For reference, when the document references "low-spec hardware", it is referring to computers running with less than 8 GB or ram and anything less powerful than an i5 processor.

Architecture:

Before running the application for development, it's important to understand the architecture so you can diagnose possible startup errors. This section serves as an introduction to the technologies used to build the architecture of this application and what the separate services of the application are.

This application uses ***Docker*** and ***Docker Compose*****.** For those unacquainted with these tools, Docker is a tool used to run applications on a computer by emulating a completely separate operating system environment, allowing an application to run on any computer such that they need only have Docker installed on their device. Docker Compose is a tool used to run multiple Docker Containers. In our application we also make use of the feature where Docker Compose creates a network separate from that of the host computer. This document will go over the basics needed to start the application, But it is important to familiarize yourself with Docker and Docker Compose if you will be developing and maintaining the backend and networking components of this application.

This application uses a microservice architecture such that each of the core services of the application are handled with a different service. The microservices are:

- **Login Microservice (Port 13126):** Used to log users in and give them the proper privileges based on their login credentials

- **Course Manager Microservice (Port 13127):** Used to create and modify any data related to a course
- **Course Viewer Microservice (Port 13128):** Used to fetch data relating to a course that gets displayed to the frontend
- **Peer Review Teams Microservice (Port 13129):** Used to create and manage the teams for a course's peer reviews
- **Professor Assignment Microservice (Port 13130):** Used to allow professors to create and modify assignments for a given course
- **Student Assignment Microservice (Port 13131):** Used for students to upload and retrieve assignment submission data
- **Student Peer Review Assignment Microservice (Port 13132):** Used for students to upload and obtain data related to submissions for specifically peer review assignments.
- **Frontend (Port 13125):** Used to interact with the backend

Each of these microservices are run in their own container in the Docker Compose network.

Inside the Docker Compose network we also host five *separate* ***MongoDB*** instances that hold all the data for the application. Each backend microservice has the ability to access any of the microservices. The databases are found at the following ports in the Docker Compose network and store the following information:

- **27037:** Student information
- **27038:** Professor and admin information

- **27039:** Course information
- **27040:** Assignment and submission information
- **27041:** Peer review teams information

Along with this, there is currently an ***Nginx*** layer that all requests made to the microservice are routed through. Nginx is a web server application and the application uses it to route all requests to the microservices inside the Docker Compose Network. The following figure shows the entire system architecture if the Nginx web server was set to run on localhost:3000. Note that the diagram lists an email microservice which is used to send email alerts. This service has been developed, but is not production ready, so it is not included in the main build of the application. There is also a Discord bot microservice that has been made, but that service has also not been deemed ready to be included in the final production environment.

