

T11: Deployment

- Due Sunday by 11:59p.m.
- Points 8
- Available until Apr 7 at 11:59p.m.

In this lab, you will build a simple React application with a backend in Express.js that allows users to register and log in. This tutorial will guide you through the steps of creating a basic full-stack application with:


- React frontend: A simple user interface for registration and login.
- Express.js backend: To handle user registration, login, and authentication.


You will also deploy both the frontend and backend to make your application publicly accessible with minimal setup.

Learning Goals

- Setting up cross origin resource sharing in the backend.
- Token authentication in the frontend.
- Deployment of a full stack web application.

Setup

Log in to [MarkUs](https://markus.teach.cs.toronto.edu/)  (<https://markus.teach.cs.toronto.edu/>) and go to CSC309. You will find the starter files for this tutorial under the **T11** assignment. Add the starter files to your repository. You will find the following folders and files:

- **backend/**: A simple backend where users can sign up, log in, see their account information, and log out.
 - **controllers/user.js**: controllers for user related endpoints, including registration, login, and viewing one's profile.
 - **middleware/auth.js**: contains the `authToken` middleware for endpoints requiring authentication.
 - **services/user.js**: provides user related services, including creating and retrieving a user, and logging in a user.
 - **routes.js**: defines all routes for the backend.
 - **server.js**: the entry point for the backend server, i.e., we run `node server.js` to start the server.
 - **index.js**: configures app-wide settings. This is the only file you need to change for the backend.
- **frontend/**: a contrived React web application created via [Vite](https://vite.dev/)  (<https://vite.dev/>).
 - **index.html**: The main HTML template file that serves as the starting point for a React application.
 - **public/**: Folder containing static assets, i.e., the favicon for the website.
 - **vite.config.js**: Used to configure Vite-specific settings.

- **src/*.css, src/**/*.css**: The stylesheets associated with each component. We will not explain each one for brevity.
- **src/**: This directory contains the source code for your application.
 - **App.jsx**: This is the main component of your app, serving as the entry point for our UI. The `App` component contains all the other components in your project.
 - **main.jsx**: This file is often the entry point for our JavaScript code, where we import and render the main `App` component.
- **src/components/Layout.jsx**: Provides the overall layout and styling for the application.
- **src/pages**: This directory contains the source code for each page of the website.
 - **Home.jsx**: A simple home page with a welcome message.
 - **Login.jsx**: The login page where the user can authenticate.
 - **Register.jsx**: A page where users can register for an account.
 - **Success.jsx**: The page that is displayed after a user has successfully registered.
 - **Profile.jsx**: This page displays the profile of the currently logged in user.
- **src/contexts**: This directory contains the source code for React contexts, which manage global state and share data across different components.
 - **AuthContext.jsx**: The context for authentication and user registration. This is the only file you need to change for the frontend.
- ***/package.json, */.gitignore**: the usual.
- **Makefile**: provide the clean command for removing generated files.
- **WEBSITE**: the tester will read this file to locate where you have deployed your web application. Fill this out once you have completed the tutorial.

Setup

To install packages for the frontend and backend, run `npm install` in each of the subfolders.

