SportsCred

CSCC01 Final Project

Jason Yuan
Zhe Fan (Jefferson) Li
Maggie Dang
Michael Do
Ohmasinh Negi
Qing Yu (Rick) Lan

December 2, 2020

Table of Contents

CRC Cards

Front-End CRC Cards

App
Responsibilities - The host container of the app - All repeated UI elements are in here
Collaborators - None
Parent Class: None
Child Classes: LoginComponent, SignUpComponent ProfileComponent, TheZoneComponent

Login
Responsibilities - Authenticate user with username/pass - Provide entry to the Signup page for new users - Direct users to the home page after successful login - Stores the authentication token in a centralized location
Collaborators - SignUp, TheZone, LoginService, **SideEffects**
Parent Class: App
Child Classes: None

Signup
Responsibilities - Record the username/password (basic info) - Ask 6 question questionnaire - Take new signed up users into the application
Collaborators - TheZone, **SignUpService,SideEffects**
Parent Class: App
Child Classes: None

Profile
Responsibilities - Display the user's profile picture - Allow user to upload a new profile picture - Display information about the user - Edit existing information about the user - Display the status of the user - Edit status of the user
Collaborators: ProfileService, **SideEffects**
Parent Class: App
Child Classes: None

TheZone
Responsibilities - Display previews of all features and redirect users to those features when clicked
Collaborators: OpenCourt, Trivia, PicksAndPredictions, Debate
Parent Class: App
Child Classes: None

OpenCourt
Responsibilities - Display posts - Open up a modal to create posts
Collaborators: Post, CreatePost OpenCourtService, **SideEffects**
Parent Class: None
Child Classes: None

CreatePost
Responsibilities - Display a form with a title field and content field - Allow user to hit submit which will create the post
Collaborators: None
Parent Class: None
Child Classes: None

Post
Responsibilities - Display the info of a post (title/content/agree/disagree) - Show buttons to agree/disagree on a post - Display comment section
Collaborators: CommentSection
Parent Class: None
Child Classes: None

CommentSection
Responsibilities - Have a comment form to reply to the post - Display a list of comments
Collaborators: Comment
Parent Class: None
Child Classes: None

Comment
Responsibilities - Display the user of the comment - Display the timestamp - Display the content of the comment - Display which post/comment this comment is replying to
Collaborators: None
Parent Class: None
Child Classes: None

Trivia
Responsibilities - Display option for setting up a solo game or multi-player game - Makes sure both players in head to head trivia hasn't surpassed their daily limits - Invite another user to play multi-player game
Collaborators: - SoloTrivia, HeadToHeadTrivia - TriviaService, **SideEffects**
Parent Class: None
Child Classes: None

SoloTrivia
Responsibilities - Display 1 question at a time (10 times) - Set question's timer to be 14 seconds - Update ACS Score after 10 questions answe
Collaborators: TriviaQuestion
Parent Class: None
Child Classes: None

HeadToHeadTrivia
Responsibilities - Display 1 question at a time (10 times) - Set question's time will be 10 seconds - Notify the other user that the game is over when completed - Update the ACS scores once both players are do - Add a tiebreaker question if they are tied - Decide the winner based on who answered first
Collaborators: TriviaQuestion
Parent Class: None
Child Classes: None

TriviaQuestion
Responsibilities - Display Question and Answers pair - Display input fields to select/submit an answe - Show if the answer was correct or not - Have a timer of x seconds
Collaborators: None
Parent Class: None
Child Classes: None

Predictions
Responsibilities - Display options to make Daily Picks , preseason predictions, or playoff brackets
Collaborators: DailyPicks, PreseasonPredictions PlayoffBracket, **SideEffects, PredictionsService**
Parent Class: None
Child Classes: None

DailyPick
Responsibilities - Display a list of historical games and results - Display a list of daily games and input fields to predict
Collaborators: Game
Parent Class: None
Child Classes: None

PreseasonPredictions
Responsibilities - Display a list of historical preseason picks and results - Display a list of predictions for various categories and input fields to select an answer
Collaborators: PreaseasonPrediction
Parent Class: None
Child Classes: None

PreseasonPrediction
Responsibilities - Shows the category - Show a dropdown or input field to select an answer - For historical predictions, show the outcome
Collaborators:None
Parent Class: None
Child Classes: None

Game
Responsibilities - Show the two teams - For historical games, Show the score (outcome) - Show the prediction - Show the input field to predict a winner
Collaborators: None
Parent Class: None
Child Classes: None

