**CS624 Full Stack Development – Mobile App**

**HOS10A: Full-Stack Mobile App**

Revised by Clark Ngo on February 23, 2025

Revised by Sam Chung on February 23, 2025

Reviewed by Naveena Moddu on May 29, 2025

School of Technology & Computing (STC)

City University of Seattle (CityU)

A blue and white logo

AI-generated content may be incorrect.A black text on a white background

AI-generated content may be incorrect. A green leaf with black text

AI-generated content may be incorrect.

**Before You Start**

* **Screenshots may be different from your environment.**
* The directory path shown in screenshots may be different from yours.
* The steps might have subtle discrepancies. Please use your best judgment while completing each step in this cookbook-style tutorial.
* Some steps may not be explained in detail. If you are not sure what to do:

1. Consult the resources from the course.
2. If you cannot solve the problem after a few tries (usually 15 -30 minutes), ask a TA for help.

#### **Readings and Examples:**

* Visit the [CS 624 Repository for Examples](https://github.com/samchung0117/cs624-examples).
  + Select the related module.
  + Visit the README.md file.
  + Find examples for your practices.

**Learning Outcomes**

* Section 1: Accessing GitHub Codespaces.
* Section 2: Setting up a backend web app
* Section 3: Setting up a connection to MongoDB Atlas
* Section 4: Setting up Server API Endpoints
* Section 5: Setting up a frontend mobile app
* Section 6: Code explanation – App.js
* Section 7: Testing CRUD operations
* Section 8: Pushing your work to GitHub.

**Upload the following to your GitHub Repository generated from GitHub Classroom.**

* + - 1. The screenshot of your ‘01 Node Server’ as ‘01\_node\_server\_firstname\_lastname.png’ using your first and last name.

1. The screenshot of your ‘02 Create Record’ as ‘02\_create\_record \_firstname\_lastname.png’ using your first and last name.
2. The screenshot of your ‘03 Update Record’ as ‘03\_update\_record\_firstname\_lastname.png’ using your first and last name.

**Section 1: Accessing GitHub Codespaces**

Refer to the steps from [the TA Center](https://cityuseattle.github.io/docs/git/github_codepsace/) to get started with this week’s module.

**Section 2: Setting up a backend web app**

**Backend Web App:**  
The server-side part of a web application that handles data processing, business logic, database interactions, and serves responses to client requests.

**Node.js:**  
A JavaScript runtime built on Chrome's V8 engine that allows you to run JavaScript on the server side, ideal for building scalable network applications.

**Express:**  
A minimalist web framework for Node.js that simplifies the process of creating robust web applications and APIs by handling routing, middleware, and HTTP utilities.

Let us now check the node version in the current Codespaces.

1. Once your Codespaces is open, you can access an integrated terminal.
2. In the terminal, you can use the following command to check the installed Node.js version:

**>>node -v**or

**>>node --version**



In this HOS, we will set up a backend for our application. To develop a server-side web application, create a folder named "backend" under the root directory in Codespaces. Then, navigate to that folder using the following commands.

**>>> mkdir backend**

**>>> cd backend**

We will also install the required dependencies. Run the following command in the terminal.

**>>> npm install mongodb express cors dotenv**

The command above installs the following dependencies:

1. The installation of MongoDB includes the MongoDB database driver, enabling your Node.js applications to establish connections with the database and manage data effectively.
2. When you install "express," you add the Node.js web framework to your environment. Express is a rapid, versatile, streamlined web application framework tailored to Node.js. It equips developers with a toolkit and capabilities to craft web and API applications.
3. The installation of "cors" introduces a Node.js package that facilitates cross-origin resource sharing, allowing resources from different origins to be shared securely.
4. By installing "dotenv," you install a module designed to load environment variables from a '.env' file into the 'process.env' file. Separating configuration details from the code promotes cleaner and more organized development practices.

We can check our installed dependencies using the “package.json” file. For example,

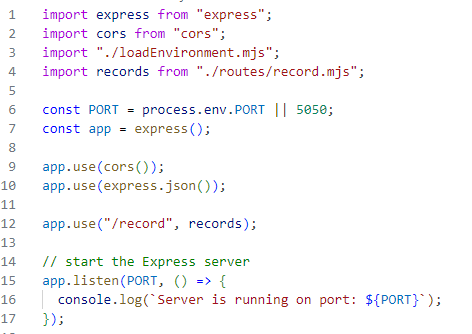
A screenshot of a computer code

AI-generated content may be incorrect.

