and Mercurial.

EXAMPLE - **Git** is the most widely used distributed version control system. It allows developers to work on the same codebase simultaneously and provides a way to manage changes to code. Each developer has a local repository on their machine, and changes can be made and committed locally. These changes can be shared with other developers by pushing them to a remote repository.

Q 3. Perform below operations using CVS

1. *Install tortoiseCVS Client*
2. cvs checkout

To checkout a module from the repository, you can use the cvs checkout command followed by the name of the module you want to checkout. For example:

cvs checkout mymodule

This will create a local copy of the mymodule module in your current directory.

1. cvs update

To update your local copy of the module with the latest changes from the repository, use the cvs update command followed by the name of the module. For example:

cvs update mymodule

This will retrieve any changes made to the mymodule module in the repository and apply them to your local copy.

1. cvs add

To add a new file to the repository, you can use the cvs add command followed by the name of the file. For example:

cvs add myfile.txt

This will add the myfile.txt file to the repository.

1. cvs remove

To remove a file from the repository, you can use the cvs remove command followed by the name of the file. For example:

cvs remove myfile.txt

This will remove the myfile.txt file from the repository.

1. cvs commit

To commit changes made to a file or files in your local copy of the module back to the repository, use the cvs commit command followed by the name of the file or files. For example:

cvs commit myfile.txt

This will commit any changes made to the myfile.txt file in your local copy of the module back to the repository. You will be prompted to enter a commit message to describe the changes made.

Q 4. Differentiate Between The Git & SVN Repository?

Git and SVN are both version control systems that are widely used for software development. While both systems share some similarities, they also have some fundamental differences. Here are some key differences between Git and SVN repositories:

Distributed vs. Centralized: Git is a distributed version control system, while SVN is centralized. This means that with Git, every developer has a complete copy of the repository on their local machine, and can work independently and offline. In contrast, with SVN, there is a central repository that all developers connect to, and changes must be made to the central repository.

Branching and Merging: Git is generally considered to be superior to SVN when it comes to branching and merging. With Git, branching is cheap and easy, and merging is generally straightforward. This makes it easy to experiment with different versions of a codebase, and to collaborate on multiple features simultaneously. SVN, on the other hand, can be more difficult to work with when it comes to branching and merging, and conflicts can be more common.

Performance: Git is generally faster than SVN, especially when it comes to large repositories or repositories with many branches. This is because Git is designed to work with local copies of the repository, while SVN requires network communication with the central repository for many operations.

Learning Curve: Git has a steeper learning curve than SVN, especially for developers who are new to version control. This is partly because Git is more flexible and powerful than SVN, but also because Git has a more complex command-line interface.

Integration with Other Tools: SVN has been around for longer than Git, and as a result, it has better integration with some development tools, such as IDEs and issue trackers. However, Git has become much more popular in recent years, and most development tools now support both Git and SVN.

Overall, both Git and SVN have their own strengths and weaknesses, and the choice of which system to use will depend on the specific needs and preferences of the development team.

Q 5. What is “branch”, “tag” And “trunk” In SVN?

In SVN, the terms "branch," "tag," and "trunk" refer to specific directories within a repository that are used to organize the codebase and track changes over time. Here's what each term means:

Trunk: The trunk is the main development line of the codebase, and is typically where the latest version of the code resides. Changes made in branches are eventually merged back into the trunk once they have been tested and reviewed.

Branch: A branch is a copy of the codebase that diverges from the main development line (also known as the trunk). Branches are typically used to experiment with new features or to work on bug fixes without affecting the main codebase. Once changes have been made in a branch, they can be merged back into the trunk or into another branch.

Tag: A tag is a specific snapshot of the codebase that is marked with a descriptive name or version number. Tags are typically used to indicate significant releases or milestones in the development process, and are often used for archival or auditing purposes. Unlike branches, tags are not intended to be modified or merged back into the main codebase.

In SVN, these directories are typically organized as follows:

/trunk: This directory contains the latest version of the codebase.

/branches: This directory contains one or more subdirectories, each of which represents a separate branch of development.

/tags: This directory contains one or more subdirectories, each of which represents a specific snapshot of the codebase (i.e. a tag).

By using branches, tags, and the trunk, developers can easily manage changes to the codebase and track the history of those changes over time.

Q 6. How CVS(Concurrent Versions System) is different from SVN(Subversion)?

|  |  |
| --- | --- |
| **CVS** | **SVN** |
| In software development, CVS is a client-oriented, free version control system. | SVN is a version control system with the most advanced, hi-tech, and latest technology for developing software. |
| Allows a user to store no other information other than files. | A file can be attached to any number of named attributes with SVN. |
| A CVS-versioned file can't be replaced with a new item of the same name without inheriting the old item's history. | Subversion can add, delete, copy, and rename files and directories. Each new file has its own history. |
| Atomic commits are not supported by concurrent versions system, but SSH (Secure Shell) is. | Apache Subversion also supports both HTTP and HTTPS. |
| The CVS process is slower than the SVN process. | SVN is far faster than CVS. Your computer runs faster since all work files are backed up. |
| When it comes to CVS, it helps to roll back any changes made to a repository. Sometimes each file should be treated independently. | With SVN, you cannot roll back changes. |

Q.7 Build a simple web app with Express and Angular.

