SW Engineering CSC648-01 Summer 2024

CodeConnect

Team 1 - PikaDevs

Max Shigeyoshi - mshigeyoshi@sfsu.edu - Team Lead
Aaron Rayray - arayray@sfsu.edu - Github Master
Noah Hai - nhai@sfsu.edu - Doc Editor
Shez Rahman - srahman2@sfsu.edu - Backend Lead
Ghadeer Al-Badani - galbadani@sfsu.edu - Backend
Majd Alnajjar - malshemari@sfsu.edu - Frontend
William Pan - wpan1@sfsu.edu - Database Admin
Phillip Ma - pma1@mail.sfsu.edu - Frontend Lead

“Milestone 2“

6/26/2024

History Table:

Date	Changes

1. Data Definitions

1. User : Class - The user that creates the account.
    1. userID - Number
    2. firstName - String
    3. lastName - String
    4. userName - String
    5. membershipType - Symbol (FREE, PREMIUM, MENTOR)
    6. email - String
    7. **Password - (data type?) char(60)?**
    8. points - Number
    9. rank - Rank object
    10. challengesCompleted - Array of Numbers(completed challenge IDs)
    11. numChallengesCompleted - Number
    12. allTrophies - Array of SpecificTrophy objects
    13. Streak of challenges completed - Number
    14. **Social Links - remove, UI only?**
    15. coins - Number
    16. mentees - Array of userIDs
    17. notificationList - Array of Notification objects
    18. bookmarks - Array of postIDs
    19. **History - (history of what?)**
    20. **Analytics - (this can be a function to display certain data items - are all data items needed here?)**
    21. numPosts - Number : number of posts + number of comments made
    22. groups - Array of Group objects
    23. groupsMentored - Array of MentorGroup objects
    24. profile - Profile object
    25. isPremium - boolean
    26. isMentor - boolean

2. PremiumUser : Class, extends User - A user that has paid the subscription fee for premium features

    1.Extends User **(userID)**

    2.paymentInfo - PaymentInformation object (25. every user has paymentInfo so premium user should have inheritance already?)

3. MentorUser : Class, extends User - A user that has become a mentor to other users. Must be approved by interview.

- Extends PremiumUser (**userID)**
- feedbackCompleted - Array of FeedbackForm objects **(feedbackID)**

4. Profile : The user's profile which has information about the user.
   - **Profile ID: number- identifier for profile table**
   - user: User object - the user who owns the profile
   - **isMentor - boolean: (used to display if mentor. Replaced Profile type data since we don't have separate profiles for recruiter, student, etc.)**
   - biography - String
   - **resume - datatype? File attachment or text page?**
   - portfolio - Portfolio object (**portfolioID**)
   - extLinks - ExternalLinks object (**externalLinksID**)

5. ExternalLinks : Class - contains external links to be used by profile
   - **externalLinksID: number - the externallinks identifiers**
   - xLogo: String - filepath to image
   - linkedinLogo: String - filepath to image
   - instagramLogo: String - filepath to image
   - tiktokLogo: String - filepath to image
   - facebookLogo: String - filepath to image
   - xLink: String
   - linkedinLink: String
   - instagramLink: String
   - tiktokLink: String
   - facebookLink: String
   - hasX: Boolean
   - hasLinkedin: Boolean
   - hasInstagram: Boolean
   - hasTiktok: Boolean
   - hasFacebook: Boolean

6. Portfolio: Class - Users will have this to display their projects
   - **portfolioID : number - identifier for portfolio userID: number**
   - **user that create portfolio**
   - projects: List of Project objects
   - visibility: Symbol (public or private)

7. Project : Class - these are data items to be stored in portfolio
   - **project ID : number - identifier for project**
   - **profolioID : number - a portfolio could have many project**
   - link: String - link to project **(projectLink)**

- desc: String - text description of project **(projectDesc)**
- title: String - title of project **(projectTitle)**
- pictures: String Array - has filepaths to images **(projectPics)**