PlayoffBracket
Responsibilities - Display a list of matchups for all rounds - Show brackets for all past years results
Collaborators: **PlayoffMatchup**
Parent Class: None
Child Classes: None

PlayoffMatchup
Responsibilities - Show the two teams - Display button to choose the winner of the matchup - Choose how many games won/lost
Collaborators: None
Parent Class: None
Child Classes: None

Debate
Responsibilities - Displays the state of the debate
Collaborators: GiveDebateAnswer, ViewDebateAnswers - **SideEffects, DebateService**
Parent Class: None
Child Classes: None

GiveDebateAnswer
Responsibilities - Diplays the question of the day - Shows input field to post analysis
Collaborators: None
Parent Class: None
Child Classes: None

ViewDebateAnswers
Responsibilities - Shows the question of the day - Shows users and their answers to the question
Collaborators: None
Parent Class: None
Child Classes: None

LoginService (DAO)
Responsibilities - Makes the HTTP request for authenticating a user
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

SignUpService (DAO)
Responsibilities - Makes the HTTP request for registering a user
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

ProfileService (DAO)
Responsibilities - requests for the profile of the user - makes the HTTP request to update the profile
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

HttpClientWrapper
Responsibilities - wrapper class to handle all HTTP requests
Collaborators: None
Parent Class: None
Child Classes: None

OpenCourtService (DAO)
Responsibilities - request for the posts to be displayed - makes the HTTP request to make a post - makes the HTTP request to get/make comments
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

TriviaService (DAO)
Responsibilities - makes the request to create a game of trivia - makes the HTTP request to get trivia questions - updates the answers of the user to the backend
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

PredictionsService (DAO)
Responsibilities - makes the HTTP requests to get all the options for making all the predictions - makes the HTTP request to save the user's predictions
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

DebateService (DAO)
Responsibilities - makes the request to get the debate question/ answers from the backend - makes the request to create the user's answer
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

SideEffects (Observer)
Responsibilities - listens for the responses to all HTTP requests and store the responses into the internal state - notify any class that requires data from the responses/state
Collaborators: **HttpClientWrapper**
Parent Class: None
Child Classes: None

Back-End CRC Card

BaseSettings
Responsibilities - Store the configuration for app server such as authentication defaults, database configuration, pagination settings, static file locations and secret keys. - Different environments can extend this class to modify the configurations.
Collaborators - None
Parent Class: DjangoDefaults
Child Classes: DevSettings, ProdSettings

Model
Responsibilities - Abstract the database so we can seamlessly change implementation - Provide python interface to creating tables - Manage relationships between tables - Handle all CRUD operations for respective table
Collaborators: ModelSerializer,
Parent Class: None
Child Classes: SportsCredUser, Post, ACS, Follows, DebatePost Agrees, SocialPost, Likes, QuestionnnaireResponse, Sports, Teams, Player, Highlights, PredictionChoice

FilterSet
Responsibilities - Handle parsing query string for GET requests - Define valid query params for models
Collaborators: Respective Model class. (UserFilter collaborates with SportsCredUser model)
Parent Class: None
Child Classes: UserFilter, PredictionFilter, PostFilter, QuestionFilter

ModelSerializer
Responsibilities - Convert json request body into respective model object - Convert model object to json to be sent over http - Define what fields a consumer is able to see from the model
Collaborators - Model
Parent Class: None
Child Classes: UserSerializer, PostSerializer, PredictionSerializer, SportSerializer, TeamSerializer, PlayerSerializer

ViewSet
Responsibilities - Define the actions for a set of resources exposed by the app server's API that will be handled by a router class - Usually map to a Model but not necessarily. Model doesn't have any direct interaction with this class
Collaborators: DefaultRouter , BasePermission
Parent Class: None
Child Classes: UserViewSet, PostViewSet, QuestionsViewSet PredictionViewSet

BasePermission
Responsibilities - Authorization on end points - Performs a series of functions based off viewset input to either allow a request or to reject it
Collaborators: ViewSet
Parent Class: None
Child Classes: AnonCreateAndUpdateOwnerOnly

ChatConsumer
Responsibilities - Connect web socket to front end for chats and notifications - Alert consumer when a new chat message or notification is received - close connection when user logs off
Collaborators - Permissions.IsAuthenticated (permission class)
Parent Class: JsonWebSocketConsumer
Child Classes: None

ACSTask
Responsibilities - adhoc updates to ACS tasks. Not directly exposed through Web API - Schedule ACS updates for users. (decay of ACS daily)
Collaborators: None
Parent Class: None
Child Classes: None