Step 1: Set up the project

First, we will create a new project directory and initialize it as an npm package. Open your terminal and navigate to the location where you want to create your project. Run the following command to create a new directory:

**mkdir my-app**

you can replace my-app with the name of your choice. Next, navigate to the new directory using:

**cd my-app**

& initialize it as an npm package by running the following command:

**npm init -y**

This will create a package.json file in the root of your project, which will contain all the necessary information about your project and its dependencies.

Step 2: Install Angular & Express

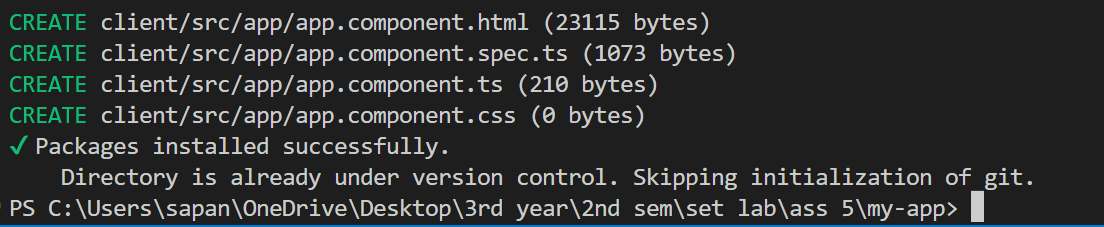
Next, we will install angular and express for our project. Run the following command to install Angular and Express:

**npm install -g @angular/cli**

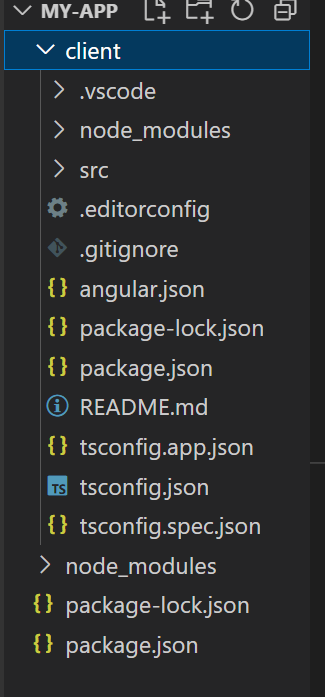
Step 3: Create the Angular client

We will use the Angular CLI (Command Line Interface) to create the Angular client. Run the following command to generate a new Angular project:

**ng new client**

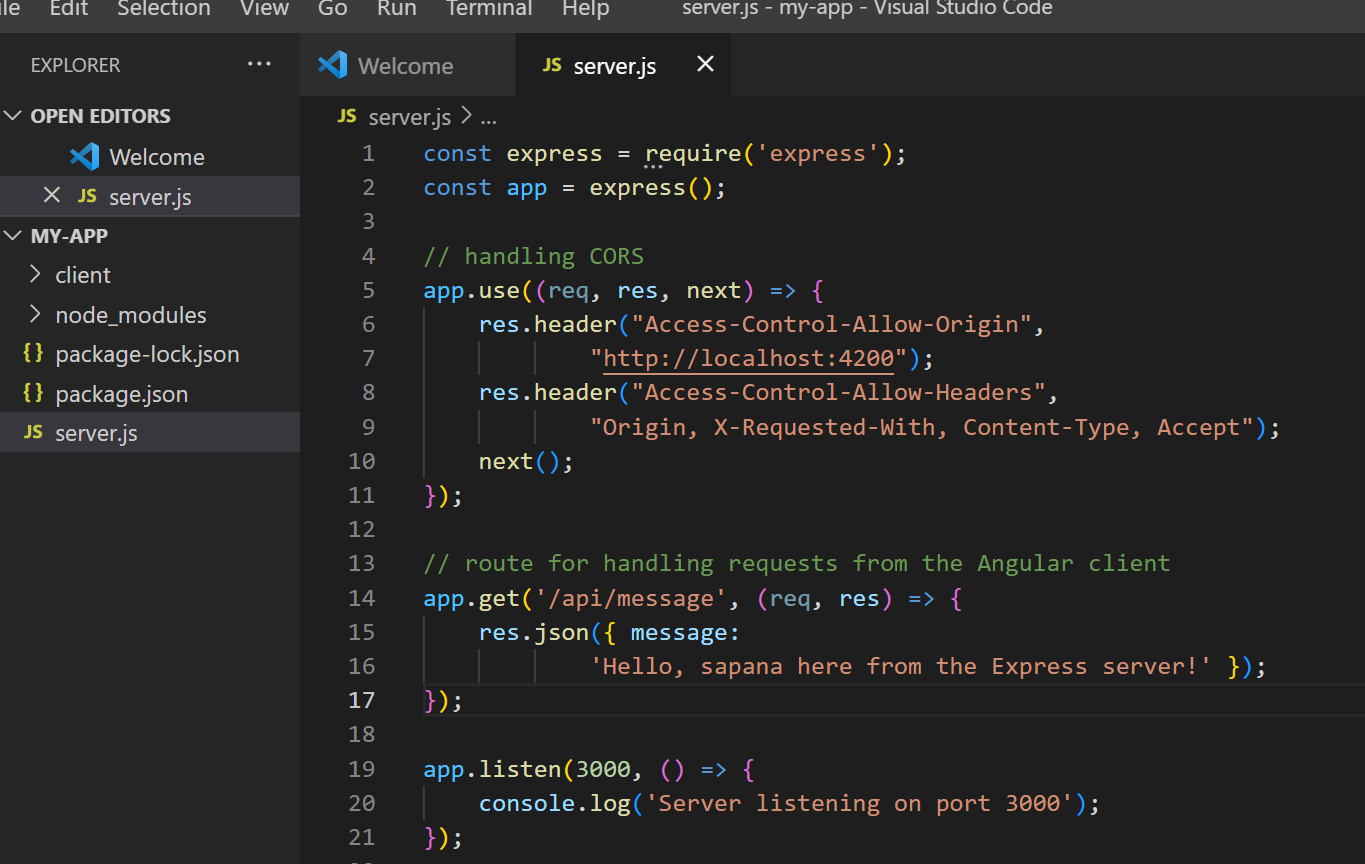
****

Project Structure: After completing the installation procedures, the following structure will be displayed:



Step 4: Create the Express server

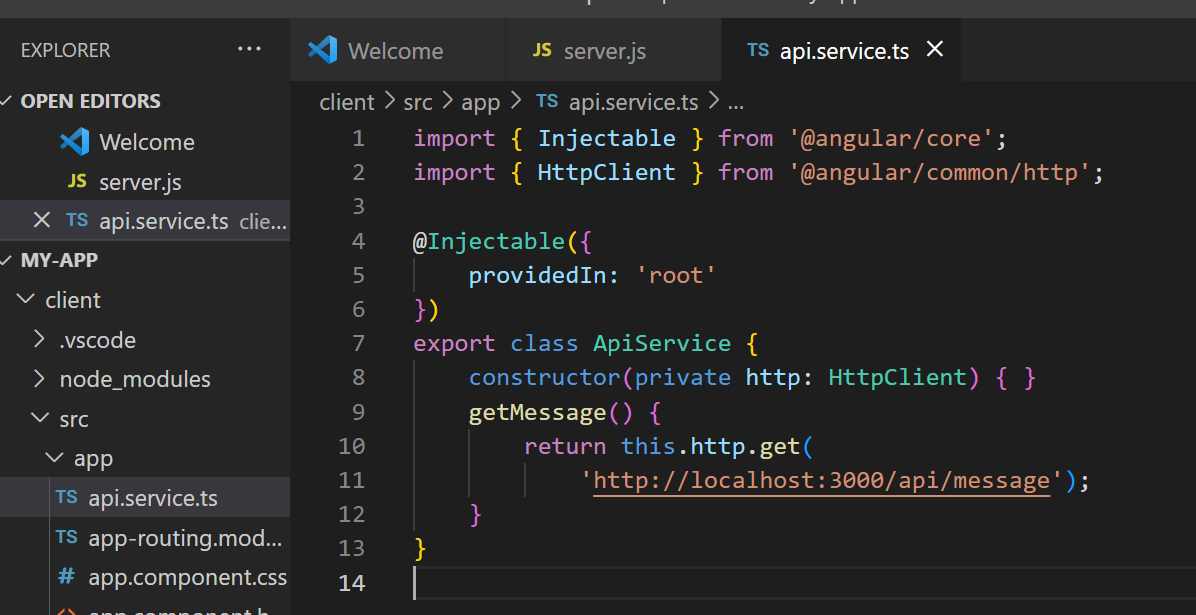
Now, we will create the Express server. Create a new file called “server.js” at the root of your project. In this file, we will set up our Express server. First, we will update the Express server to handle requests from the Angular client. In the “server.js” file, add the following code to handle CORS (Cross-Origin Resource Sharing) and to allow the Angular client to make requests to the Express server:



Step 5: Update the Angular client

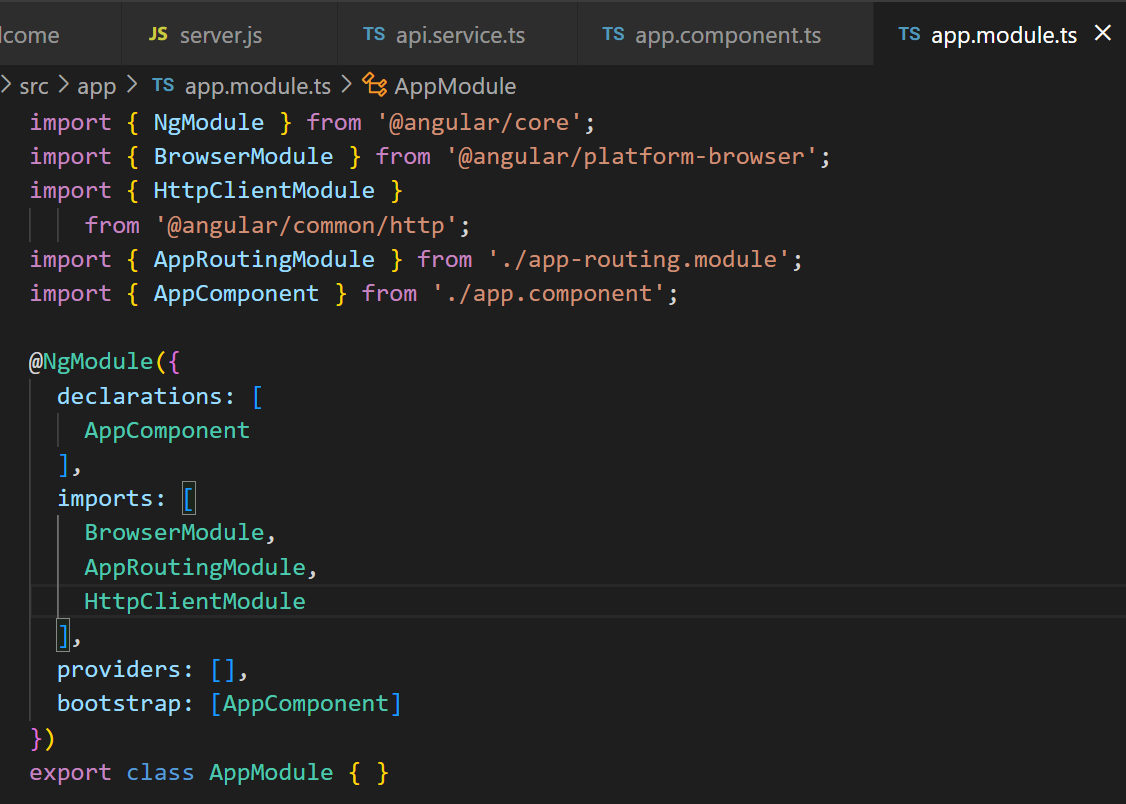
Next, we will update the Angular client to make requests to the Express server. In the “src/app” directory, create a new file called “api.service.ts”. In this file, we will create a service that will handle the requests to the Express server.

api.service.ts



Here, we created a service that has a single method, “getMessage()”, which makes a GET request to the Express server’s ‘/api/message’ route. You will also require to import the HttpClient Module into your app.module.ts file, add the following code to import it:

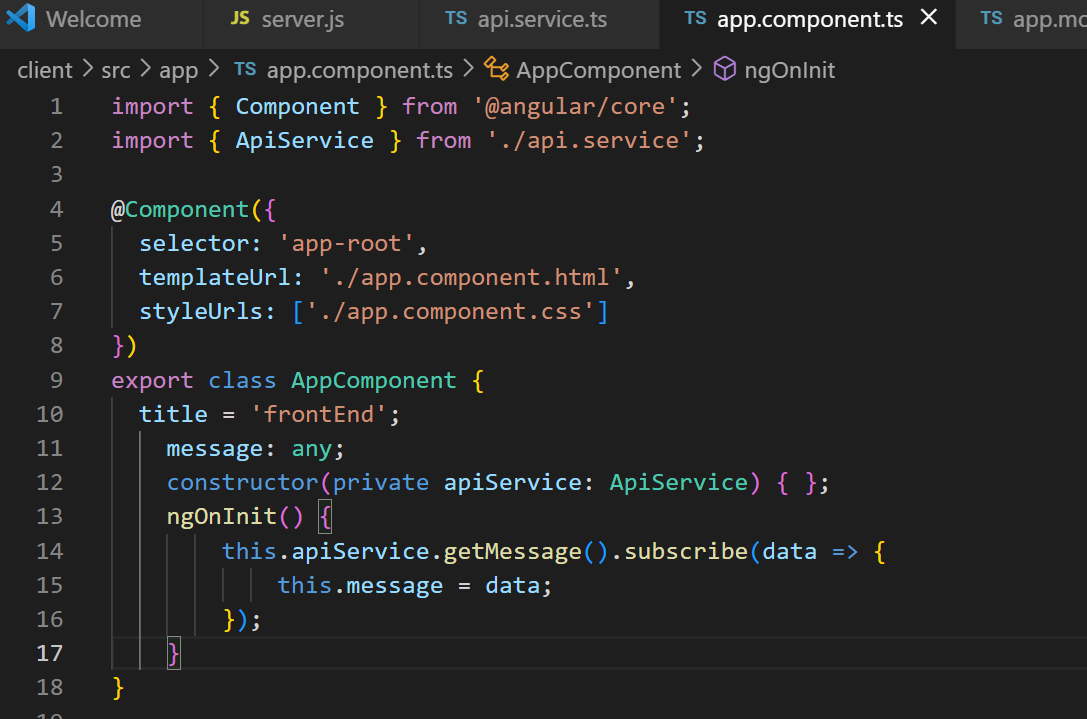
app.module.ts



Step 6: Use the service in the Angular component

Finally, we will use the service in an Angular component to display the message from the Express server. In the “src/app” directory, open the “app.component.ts” file.

app.component.ts



Step 7: Display the message on the Angular template

Open the “src/app/app.component.html” file and add the following code to display the message from the server:

app.component.html

<h1 style="color: rgb(227, 154, 237);

    font-style: italic;">

  Welcome to my Angular-Express Web App

</h1>

<h3>

  Build a Simple Web App with Express & Angular

</h3>

<h5>

  The below message is fetched from the express backend

</h5>

<p \*ngIf="message">

  {{ message.message }}

</p>

This will display the message from the Express server on the Angular template.

