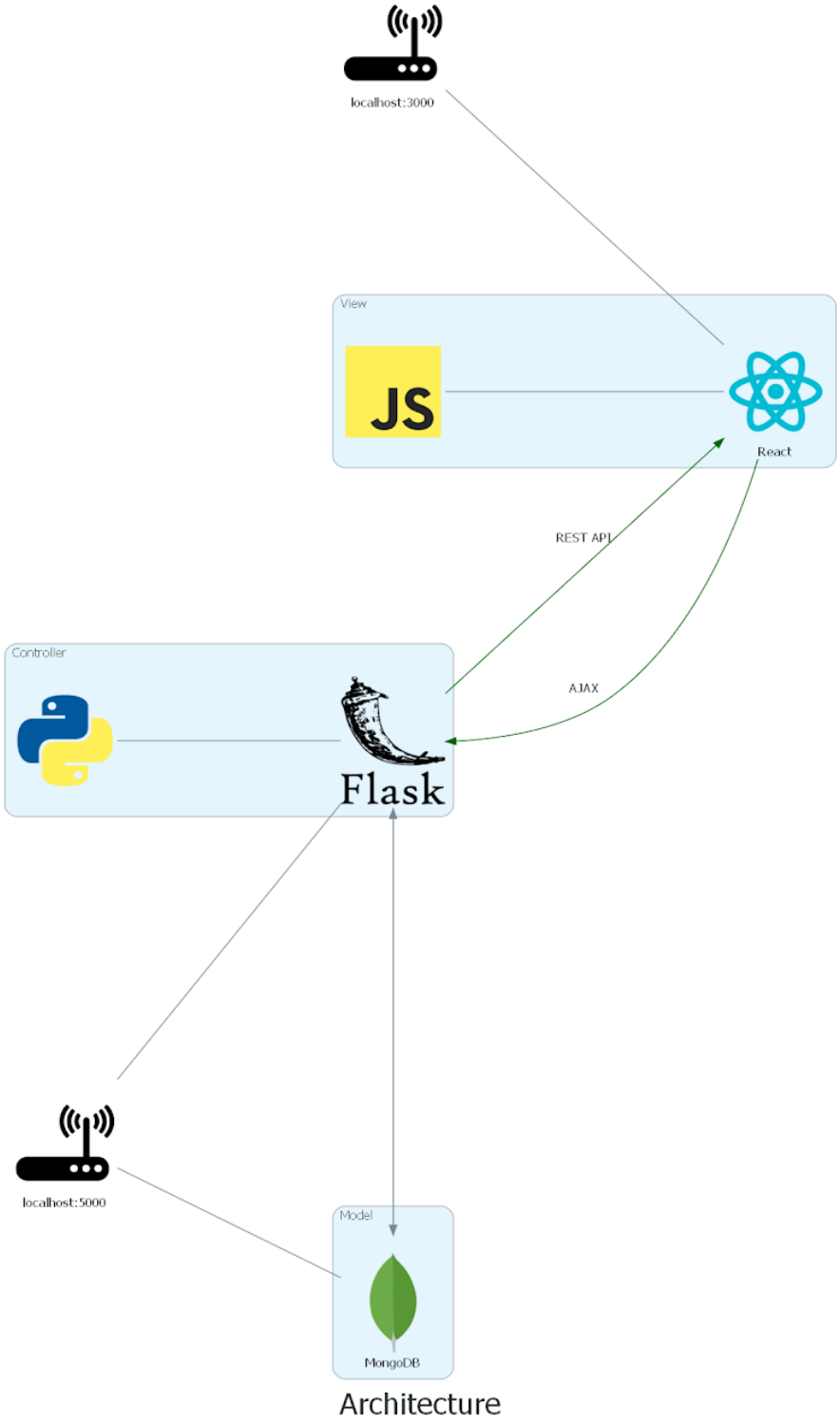
LUKE WARM BEANS

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: **React**
Responsibilities:

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

Collaborators:

- *Controller

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:

- UserID
- Email
- Pass
- Name
- Phone
- Roles -- replaces Access in the template code -- [“Human”, “Instructor”, “Organization”]

Responsibilities (if any):

- Generating a password hash
- Check the password hash

- Saves user information to the database

Collaborators:

- UserController

Class name: **HumanController**

Responsibilities:

- Get all enrolled in courses
- Get all applied to opportunities

Collaborators:

- HumanModel
- CourseController

Class name: **HumanModel**

Schema:

- UserID - required from user model
- EnrolledCourses

Collaborators:

- HumanController

Class name: **InstructorController**

Responsibilities:

- Can create courses
- Get all instructed courses

Collaborators:

- InstructorModel
- CourseController

Class name: **InstructorModel**

Schema:

- UserID - required from user model
- TaughtCourses

Collaborators:

- InstructorController

Class name: **OrganizationController**
Responsibilities:

- Get all organization information
- Receive organization information for storage in database
- Delete organization from database
- Add endorsements for courses
- Remove their endorsements from courses
- Add volunteer opportunities
- Remove volunteer opportunities
- Add job opportunities
- Remove job opportunities
- Add training courses
- Remove training courses

Collaborators:

- OrganizationModel
- CourseController
- HumanController

Class name: **OrganizationModel**
Schema:

- UserID - required from user model
- EndorsedCourses
- TrainingCourses
- VolunteerOpportunities
- JobOpportunities

Collaborators:

- OrganizationController

Class name: **OpportunityController**
Responsibilities:

- Query for volunteer opportunities
- Query for paid opportunities (jobs)
- Query for list of applicants to opportunity

- Receive an application from a Human

Collaborators:

- OpportunityModel

Class name: **OpportunityModel**

Schema:

- IsPaid (boolean)
- Description
- Applicants

Collaborators:

- OpportunityController

Class name: **CourseController**

Responsibilities:

- Get all course information
- Receives course information for storage in database
- Delete course from database
- CRUD course content
- Enrol humans (students) in courses
- Drop humans (students) from courses

Collaborators:

- CourseModel
- HumanController
- InstructorController
- ContentController

Class name: **CourseModel**

Schema:

- Course ID
- Name - text
- Objective - Text field
- LearningOutcomes - Text field
- Instructor - id
- Students - list of human ids

- Published (whether the course can be enrolled in or not)
- CourseContent - array of content ids []

Collaborators:

- CourseController

Class name: **ContentController**

Responsibilities:

- Know its ID

Collaborators:

- ContentModel
- CourseController
- AssignmentController
- QuizController
- VideoController
- EmbeddedTextController

Class name: **ContentModel**

Schema:

- ContentID
- Name

Collaborators:

- ContentController

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: **QuizController**
Responsibilities:

- Get quiz average
- CRUD quiz-questions
- Set timer
- Get submissions

Collaborators:

- ContentController
- QuizModel

Class name: **QuizModel**
Schema:

- QuizQuestions
- QuizTimer
- StudentGrades
- StudentSubmissions
- QuizSubmit

Collaborators:

- QuizController

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

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