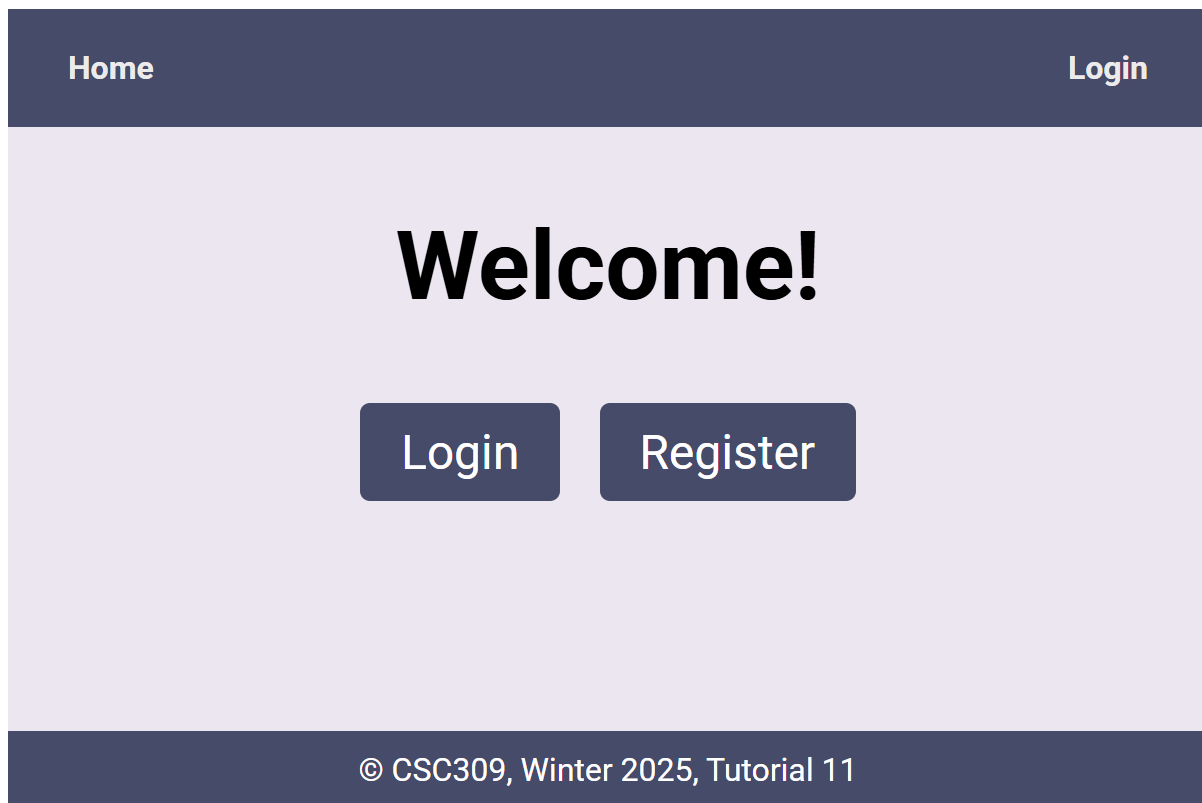
To start the backend server, run the following command in the `backend` folder:

```
node server.js
```

To start the React development server for the frontend, run the following command in the `frontend` folder:

```
npm run dev
```



Next, open your browser and visit <http://localhost:5173/> (<http://localhost:5173/>) to see the home page of this web application.



Your Tasks


The starter code already contains most of the implementations. Your main objective is to connect the frontend with the backend such that they can communicate with each other in a production environment.

Task 1

Currently, the frontend code is unable to interact with the backend API endpoints because they are hosted under different origins (i.e., different domains, protocols, or port numbers). Even during development, the backend server typically runs on <http://localhost:3000> , while the React development server runs on <http://localhost:5173> . This difference triggers the **Same-Origin Policy (SOP)**, a security mechanism enforced by web browsers to prevent unauthorized requests between different origins. As a result, the browser blocks the frontend from making requests to the backend by default.

To enable communication, the backend must explicitly allow cross-origin requests using **Cross-Origin Resource Sharing (CORS)**. CORS works by adding special HTTP headers to responses, specifying which origins are permitted to access the backend.

Make the appropriate changes to `backend/index.js` to use the **`cors`** middleware, so that you can configure the server to only accept requests only from your frontend.

Please refer to <https://expressjs.com/en/resources/middleware/cors.html>  [\(https://expressjs.com/en/resources/middleware/cors.html\)](https://expressjs.com/en/resources/middleware/cors.html) for more information on using the cors middleware.

