

Design Document - Team 33

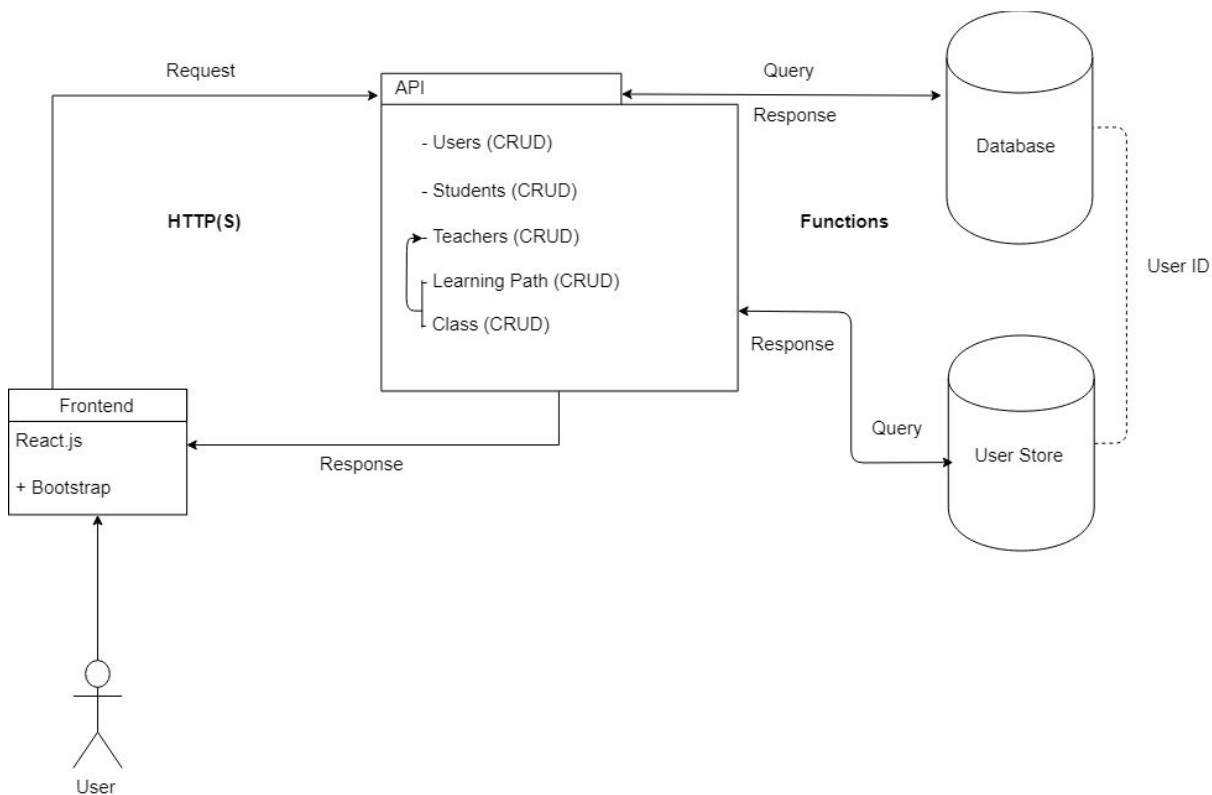
Ludus

Alex Brooke, Shaural Patel, John Shen, Alex Willis, Kevin Xue

Purpose:

Students will be able to search for and learn from a variety of different courses with different teaching styles pertaining to the subject of their choice. Teachers will be able to create unique content for their classes and present all their work to the students in an organized structure based on a tree diagram, building a base for knowledge and stemming outwards into any more complex sub-topics a student is interested in. The goal is to create a unified location for any learning related needs on any topic, while supporting the students as they progress.

Design Outline:



The application will be split into two major sections— the frontend and the backend.

Frontend

The frontend section of the application will contain all code related to the user interface. This section will be written in React.js (a component-based JavaScript framework) and use Bootstrap (an HTML/CSS framework for reactive styling of web pages). The frontend will communicate with the backend through the use of an API (described below), by making HTTP(S) requests to specific URL endpoints. In doing so, the user interface can fulfill its purpose to organize and display data to the user in an intuitive manner.

Backend

The backend section of the application will be separated further into three major components— the Database, the User Store, and the Application Programming Interface.

Database

The database will store all data necessary for the application to function. This database will be an instance of Firebase's Real-Time Database service. A description of the schema can be found below.

User Store

The user store will securely store user account credentials. Each user will be associated with a Unique ID (UID) that can be linked to a user record in the database. This store will be an instance of Firebase's Authentication service. Using this service will allow users to log in to our web application using accounts from other providers, including Google, Facebook, Twitter, as well as their email and a password.

Application Programming Interface (API)

The API is intended to provide a layer of abstraction that bridges the gap between the backend and the frontend (graphical user interface). The API will consist of a series of functions meant to map database and user store queries and responses to HTTP(S) requests and responses that exist on the internet as a set of *endpoints*, represented by URLs. These functions will provide a consistent and simple method of reading and updating information in the database to the frontend code, reducing the risk of programming errors. The functions will be hosted using an instance of Firebase's Functions service. A detailed description of the API's functions can be found below.

Design Considerations:

Issue: Hosting of large files (videos etc.)

Option A: Outsourcing/hosting offsite

Option B: Storing and hosting files in the cloud

Option C: Storing and hosting files on a physical server

Justification: Choosing to outsource hosting of media to other sites would reduce the technical overhead of managing content ourselves, but this would lessen our control over the content and would also require users of our site to have accounts on other sites. Storing and distributing media from a physical server is another solution, but this has the drawback that our application would be tied to some physical infrastructure, which makes resource scaling a tricky— upgrading may require some server downtime and would be an added expense. Therefore, we decided to host media content in the cloud, specifically using Firebase's Storage service. This is a compromise between the other options, whereby we control our own delivery system, but we don't need to worry as much about resource scaling or upkeep cost. Further, using Firebase's solution for storage will allow us to keep our application's components well-integrated (see below).