<https://stackoverflow.com/questions/22343224/whats-the-difference-between-tilde-and-caret-in-package-json>

* ~version **“Approximately equivalent to version”**, will update you to all future patch versions, without incrementing the minor version. ~1.2.3 will use releases from 1.2.3 to <1.3.0.
* ^version **“Compatible with version”**, will update you to all future minor/patch versions, without incrementing the major version. ^1.2.3 will use releases from 1.2.3 to <2.0.0.

Now let’s create a file called “**server.mjs**” with the following code,



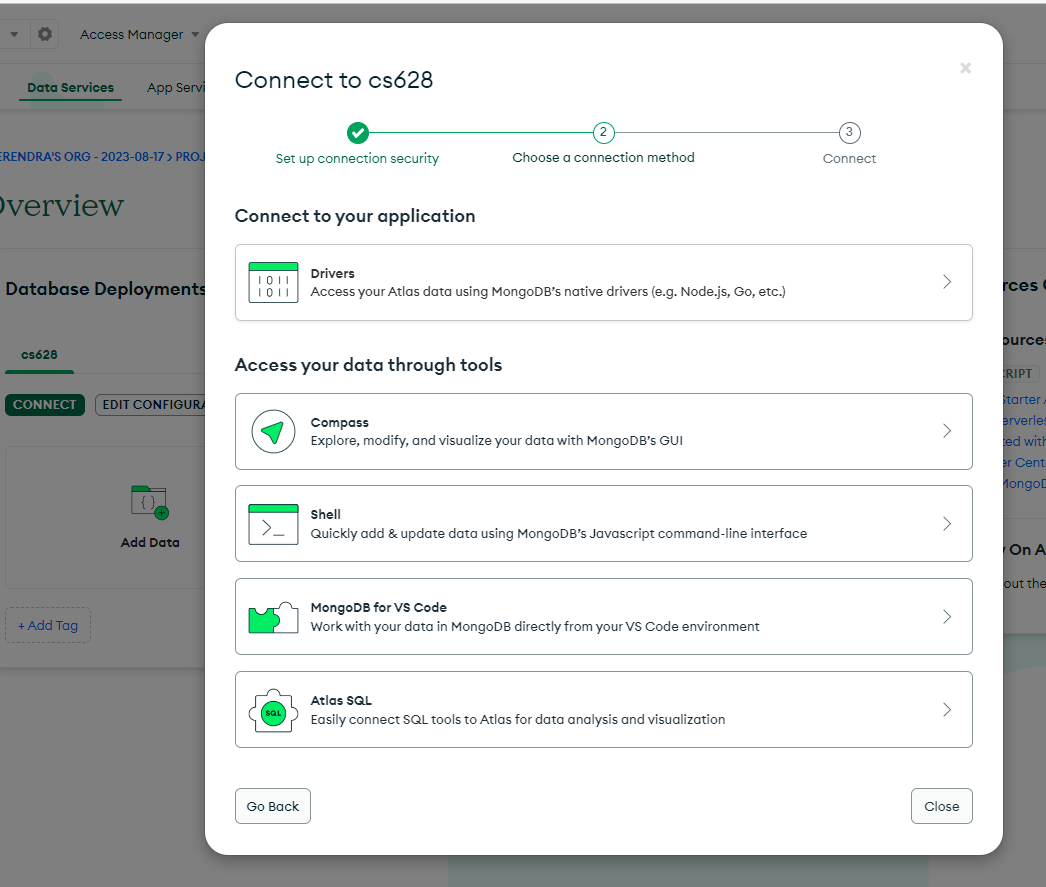
If you're wondering about the meaning of .mjs, it's important to note that both “.js” and “.mjs” files are utilized in Node.js to run JavaScript code. Nevertheless, they diverge significantly in how they manage modules. In “.js” files, Node.js employs the CommonJS module system, requiring the use of require() to import modules and module.exports to export them. Conversely, “.mjs” files utilize the ES (ECMAScript) Modules (ESM) system, employing import and export statements for handling modules.(<https://stackoverflow.com/questions/57492546/what-is-the-difference-between-js-and-mjs-files>)

In this code, we are importing express and cors.

“const port process.env.PORT” will access the port variable from the “config.env.

**Section 3: Setting up a connection to MongoDB Atlas**

We first need the connection string to set up a connection to MongoDB Atlas. Log in to your MongoDB Atlas account. And then, under the overview section, click Connect and then Drivers.