8. Notification: Class
- **notificationID : number - identifier for project**
- title: String - title of notification with template
- redirectLink: String - clicking notification takes you to link
- date: Date object
- time: Date object

9. Group : Class - users can join these to bond over commonalities such as being alumni from the same school, or having interest in a certain technology
- allMembers: Array of User objects who have access
- forum: Forum object
- **GroupsID: number - every group have one groups**

10. MentorGroup : Class extends Group - a group for mentors to communicate with their mentees
- mentorMembers: Array of User objects
- **Resource page (what is this? needs development- page for mentor groups to have exclusive collection to resources?)**
- **GroupsID : number - every mentorGroup have one groups**

11. Groups : Class - a list of all groups for access
- **GrouopsID: number - identify the group**
- mentorGroups : Array of MentorGroup objects
- groups: Array of Group objects

12. Post : Class - Posts that users can create in order to interact with the larger community.
- user: User object of user who posts
- postID: Number
- content: String - the text content
- comments: Array of Post objects that are comments/replies to this post
- **codeBlock: data type?**
- date: Date object

- time: Date object
- likes: Number - number of likes on the post
- **threadID : number: label thread for post**

13. Forum : Class - A collection of posts and data about the forum
    - **forumID: number: unique identifier for each thread**
    - **threadID : number: refer to the thread**
    - threadTitle: String - Title of the forum thread
    - date: Date object
    - time: Date object
    - List of members who have access
    - threads: Array of ForumThread objects

14. ForumThread: Class - used to contain all posts under a thread topic
    - **threadID: number - label each thread in unique identifier**
    - originalPoster: User object who started the thread
    - threadTitle: String - title of the forum thread
    - posts: Array of Post objects which make up the thread
    - date: Date object
    - time: Date object

15. CodeChallenge : Class - The coding challenges that each user has access to and can attempt to solve.
        - userCreator: User object**(userID : in userchallenge associated relation)**
        - title: String
        - challengeID: Number
        - description: String
        - language: Symbol
        - difficulty: Symbol
        - **codingBlock**
        - deadline: Date object
        - completionPoints: Number - points gained for completing this challenge
        - solutions: array of ChallengeSubmission objects - record of X successful solutions
        - **codingTests**
        - pseudocodeHint: String

16. ChallengeSubmission: Class - contains submission data
    - challenge: CodeChallenge object - what this is submitted in response to**(challengeSubID)**
    - user: User object - who's submitting the solution**(userID)**
    - **codingID: int - which coding challenge it's submitted**
    - **codSub: string- the code is submitted**
    - date: Date object
    - time: Date object
    - verifiedSolution: Boolean - true if the submission was successful

17. SubmissionShare : Class (FR 1.18) - unique class to share with peers when you complete a challenge
    - submission: ChallengeSubmission object**(shareID)**
    - **userID : int - weak key for user submission**
    - likes: Number
    - **Comments?**

18. Feedback : Class - The review form that mentors use to give feedback to mentees on their coding challenge solutions.
- **feedbackID : the primary key for identify feedback class**
- **userID : the mentor that review the feedback**
- **challengeSubID : the challenge feedback is reviewed to**
- solutionSubmission: ChallengeSubmission object the feedback is in response to
- organizationFeedback: String - mentor's written feedback on formatting/organization of code
- organizationScore: Number - mentor's scoring on scale of 1-5
- logicFeedback: String - mentor's written feedback on code logic and efficiency
- logicScore: Number - mentor's scoring on scale of 1-5
- commentFeedback: String - mentor's written feedback on code comments
- commentScore: String - mentor's scoring on scale of 1-5

19. Leaderboard : Class - list of all users organized by ranking points to be displayed as a table
- allUsers: Array of User objects - all users in the system
    - numPoints : Number - Each user's number of points held
    - rankTitle: String - Each user's rank title
    - rankIcon: String - corresponding icon for user ranking