Profile
Responsibilities - knows its indentifier - knows passwords/usernames - knows whether a user is admin (just a boolean) - knows which ACS it has for each sport - knows what posts it's created - know what predictions it's made - knows its highlights - knows the posts it likes for SocialPost - knows the rating it gave for DebatePost
Collaborators: Post, ACS, Question, Sports, DebatePost
Parent Class: None
Child Classes: None

DebatePost
Responsibilities - knows its Identifier - knows its Agreement Average
Collaborators: SportsCredUser
Parent Class: Post
Child Classes: None

ACS
Responsibilities - knows its indentifier - knows the ACS score of a user for a sport
Collaborators: SportsCredUser, Sports
Parent Class: None
Child Classes: None

QuestionaireQuestion
Responsibilities - knows the question - knows if the answer is quanitative or qualitative
Collaborators: QuestionaireResponse
Parent Class: None
Child Classes: None

Post
Responsibilities - knows its indentifier - Given a user_id can find all the posts for by user - knows its title - knows its content - knows any attachments
Collaborators: SportsCredUser
Parent Class: None
Child Classes: DebatePost, SocialPost

SocialPost
Responsibilities - can determine the total number of likes this post has
Collaborators:
Parent Class: Post
Child Classes: None

Likes
Responsibilities - knows the user who gave a like/dislike to the post - knows which post the like/dislike was given to
Collaborators: SportsCredUser, SocialPost
Parent Class: None
Child Classes: None

Sports
Responsibilities - knows the name of the sport - knows its indentifier - knows all the related posts from debates or social
Collaborators: DebatePost, SocialPost, ACS, Question, Prediction
Parent Class: None
Child Classes: None

QuestionaireResponse
Responsibilities - knows the response given by the user to a questionaire question - knows its user - knows the parent question
Collaborators: QuestionaireResponse, SportsCredUser
Parent Class: None
Child Classes: None

Question
Responsibilities - knows its indentifier - knows the correct answer to the question - knows the content of the question
Collaborators: Answer, Sports
Parent Class: None
Child Classes: None

Answer
Responsibilities - knows its indentifier - knows the text content of the answer
Collaborators: Question
Parent Class: None
Child Classes: None

Agrees
Responsibilities - knows the user who agreed - knows the post the user agreed to - knows the agreement value from 1 to 10
Collaborators: SportsCredUser, DebatePost
Parent Class: None
Child Classes: None

Team
Responsibilities - knows the name of the team - knows which sport the team plays
Collaborators: Sports, Players
Parent Class: None
Child Classes: None

Player
Responsibilities - knows its players name
Collaborators: Team
Parent Class: None
Child Classes: None

PlaysOn
Responsibilities - knows the relation between player and team - knows when a player joined a team - knows when a player leaves the team
Collaborators: Player, Team
Parent Class: None
Child Classes: None