It is important to use an environment variable to specify the frontend URL, otherwise you will not be able to change it during production. Your server should load environment variables from the `.env` file, and use a default value when the file is missing, e.g.:

```
const FRONTEND_URL = process.env.FRONTEND_URL || "http://localhost:5173";
```

Task 2

At the moment, the frontend is missing code for making API calls to the backend. You need to complete the `AuthProvider` function in `frontend/src/contexts/authContext.jsx`. Before that, you need to specify the backend URL. For Vite, all environment variables must be prefixed with `"VITE_"`, and they can be accessed under `import.meta.env`, e.g.:

```
const BACKEND_URL = import.meta.env.VITE_BACKEND_URL || 'http://localhost:3000';
```

There are three backend API endpoints that you will need to access to complete task 2, as listed below. When an error occurs, all API endpoints will return a 400 level status code, with the following response format:

```
{
  "message" : "the error message"
}
```

/login

- **Method:** POST
- **Description:** Authenticate a user and generate a JWT token
- **Content-Type:** application/json
- **Payload:**

Field	Required	Type	Description
username	Yes	string	The username of a registered user
password	Yes	string	The password of the user

- **Responses**
 - **200 OK** on success

```
{
  "token": "jwt_token_here"
}
```

/register

- **Method:** POST
- **Description:** Register a new user
- **Content-Type:** application/json
- **Payload:**

Field	Required	Type	Description
username	Yes	string	Must be unique
firstname	Yes	string	The first name of the user
lastname	Yes	string	The last name of the user
password	Yes	string	The password that can later be used to authenticate the user

- **Response:**

- **201 Created** on success

```
{
  "message": "User registered successfully"
}
```

- **409 Conflict** if a user with that username already exists

/user/me

- **Method:** GET
- **Description:** Retrieve the information of the authenticated user
- **Payload:** None
- **Response:**
 - **200 OK** on success

```
{
  "user" : {
    "id": 1,
    "username": "johndoe",
    "firstname": "John",
    "lastname": "Doe",
    "createdAt": "2025-02-22T00:00:00.000Z"
  }
}
```

You need to implement the three functions defined in `AuthProvider` and manage the user context state as follows:

- When the user is logged in (i.e., `localStorage` contains a valid token), fetch the user data from `/user/me` and update the user context state with the returned user object.
- When the user is not logged in (i.e., `localStorage` does not contain a token), set the user context state to `null`.

Additionally, ensure that the authentication state persists across hard reloads (e.g., when the user refreshes the page). **Hint:** `useEffect` can be useful for checking the stored token and fetching user data when the component mounts.

function login(username, password)

Implement the login function using the Fetch API to send a request to `/login`. If login fails, return the error message from the response. If it succeeds, you need to do three things:

- Store the received token in `localStorage`. Please use **'token'** as the key, otherwise the autotester will have trouble restoring session state across hard reloads.
- Update the user context state.
- Lastly, redirect the user to `/profile`.

function register({username, firstname, lastname, password})

The `register` function takes an object with four fields, and can be implemented with the `/register` endpoint. Upon success, navigate to `/success`. Otherwise, return the received error message.

function logout()

This function does not require making any API calls. Simply remove the token from `localStorage`, set the user context state to `null`, then navigate to `/`.

Testing

You should make sure that task 1 and task 2 are completed and tested before continuing with task 3. Debugging is much harder in production mode.

Task 3

Finally, you will be deploying your website. For this tutorial, we will use [Railway](https://railway.com/), a Platform-as-a-Service (PaaS) that takes care of most of the deployment details, such as package installation, choice of operating system, physical resource allocation, etc. To deploy, follow these steps:

1. Create a GitHub Repository

Create a GitHub repository with your completed Tutorial 11 code. You can name this repository anything you like. If you do not have a GitHub account, you will need to create one:


- Go to [GitHub](https://github.com/) and sign up for a free account.
- Use your university email address if possible, as it will help you access student benefits.