Step 8: Start the development server

Now that we have integrated the Angular front-end with the Express back-end, we can start the development server. To start the Angular development server, navigate into the “client” directory and run the following command:

**ng serve**

This will start the development server and make your Angular client available at “http://localhost:4200/”.

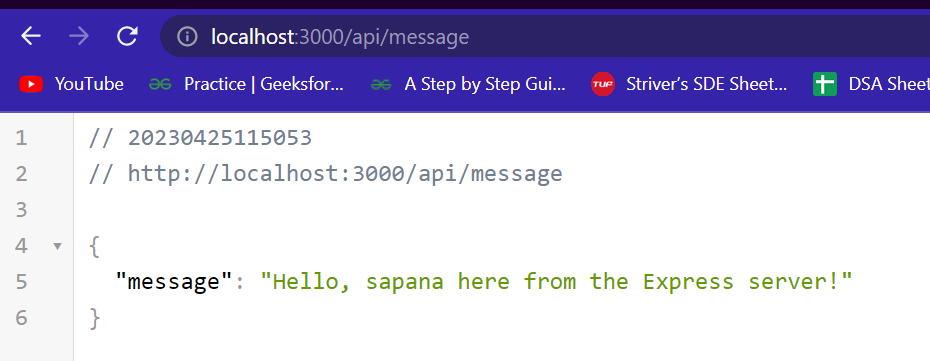
To start the Express server, open a new terminal window and navigate to the root of your project. Run the following command:

**node server.js**

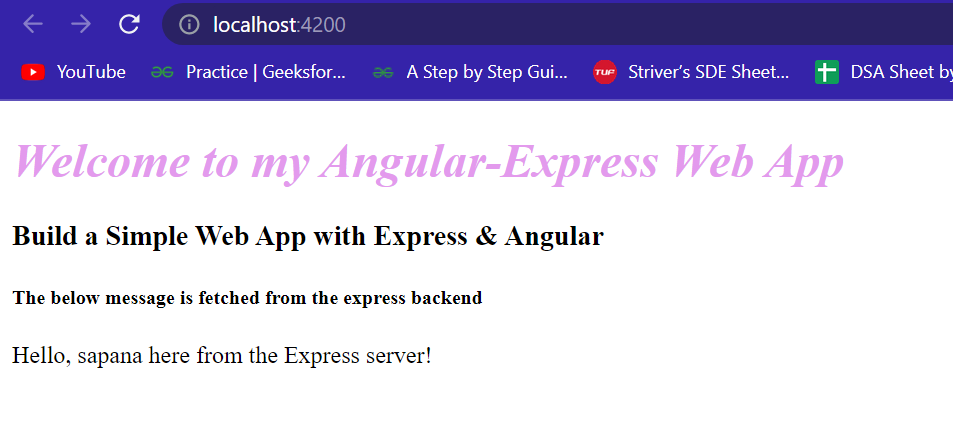
With this, we have successfully created an example application that integrates the Angular front-end with the Express back-end. The Angular client makes a GET request to the Express server, which returns a JSON object containing a message. The message is then displayed on the Angular template.

Output: The following output will be displayed by the Angular and Express servers:

**Express:**

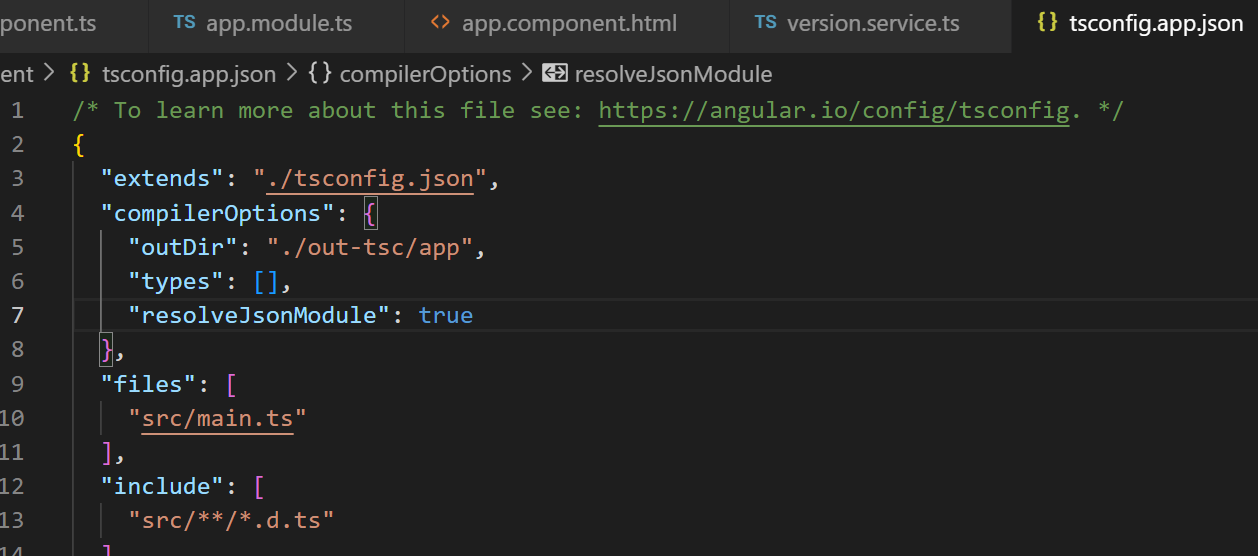
****

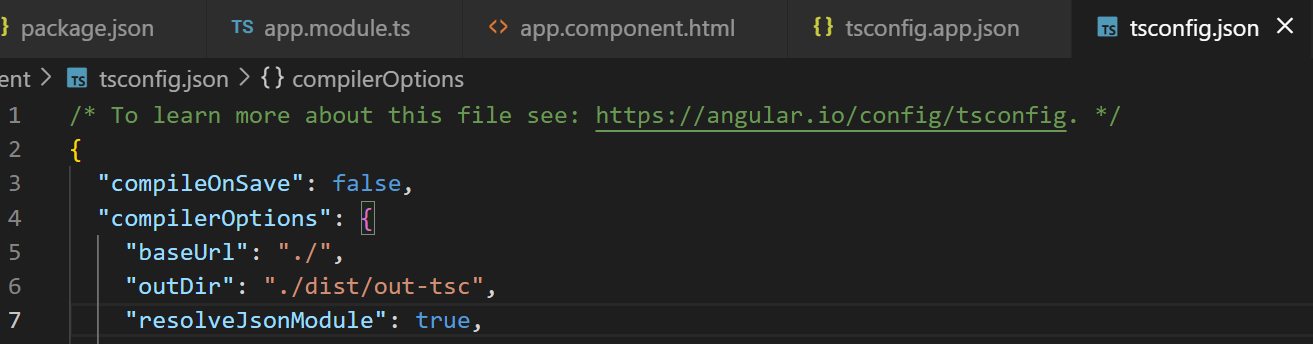
**Angular:**

****

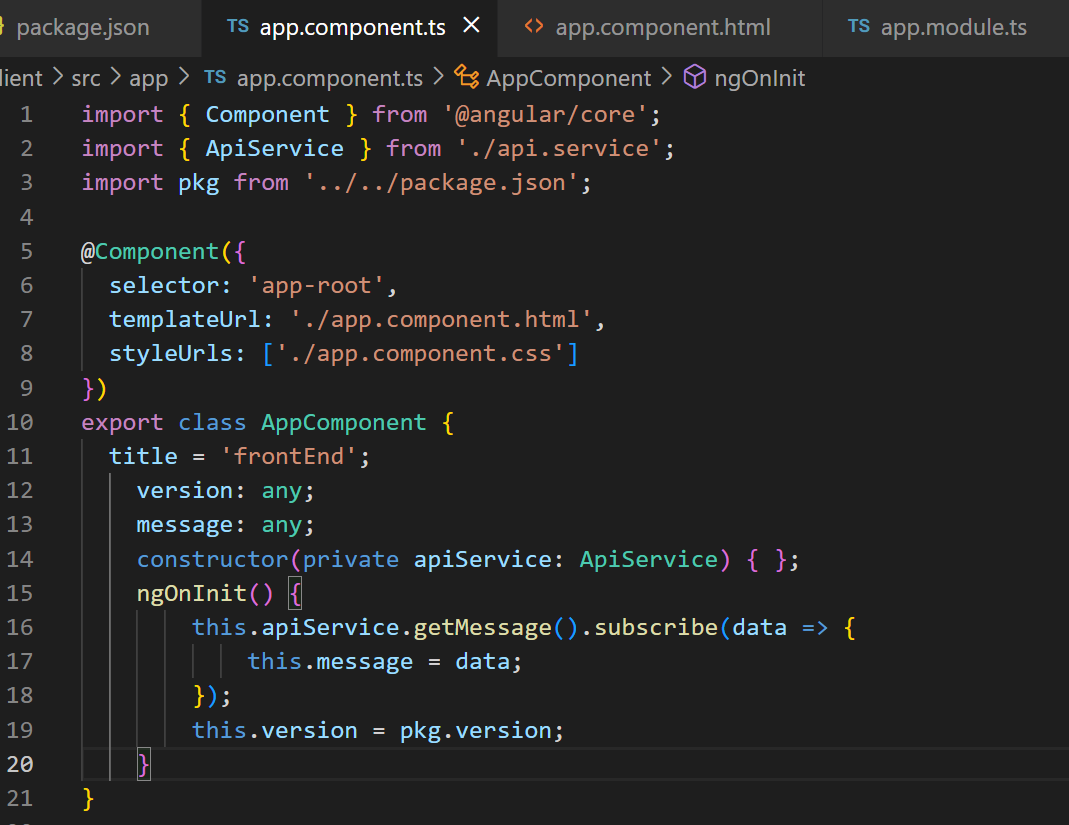
Q.8 Demonstrate and display the app version in angular.

