# Starting and Debugging JS Applications

The following information serves as a guide for **starting and debugging the applications** you will be working on. During the course, you will be given **a Word document with the problem description(s) and an archived folder with the needed resources for each task.**

# Part I: Start the Application

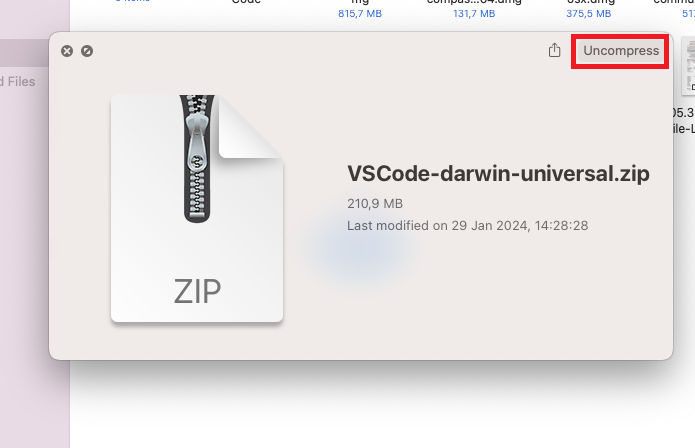
## Open the Resources

* Download the resources from the **course web page**:

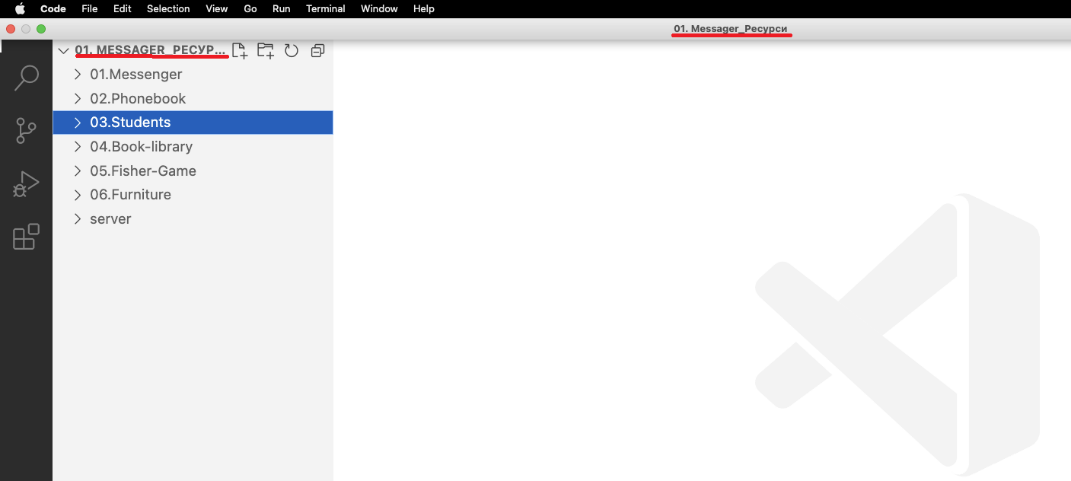
A screenshot of a computer

Description automatically generated

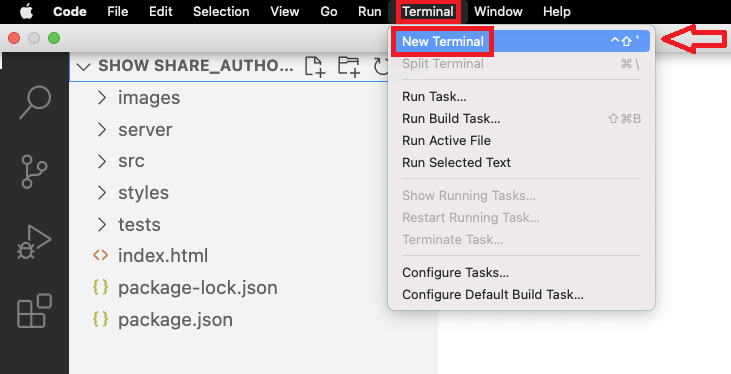
* You need to **unarchive the folder** in order to work on the task:

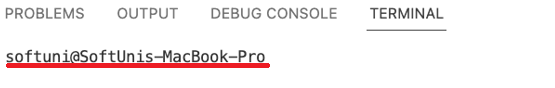


* Then, open the folder in **Visual Studio Code**:



## Install the Dependencies

* **Open a new terminal** window by selecting Terminal > New Terminal or using the keyboard shortcut: Cmd + ` (backtick):  
  **
* Always pay attention to **the folder the terminal is open in**, because the commands you write will be executed in this folder:

  
If you need to go to the parent folder of the current folder (one level up), use the command cd .., if you need to work in a folder that is in the current folder (one level down), use cd + the name of the target folder (e.g., cd server). An easier way to go one level down is to type "cd " (with the space!) and then press the Tab key once for the first subfolder, second for the second one, etc. At the end, always press Enter to execute the command.

