FlyNext: your most reliable travel companion!

Please find the general information about the project, features, interviews, mentor sessions, rules, etc. on the Part 1 handout. All rules and restrictions apply to this part as well.

In this part, you will implement the React frontend of FlyNext and connect it to your backend server. You will also make your app portable via Docker and deploy it to the internet. The deliverable is a portable, deployed, fully functional website that is polished, user-friendly, and production-ready.

Deadline

The submission deadline for Part 2 is **Friday, April 4th at 22:00**. Late submissions are subject to a 10% penalty per day, up to a maximum of three days.

Students registered with Accessibility Services may qualify for deadline extensions. In that case, email the instructor at least one week before the deadline and request the extension. Note that individual entitlements will be adjusted to the size of the team. For example, if one member from a team of two is entitled to 7 days of extensions, the team will be granted $\lceil 7 \div 2 \rceil = 4$ days.

Academic Integrity

Honesty and fairness are fundamental to the University of Toronto's mission. Plagiarism is a form of academic fraud and is treated with utmost seriousness. You are expected to review the handout How Not to Plagiarize and to be familiar with the Code of Behavior on Academic Matters.

You may use online resources, including open-source codes and generative AI tools like ChatGPT, to support your project. However, **sharing any portion of code with other teams, whether giving or receiving, is strictly prohibited**.

Note that all work that was not written by you or your teammates must be cited in the code, including open-source codes and code created by generative AI.

Environment

Your application must run in an Ubuntu 22.04 machine that has Docker and Docker Compose installed. **You should not make assumptions about the existence of Node.js or a database anymore**. It is fine if you are developing your code on other operating systems, although Windows is **strongly discouraged** as its Dockerfiles might not work on Linux. Remember that it is your responsibility to make sure that your server and scripts (described later) run without any problem on Ubuntu 22.04 before submitting.

Part 2 deliverables

At this part, you will implement **all** user stories listed in the part 1. Use React and TailwindCSS to deliver a single-page and responsive UI. You can modify any part of the backend code as well.

The interviews will simulate the real-world delivery of your application to the business owner. That is, a hypothetical business owner will be looking for a complete website that is seamless, secure, efficient, and bug-free.

Updated AFS

Some improvements have been made to AFS, including the addition of new APIs to fetch a flight and cancel a booking. You can access the updated API docs here and the updated Postman collection here, importable to Swagger Editor. All existing APIs are unchanged.

Given the new API, you will need to support the user story where a user can cancel their flight booking as well.

Containerization

Your deliverable is a portable app. a `docker-compose.yaml` file along with your Dockerfiles should be present. The Docker Compose file should include **every software** needed to run your app in a new environment. That includes your Next.js app, the Postgres database (described below), an nginx server to manage the incoming requests, etc.

The Docker build phase should build your Next.js app. In other words, the servers must run in **production** mode to take advantage of all available optimizations.

Using TypeScript

Implement all of your part 2 code in TypeScript. To fully leverage the advantages of TypeScript and Monorepo, migrating your Part 1 code to TypeScript is strongly recommended. However, since that might take a considerable amount of time, you will receive a small bonus if your system is fully type-safe (backend, frontend, and their interaction).

Change of database

In this part, you will switch to Postgres as your database system. You will need to modify the schema file and delete all previous migrations. Please note that all your existing data will be lost. If that data is important, ensure that you create a backup before proceeding.

Local AFS (bonus)

You will receive bonus marks for running AFS locally. To do so, download the code from here and add the AFS setup to your Docker Compose file.

Consult the README file of the repository for instructions on running the AFS server and importing the data.

User experience

You must prioritize user experience (UX) to ensure that users can navigate the application seamlessly without the need for external guidance. Think about the websites you have left because you could not find the page you were looking for, or where parts of buttons were unresponsive due to frontend bugs. In such cases, you likely did not consider the effort spent by the backend and frontend engineers, or the functionality of other parts of the app. You left the website, and that was a loss of a potential customer for that business.

While our evaluation will be constructive and not as harsh, you should be aware that the marking scheme differs from traditional projects, where the primary focus is on whether the code works. You will lose marks if your web app is slow (takes too long for actions to complete), incomplete (lacks essential functionality), or contains major or minor infrastructure/UI/UX issues.

The web application should provide a smooth experience across devices with varying screen sizes. A smooth UX also involves a seamless single-page experience, front-end validations, in-place error handling, asynchronous requests, and efficient re-rendering.

Note: This course is not focused on UI design, so you will not be graded on the visual appearance of the website as a designer would be. However, you must adhere to the common sense in your UI design. Your UI should make a positive first impression, and navigating your website should be an enjoyable experience for

users. A basic UI that clearly reflects the default styles of a framework (e.g., Bootstrap) is not considered an engaging or delightful user experience.

Note 2: Even though this is a single-page application, your website must include distinct pages. The browser's URL should be updated to reflect navigation, allowing users to move through different sections with the back/forward buttons or revisit specific pages via their URL. Different sections of the app must be easily accessible through a navigation bar. A user dropdown for viewing/editing the profile, manage bookings and hotels, and logout is also required. Users should never need to manually modify the URL. Beyond these requirements, the content of the pages and the choice of the landing page are at your discretion. We recommend that the landing page includes a search feature with two tabs for switching between flights and hotels, with results displayed immediately below.

Pre-populated database

Unlike in the previous part, your application should not start with an empty database. Pre-existing data significantly enhances the presentation of your project. In a real-world scenario, business owners would not want to navigate a blank website with only a few entries. To effectively showcase your project, it is recommended that you include at least 50 hotels with complete information across various cities. You may utilize generative AI to assist with generating this data.

However, note that data should be imported as part of Docker compose setup. Do not attempt to push the database into your repository.

Deployment

Your web app should be deployed to the internet. If your Docker Compose is correctly set up, the deployment process should be quick and straightforward. Simply SSH into a cloud virtual machine, clone the repository, and start Docker Compose.

Hosting your website on the internet will be a valuable asset in future presentations or job interviews. However, note that **the TAs still run and test your application locally**. This ensures that your application is truly portable and can easily be started on fresh environments.

Note: You do **not** need to buy a domain. You may provide the IP address of your server instead. Also, major cloud provider (e.g., AWS, GCP) offer free plans. **Do not opt for paid options**, as we will not be able to reimburse any incurred expenses.

Note 2: If your deployed application is accessible via the HTTPS protocol, you will receive a small bonus.

What to submit

You should push the following to your repository:

- The FlyNext source code.
- The AFS source code (if applicable), including the data generation scripts.
- Dockerfiles and `docker-compose.yaml`.
- Hotels data (i.e., pre-populated database) and import scripts.
- A `start.sh` script: This script should be fairly minimal. It should call `docker-compose up -d --build` to build and start the system.
- An `import-data.sh` script: This script call the specific Docker Compose services that import data into AFS (if applicable) and FlyNext.

A `stop.sh` script: This script should be fairly minimal and simply call `docker-compose down` to shut down the system.

A file named `url.txt` that contains the URL of your deployed system.

Important note: Before submitting, ensure that your startup and run scripts function correctly in the described environment. The TAs will test your application on clean instances of that environment. If you have developed on a different operating system, it is your responsibility to double-check that your server works fine on our environment as well.