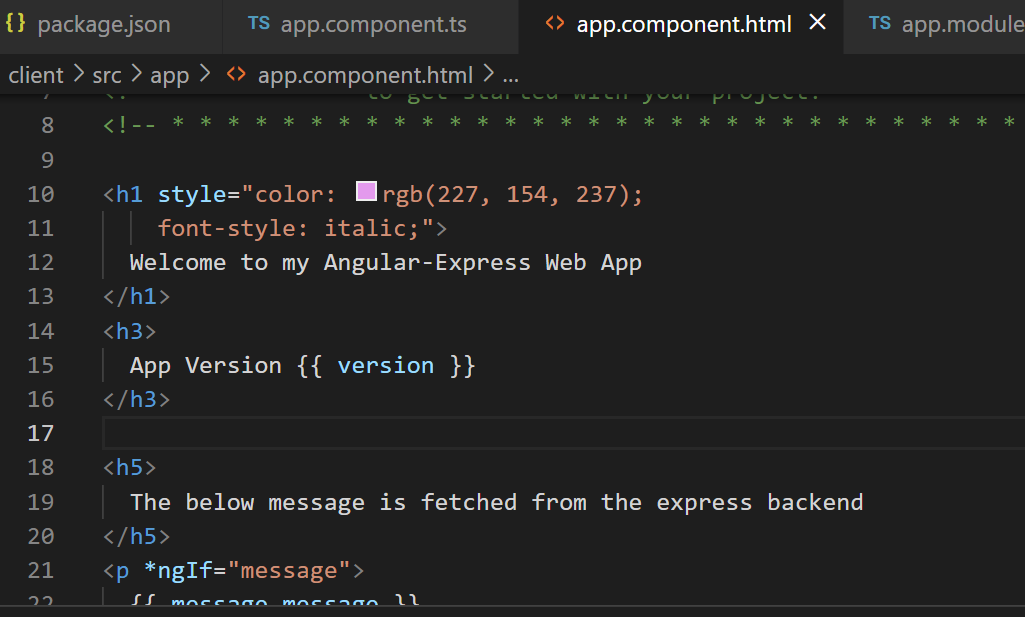
We’d like to import the version value from the package.json file. However, that’s not possible at the moment. We need to tell Typescript to allow that to happen using the resolveJsonModule setting. In **tsconfig.app.json** file add “resolveJsonModule” : true



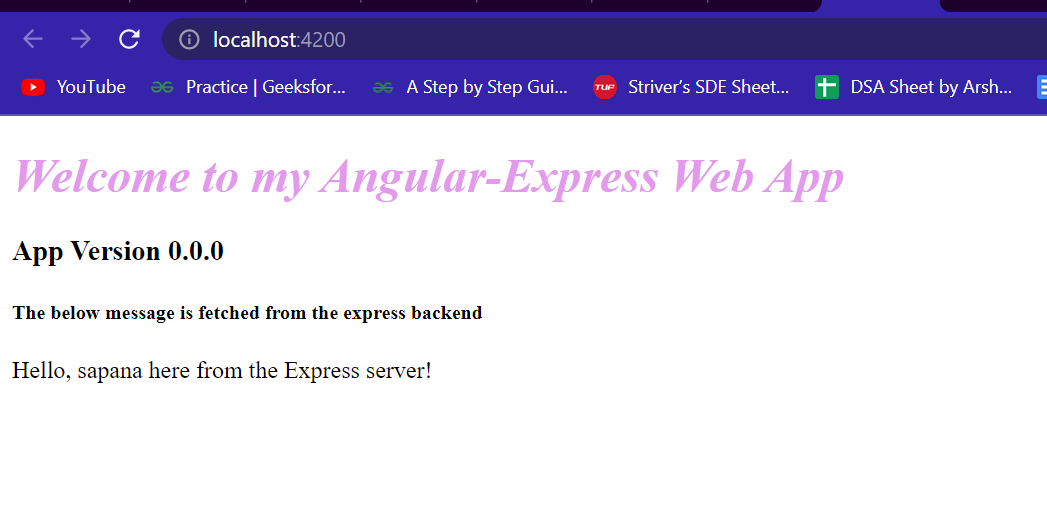


In the component where you’d like to display the version number, now you can import it successfully.





OUTPUT:



Q.9 What is git version control?

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It was created in 2005 by Linus Torvalds, the creator of Linux, and has since become one of the most popular version control systems in use today.

Git allows multiple developers to work on the same codebase simultaneously and track changes made to the code over time. Each developer has their own copy of the repository on their local machine, and changes can be made and committed independently. This means that developers can work offline and collaborate seamlessly, without the need for a centralized server.

Git uses a branching model that allows developers to experiment with different versions of the codebase without affecting the main development line. Branches can be created and merged back into the main development line as needed, making it easy to work on multiple features or bug fixes simultaneously.

One of the key features of Git is its speed and efficiency. Because each developer has a complete copy of the repository on their local machine, operations like branching and merging are fast and can be done offline. Additionally, Git uses a unique data storage system that ensures that every change made to the repository is tracked and can be easily accessed and reverted if necessary.