Issue: Course Rating Scale

Option A: Stars

Option B: Likes

Justification: A 0-5 star rating is intuitive and can better describe the quality level of a course or learning path than a simple like or dislike. The information is more useful and can also be stored to provide further analytics later on in the project if desired.

Issue: Should we prevent certain malicious content from being added to the site?

Option A: Hard Restriction

Option B: Content Warning

Option C: No restriction

Justification: Creating a filter to prevent any offensive content from being posted to the website would be tedious and not fit within the scope of our project. Inappropriate material is still an issue however, so we decided to implement a content warning/user filter to allow users to opt in/out of potentially offensive content (18+ content gets flagged, user has option for the content to be visible to them).

Issue: Should we use a relational database?

Option A: Non-relational

a) MongoDB

b) Firebase Real-Time Database

Option B: Relational

a) MySQL

b) PostgreSQL

Justification: Our logical design involved having a tree-like structure which will be holding large volumes of information. A non-relational database like firebase makes this easier to implement and scale out for maintainability. Furthermore, our other components are going to leverage firebase services, so using Firebase's database service allows us to maintain tight integration between components. Finally, we decided to go with the 'serverless' implementation which would prevent us from having to maintain a server ourselves while also solving the problem of resource scalability (see below).

Issue: Programming language for the front-end:

Issue: Frameworks for the front-end

Option A: ReactJS

Option B: AngularJS

a) Sub option 1: Use Bootstrap

b) Sub option 2: Design from scratch

We will use Bootstrap because it provides us with an easy way to style UI elements in a clean and sophisticated manner. We do not have an experienced UI designer and therefore it would be much more challenging creating a good design from scratch using HTML and CSS. We will use ReactJS to structure the interactive parts of the user interface because it will allow us to write code in small, encapsulated components that can be easily reused.

Issue: Programming language for the backend**Option A: JavaScript/Node.js**

Option B: Python

Option C: Java

Justification: Most of our team has had experience using Node.js in our previous web development projects, so the technical overhead of building the project will be relatively low. NodeJS also provides us with all necessary functionality, has a robust package ecosystem, and allows for smooth integration with Firebase.

Issue: Serverless or Hosted model**Option A: 'Serverless' implementation**

Option B: Hosted implementation

Justification: The 'serverless' model of backend design is gaining popularity in the industry. When an application is run on a traditional server ('hosted' model), resource scaling becomes a problem. As the load on the server increases past its rating, response times slow until the application can be migrated to another server with more resources. The 'serverless' model generally involves defining a set of functions, hosted by a cloud provider, that run only when certain triggers are activated (generally these are HTTP(S) requests), as opposed to the traditional server model that necessitates running a webserver at all times.

Issue: Authentication and secure password storage

Option A: Store user information directly in the database (after encryption)

Option B: Use an authentication framework (e.g. an OAuth implementation)

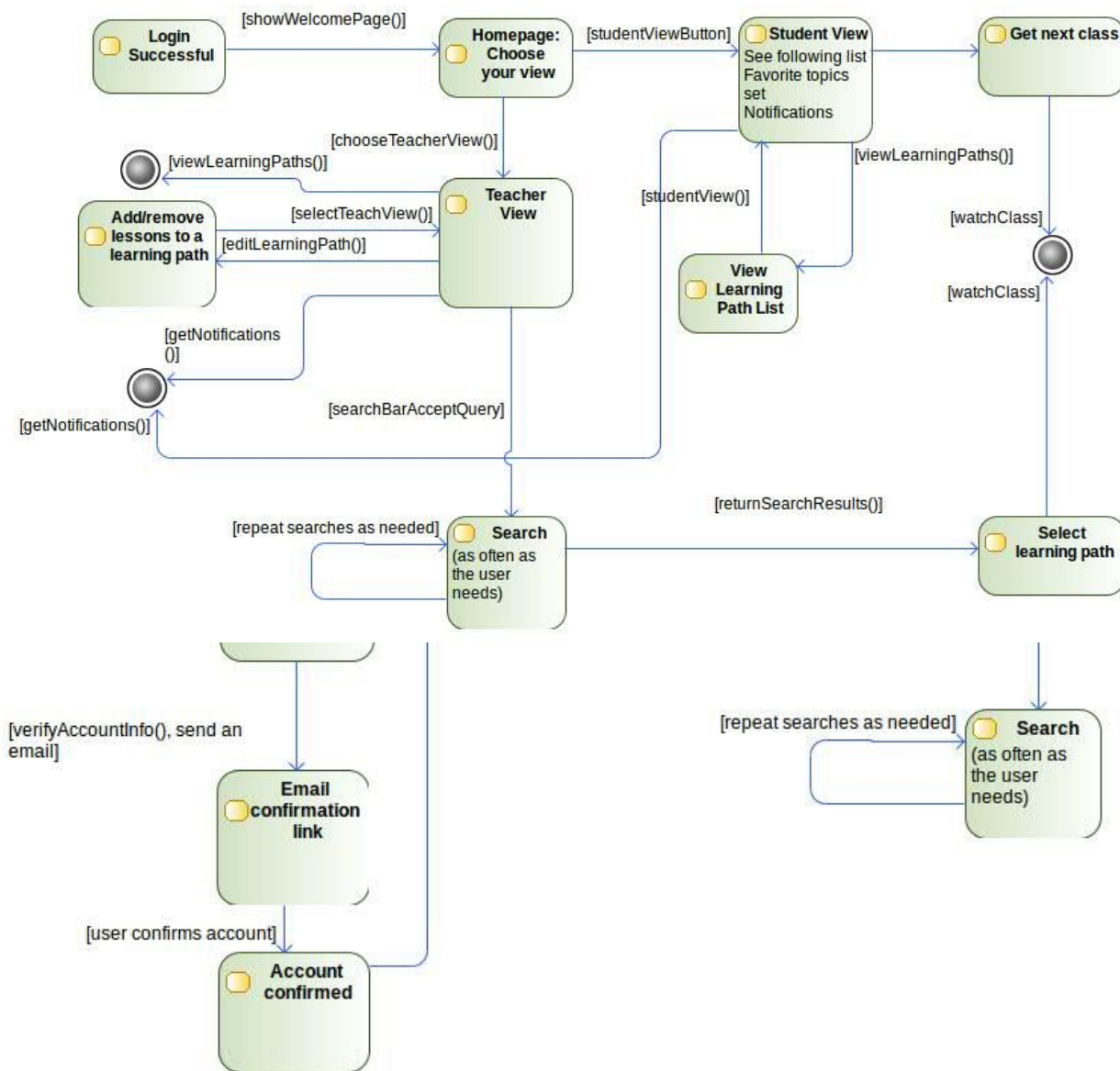
Option C: Use Firebase's Authentication Service