20. Rank : Class - the different icons and titles that users can acquire as they gain more points

- icon : String - path to image file
- title : Symbol
- rankID: Number **- that identify the rank**
- **ranksID: Number - which ranks the rank acquire**
- pointsRange: function returning true if User collected points fall in range

21. Ranks : Class - list of different rank objects that users can acquire as they gain more points, and functions involving the ranks
    - **ranksID : number - identify what ranks it is**
    - Collection of rank objects
    - checkRequirements: function to check what rank User has and return the correct rank

22. Trophy : Class - inherited by SpecificTrophy class. Users earn these for specific achievements
    - name: String
    - trophyID: Number
    - description: String - describes requirements to earn trophy
    - Trophy flag : true if user possesses trophy
    - **userID: number- identifier for user trophy**
    - checkRequirements : abstract method, implemented by SpecificTrophy

23. SpecificTrophy: Class extends Trophy - users can earn trophies for different achievements
    - InheritTrophy class **(trophyID)**
    - hasTrophy : Boolean - used by User, is 1 if checkRequirements returns 1
    - checkRequirements : function to be implemented

24. JobListing: Class - Job listings that are available for any user to apply for.
    - user: User object - the user who posts the listing
    - jobID: Number
    - title: String
    - company: String
    - location: String
    - description: String
    - requirements: String
    - date: Date object
    - applicationLink: String

25. PaymentInfo: Class - Users will be able to store their payment information for purchases

- **paymentID: int - primary key for each payment info**
- **cardNumber: Number**
- **cardName: String**
- **zipCode: Number**
- **backCode: Number - CVV/CVC code**

26. Message : Class - Sent between users as direct messaging

- **MessageID: number - identifier for message table**
- **inbox_userID : number - the inbox the message is sending to**
- **messageID: number - the thread the message belong**
- sendingUser: User object - who's sending the message
- receivingUsers: Array of User objects - who's receiving the message
- time: Date object
- date: Date object
- content: String - content of the message

27. MessageThread: Class - includes all past replies to a single message thread

- **messageThread : number - identifier for each thread**
- messages: Array of Message objects
- participatingUsers: Array of User objects who are in the message thread

28. Inbox : Class - Every user contains an inbox of messages that other users have sent them

- **inboxID int: primary key for inbox**
- **userID int: weak key for users inbox**
- messageThreads: array of MessageThread objects

29. SupportForm : Class - All users can use these to communicate with the company

- **supportFormID : number - unique identifier for supportForm**
- **userID : number - user that send supportForm**
- **userHiring_userID - company(userHiring)that receive the supportForm**
- name : String - populated by user from UI
- date: Date object
- time: Date object
- message: String - populated by user from UI

30. userHiring: Class extends User - A company user that hiring user

- **userID : number : to identify company**

- **Company : string - the company that user work for**
- **Position : string - The position of user is in company**

31. Meeting : Meeting rooms which users can use to meet with other users (free or premium). (Assuming this relates to workspace idea - what separates this from an inbox thread with multiple recipients?)

- **meetingID the meeting room user want join**
- **User ID (from user who posted)**
- **Meeting date**
- **Meeting title**
- **Meeting invitees (list of user_ID's)**

32. Chatbot : alternative to support form for users to communicate with an AI rep for the company

2. Prioritized Functional Requirements

Priority 1:

1. **All Users:**

*1.1 Users shall be able to explore some portions of the product without a profile
*1.2 Users shall be able to solve an example problem.
*1.3 Users shall create a profile
*1.4 Users shall be able to Log in/Log out (only with created profile)
*1.5 Users shall be able to use a SSO login for companies/schools
*1.6 Users shall be able to upload profile picture
*1.7 Users shall be able to Delete profile
*1.8 User shall be able to update payment information
*1.9 Users shall be able to make their profile private or public
*1.12 Users shall be able to check other users profiles/stats
*1.13 Users shall be able to do coding challenges
*1.15 Users shall be able to award different profile trophies for achievements
*1.17 Users shall be able to earn points for coding challenges
*1.18 Users shall be able to like/comment on challenge posts