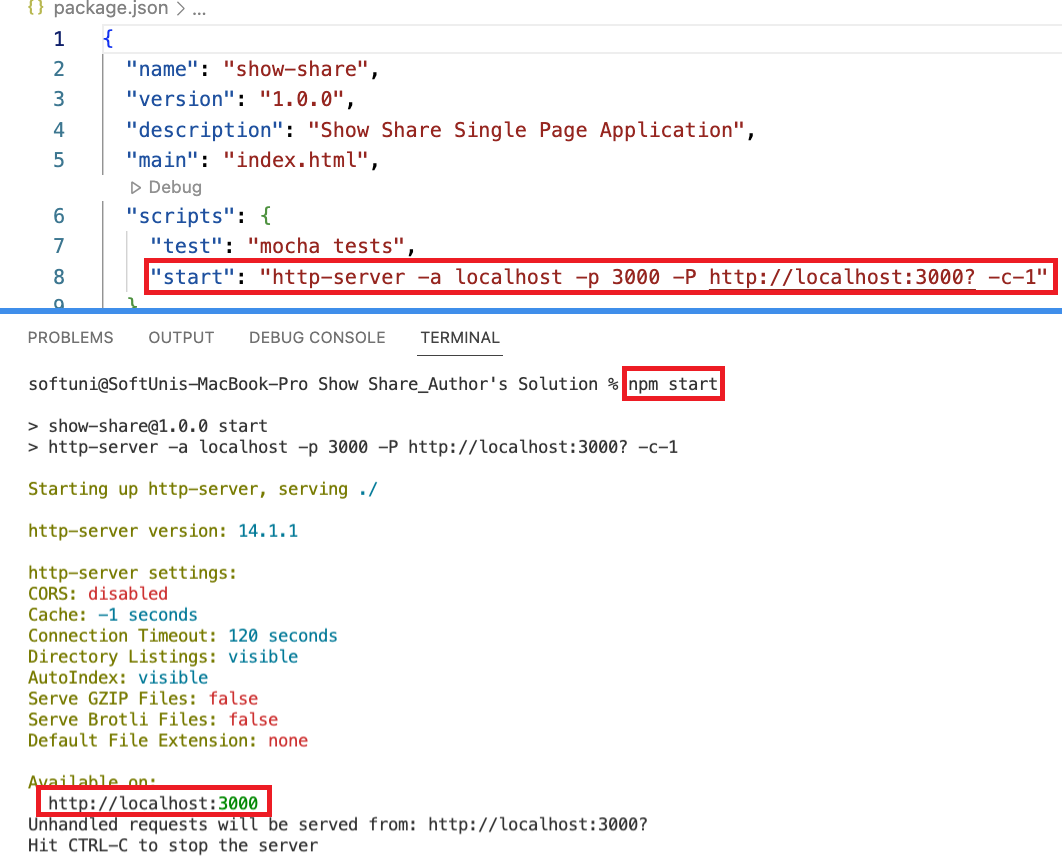
* **Navigate to** **the folder of the task** you will be working on. This is the folder which contains the package.json file for the application. In the example below, you have several tasks in the provided resources. When you first open the terminal, it is in the main folder of the resources, where there is no package.json file. In order to go to the folder of the first task (01.Bus-Stop), you need to go one level down. Type "cd ", press Tab once to select the first subfolder (01.Bus-Stop), and press Enter to navigate to the selected folder:

**IMPORTANT: Do NOT change, move, or delete the provided resources! You are only allowed to work in app.js and (if necessary) index.html.**

* When you are in the folder of the task you will be working on, execute the command npm install (or npm i) in the terminal. This command will create a folder "node\_modules" and **install in it all the needed packages** from the package.json file ("dependencies" and "devDependencies"):  
  

## Start the Application

* In the folder of the task, use the command npm run start (or npm start) to start the application. This will execute the script "start" from the package.json file and will give you a link to open the application in the browser (http://localhost:3000):



The application will be running until you press Ctrl + C or close the terminal window.