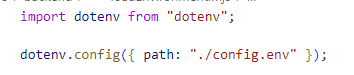


Copy the connection string, update your password like we did in our previous module, and save it. We will use it in the next steps.

Having obtained the connection string, return to the 'backend' directory and create a '**config.env**' file. This file assigns the connection string to a fresh variable named ATLAS\_URI, as shown below.



You will additionally require a module for loading this environment variable. Create a `loadEnvironment.mjs` file in the backend directory and input the provided code.



Create a new directory named '**db**' inside the backend directory. Inside this directory, create a file named '**conn.mjs**'. Within 'conn.mjs', insert the following code to establish a connection with the database.



**Section 4: Setting up Server API Endpoints**

Return to the "backend" directory and generate the required directory and files using the following commands in the terminal for setting up API Endpoints:

**>> cd ../backend**

**>> mkdir routes**

**>> touch routes/record.mjs**

We use the Linux touch command to update the timestamps on existing files and directories as well as create new, empty files. (<https://linuxize.com/post/linux-touch-command/#google_vignette>)

Now insert the code in your “record.mjs” from here. <https://github.com/samchung0117/cs628-examples/blob/main/Module%2008/backend/record.mjs>

The code defines a router with various routes for performing CRUD (Create, Read, Update, Delete) operations on a collection named "records."

Verify folder and file structure:

A screenshot of a computer

Description automatically generated

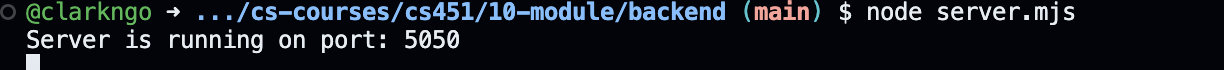
Express Router is a feature provided by the Express web application framework in Node.js that allows you to modularize and organize your application's routes and middleware in a structured manner.

* The first route responds to a GET request at the root path ("/") and retrieves all records from the "records" collection.
* The second route handles a GET request with a parameterized route ("/:id"), fetching a single record based on the provided ID.
* The third route deals with a POST request, allowing the creation of a new record in the "records" collection.
* The fourth route responds to a PATCH request ("/:id") and updates a record based on the provided ID with the data in the request body.
* The fifth route handles a DELETE request ("/:id"), deleting a record from the "records" collection based on the given ID.

Let us now run the server using the following command in the Terminal.

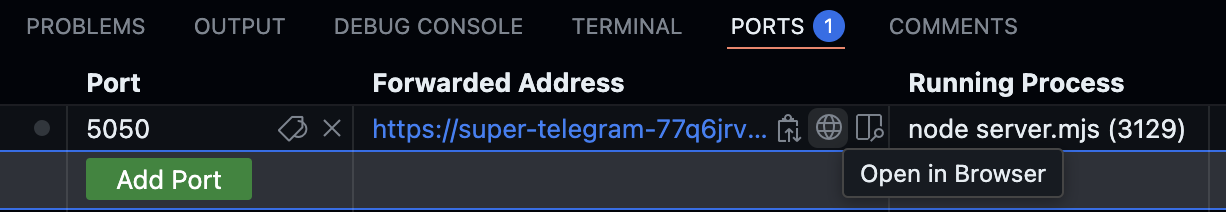
**>>> node server.mjs**

If everything works as expected, you will see this message in the terminal,

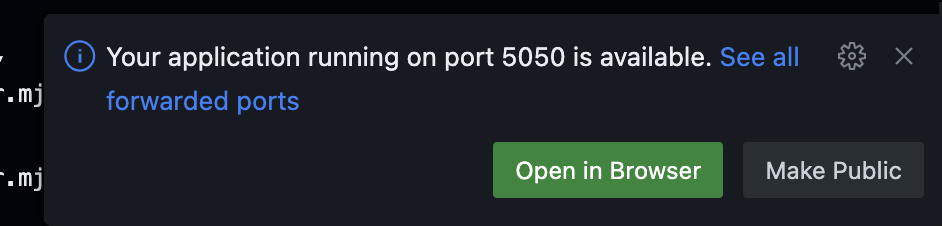


Take a screenshot of your ‘01 Node Server’ as ‘01\_node\_server\_firstname\_lastname.png’ by using your first and last name.

Now click on the Ports tab beside the Terminal and open the local address in the new tab.