Justification: While it would be possible to store user login information directly in our database, we would be liable for making sure said information is kept secure. Even if we encrypt the information, this option would likely be less secure than other options. Using a pre-existing authentication framework is a possible solution to this, but we would still need to store references to users somewhere, and this would likely force users to use an account from an external provider in order to log into our site, which may be undesirable. Using Firebase's Authentication Service solves all of these issues. It is hosted and managed by Google, a trusted provider, and it would allow users to use their email as well as external providers, such as Google, Facebook, and Twitter to log into our site. Further, each user who registers is allotted a Unique ID which can be used to easily match an authenticated user with a user record in our database.

Design Details:

Site Layout:

Continued (to the right of "Login Successful")...



The UML diagrams above detail the various actions a user may take in the use of the site. A user will initially be presented with a site homepage [abstracted by the starting node]. If a user does not already have an account, they would then be directed to create an account. A new user shall provide credentials as previously described by the database schema. After submitting the account data, the user will be sent a link to their email account, which they would then need to click to confirm their account

Assuming a user either already has an account, or has followed the steps in the previous paragraph, logging in will present a homepage. The homepage has the options to choose either student view or teacher view. In student view, the user can

- Search for new learning paths
- Access notifications, as well as interact with them as needed (for example, reply to message)
- View and alter the teachers that they are following, represented as the “Following list” in the diagram
- View and alter topics that they are interested, in order to better locate learning paths that interest them

For teacher view, the teacher has the following paths on the site

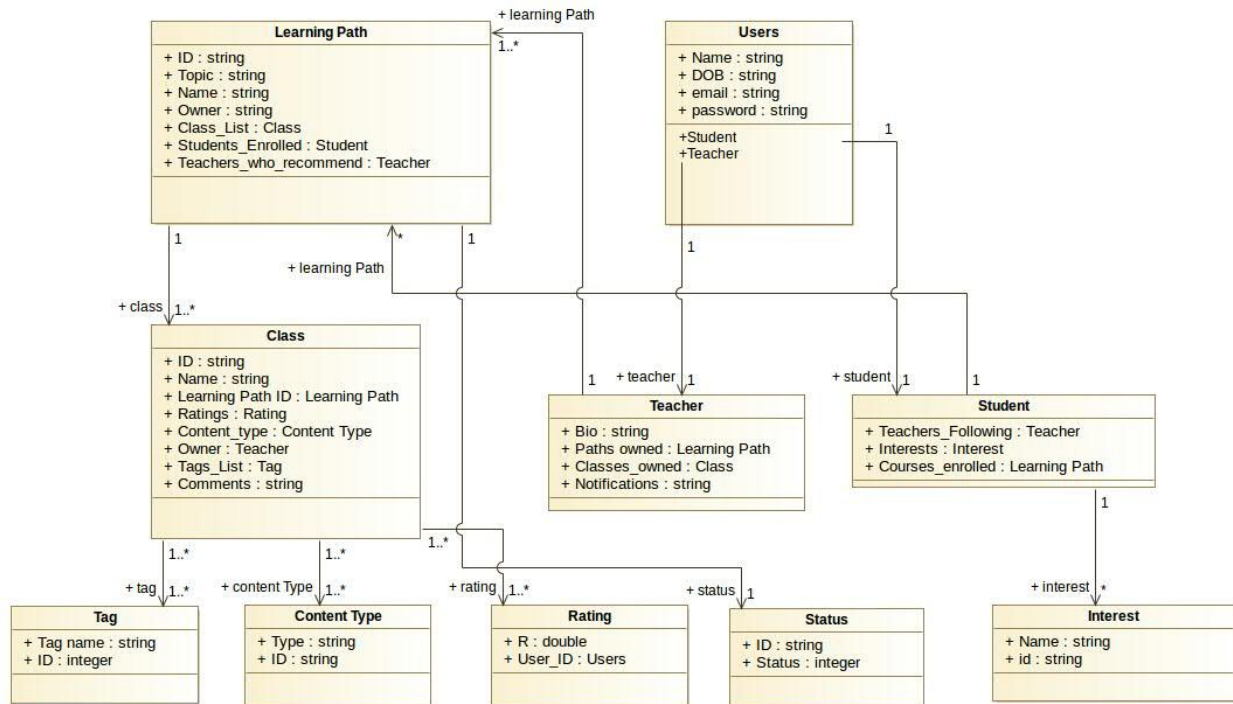
- Add/Edit learning paths
- Get notifications
- Search for content
- Special permissions to make recommendations for learning paths [abstracted under add/edit learning paths in the diagram]

For searching, a teacher or student may make an arbitrary number of searches for learning paths, and once a desired learning path is found, the user may select “watch class” for any given class in a learning path. Once a user does that, they may continue to follow the guidelines of the learning path for as long as necessary

Our functional requirements consists of the following:

1. Users shall be able to make accounts
 - a. Each account shall be able to enroll in as many courses as the user wants
 - b. Each account shall be able to upload material to create a new course
 - c. An account shall be able to both enroll in courses and act as a teacher for other courses
2. Each course must fulfill certain criteria
 - a. Each course must be accurately tagged in accordance with the subject matter
 - b. Potentially offensive content must be correctly tagged, so that the user has the option to filter it out of search results
3. The user shall be able to search for learning paths
 - a. There shall be a search engine that allows users to search by tags
 - b. Each user shall also be able to search according to their self-reported interests and learning styles
4. User account creation
 - a. Passwords for new accounts must meet certain criteria [length ≥ 10 , at least one upper/lowercase letter, and at least one special character]
 - b. All new users must be sent an email with a link to confirm their accounts to prevent spam
 - c. Users shall be able to delete their accounts on demand

Database Schema Mockup:



Description of database hierarchy:

1. User
 - a. Each user account needs to have a name, password, email, and date of birth
2. Student
 - a. A student is created by default. Also, a student needs to have a list of learning paths that they are enrolled in (can be empty), a list of interests, and a list of teachers that they are following (but not necessarily enrolled in their learning paths)
3. Teacher
 - a. A teacher creation requires a short “about me” as their bio, preferably to list their qualifications in the field they want to create learning paths for.
 - b. Upon creation, a teacher will have a list of classes and learning paths owned, and a list of notifications pertaining to them
4. Learning Path
 - a. A learning path is a complete course created by a teacher for an arbitrary topic
 - b. A learning path can only be made by a teacher, never by a student
 - c. A learning path may have any number of students enrolled in it, as well as an list of teachers who recommend it
5. Class
 - a. A class is an individual lesson in a learning path
 - b. A teacher may add an arbitrary number of lessons to a learning path
6. Tags
 - a. Classes must be tagged with the subject matter of the lesson (such as physics, chemistry, or juggling)
7. Comments
 - a. Users may comment on a class, with notifications going to the teacher
8. Rating
 - a. Classes may be rated from zero to five stars. Teachers shall be notified of ratings as they appear (but not of the user who gave the rating)
9. Content Type
 - a. Each class must be marked with the correct content type (video, quizlet link, pdf, etc). This allows users to search by content that may be more appropriate for their learning style
10. Status
 - a. The ID field shall contain either true or false to mark enrolled or not enrolled