* Use Ctrl + click on the link or type <http://localhost:3000> in the browser address bar to see the application in the browser:

A screenshot of a computer

Description automatically generated

* If you are provided with an **app.js** file, check the initial code in it. If there is no app.js file in the resources, create one and use console.log('…') to log something to the console of the browser:

A screenshot of a computer

Description automatically generated

* Check the content provided for you in the file **index.html**. Pay attention to the way it is linked to the .css file with the styles and to the script (if provided). If you just created the app.js file, you need to link it here. If you are planning to use several .js files for your code, you need to include type="module" in the <script> tag. Locate the function from app.js or add it at the appropriate place. In the example below, this function will be executed when you click on an input field of type "button" with the text "Check":

A screenshot of a computer

Description automatically generated  
  
**IMPORTANT: You are only allowed to cut sections of the index.html file to use them in the exact same place as different views, add attributes, and change the value of the "href" and "action" attributes. Any other change to the index.html file might prevent the tests from running correctly.**

* Use F12 in the browser to open the DevTools, open the Console tab, execute the function from app.js (in the example below, click on the button), and **make sure everything is set up correctly**:  
  A screenshot of a computer

  Description automatically generated

## Start the Server

* To be able to perform requests to the provided server, you need to have both the front-end application (that you have already started) and the server running simultaneously. **Open a new terminal window** by clicking on the "+":

A screenshot of a computer

Description automatically generated

* At this point, you will see both open terminals on the right side; you can click on them to visualize one or the other. **Navigate to the folder "server"** and execute the command node server.js:

  
  
The server will be running until you press Ctrl + C or close the terminal window.

You are now ready to start writing your solution to the task!

# Part II: Debug the Application and Run the Tests

## Debug the Application

* While working on your application, **keep an eye on the browser console**. Below you can see the error that you get when you try to retrieve some data from a server that is not running. You have an indication of the file and line of code where the error was thrown (line 7 in the app.js file):

A screenshot of a computer

Description automatically generated

* Use console.log(…) in your code to log the value of the variables to the browser console and be able to check it when testing the application. For your convenience, the browser console will even tell you the file name and the line number where the specific console.log(…) is located:

A screenshot of a computer

Description automatically generated

* When you use modules (your code is distributed between several .js files), **pay attention to the way you import the files**. Even a single missing letter could cause the whole application to stop working. If you use ES modules (the import statement), **always add the file extension** (".js"). Pay attention to the use of capital/small letters as well as this could be a problem for our Judge system:  
  A screenshot of a computer

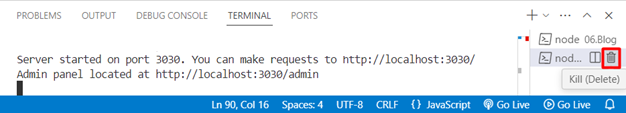
  Description automatically generated
* When you have your **base URL** stored in a variable, beware of the way you are using it. It is easy to miss a "/" or insert an unnecessary one:  
  A screenshot of a computer

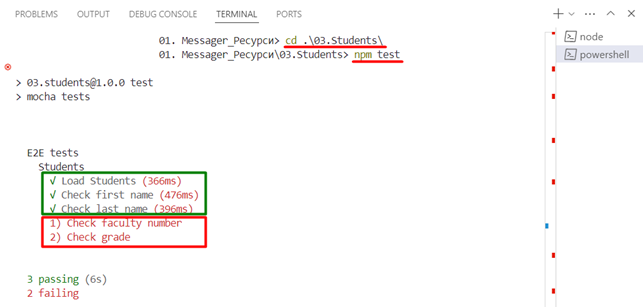
  Description automatically generated
* Check the **HTML structure of the newly created elements**. Adhere to the structure provided in the document; all the elements must be present, each element must have the exact same attributes (especially id and class) as in the document with the problem description:

A screenshot of a computer program

Description automatically generated

## 2. Run the Tests

* **Stop the server** by pressing Ctrl + C in the terminal where it is running or by selecting "Kill (Delete)":  
  
* Open a new terminal, **navigate to the folder of the task**, and run the tests using the command npm run test (or npm test). This command must be executed in the folder containing the folder "tests" for the specific task. The application must be still running in the first terminal. This will execute the script "test" from the package.json file (referencing to the file **e2e.test.js** in the folder "tests"). You will see the names of the tests appear below as they are executed. The ones that passed successfully will be marked in green, the ones that failed – in red:



* Usually, the name of the test is descriptive enough and gives you information about where the error in your code is. In the example above, the test "Check faculty number" failed. You can get additional information if you scroll down in the terminal window. In this case, you can see that at **line 133** of the code in the **e2e.test.js** file (in the folder "tests") the test expected the faculty number to be a string, but it was a number instead:  
  A screenshot of a computer