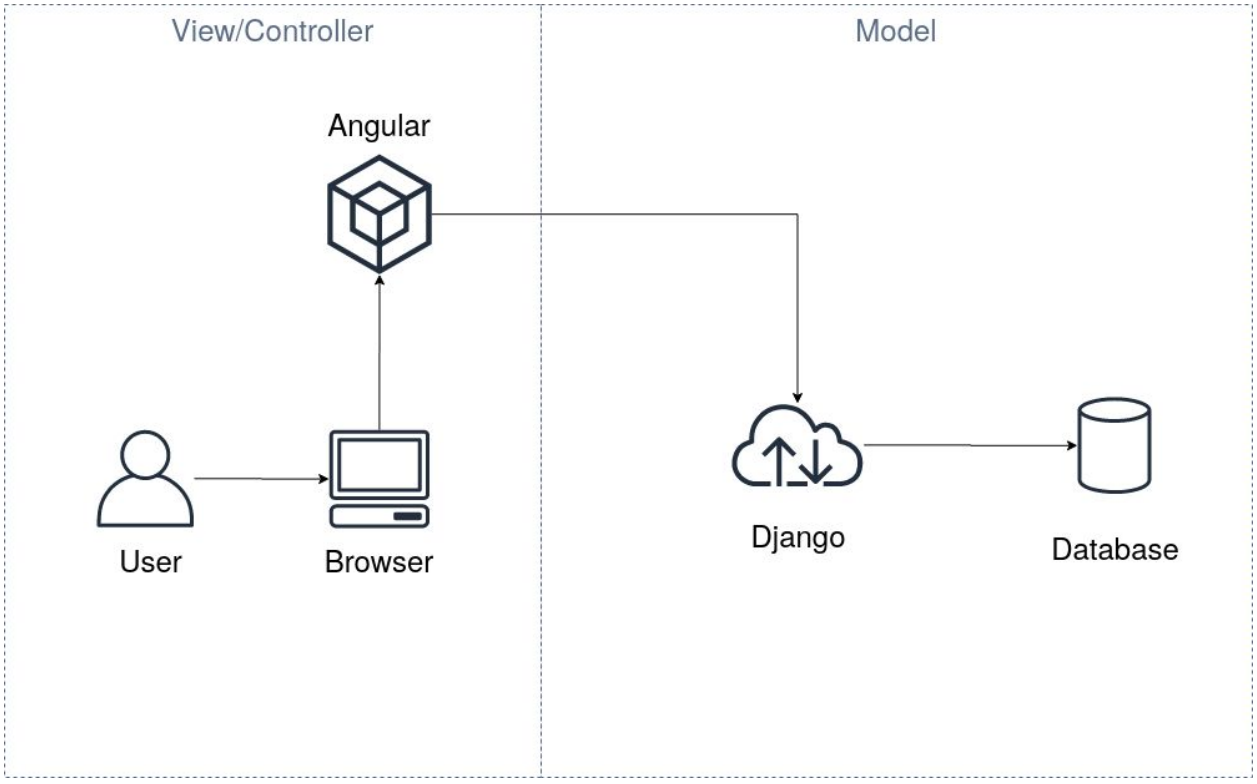
Prediction
Responsibilities - knows its indentifier - knows its Title - knows its type - knows the deadline - knows what other prediction this prediction depends on
Collaborators: PredictionChoice, Sports
Parent Class: None
Child Classes: None

System Design

System Interaction With Environment

- This system runs best on linux like systems but will work on windows computers as well.
    - To Install the the front end dependencies run `npm install` in the frontend/ directory. To start the application run `ng serve` in the same directory. This will start a development server on http://localhost:4200/. We will also need to have the backend server running.
    - To install the backend dependencies, navigate to the backend directory and run the following command: `pip3 install - r requirements.txt`.
    - Note for the following steps we make use of environment variables to choose between development or production settings to setup the application. **DO NOT** run this in powershell. If you are on windows either use cygwin or WSL to run the following steps. If you are on any other OS there should be no issues.
    - To initialize the database and populate the data run `./reset_dev.sh -t` in the backend/ folder.
    - Once this is done you can run `./runserver_dev.sh` and this will start the development server and you can use the application.
- The application will run well on any browser and will only require the user to load http://localhost:4200/
- The system also requires python3 installed but this should be met by most computers as python2 has reached the end of life.
- Some python dependencies require certain versions, so it's recommended to use our pip command. If you require to install the dependency from other methods, please pay close attention to the version require

Architecture of the System



Components

- User: Interacts with the application through a browser.
- Angular: Angular handles the view and the controller through templates and handling the HTTP requests/responses.
- Django: The model in our system interacts with the database directly and serves the data to angular. Also deals with authentication.
- Database: We are using a sqlite database in development because it is lightweight. We will use PostgreSQL for production when our application is deployed.
- ~~Background scheduler: (not yet implemented) Will schedule events such as daily ACS score deductions and scheduled deductions/rises for ACS score with Trivia and debates.~~ Replaced with polling from the front end.
- ~~Redis: (not used yet) it will act as the message broker between our 2 running instances of django.~~ No longer required since we aren't using the background scheduler

System Decomposition

Angular Application

The Angular application serves as the view to the user. This component will retrieve information from the model and will also act as the controller(TypeScript) to fill and update parts of the view. The application will not handle the errors, but will be alerted of any errors from the model (Django) and will display these errors in the view to the end user. This is so the user can get feedback on the current state of the application.

Django

Although base Django is a full framework that uses a (MVT) framework, we are only leveraging certain components of the framework and letting Angular handle the views and the controller aspect. Specifically, Django offers security from the most common attacks (CSRF, SQL Injection, XSS), so we will be leveraging it's security middleware for authentication. We use the REST framework to route and serve data to angular as a web API. Lastly, we use the model class to pull data from our database.

Database

For development we are using a sqlite database. The database is lightweight and portable so it allows for quick changes as we update our models. The database interacts with our main Django application and the background scheduler. This means it stores everything such as user information, profile information, questionnaire information, etc. It will even store information such as tokens ~~and the background task history~~. Once we deploy to a production environment we will use a client/server database such as PostgreSQL.