African Impact System Design

TeamNov

Colin Lin, Mitravasu Prakash, Shammo Talukder, Simon Chau, Jahin Promit, Ka Fai Yuen, Brandon Lo

CSCC01

Table of Contents

CRC cards

DmsModel:

- Responsibilities
    - Create a DM in the database
    - Update the contents of a DM in database
    - Read contents the of a DM from database
    - Delete a DM from database
- Collaborators
    - None

DmsController:

- Responsibilities
    - Handle HTTP Requests for DMs resource
    - Collaborate with DMsModel to perform CRUD operations for DMs
    - Collaborate with DMsView to update UI
- Collaborators
    - DmsModel
    - DmsView

DmsView:

- Responsibilities
    - Obtains new text input from user
    - Sends information about DMs to controller
    - Displays all DMs on page
- Collaborators
    - DmsModel
    - DmsController

UsersModel:

- Responsibilities
    - Modifies user information on the database.
- Collaborators
    - None

UsersController:

- Responsibilities
    - Handle HTTP Requests for Users resource
    - Collaborate with UsersModel to perform CRUD operations for Users
    - Collaborate with UsersView to update UI
- Collaborators

        - UsersModel
        - UsersView

UsersView:

- Responsibilities
    - Takes in user information from the user.
    -
- Collaborators
    - UsersModel
    - UsersController

InterestsModel:

- Responsibilities
    - Modifies interest information on the database.
- Collaborators
    - None

InterestsController:

- Responsibilities
    - Handle HTTP Requests for Interests resource
    - Collaborate with InterestsModel to perform CRUD operations for Interests
    - Collaborate with InterestsView to update UI
- Collaborators
    - InterestsModel
    - InterestsView

InterestsView:

- Responsibilities
    - Takes in interest information from the user.
- Collaborators
    - InterestsModel
    - InterestsController

CompanyModel:

- Responsibilities
    - Create a company in DB
    - Read company information from the DB
- Collaborators
    - None

CompanyController:

- Responsibilities
    - Handle HTTP Requests for company resource
    - Collaborate with CompanyModel to perform get & post operations for companies

- Collaborators
    - CompanyModel
    - CompanyPageView
    - CompaniesPageView

CompanyPageView:

- Responsibilities
    - Takes in company information from a registered user
- Collaborators
    - CompanyModel
    - CompanyController

CompaniesPageView:

- Responsibilities
    - Display all companies within the db
- Collaborators
    - CompanyModel
    - CompanyController

DiscussionModel:

- Responsibilities
    - Save discussion post information and media in db
    - Update discussion post information and media in db
    - Delete discussion post from db
    - Retrieve discussion post information and media from db
- Collaborators
    - DiscussionController
    - DiscussionView

DiscussionController:

- Responsibilities
    - Sends new post information and media to the API
    - Gets post information and media from the API
    - Update post information and media in the db through the API
    - Delete post information and media in the db through the API

- Collaborators
    - DiscussionModel
    - DiscussionView

DiscussionView:

- Responsibilities
    - Sends information about discussion post to controller
    - Obtains new post information and media from user

        - Displays the different discussion posts and comments on different pages

- Collaborators
    - DiscussionModel
    - DiscussionController

DiscussionCommentsModel:

- Responsibilities:
    - Save comment information in db
    - Update comment information in db
    - Delete comment from db
    - Retrieve comment information from db
- Collaborators:
    - None

DiscussionCommentsView:

- Responsibilities
    - Sends information about comments to controller
    - Obtains new comment information from user
    - Displays comments and replies on discussion page
- Collaborators
    - DiscussionCommentsController
    - DiscussionView

DiscussionCommentsController:

- Responsibilities:
    - Sends new comment information to the API
    - Gets comment information from the API
    - Update comment information in the db through the API
    - Delete comment information in the db through the API
- Collaborators
    - DiscussionCommentsModel
    - DiscussionCommentsView

VideoCommentsModel:

- Responsibilities
    - Create a comment in the database
    - Update the contents of a comment in database
    - Read contents the of a comment from database
    - Delete a comment from database
- Collaborators
    - None

