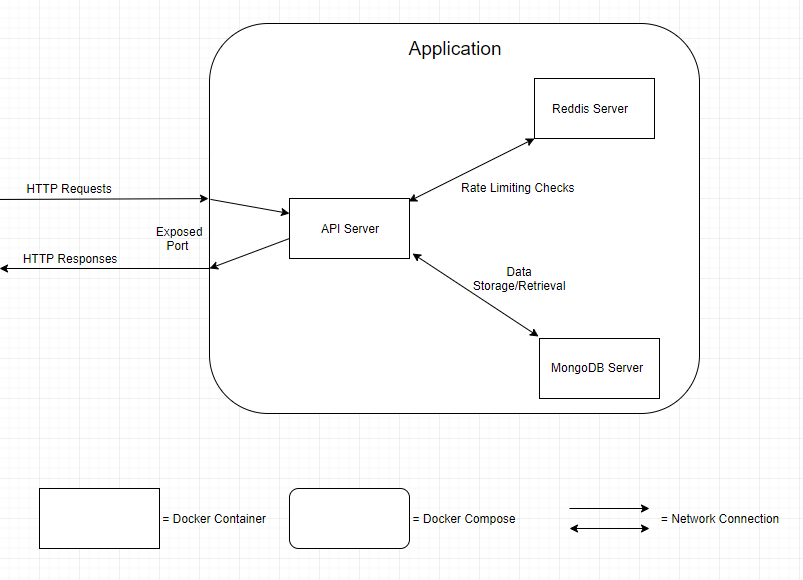
CS 493 Final Project  
Team Clout Dev  
-Christopher Hoskins, Kennedy Vandel, Michael Zavalza

# API architecture/design document

In addition to your API implementation, your team must also submit an API architecture/design document. This document should include the following sections:

* **API architecture diagram** – this should, at a high level, depict the major components of your API (e.g. API server(s), database server(s), etc.) and indicate the relationships between these components.
* **API data layout** – this should describe, either pictorially or in text, how application data is organized within your API database (e.g. data schema, links between entities, etc.).
* **API design reflection** – this should describe why you chose the architecture and data layout you chose and reflect on what parts of your design worked well and what parts you would change, and how you would change them, if you were to implement the same project again.

**API Architecture Diagram**



**API Data Layout**

The data is all stored in a mongodb server. There are 3 collections in the database for dealing with normal structured data. These are users, courses, and assignments

The Users collection contains 5 fields: \_id, name, email, role, password. The id field, which represents the user’s unique id number, is of data type ObjectId. There name field contains a single string with the user’s full name. The email field contains a string of the user’s email, which must be unique to the user. The role field contains a string of the user’s role, which must be either “user”, “instructor”, or “admin”. The password field contains a string of the user’s hashed password.

The Courses collection contains 7 fields: \_id, subject, number, title, term, instructorId, enrolled. The id contains a unique course id of type ObjectId. The subject contains a string for the subject of the course, and the number contains an integer for the specific number of the course within the subject. The title contains a string with the name of the course. The term contains a string with which term the course is being offered. The instructorId contains the id of the instructor user who is assigned as teaching the course. The enrolled field contains an array. This array contains the ids of the students who are enrolled in the course.

The Assignments collection contains 5 fields: \_id, courseId, title, due, points. The id contains the assignment’s unique id of type ObjectId. The courseId contains the id of the course for which the assignment is for. The title contains a string of the name of the assignment. The due field contains a (date format) for when the assignment is due. The points field contains an integer of how many points the assignment is worth.

**API Design Reflection**

Overall, we decided to play it fairly safe and stick with tools we learned in the class. Even while structuring the files of our API server, we chose to mirror the layout of the code written in lectures as well as the starter code given for assignments.

The primary reason we decided to use a mongodb server for data storage was the file storage requirement with the assignment submissions. The only good way we learned for storing files in a database was with GridFS in mongo, so that is what we went with. Without the file storage requirement, we would have likely gone with mysql.

For the database, we relied heavily on the ObjectId provided to every document in hardcoding the initial database, we created ObjectIds using a generator created by “@hugodf”1. Hardcoding these values made it easier to create tests, and easier to reference documents while building the project. In addition to using the distinct ObjectIds as a key, we also kept some unique fields. In the *users* collection, the email field is unique, as the email is what is used for logging in users. Similarly, the *courses* collection is unique on the subject and number fields. Even though there would be several instances of a class across multiple terms or sections, for this project we felt it was reasonable to keep these fields unique.

One thing that ended up being somewhat suboptimal at times was the way we chose to implement student enrollment in classes. Having the enrollment field as a part of the course object added a lot more work when doing things such as finding all courses that a specific student is enrolled in. It might have been more efficient to have enrollment be its own table, going with somewhat of a more sql like schema for it all.

The rate limiting was implemented using a reddis server similar to the extra credit for assignment three, but in this case the rate limiting was performed by IP regardless of whether an authentication token was provided or not. A majority of the code was adapted from the reddis lecture.