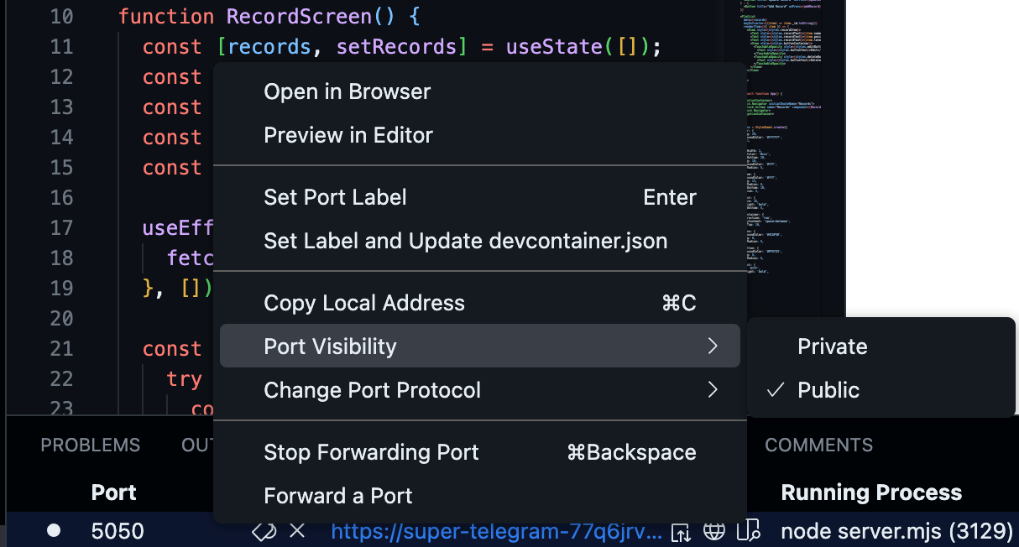


If you see this popped up, click Make Public.

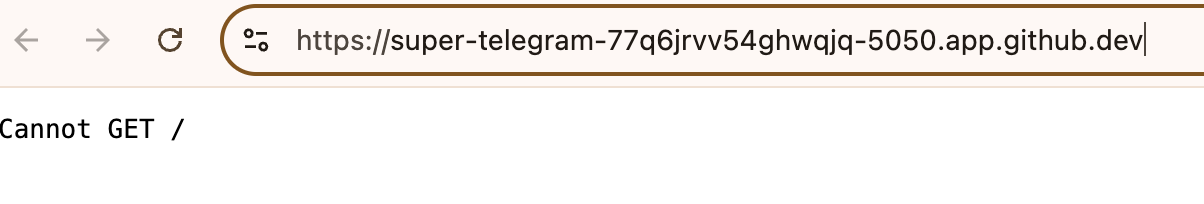


If you missed the popped up, an alternative way to change the Port Visibility:

Right-click on the selected port to change visibility to Public.



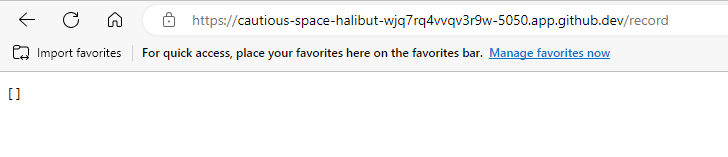
You will see the following message,



In our “server.mjs” file, we are using **“/record**” to mount the records router to a specific route path within the Express application. This means that whenever a request is made to the "/record" route, the functionality defined in the records router will be executed.



Append **“/record”** to the above URL and refresh the browser. The error will go away, and an empty array will be returned. This is expected as we have not inserted any records into our database.



**Section 5: Setting up a frontend mobile app**

1. Open another Terminal. Assume you are under the “HOS10” directory.
2. Run the command “**npx create-expo-app frontend”** to create a new Expo project.  
   Then select a blank template for your mobile app.
3. You can see a directory called “**frontend**” was created.

Move to the “frontend” directory.  
>>> cd frontend

1. Test the default mobile app.  
   >>> npx expo start --tunnel

A screenshot of a cell phone

AI-generated content may be incorrect.