Overall, Git is a powerful and flexible version control system that has become a fundamental tool for modern software development. It is used by millions of developers and organizations worldwide, and has spawned a vibrant ecosystem of third-party tools and services to support its use.

Q.10 Demonstrate creation of repository in git

Here are the basic steps to create a new Git repository:

Open your terminal or command prompt and navigate to the directory where you want to create your new repository.

Use the git init command to initialize a new Git repository in the current directory. This will create a new .git directory that will hold all of the metadata and version history for your project.

**git init**

Add some files to your repository. You can use any text editor to create a new file, or copy an existing file into the repository directory. For example, let's create a new file called README.md:

**echo "# My new project" > README.md**

Add the file(s) to your Git repository using the git add command. This will stage the file(s) for the next commit.

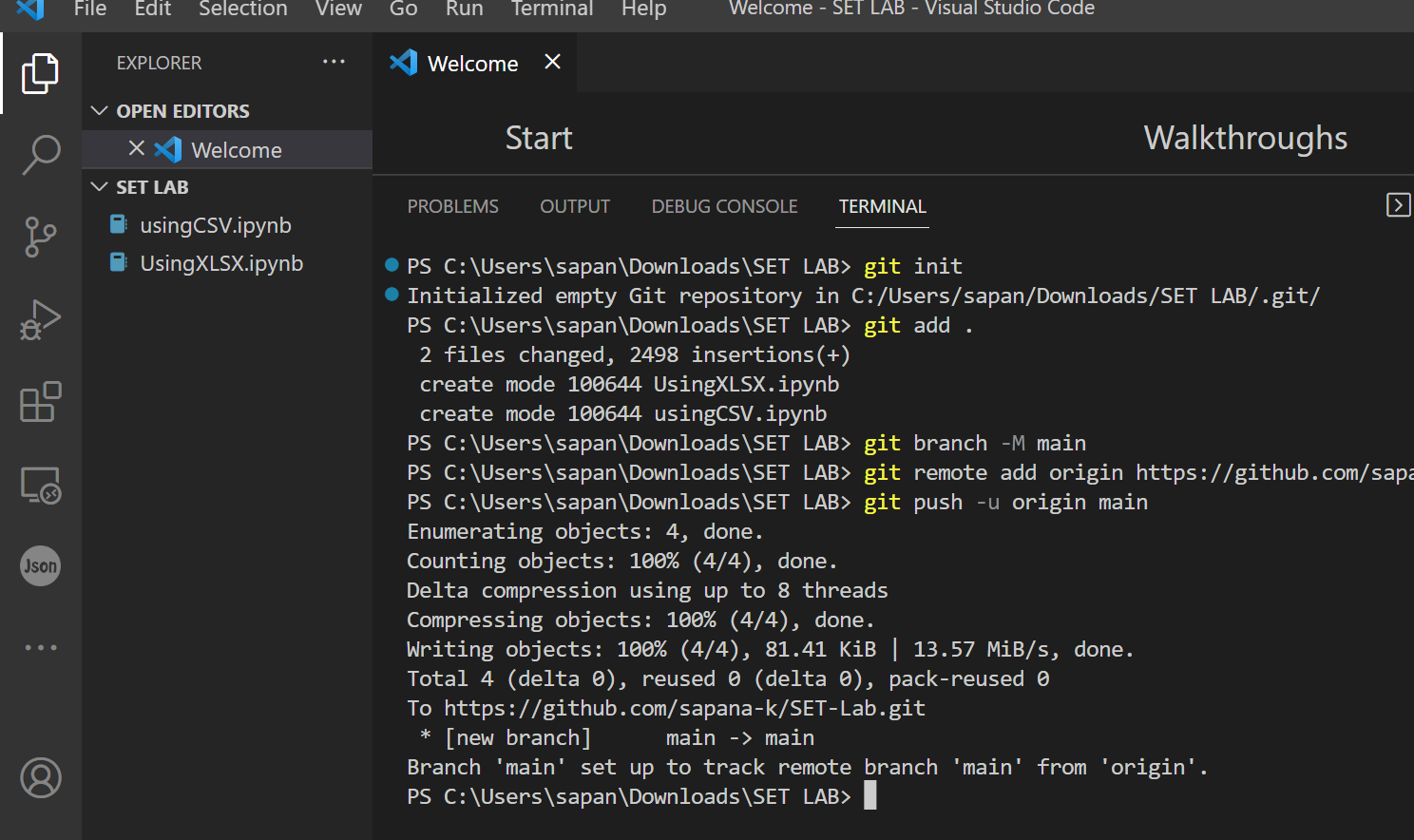
**git add README.md**

Commit the changes to your repository using the git commit command. This will create a new commit with a message describing the changes you made

**git commit -m "Initial commit"**

Your Git repository is now set up and ready to use! You can continue to make changes to your files, add them to the repository with git add, and commit them with git commit as needed.

Note that this is just a basic example of creating a Git repository. There are many other Git commands and options you can use to customize your workflow and manage your repository more effectively.

****