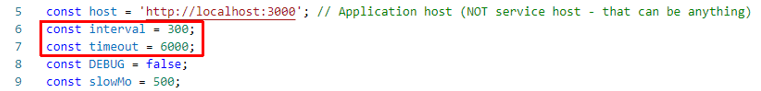
  Description automatically generated
* Below you can see **another example of a failing test** while the application seems to be working as expected. The test checks the forecast for the upcoming days and **cannot find any** (expects their number to be 3, but finds 0), even though you can clearly **see three of them in the browser**. In the code, you can see that the test is looking for an element with the class "upcoming" which is in an element with the class "forecast-info" which is in the element with the ID "upcoming" ('#upcoming .forecast-info .upcoming'):

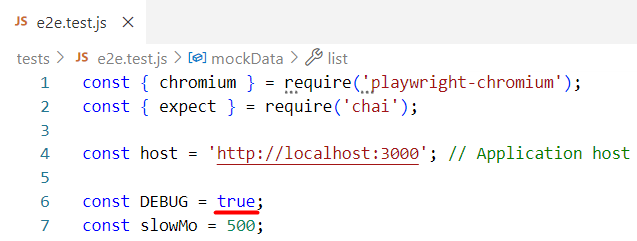
A screenshot of a computer

Description automatically generated  
  
**Inspect the elements** in your code and **compare them carefully with the provided HTML structure** from the document. You will find some discrepancy, for instance:  
A screenshot of a computer

Description automatically generated

## 3. Debug After Failed Tests

* On slower machines, some of the tests may require more time to complete. You can instruct the tests to run more slowly by slightly increasing the values of the variables **interval** and **timeout**:  
    
    
  Note that **interval** values greater than 500 and **timeout** values greater than 10000 are not recommended.
* If some tests are still failing, try **running the tests in debug mode**. Open the **e2e.test.js** file (in the folder "tests") and change the value of the DEBUG variable to true:

  
  
If the actions are happening too fast, you can increase the value of **slowMo**. If the browser hangs, you can just close it and abort any remaining tests by focusing the terminal window and pressing Ctrl + C followed by the letter "y" and Enter.

* If some tests failed in debug mode as well, **run the tests one test at a time** by adding ".only" after the "it" reference:

A screenshot of a computer program

Description automatically generated  
This way, it will be easier for you to test changes in your code about a single failed test.

## 3. Understand the Tests

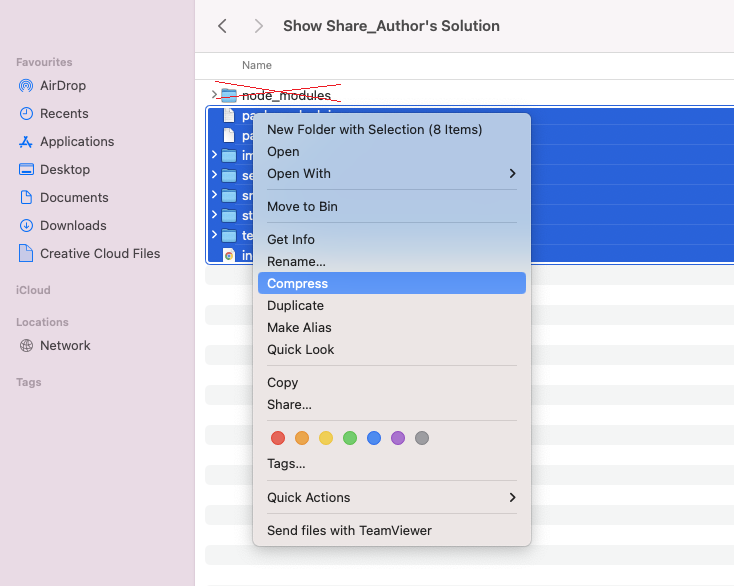
Whenever you have a test failing, you should **read the code of the test** to retrieve more information about the error that could cause it to fail. Below you can find some explanation of a typical test code in the comments:  
A screenshot of a computer code

Description automatically generated

# Part III: Submit Your Solution

## Archive

Do NOT use WinRAR version 6 or below to archive your solution. Archive **all the resources**, including the files and folders you created. **Exclude only the folder "node\_modules"**. Do NOT remove the tests or the server (if provided in the task folder):

  
  
**IMPORTANT: Do NOT archive the folder of the task, only its content! If you archive the outer folder, there will be an extra folder and the tests won’t be executed in the Judge system.**

## Submit to the Judge System

In the Judge system, go to the page of the task you’ve been working on. Click on the "Select files…" button, select the archived folder and submit it: 