As a student, you are eligible for GitHub Student Developer Pack, which provides free access to various developer tools, including upgraded GitHub features. To apply:

- Visit the [GitHub Student Developer Pack page](https://education.github.com/pack).
- Click "Get Student Benefits" and follow the instructions to verify your student status.
- Use your university-issued email or upload proof of enrollment if required.

Once your GitHub account is set up, proceed with creating a new repository and pushing your tutorial code.

2. Set Up a Railway Account

Sign up for a [Railway account](https://railway.com/)  (<https://railway.com/>) by selecting the Sign in with GitHub option. This will link your Railway account to GitHub. During the setup process, grant Railway the necessary permissions to access your repositories. This will enable Railway to deploy your project directly from GitHub.

3. Create a New Project

In the Railway dashboard, create a new project where your application will be hosted. Again, you can name this anything you want. Link this project to the tutorial 11 GitHub repository you created during step 1.

Project Settings

×

General

Usage

Environments

Shared Variables

Webhooks

Project Info

Name

csc309-t11

Description

CSC309 Winter 2025, Tutorial 11

4. Frontend Setup

You should first create the frontend service. This can be done by clicking on the "+ Create" button, then choose GitHub repo, then choose your tutorial 11 repo. You can name the service whatever you want, by editing its name. Then, click on the frontend service, go to Settings, and make the following changes:

- **Root Directory:** This should be set to `frontend`, which is the folder containing your React frontend project.
- **Build Command:** `npm run build`. This command compiles your React project
- **Start Command:** `npm run serve`. This serves the built files from the `dist` directory.

Click the **Deploy** button to start deploying your frontend. Once the deployment is complete, navigate back to the **Settings** tab. Under the **Networking** section, click **Generate Domain** to make your frontend publicly accessible. Railway will provide a URL with the following format:

```
https://<service-name>.up.railway.app
```

Open this URL in your browser to view your deployed website. At this stage, the website is live, but it is not yet connected to the backend, so features like login and registration will not work. We will address this next.

5. Backend Setup

Now, let's create the backend service. Then, make the following changes to its settings:

- **Root Directory:** This should be set to `backend`, which is the folder containing your Express backend project.
- **Start Command:** `npm run start`. This will start the Express server.

To allow the frontend to communicate with the backend, we need to configure CORS by setting the `FRONTEND_URL` environment variable in Railway. Go to the **Variables** tab in your backend service. Click on **Raw Editor** and add the following entry:

```
FRONTEND_URL="https://your-frontend-domain"
```

Replace `your-frontend-domain` with the actual domain provided by Railway for your frontend service. Then, deploy your backend. and once complete, make the service publicly available by generating a domain under the **Networking** section.

6. Update Backend URL

Now that the backend has a public domain, we need to configure the frontend to communicate with it. Go to your frontend service in Railway and add the following environment variable:

```
VITE_BACKEND_URL="https://your-backend-domain"
```

Replace `your-backend-domain` with the actual domain of your deployed backend. Then, redeploy the frontend to apply the changes.

Your web application should now be fully functional! You can access it using the public domain of the frontend service, and it should properly communicate with the backend.

7. Update WEBSITE

In your MarkUs repository for CSC309, navigate to the **T11** folder. In it, you will find a file named `WEBSITE`. Edit this file to include the URL of your deployed website. For example:

```
https://your-frontend-domain
```

Make sure the URL is correct and publicly accessible before submitting. This will allow us to verify your deployment.

Submission

We will be looking for the following files in the **T11** directory of your repository:

- `WEBSITE`
- `backend/index.js`
- `frontend/src/contexts/AuthContext.jsx`

We will be using a testing framework, e.g., selenium, to automatically grade your submission. On MarkUs, we will provide *all of the test cases* that will be used to grade your submission. There will be no hidden test cases. You will not be graded on your coding style.