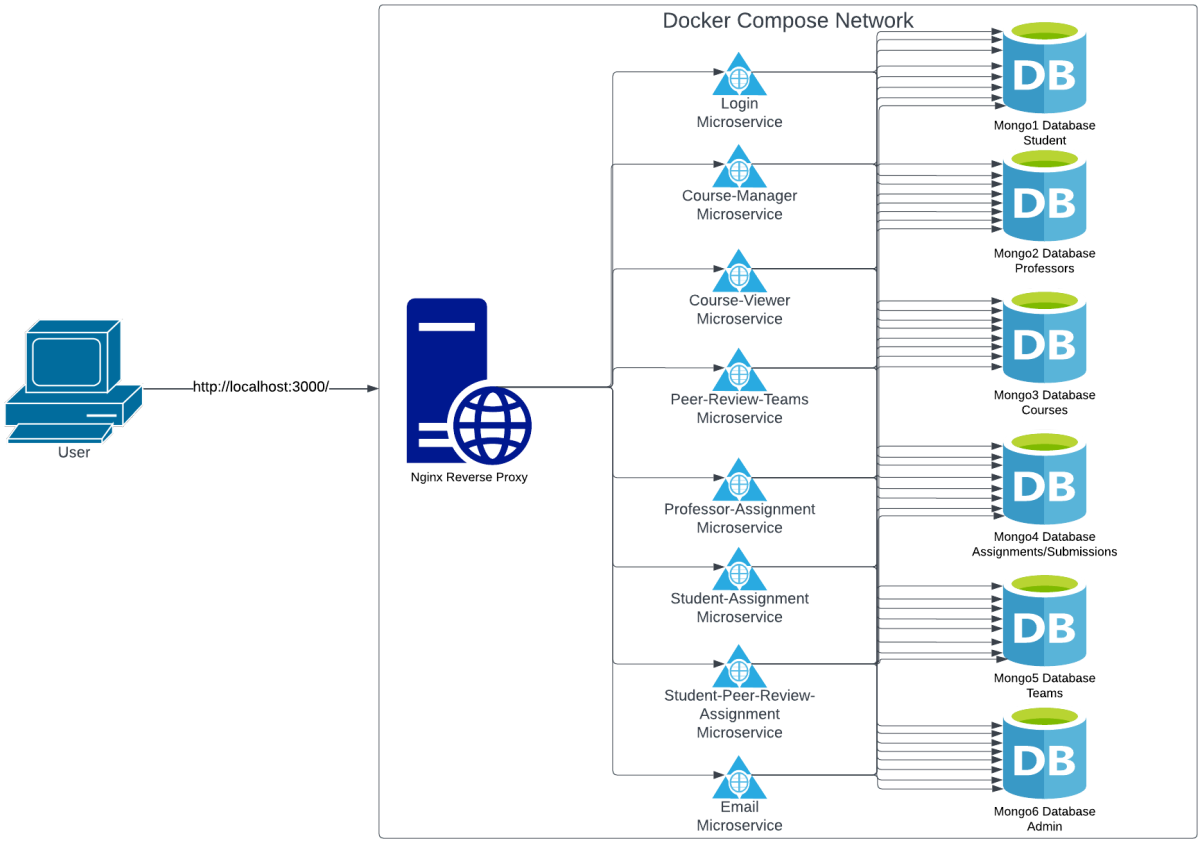

Figure 1: Local CPR Networking and Database Architecture (Diagram Link)

Building & Running the Application:

Before building, there are some things to first consider. This application is running a lot of services at the same time on top of Docker. The application may run very slow if you are on a lower-tier computer. Consider using resources available to you by the school if the application does not run well on your computer. In the HCI lab, for example, there are Desktop computers available that should be able to run the application fine. It is also important to note that students running on Apple Silicon (i.e M1 or M2 Macs) may have trouble running the application. Docker works differently for computers running with this chipset, and we did not have enough people with Apple silicon computers to be able to properly develop environments to run the application in, so there is potential for the application to not run on these computers and further deployment development will be required for them.

The main ReadMe file explains how to run the application using the provided shell scripts. These Shell scripts were made with the thought in mind that someone will be able to redeploy the application without needing any prior docker knowledge, so that is why there are so many of them. The functions of each of the scripts are as follows:

build-app.sh: Runs the script to build the app for a production environment.

build-app-local.sh: Runs the script to build the app for a local environment.

build-app-db-and-proxy.sh: Runs the script to build only the Nginx proxy server and databases so that the developer can run the backend services and the frontend outside of the Docker Compose network.

independently-run-db.sh: Runs the script to only turn on the databases used for the application. This is handy when just needing to check how data is formatted in the databases.

mongo-changepwd.sh: Runs the script to change the database passwords for the application

mongo-init.sh: Runs the script to initialize the databases. It is required to run this script the first time the databases are established on the system the application is running on.

rebuild-app.sh: Runs the script to rebuild the app for a production environment. This is needed in order to redeploy the application for production without needing to acquire the PEM keys needed for HTTPS connection every time an update to the application is made.

run-frontend-proxy.sh: Runs the script to start up the Docker Compose network so that it only contains the databases, Nginx reverse proxy, and backend microservices. This is so that the frontend can be run outside of the Docker Compose network for ease of development for the GUI team.

tear-down-app.sh: Runs the script to tear down the Docker containers spun up by the application.

Note that in the scripts folder, there are “M1” variants of the necessary shell scripts in an attempt to make the application Apple silicon compatible, but they are largely untested since we did not have the required resources to test them.

Tips For Successful Development:

This application has a lot of points of failure initially because it requires having all the technologies. Our team was mostly successful in getting the app to run for everyone, but there were still a few people who had problems running the app throughout the semester. We have compiled a list of tips to help remedy this for the next group so that problems are caught before they become too unmanageable.

- If you can, consider running the application on a Linux or Unix operating system. This application was initially designed to primarily run on a Linux server, so it has been modified to be able to run on nearly any other operating system without breaking the deployment functionality that already preexisted. It still runs the best on a Linux/Unix-based OS since Windows requires Docker Desktop, which in turn requires WSL. The app is still fine to run on Windows, but it may be more smooth if you are able to run it with, for example, Ubuntu.
- If you are on Windows, WSL 2 and Ubuntu for Windows are tricky to get working right the first time around. The guide posted in the ReadMe to install these applications works for some computers but not others, as we have come to find out.If you are a developer on Windows and find that the app is not working properly or running particularly poorly, ensure that WSL 2 and Ubuntu for Windows are properly installed and configured with Docker Desktop. Though WSL and Ubuntu for Windows are resource heavy, they should not completely cripple your computer - unless you are running on low-spec hardware - and are likely not installed properly if they are.
- For the production build, you will need the required PEM keys since Google login requires an HTTPS connection. Email Dr. Lea for these keys and he will make

them available to you. After this, anytime you update the application on the production server, use the **rebuild-app.sh** script, it will redeploy the app with the PEM keys you initially received from Dr. Lea so that you do not need to keep requesting access to the keys every time you redeploy.

- Get some Continuous Integration tools up and running. We had a few instances in the semester where redeployment would fail because code would fail to compile. This would have been caught using some Continuous Integration tool like CircleCI or even just the GitHub actions CI tools to make sure code would compile before merges. It would be a good idea to make this a priority in the future.
- Assemble a small deployment team to help people with start up and to fix any deployment issues that may be found. We had one person who was handling deployment of the application for everyones' computers and it had eventually become unmanageable. A lot of issues would have been fixed if there were two or three people who had a good knowledge of how deployment for this application works. At the start of the semester, it would be a good idea to volunteer three people to focus on the deployment of the application and how it works so that the team has any of their deployment issues fixed as soon as possible.