*1.20 Users shall be able to check their coding ranking
*1.21 Users shall be able to check leaderboards
*1.22 Users shall be able to subscribe to Premium
*1.23 Users shall be able to unsubscribe to Premium
*1.32 Users shall be able to connect other socials
*1.34 Users shall be able to request additional features from the dev team
*1.35 Users shall be able to utilize live chat w/ other users
*1.36 Users shall be able to direct message other users

***1.38 Users shall be able to check coding streak counter of daily challenges**
*1.39 Users shall be able to create a portfolio
*1.40 Users shall be able to check/update portfolio
*1.41 Users shall be able to change portfolio visibility (public/private)
*1.42 Users shall be able to access portfolio review
*1.45 Users shall be able to submit support forms to submit feedback regarding the app
*1.55 **Users shall be able to link account to other socials(github,linkedin,etc)**
*1.61 Users shall be able to use an IDE without having to create an account
1.64 Users shall be able to choose their own country/region (Nonfunctional req? Why is this needed?)
*1.65 Users without premium shall be able to view three solutions per month
*1.66 Users shall be able to see the difference between premium and free membership perks
*1.67 Users shall be able to gain points by starting forum threads
*1.68 Users shall be able to gain points by commenting in threads
*1.69 Users shall be able to gain points by gaining friends
*1.70 Users shall be able to gain points by gaining mentees
*1.71 Users shall be able to gain points by gaining mentors
*1.72 Users shall be able to post text and images to their profiles
*1.73 Users shall be able to view a general feed of other users' updates and activities
*1.76 Users shall be able to search for other users
*1.77 Users shall be able to search for groups
*1.78 Users shall be able to receive notifications for new coding challenges.
*1.78 Users shall be able to receive notifications for messages and comments.
*1.79 Users shall be able to customize notification preferences.
*1.81 **Users shall be able to save forum posts for later reading.**
*1.82 Users shall be able to report inappropriate content.
*1.83 Users shall be able to block or mute other users.
*1.84 Users shall be able to set privacy settings for profile visibility.
*1.85 **Users shall be able to view the history of coding challenges attempted.**
*1.86 Users shall be able to request a mentor recommendation.
*1.90 Users shall be able to subscribe to notifications for specific forums or groups
*1.93 Users shall be able to view a scrolling list of job listings from home page
*1.94 Users shall be able to view other users' activity from their profiles

2. **<u>Free Users</u>**
   *2.2 Free Users shall be able to check code challenge repo
   *2.3 Free users shall be able to view three pseudocode hints per month

3. **<u>Premium Users</u>**

***3.0 Premium Users shall be able to check a single system chosen solution after completing a challenge**

*3.2 Premium Users shall be able to request mentorship (premium)

***3.3 Premium Users shall be able to Match mentors with similar coding language experience (search? Automatic matching?)**

*3.4 Premium users shall be able to view one pseudocode hint per challenge

*3.7 Premium Users shall be able to request code review from a mentor

4. **Mentor Users**

*4.1 Mentor Users shall be able to review code solutions of other users they are mentoring.

*4.4 Mentor Users shall be able to review Free/Premium users resumes/portfolios

*4.5 Mentor users shall be able to complete challenge feedback on coding challenges completed by mentees

*4.6 Mentor users shall be able to gain points by completing challenge feedback on mentee solutions

*4.7 Mentor Users shall be able to upload videos

*4.8 Mentor users shall have their number of mentees displayed on their profiles

*4.9 Mentor users shall have their number of solutions reviewed displayed on their profiles

*4.10 Mentor users shall have the groups they lead displayed on their profiles

Priority 2:

1. **All Users:**

**1.11 Users shall be able to follow other users/mentors

****1.16 Users shall be able to allow switching coding languages for challenges (cross compatibility) (different compilers - maybe 2 or 3 priority here)(would prefer just different sets of challenges based on language)**

**1.24 Users shall be able to see pop-up ads for premium/paid utilities

**1.48 Users shall be able to take mock interviews (practice interviews with real-time communication and feedback from peers and other users)

**1.49 Users shall be able to view featured users (spotlight user that's completed most challenges/stayed the most active for the past month. Featured user gets changed monthly)

**1.51 Users shall be able to utilize discounts/offers on apps premium features

**1.54 Users shall be able to join group session workshops led by mentors

1.56 **Users shall be able to view/update their own calendar(scheduled hackathons, virtual meetings..) (API or creating our own calendar?)

****1.59 Users shall be able to utilize a button to change the screen to dark mode.**

****1.60 Users shall be able to get assessed to become a mentor**
****1.63 Users shall be able to choose their preferred speaking language(How does that look in implementation?)**
1.74 **Users shall be able to earn coins
1.75 **Users shall be able to trade coins for premium subscriptions
-Or top leaderboard users get a premium reward
1.80 **Users shall be able to bookmark coding challenges.
**1.87 Users shall be able to participate in live coding sessions.
1.91 **Users shall be able to access analytics on their coding performance.
****1.92 Users shall be able to receive notifications for new coding challenges.**

2. **Free Users:**
   ****2.1 Free Users shall be able to check the most average solutions after completing a challenge.**

3. **Premium Users:**
   ****3.1 Premium Users shall be able to check a system chosen set of 2-3 solutions after completing a challenge**

4. **Mentor Users:**
   **4.3 Mentor Users shall be able to create coding challenges.

Priority 3:

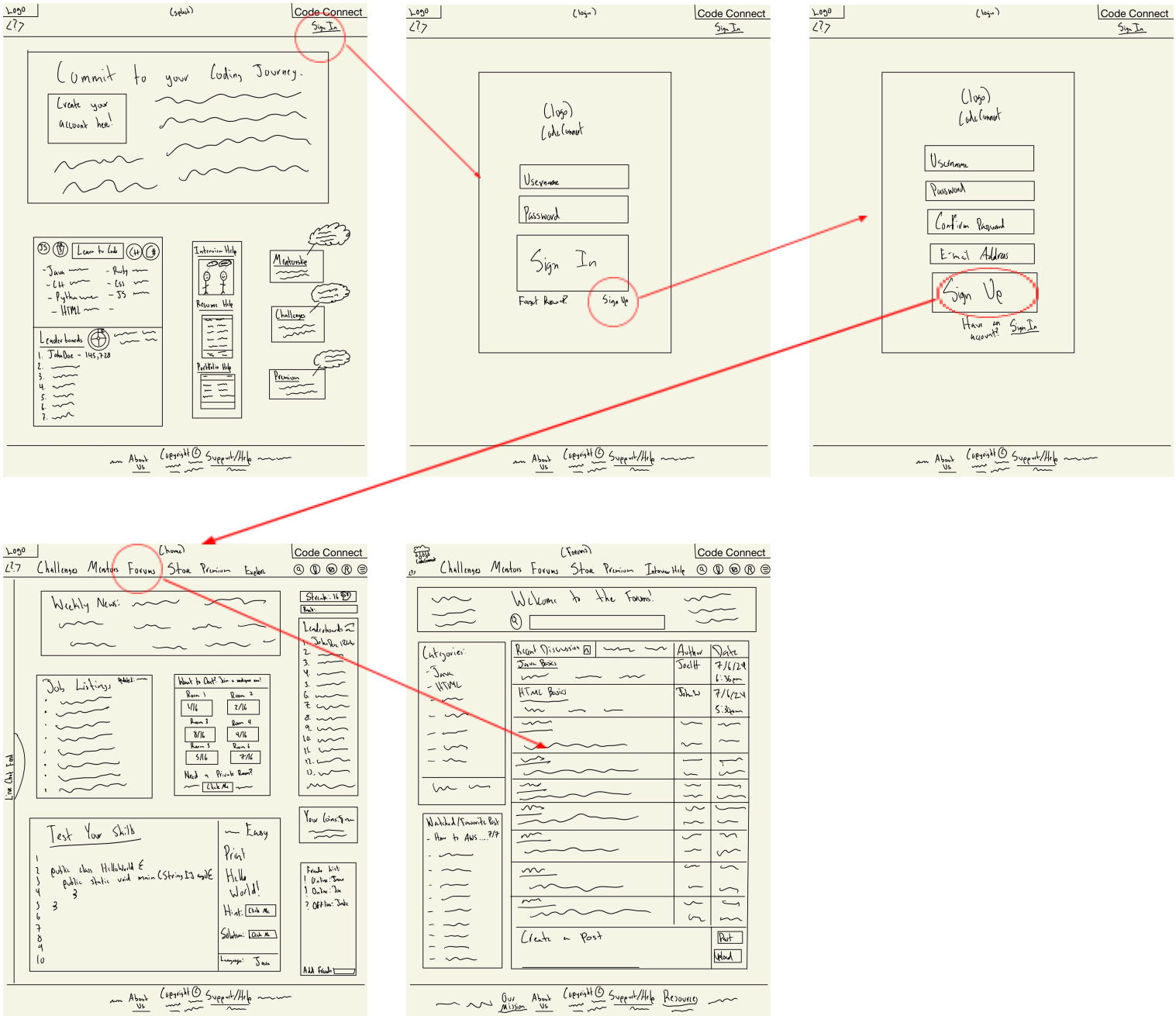
1. **All Users:**
   *****1.14 Users shall be able to see popular submissions (how is popular measured?)**
   *****1.25 Users shall be able to application survey for mentorship (?)**
   *****1.28 Users shall be able to take hiring questionnaire challenges (which are siblings to coding challenges**
   *****1.37 Users shall be able to access free/paid resources (videos/textbooks)**
   *****1.43 Users shall be able to utilize chatbot (this function would be cool but bold because need to know the data items) (Use support form instead)**
   ***1.44 **Users shall be able to access virtual workspace (define virtual workspace?)**
   ***1.46 **Users shall be able to open source/collaborative coding projects (users can participate in other people's coding projects) (Similar to 1.44?)**
   ***1.47 Users shall be able to read weekly digests (recommendations created for users based on interests/what they worked on)
   ***1.50 **Users shall be able to utilize pair programming sessions (allows users to work on projects/code collaboratively) (similar to 1.44 and 1.46)**
   ***1.52 **Users shall be able to read technical news (similar to 1.47)**

***1.57 **Users shall be able to create Recruiter/Hiring Manager specific profile**
***1.58 Users shall be able to pass a Recruiter/Hiring manager verification/background check
***1.88 Users shall be able to join virtual coding bootcamps.
***1.89 **Users shall be able to create and manage a personal blog.**

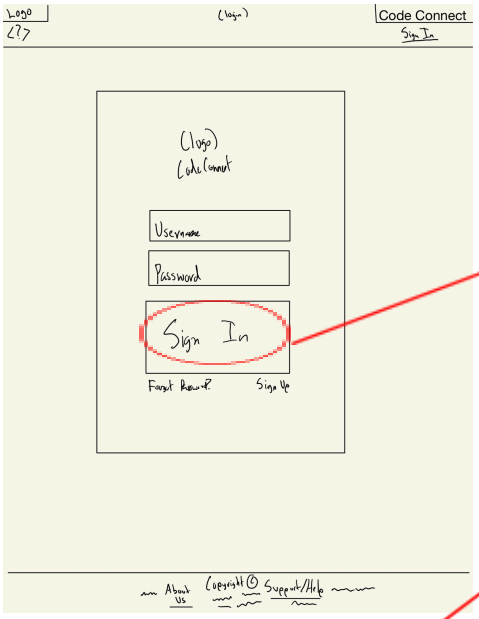
2. **Free Users: N/A**
3. **Premium Users:**
   ***3.5 Premium Users shall be able to create code challenge repo, which are reviewed by the dev team prior to publication. (with a full repo it could be harder to check solutions vs a simple IDE plug in/compiler. Resources may be difficult on this topic…maybe talk to prof)**
   ***3.6 Premium Users shall be able to check/update code challenge repo(^see note for 3.5)**
4. **Mentor Users:**
   ***4.2 Mentor Users shall be able to set up group work stations. (what is a group work station? Also potentially lower priority)**