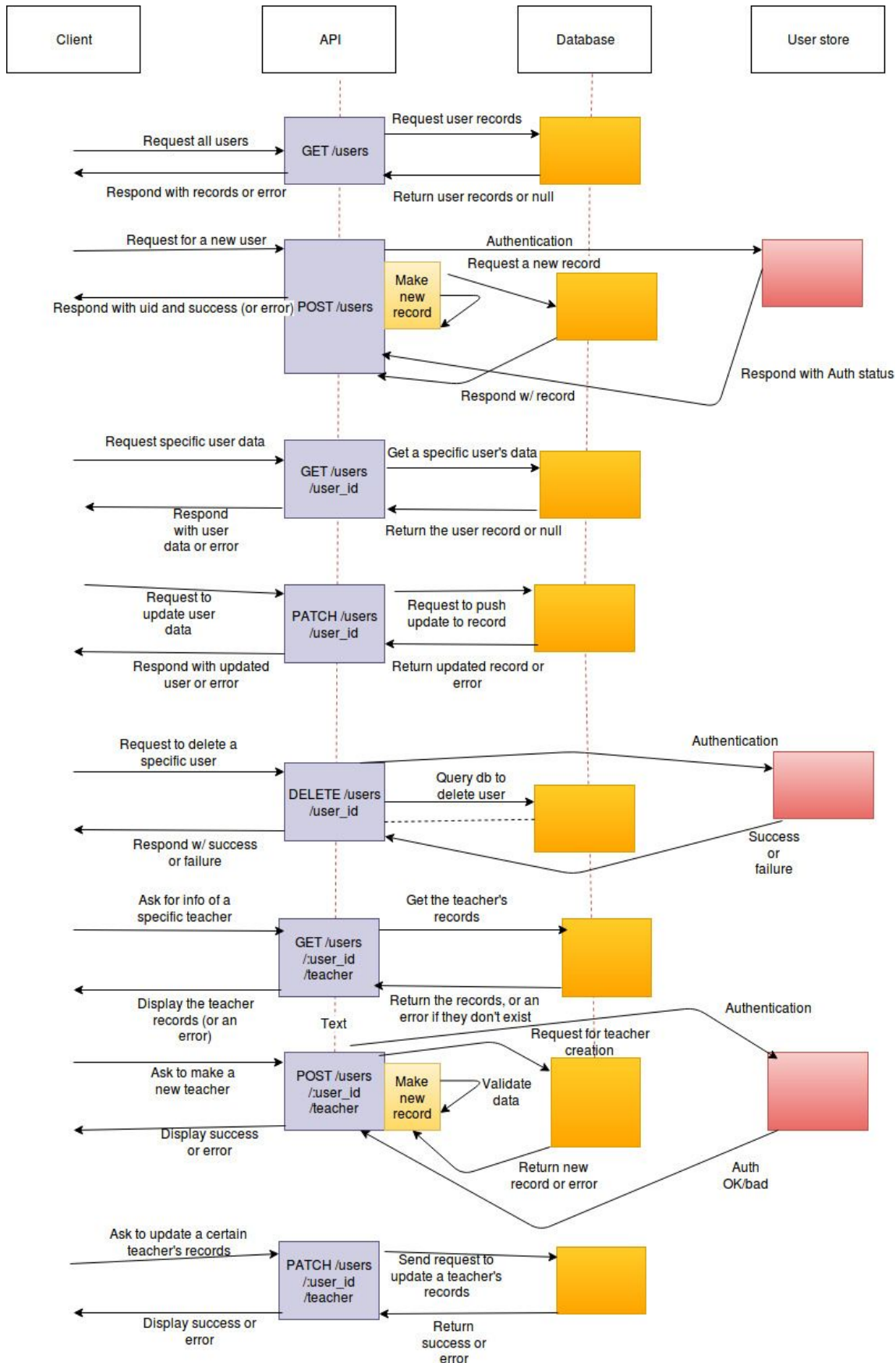
API Detail

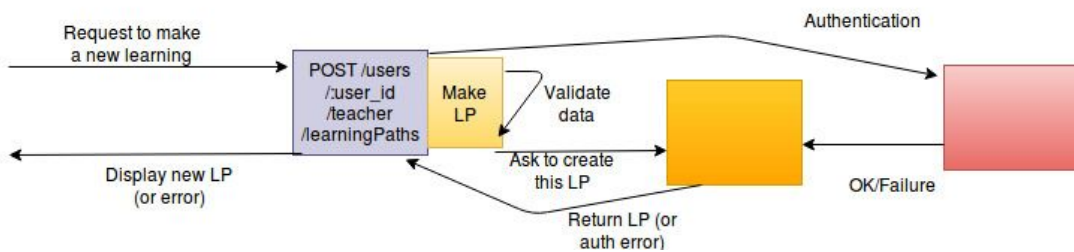
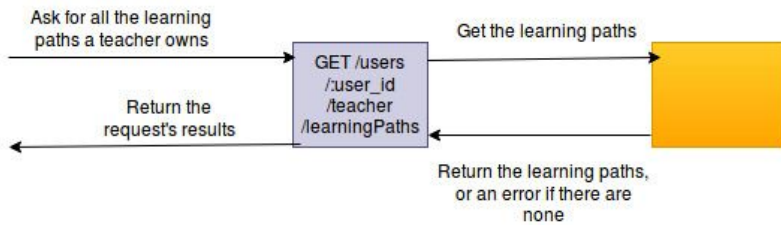
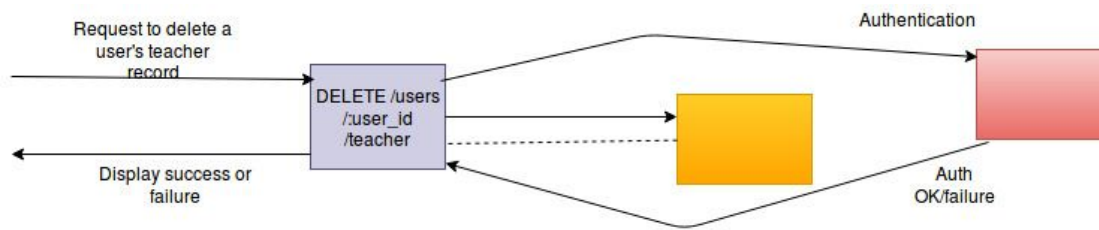
The following is a list of the endpoints that will be provided by the API, and the HTTP(S) methods each will support:

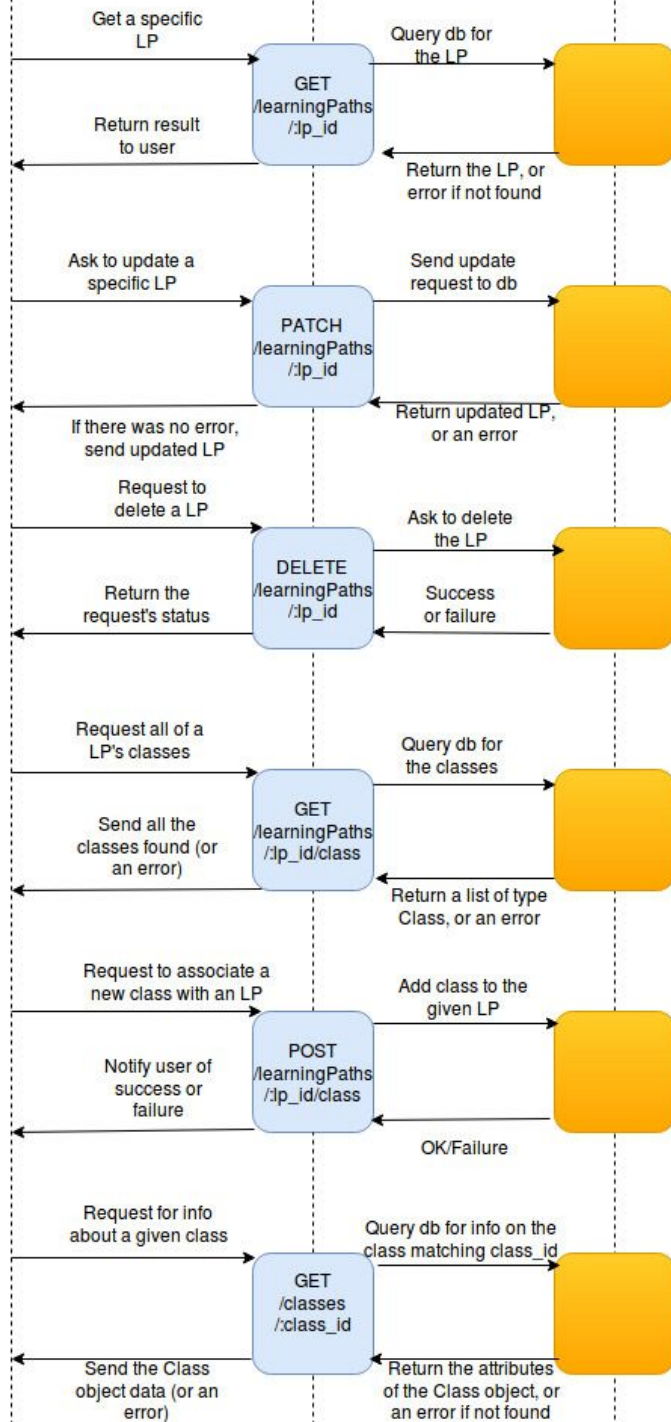
- /users
 - Methods:
 - GET: list all users
 - POST: create a new user
- /users/:user_id
 - Methods:
 - GET: list top-level information about a specific user
 - PATCH: update top-level information about a specific user
 - DELETE: delete a user
- /users/:user_id/teacher
 - Methods:
 - GET: list specific information about a user's teacher record
 - POST: create a new teacher record associated with a user
 - PATCH: update information about a user's teacher record
 - DELETE: delete a user's teacher record
- /users/:user_id/teacher/learningPaths
 - Methods:
 - GET: list all learning paths owned by a teacher
 - POST: create a new learning path owned by a teacher
- /learningPaths/:lp_id
 - Methods:
 - GET: list information about a specific learning path
 - PATCH: update information about a specific learning path
 - DELETE: delete a learning path
- /learningPaths/:lp_id/class
 - Methods:
 - GET: list all classes associated with a learning path
 - POST: create a new class associated with a learning path
- /classes/:class_id
 - Methods:
 - GET: list information about a specific class
 - PATCH: update information about a specific class

