HUST Remote Study Project

# Functionalities:

## Student:

* Login, logout.
* View class list, lecture list of each class.
* Attend an on-going lecture: allowed actions:
  + Chat.
  + Audio/video conference.
* View previous lecture on demand.

## Teacher:

* Login, logout.
* View class list.
* Create/Delete lectures.
* Modify lectures’ content (slides/materials) and schedule.
* Start a live lecture. Allowed actions:
  + Live stream video.
  + Control slides.
  + Chat.
  + Control audio/video conference.

# Architecture: Web 2.0:

## Client:

* Provide UI for each functionality:
  + Login, logout.
  + Live/on demand lecture.
  + View classes.
  + Manage lectures.
* Exchange data with server.

## Server:

Server will be divided into components which in charge of only one domain business logic, also known as microservices architecture.

1. User service:

Business logic related to users (teachers and students) such as log in, log out, change preferences,..

1. Lecture service:

Core service control logic related to lectures such as CRUD lecture, slides, …

1. Chat service:

Messages (text, voice or video) communication in and outside a lectures

1. Main service:

Serve static page, …

Some main functions

* Communicate with database & data storage.
* Manage lecture info and lecture resources (video, audio, slides).
* Communicate with other existing services (hust-edu…)
* Provide APIs to communicate with client.
* Stream live video.  
    
  Tech stack:
* Services are written in NodeJS
* Client uses ReactJS
* Database: Google Cloud Firestore (next generation Datastore) store data as document and collection in JSON format (similar to MongoDB)
* Cache: Google Cloud Memorystore
* Serve file: Google Cloud Storage
* In future (if possible) CICD: Cloud Build + Deployment

## Communication methods between client and server:

* HTTP request for basic functionalities.
* Web socket for real-time functionalities (live lecture).

# Database Design:

## Lectures:

// TODO

## Action:

* lectureId
* username/id.
* timestamp.
* actionType.
* actionData.

# APIs:

## WebSocket APIs:

* 2 types of broadcast rooms:
  + Course: users will be notified when a live lecture starts.
  + Live lecture: Synchronize client UI between users attending a live lecture.
* Event types:

|  |  |  |
| --- | --- | --- |
| Event | Client | Server |
| **Student login** | Emit “logged-in”.  Data: username, auth data. | Register to broadcast room of student’s available courses. |
| **Student logout** | Emit “logged-out”.  Data: username, auth data. | Remove from broadcast room of student’s available courses. |
| Student join lecture | Emit “join”. Data: lecture id, auth data. | Emit “join” to all users in room. Data: lecture id, user info, (error).  Register to broadcast room of live lecture. |
| Student left lecture | Emit “leave”. Data: lecture id, auth data. | Emit “leave” to all users in room.  Data: success/fail. Remove from broadcast room of live lecture. |
| Lecturer start lecture | Emit “start”. Data: lecture id, auth data. | Emit “start” to all online user belongs to the class. Data: lecture id. |
| User perform action | Emit “action”. Data: lecture id, auth data, action type, action data. | Broadcast “action” to all users in room. Data: username, timestamp, action type, action data. Save action for replay purpose. |

## REST APIs:

// TODO