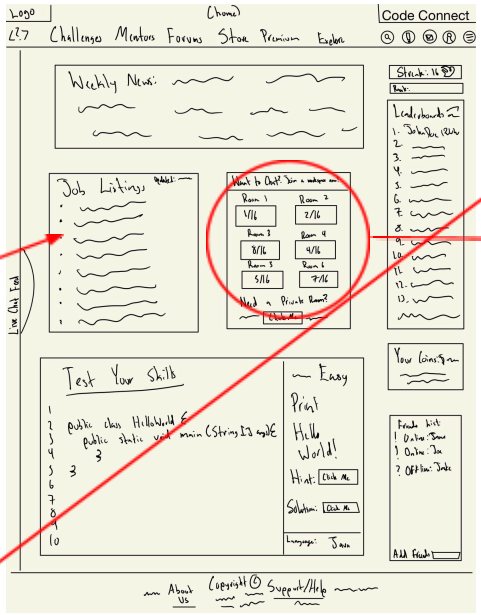
3. UI Mockups and Storyboards (high level only)

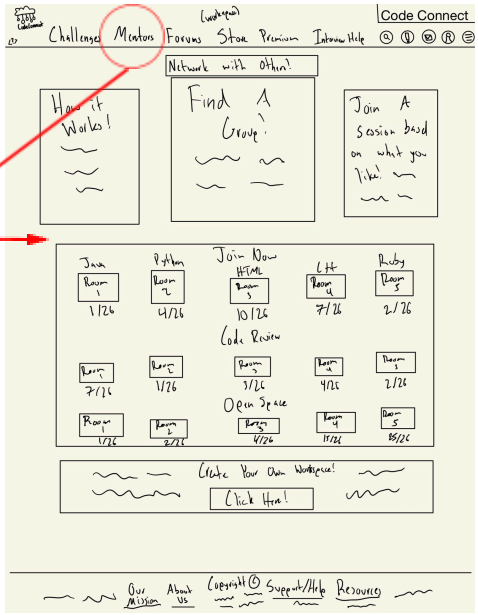
Use Case 1: Toby (Unregistered User) enters the forums

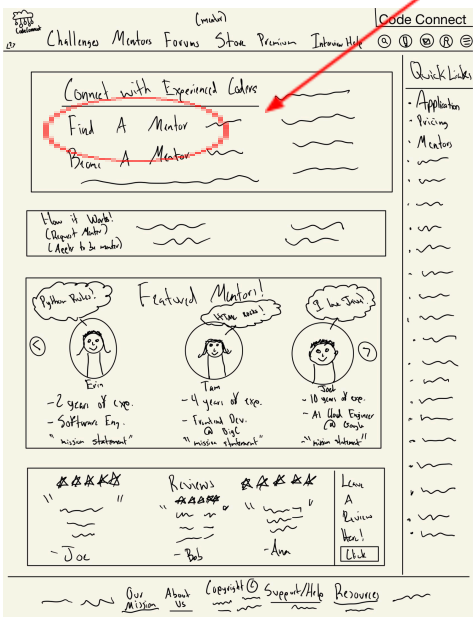


Use Case 2: Harold (Registered User) Attends a group session, meets with a mentor