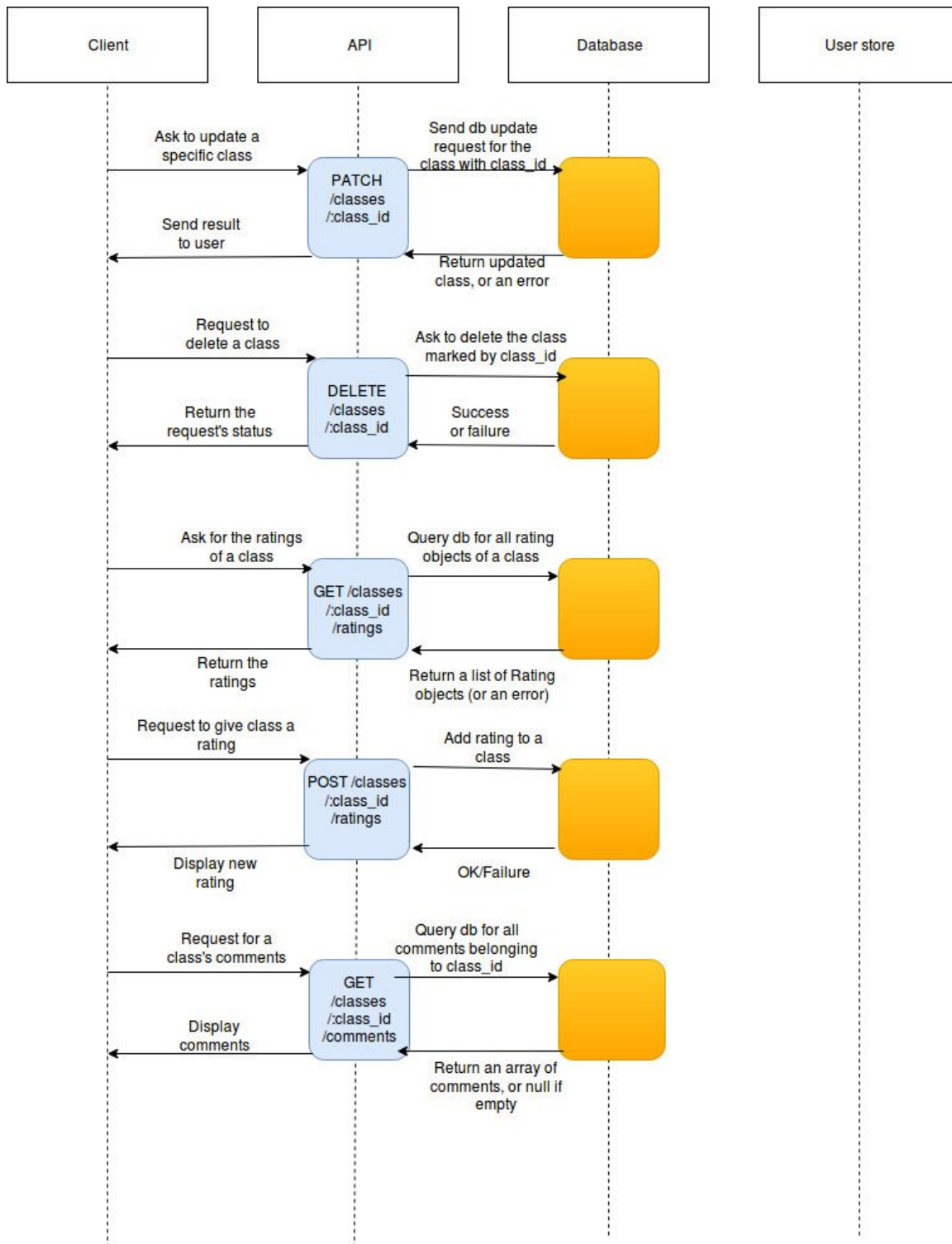
- DELETE: delete a class
- /classes/:class_id/ratings
 - Methods:
 - GET: list rating information about a specific class
 - POST: create a new rating for a specific class
- /classes/:class_id/comments
 - Methods:
 - GET: list all comments associated with a class
 - POST: create a new comment on a class
 - DELETE: delete a comment on a class
- /ratings/:rating_id
 - Methods:
 - GET: list information about a specific rating
 - PATCH: update information about a specific rating
- /users/:user_id/student
 - Methods:
 - GET: list specific information about a user's student record
 - POST: create a new student record associated with a user
 - PATCH: update information about a user's student record
 - DELETE: delete a user's student record

