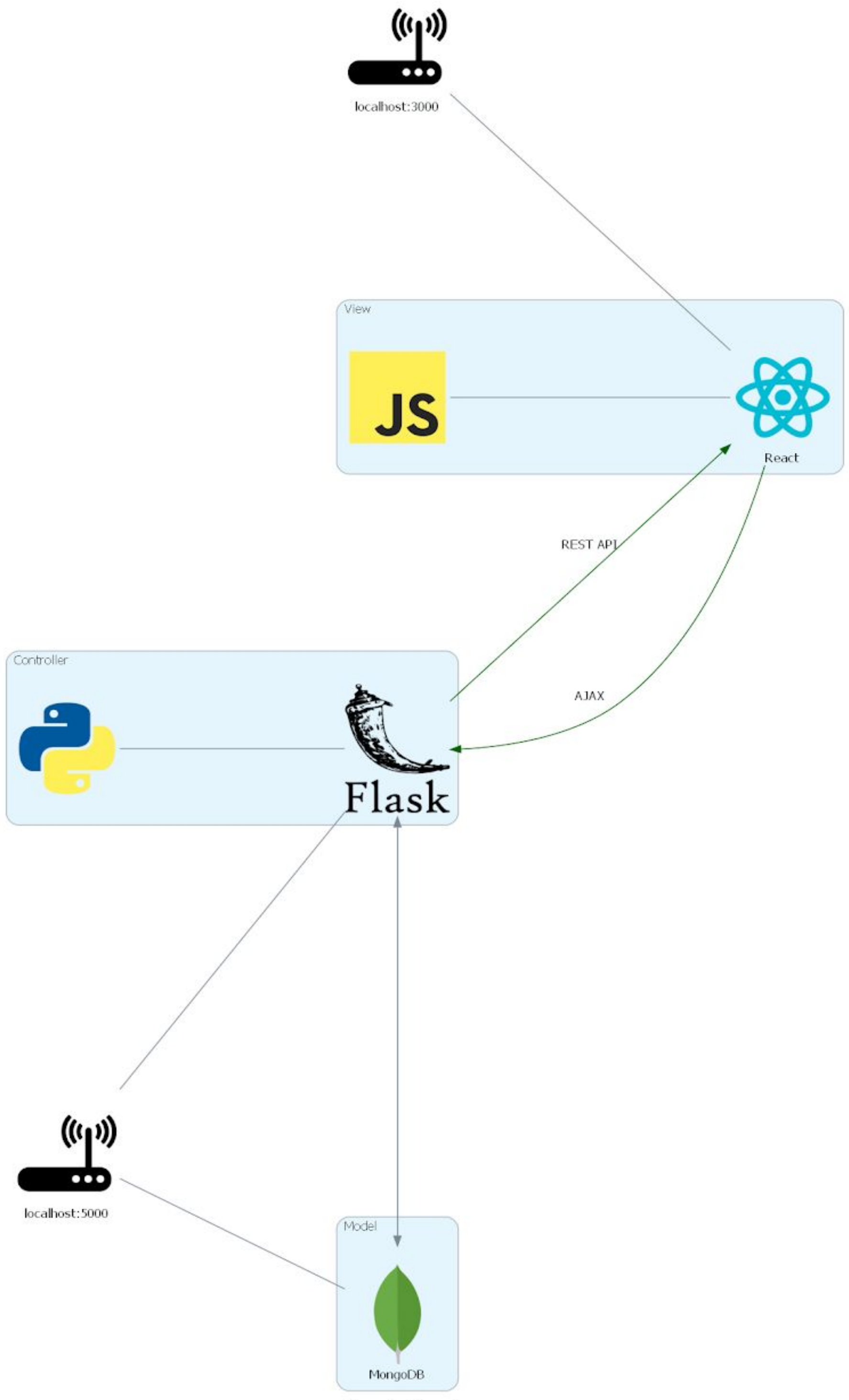
LUKE WARM BEANS

UIMPACTIFY

SYSTEM DESIGN DOCUMENT

Table of Contents:

System Architecture



Architecture

CRC Cards

NOTES:

- Model represents the data in code.
- Database holds the data.
- Schema implies that the class is responsible for saving its information to the database

Class name: **UserController**

Responsibilities:

- Get all user information
- Receives user information for storage in database
- Delete user from database

Collaborators:

- UserModel

Class name: **UserModel**

Schema:

- email
- pass
- name
- phone
- roles -- replaces Access in the template code -- [“Student”, “Instructor”, “Organization”]

Responsibilities (if any):

- Generating a password hash
- Check the password hash
- Saves user information to the database

Collaborators:

- UserController

Class name: **OpportunityController**

Responsibilities:

- CRUD Opportunities
- Query for volunteer opportunities

- Query for paid opportunities (jobs)
- Query for list of applicants to opportunity
- Receive an application from a Student

Collaborators:
- OpportunityModel

Class name: **OpportunityModel**
Schema:
- paid (boolean)
- description -> “paid hourly , internship, beanbags...”
- applicants
- organization - user ref
- published (boolean)

Collaborators:
- OpportunityController

Class name: **CourseController**
Responsibilities:
- CRUD course
- Get all courses taught by a particular instructor
- Get all courses containing a particular student
- Get all courses endorsed by a particular organization
- Get all courses used for training by a particular organization
- Enrol Students in courses
- Drop Students from courses
- Add/Remove endorsements by organizations
- Add/Remove courses as training for organizations

Collaborators:
- CourseModel
- ContentController

Class name: **CourseModel**
Schema:
- name - text

- objective - Text field
- learningOutcomes - Text field
- instructor - id
- students - list of student user ids
- endorsedBy - list of organization user IDs
- trainingFor - list of organization user IDs
- published (whether the course can be enrolled in or not)

Collaborators:

- CourseController

Class name: **FeedbackController**

Responsibilities:

- Add feedback to a course

Collaborators:

- FeedbackModel
- CourseController

Class name: **FeedbackModel**

Schema:

- comment - string
- user - user reference
- course - course reference
- public - boolean

Collaborators:

- FeedbackController

Class name: **QuizController**

Responsibilities:

- CRUD quiz info
- Get all quizzes for a specific course (by id)
    - Restrict access to course creator

Collaborators:

- QuizModel

Class name: **QuizModel**
Schema:

- name
- quizQuestions - list of multiple choice (embedded doc)
    - question - string
    - index - index of question
    - options - list of multiple choice options (embedded doc)
        - index
        - option - string
    - answer - index of correct answer
- published (whether the quiz is available to students or not)
- course - course reference

Collaborators:

- QuizController

Class name: **QuizSubmissionController**
Responsibilities:

- Create a submission for a quiz by a student
- Get a submission for a quiz by a student (detailed info)
- Get all submissions by a user (overview list)
- Get all submissions to a particular quiz
- Get quiz average

Collaborators:

- QuizSubmissionModel
- QuizController

Class name: **QuizSubmissionModel**
Schema:

- quiz - quiz ref
- user - user ref
- answers - list of answers (embedded doc)
    - question - index
    - answer - index of selected answer
- grade

Collaborators:

- QuizSubmissionController

Class name: **AssignmentController**
Responsibilities:

- Get assignment average
- Get all submissions
- CRUD assignment file / UploadFormat

Collaborators:

- ContentController
- AssignmentModel

Class name: **AssignmentModel**
Schema:

- dueDate
- studentGrades
- studentSubmissions
- uploadFormat
- assignmentFile (for download)

Collaborators:

- AssignmentController

Class name: **VideoController**
Responsibilities:

- CRUD url

Collaborators:

- ContentController
- VideoModel

Class name: **VideoModel**
Schema:

- url

Collaborators:

- VideoController

Class name: **EmbeddedTextController**
Responsibilities:

- CRUD text content

Collaborators:

- ContentController
- EmbeddedTextModel

Class name: **EmbeddedTextModel**
Schema:

- Used for articles / readings / random shit

Collaborators:

- EmbeddedTextController

Class name: **React**
Responsibilities:

- Gets all data from server
- Sends data to server
- Is pretty :) <3

Collaborators:

- *Controller

Design Patterns

Application Factories

https://flask.palletsprojects.com/en/1.1.x/patterns/appfactories/

- Provide a malleable structure to our application configuration
- Allows use of different settings in different environments
- Improves testing by allowing different settings for different test cases
- Can run multiple instances of the application in one process

High Order Functions / Decorator Functions

- Allows writing class methods with single purpose in mind (serving resources) instead of constantly rewriting code for authentication, command creation, etc.
- Allows dynamic authentication requirement assignment at run time in a flexible way (not tied to a subclass structure)
- Examples:
    - @jwt_required() to authenticate api routes on flask server
    - @click.command('cmd') to create commands usable in CLI

Application Structure

Frontend Architecture:

- Courses -- Components that represent interacting with courses (creating courses, listing available courses, etc.)
- Course Content -- Components that represent interacting with course content like videos, quizzes, assignments
- Landing -- Components that represent the landing page on the website (home page, login, sign up)
- Dashboards -- Pages which collect a bunch of components to provide a more unified UI
- Jobs -- Components that represent interacting with job and volunteer opportunities

- App -- Root component. Provides a router for all urls on the react application and serves relevant components for those urls (Courses.js, Home.js, etc.)

Structure:
Public folder: holds files that are served at the root of the single page
- Favicon.ico - is the icon that appears in a chrome tab
- Index.html - root html file that servers the built javascript from React
- Robots.txt - relevant for preventing webscrapers, not important for our purposes
- Manifest.json - more icon settings / config

Src folder:
- Courses -- Holds components and relevant files that represent interacting with course content
    - Courses.js -- overall view for a list of courses and creation
        - Holds CourseList & CreationForm components
    - CourseList.js -- view of a list of courses and an add new course option
    - CreationForm.js -- view of form for creating courses
- Landing -- Holds components and relevant files that represent the landing page on the website
    - Home.js -- view for the website homepage
    - About.js -- view for additional information about the homepage
- App.* -- root component, test and css files

Provides a router for all urls on the react application and serves relevant components for those urls (Courses.js, Home.js, etc.)
- index.js root javascript file provides App.js
- index.css root css file, affects styles of all other components

Package.json -- holds information on relevant node dependencies as well as describes the npm commands that are available (npm start, npm test)

Backend Architecture:

- Controllers and Models as described in CRC Cards

Structure:
Backend root:

- .flaskenv -- settings to be used in the virtual environment pertaining to flask. Since python-dotenv is installed, the python environment has access to the .env files (which are necessary for running flask commands)
- README.md -- explaining parts of the backend service that are more complicated and links to relevant documentation
- requirements.txt -- dependencies for a prod environment
- setup.py -- setting up dependencies and ensuring that development environment meets certain version requirements

uimpactify -- project folder:

- controller -- files pertaining to the controller of the MVC structure
  - routes.py -- defines all api routes made available by the server and links them to specific controller classes
- models -- files pertaining to the model of the MVC structure
- __init__.py -- creates the application factory and a default configuration to create a basic development environment
- api.py -- creates some flask commands to run some API requests pythonically (login, signup, etc.)
- db.py -- defines a init-db command to create an admin user on the database. Also defines methods for getting and setting the global database object available in the flask app

Change Log

Sprint 2 Changes

- Added Feedback which references the course it is feedback for
- Removed CourseContent (split into individual content types)
- Reworked Quiz and split into Quiz Submissions

Sprint 3 Changes

- Slight changes to OpportunityModel to be more consistent with field naming
- Added a responsibility to QuizController