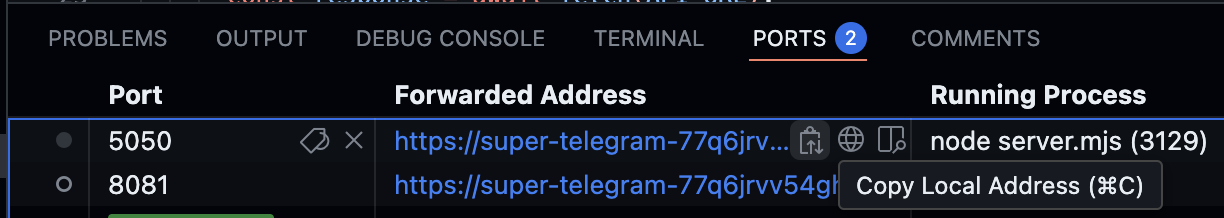
1. Update your “package.json” to make App.js the entry point of the application.  
   Replace "main": "expo-router/entry" with the following info:

"main": "node\_modules/expo/AppEntry.js",

1. Create App.js under the “frontend” directory. We will keep App.js simple this time around and have all the components in one file. Feel free to refactor and move components to new files. App.js code is [here](https://github.com/cityuseattle/cs451-examples/blob/main/Module10/frontend/App.js).
2. In your App.js, please change your API\_URL to your own backend url.



To find it, go to your Ports Tab and click the copy  icon from Port 5050’s Forwarded Address.



Make sure to add **/record** at the end of the copied url when replacing the API\_URL.

1. Run the command **npx expo start --tunnel** to start the development server.

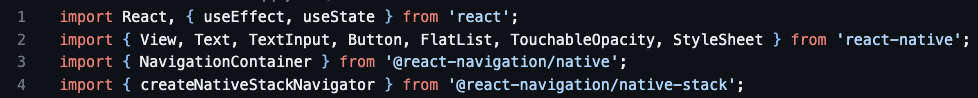
A screenshot of a computer

AI-generated content may be incorrect.

**Section 6: Code explanation – App.js**

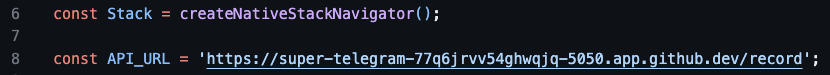
**Imports**

* Brings in React, React Hooks (useEffect, useState), and various React Native components.
* Imports navigation utilities (NavigationContainer, createNativeStackNavigator) to set up screen navigation.



**Stack and API URL**

* createNativeStackNavigator() is used to create a stack-based navigation flow.
* API\_URL stores the endpoint for CRUD operations.



**RecordScreen Component**

1. **State Hooks**:
   * records (list of items), name, position, level (input fields), and editingRecord (tracks the record being edited).
2. **useEffect**:
   * Calls fetchRecords once when the component mounts to load existing data.
3. **fetchRecords**:
   * Asynchronously fetches records from API\_URL and updates records.
4. **addRecord**:
   * Creates a new record via a POST request, then refreshes the list and clears inputs.
5. **deleteRecord**:
   * Deletes a record by ID with a DELETE request, then refreshes the list.
6. **updateRecord**:
   * Updates an existing record via a PATCH request, then refreshes the list and resets edit mode.
7. **startEditing**:
   * Populates input fields with the selected record’s data for editing.
8. **UI Rendering**:
   * Renders input fields, either “Add Record” or “Update Record” button depending on editingRecord, and a FlatList displaying each record with “Edit” and “Delete” actions.

A screen shot of a computer program

AI-generated content may be incorrect.

**App Component**

* Wraps everything in a NavigationContainer.
* Defines a stack navigator with one screen (“Records”), which renders RecordScreen.

**A screen shot of a computer code

AI-generated content may be incorrect.**

**Section 7: Testing CRUD operations**

Let’s test our full stack mobile application!

**To test Create:**

Add a new record with your name, Software Engineer for position, and Junior for level.

Take a screenshot of your ‘02 Create Record’ as ‘02\_create\_record \_firstname\_lastname.png’ by using your first and last name.

Refresh your app if you do not see any changes.

**To test Read:**

Here’s we can already see list of records. So, we are good here.

Refresh your app if you do not see any changes.

**To test Update:**

Edit your newly created record and change Software Engineer to AI Software Engineer position and Junior to Senior level.

Take a screenshot of your ‘03 Update Record’ as ‘03\_update\_record\_firstname\_lastname.png’ by using your first and last name.

Refresh your app if you do not see any changes.

**To test Delete:**

Hit the delete button and you should no longer see that record you deleted.

Refresh your app if you do not see any changes.

**Section 8: Pushing your work to GitHub**

[How to submit your work in GitHub](https://cityuseattle.github.io/docs/git/github_upload_files/)