System Design of EntreE
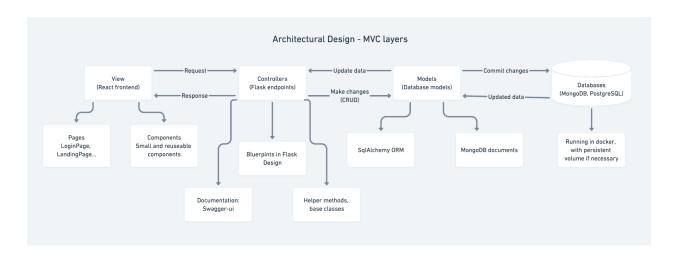
JunXing Xu
Xianghu Dai
Jiale Yu
Houde Liu
Zhenye Zhu

System Design Architecture

The following diagram illustrates the architecture of our system and it makes use of the MVC layers design architecture.

- **M: Models layer**. We make use of a python library named SqlAlchemy to create models for different classes, for instance, the User class. It also represents the DAO class in the application since it handles all commits to the database. We may also want to use MongoDB in the future and it also contains methods for us to define models, they are called documents however. We separate files into two parts, components and pages and we write documentation in the code without using any third party library.
- **V: View layer**. We choose React as our frontend framework and it is good for creating SPA and client-side rendering. It does not interact with the models directly but through the controllers, therefore, this architecture is considered as MVC layer instead of traditional MVC.
- **C: Controller layer**. We choose Flask as our backend framework and we make use of a module called flask_apispec to create RESTful APIs. It handles requests coming from the frontend and returns the requested data and a status code indicating the status of the request. For documentation, we use a famous library called swagger-ui and one can navigate to the backend url + /swagger-ui to check out a list of available endpoints including the request and response schemas.

System Interaction

The following are information about the system interaction of our project
- Our project user interface is a webapp and therefore does not have any requirements about the user system. We are using the react framework and it has been tested on Safari, Firefox, Chrome and IE, all of the modern browsers should work.
- For the backend part we are using the flask framework which recommends the newest python version, Python 3.8+ and it will work on the majority of the systems including Windows, MacOS and so on.
- Docker will be required in order to successfully set up the databases at local
- For the database part we are using MongoDB and PostgreSQL through docker images so that we do not need to maintain the databases at local.

In developer perspective
- Frontend requirements:
    - Tools: Node, npm
    - Platform: Most of the modern browsers: Firefox, Chrome, IE
- Backend requirements:
    - Language: Python 3.8+
    - Platform: Majority of the systems including MacOS, Windows and so on
- Database requirements:
    - Tools: Docker
    - Others: openssl for PostgreSQL

CRC Cards

The purpose of the CRC cards is to give a high level description of classes in the project. The following are some of the classes that we may need:

Interface: User
Subclasses: EntrepreneurUser, PartnerUser, CompanyUser
Responsibility: Representing a user, which is the essential part of our project
- Knows UserID
- Knows email
- Knows password (encrypted)
- Knows first name, last name
- Knows join datetime of joining the platform

Class Name: EntrepreneurUser
Parent Class: User
Responsibility: Representing an entrepreneurUser, one type of the user, implementation of User

Class Name: PartnerUser
Parent Class: User
Responsibility: Representing a partner, implementation of User

Class Name: CompanyUser
Parent Class: User
Responsibility: Representing a company, implementation of User

Class Name: Post
Responsibility: Representing a post
- Knows post tile
- Knows post content
- Knows post date
- Knows the user who posted it
- Knows a list of tags related with it

Class Name: JobPost
Responsibility: Representing a job post

- Knows job post title
- Knows job post description
- Knows job post requirements
- Knows job post company
- Knows job post location

Class Name: PostDAO
Collaborators: Post, User
Responsibility: Data access object for Post
- Create a Post for the user
- Get all posts posted by the user
- Delete posts posted by the user

Class Name: JobPostDAO
Collaborators: JobPost, User
Responsibility: Data access object for JobPost
- Create a Job Post for the user
- Get all job posts posted by the user
- Delete job posts posted by the user

Class Name: Course
Collaborators: Video
Responsibility: Representing a course
- Knows course name
- Knows course description

Class Name: Video
Collaborators: Course
Responsibility: Representing a video
- Knows url to the video
- Knows name of the video
- Knows description of the video

Class Name: CourseDAO
Collaborators: Course, Video

Responsibility: Data access object for Course
   - Create a Course
   - Create a Course with a list of Videos
   - Get a list of Videos associated with the Course

Class Name: VideoDAO
Collaborators: Video, Course
Responsibility: Data access object for Video
   - Create a Video
   - Get the Course associated with the Video

Interface: UserDAO
Collaborators: User
Responsibility: Data access object for User and have access to the database, perform CRUD operations here
   - Create a user, given first name, last name, email and password
   - Get an user by its UserID
   - Change password
   - Given an user, tell if it is either a EntrepreneurUser, PartnerUser or CompanyUser

Class Name: UnAuthPage
Subclasses: LandingPage, LoginPage, SignupPage
Responsibility: A fixed header at top to let users that are not signed in yet to navigate between pages

Class Name: LandingPage
Collaborators: LoginPage, SignupPage
Responsibility: Provide general information about the platform
   - Knows information to display
   - Has Login button
   - Has Signup button
   - Clicking login and redirect user to the LoginPage
   - Clicking Signup and redirect user to the SignupPage

Class Name: LoginPage

Collaborators: ForgotPasswordPage

Responsibility: A form asking for email and password
   - A field to let user enter email
   - A field to let user enter password
   - A link with the label "Forgot your password?" and clicking it will redirect user to the ForgotPasswordPage

Class Name: SingupPage

Collaborators: DashboardPage

Responsibility: A form asking for first name, last name, email and password
   - A field to let user enter first name
   - A field to let user enter last name
   - A field to let user enter email
   - A field to let user enter password
   - A button with the label "Create Account" and clicking it will send the request to the backend, if success, redirect user to the DashboardPage

Class Name: DashboardPage

Collaborators: PostComponent, JobPostComponent, SendPostComponent

Responsibility: Display everything related to the user on the dashboard
   - A post component to show recent posts
   - A job post component to show recent job posts
   - A send post component to send post

Class Name: PostComponent

Collaborators: DashboardPage

Responsibility: Show recent posts or all posts or filtered posts
   - Knows post title
   - Knows post date
   - Knows poster
   - Knows post image
   - Knows post content
   - Knows post comments
   - Knows post likes

Class Name: JobPostComponent

Collaborators: DashboardComponent
Responsibility: Show recent job posts or all posts or filtered posts
- Knows job title
- Knows company
- Knows location
- Knows job description
- Knows job requirements
- Knows job image

Class Name: SendPostComponent
Collaborators: DashboardPage
Responsibility: Allow users to send post
- A field to let user enter post title (required)
- A field to let user enter post content
- A button to let user select images

Class Name: AllCoursesPage
Collaborators: CourseVideoPage
Responsibility: Show a list of available courses
- Show a list of courses in grid layout
- Clicking any of the course will redirect to CourseVideoPage

Class Name: CourseVideosPage
Collaborators: AllCoursesPage
Responsibility: Show all videos associated with the course
- Show a list of videos with name