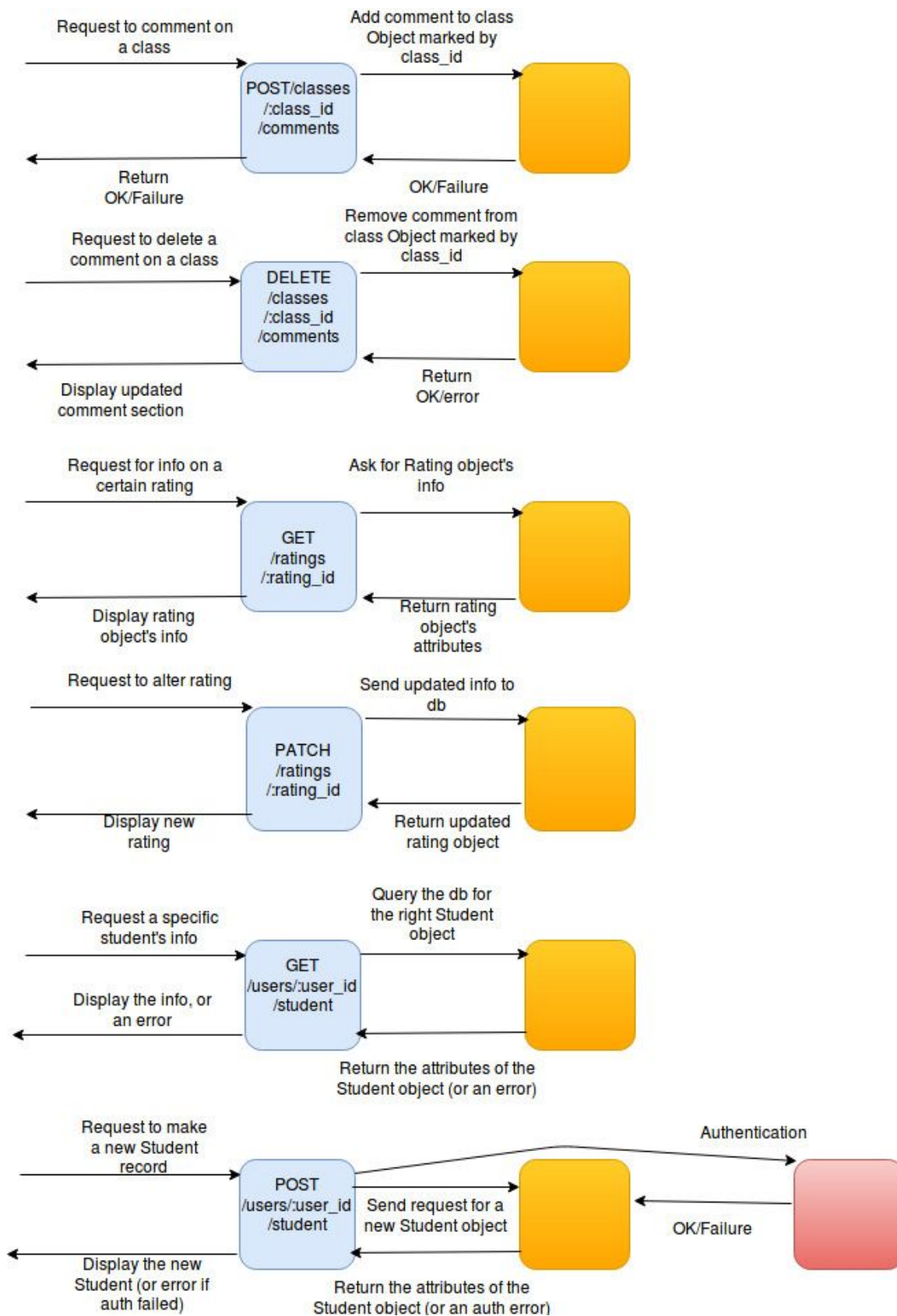
Below is a functional diagram of each API endpoint:

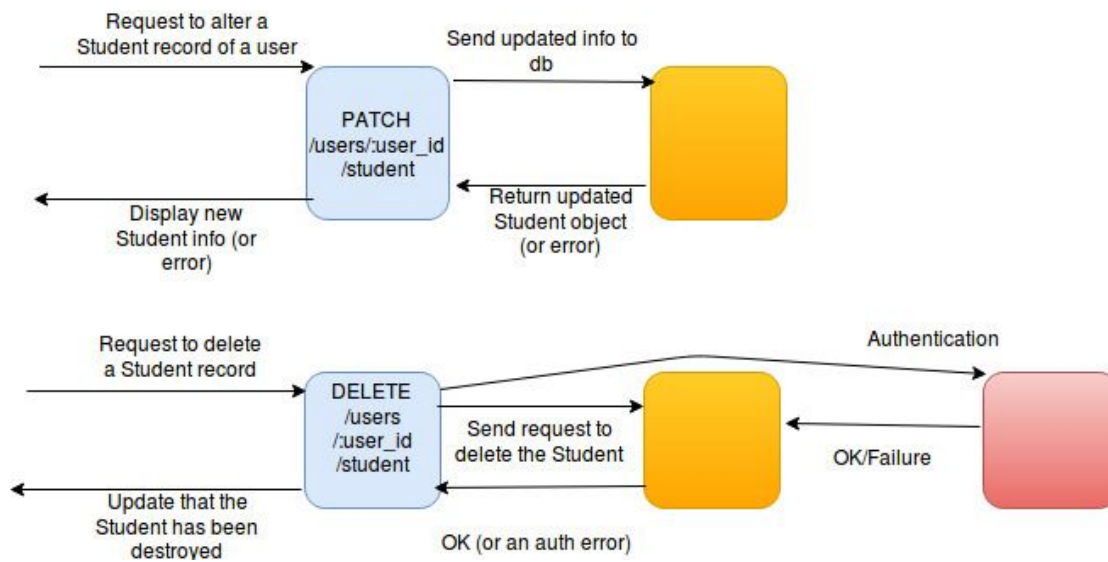






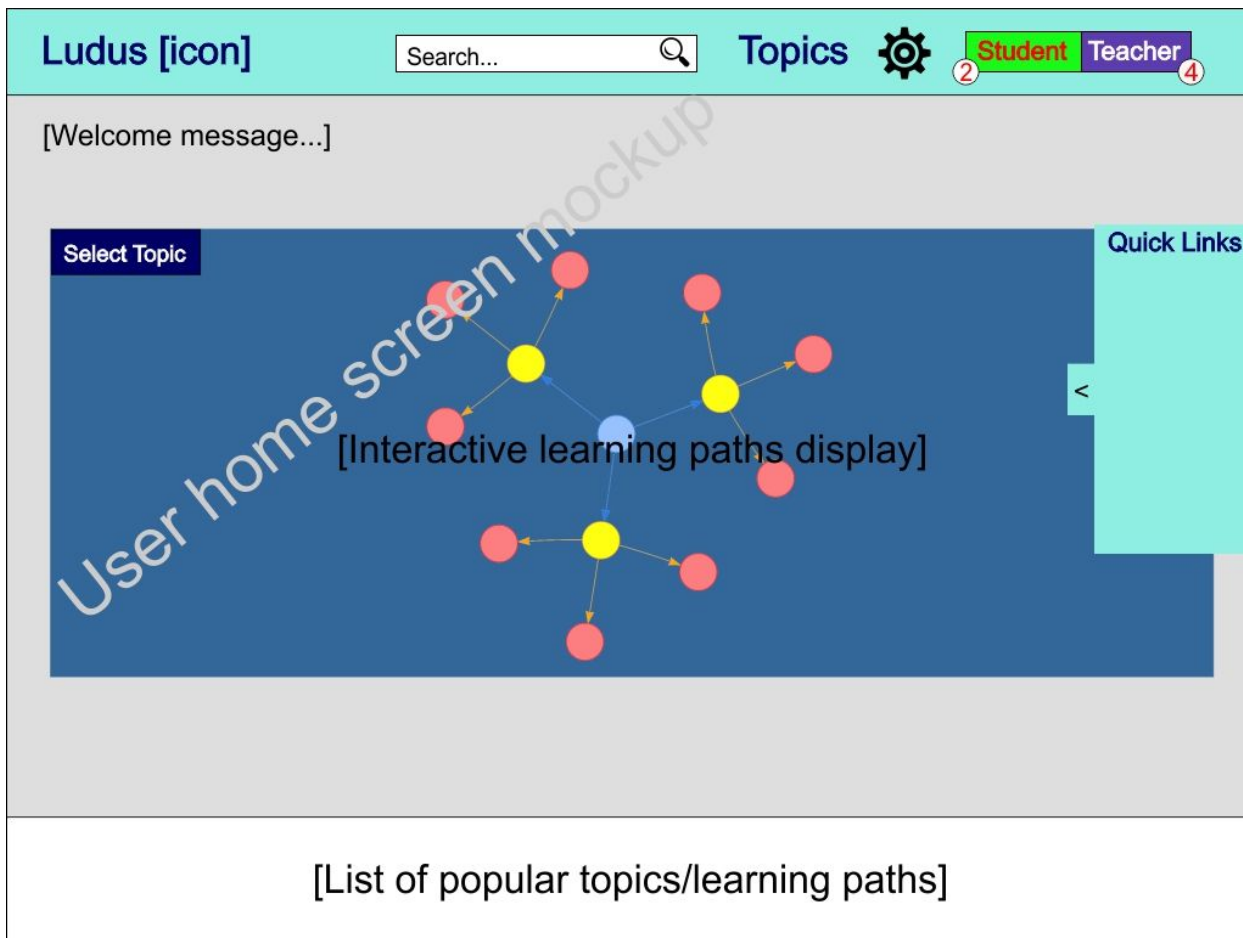






UI Mockups:

Home Screen after login:



After you have logged in, you will be routed to this page, which will contain some quick links on the right which can be minimized, your notifications will be available on hover of “Student” or “Teacher” (as well as on click on the little red numbers).

We will display an interactive map of the learning paths associated to the topic you select (From the dropdown at the top left).

At the footer, there will be a list of the most popular topics and learning paths, as well as some other links like the contact us and site map.

Prior to logging in the section from “Topics” to “Teacher” will be replaced with Login/Signup links which will trigger a popup to sign in or log in.









Teacher Dashboard:

Ludus [icon] **Topics** **Student** **Teacher** ² ⁴








Teacher Dashboard [Create a new Learning Path](#)

My learning paths: Owned | Not Owned

Sailing

 →  →  →  →  →  →  → 

Surfing

 →  →  →  →  →  → 

My Classes: [Create a new Class](#)


Sailing 101 Tags:tag1, tag2,...	LP: Sailing Comments: [comment...]	Content Type: Video Lecture	4.3/5
Catamaran Basics Tags:tag1, tag2,...	LP: Sailing Comments: [comment...]	Content Type: Video Lecture	4.7/5

[List of popular topics/learning paths]

This is the teacher dashboard, you can do all the functions of a teacher here. Create a new learning path on the top right as well as creating classed, deleting, editing them, managing learning paths, approving classes suggested for the learning paths you own, adding a class you made to someone's learning path and more...

Student Dashboard:

Ludus [icon]

Topics 

Student 2

Teacher 4

Student Dashboard

View my Skill Tree

Next Course in learning path:

Catamaran Basics
Tags:tag1, tag2,...

LP: Sailing
Comments: [comment...]

Content Type: Video Lecture
Teacher: Antony Smith

4.7/5

Threads/Concurrency
Tags:tag1, tag2,...

LP: C/C++
Comments: [comment...]

Content Type: Video Lecture
Teacher: Prof. Gustavo

4.7/5

Random Interesting Learning paths:

Notifications

[List of popular topics/learning paths]

This is the student dashboard, learning paths will be suggested based on learning style and topics of interest. For students already enrolled in a learning path, the next class in their learning path will be suggested. Notifications, if applicable, will be presented here.