VideoCommentsController:

- Responsibilities
    - Handle HTTP Requests for comments resource
    - Collaborate with CommentsModel to perform CRUD operations for DMs
    - Collaborate with CommentsView to update UI
- Collaborators
    - CommentsModel
    - CommentsView

VideoCommentsView:

- Responsibilities
    - Displays the comments for the e-learning videos
- Collaborators
    - CommentsModel
    - CommentsController

VideosView:

- Responsibilities
    - Displays videos
- Collaborators
    - VideosController
    - VideosModel

VideosController:

- Responsibilities
    - Handle HTTP requests for videos resource
    - Collaborate with VideosModel to perform CRUD operations for videos
    - Collaborate with VideosView to update UI
- Collaborators
    - VideosView
    - VideosModel

VideosModel:

- Responsibilities
    - Create videos in DB
    - Read video content in DB
    - Update video content in DB
    - Delete videos in DB
- Collaborators
    - None

AppBackend

- Responsibilities
    - Route URIs to the correct controller

- Collaborators
    - DmsController
    - UsersController
    - VideoController
    - VideoCommentsController
    - DiscussionController
    - DiscussionCommentsController

AppFrontend

- Responsibilities
    - Display all views
    - Handle page routing
- Collaborators
    - DmsView
    - UsersView
    - VideoView
    - DiscussionView

SearchFrontend

- Responsibilities
    - Display search suggestion
    - Send search queries to SearchBackend
- Collaborators
    - VideoView
    - SearchBackend

SearchBackend

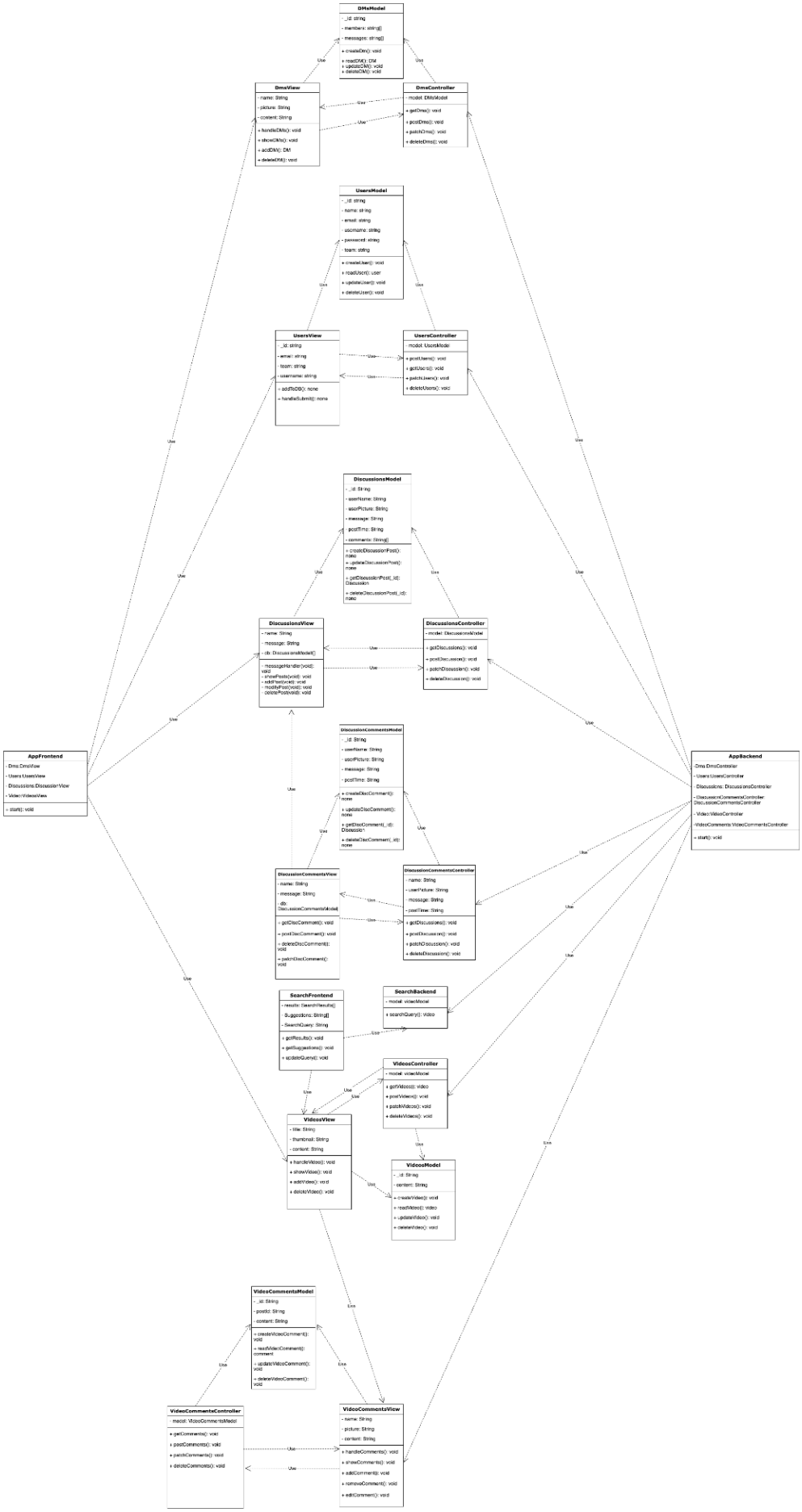
- Responsibilities
    - Search Db for elements matching the search query
    - Retrieve search queries from frontend
    - Send search results to frontend
- Collaborators
    - SearchFrontend

Description of the Architecture of the System

MVC (Model, View, Controller) is the architecture of choice for the application. There are separate sets of MVC for each major component of the app, such as Users, Discussions, and Videos. The view is responsible for displaying the information received by the controller, or model. The controller accepts input from the front end components and makes various requests to the model, and formats and returns the output to the view. The overarching components AppFrontend and AppBackend are responsible for displaying all views. It interacts with and relays the updates from the individual views to the application interface. AppBackend handles the routing api calls to use the correct controller. Other components of the app operate alone within their MVC triangle. An illustration of this architecture design can be found below.

UML Diagram

https://drive.google.com/file/d/1MpiyuRQyr7vPwGDSlwOZVsAT1O0r6OPJ/view?usp=sharing


DMsModel
- _id: string
- members: string[]
- messages: string[]
+ createDm(): void
+ readDM(): DM
+ updateDM(): void
+ deleteDM(): void
DmsView
- name: String
- picture: String
- content: String
+ handleDMs(): void
+ showDMs(): void
+ addDM(): DM
+ deleteDM(): void
DmsController
- model: DMsModel
+ getDms(): void
+ postDms(): void
+ patchDms(): void
+ deleteDms(): void
UsersModel
- _id: string
- name: string
- email: string
- username: string
- password: string
- team: string
+ createUser(): void
+ readUser(): user
+ updateUser(): void
+ deleteUser(): void
UsersView
- _id: string
- email: string
- team: string
- username: string
+ addToDB(): none
+ handleSubmit(): none
UsersController
- model: UsersModel
+ postUsers(): void
+ getUsers(): void
+ patchUsers(): void
+ deleteUsers(): void
DiscussionsModel
- _id: String
- userName: String
- userPicture: String
- message: String
- postTime: String
- comments: String[]
+ createDiscussionPost(): none
+ deleteDiscussionPost(): none
+ getDiscussionPost(_id): Discussion
+ deleteDiscussionPost(_id): none
DiscussionsView
- name: String
- message: String
- db: DiscussionsModel[]
- messageHandler(void): void
- showPosts(void): void
- addPost(void): void
- modifyPost(void): void
- deletePost(void): void
DiscussionsController
- model: DiscussionsModel
+ getDiscussions(): void
+ postDiscussion(): void
+ patchDiscussion(): void
+ deleteDiscussion(): void
DiscussionCommentsModel
- _id: String
- userName: String
- userPicture: String
- message: String
- postTime: String
+ createDiscComment(): none
+ updateDiscComment(): none
+ getDiscComment(_id): Discussion
+ deleteDiscComment(_id): none
DiscussionCommentsView
- name: String
- message: String
- db: DiscussionCommentsModel
+ getDiscComment(): void
+ postDiscComment(): void
+ deleteDiscComment(): void
+ patchDiscComment(): void
DiscussionCommentsController
- name: String
- userPicture: String
- message: String
- postTime: String
+ getDiscussions(): void
+ postDiscussion(): void
+ patchDiscussion(): void
+ deleteDiscussion(): void
AppFrontend
- Dms:DmsView
- Users:UsersView
- Discussions:DiscussionView
- Video:VideosView
+ start(): void
AppBackend
- Dms:DmsController
- Users:UsersController
- Discussions: DiscussionsController
- DiscussionCommentsController: DiscussionCommentsController
- Video:VideoController
- VideoComments:VideoCommentsController
+ start(): void
SearchFrontend
- results: SearchResults[]
- Suggestions: String[]
- SearchQuery: String
+ getResults(): void
+ getSuggestions(): void
+ updateQuery(): void
SearchBackend
- model: videoModel
+ searchQuery(): video
VideosController
- model: videoModel
+ getVideos(): video
+ postVideo(): void
+ patchVideos(): void
+ deleteVideos(): void
VideosView
- title: String
- thumbnail: String
- content: String
+ handleVideo(): void
+ showVideo(): void
+ addVideo(): void
+ deleteVideo(): void
VideosModel
- _id: String
- content: String
+ createVideo(): void
+ readVideo(): video
+ updateVideo(): void
+ deleteVideo(): void
VideoCommentsModel
- _id: String
- postId: String
- content: String
+ createVideoComment(): void
+ readVideoComment(): comment
+ updateVideoComment(): void
+ deleteVideoComment(): void
VideoCommentsController
- model: VideoCommentsModel
+ getComments(): void
+ postComments(): void
+ patchComments(): void
+ deleteComments(): void
VideoCommentsView
- name: String
- picture: String
- content: String
+ handleComments(): void
+ showComments(): void
+ addComment(): void
+ removeComment(): void
+ editComment(): void
Use

System interaction with the environment

- OS
    - Web browser
    - Chrome/firefox
- React Framework
    - Javascript
- Express
    - Nodejs
    - Javascript
- MongoDB
    - MongoDB atlas to host
    - Use Mongoose to work with MongoDB in NodeJs
- Network Configuration
    - Backend on localhost port 5000
    - Frontend on localhost port 3000
    - All backend resources start with /api/
    - Must follow instructions in README to run both backend and front end for the application to work.
- Dependencies
    - All dependencies in the package.json files in the frontend folder and backend folder should be installed

System decomposition

- Backend will be implemented with Express running in a NodeJs environment.
    - The api will receive requests, validate the request body then tells the model to perform the appropriate action
    - The api will send a response when an operation has completed or if there are any errors
- We will use MongoDB for our database and use mongoose to interact with it
    - This will store our application's data and allow us to to interact with our data
- For the frontend we will use react to build the user interface and implement any logic involved
- We will be following the MVC architecture, so our classes are grouped into three categories: Model, View, and Controller.
- The model consists of the of the DmsModel, UsersModel, DiscussionsModel, VideoCommentsModel, DiscussionComments model, and VideosModel classes
- The view consists of the DmsView, UsersView, DiscussionsView, VideoCommentsView, DiscussionCommentsView, and VideosView classes

- The controller consists of the DmsController, UsersController, DiscussionsController, VideoCommentsController, DiscussionCommentsController, and VideosController classes
- The classes that belong in the view, are the frontend, and the classes that belong under model and controller are the backend of our application
- AppFrontend and AppBackend handle the routing and aggregation of the frontend and backend respectively
- Handling errors
    - Input errors
        - Give users an error message indicating the problem with their input
    - Network errors
        - For routes that do not exist, API should return 404 status code and users should be alerted that page does not exist
        - Give users a prompt when there is a connection error with the server
    - Server error
        - API should return 500 status code and users should be alerted that there is a server error
    - Authentication error
        - Alert users when authentication fails. Prompt them to check the credentials they inputted