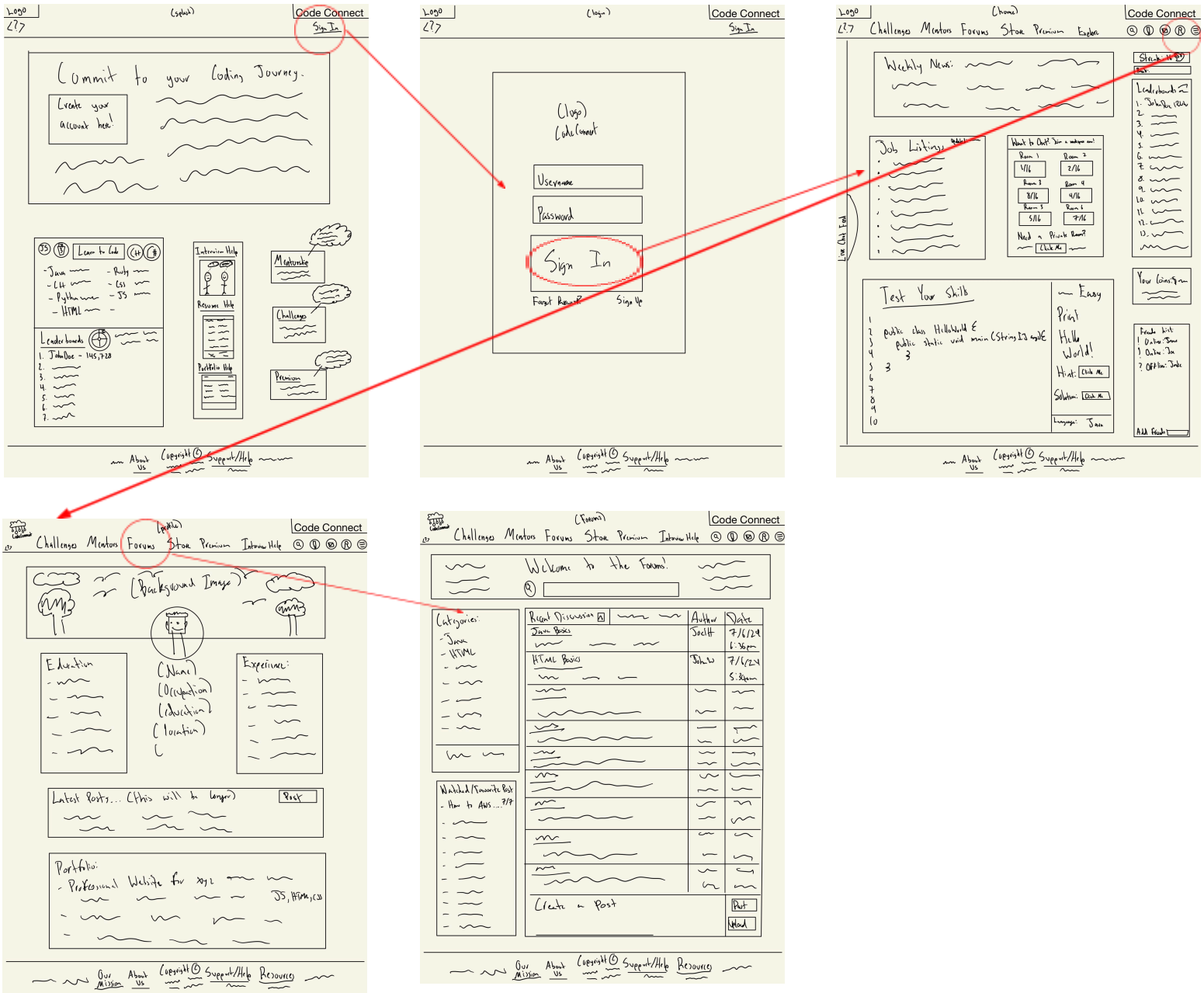




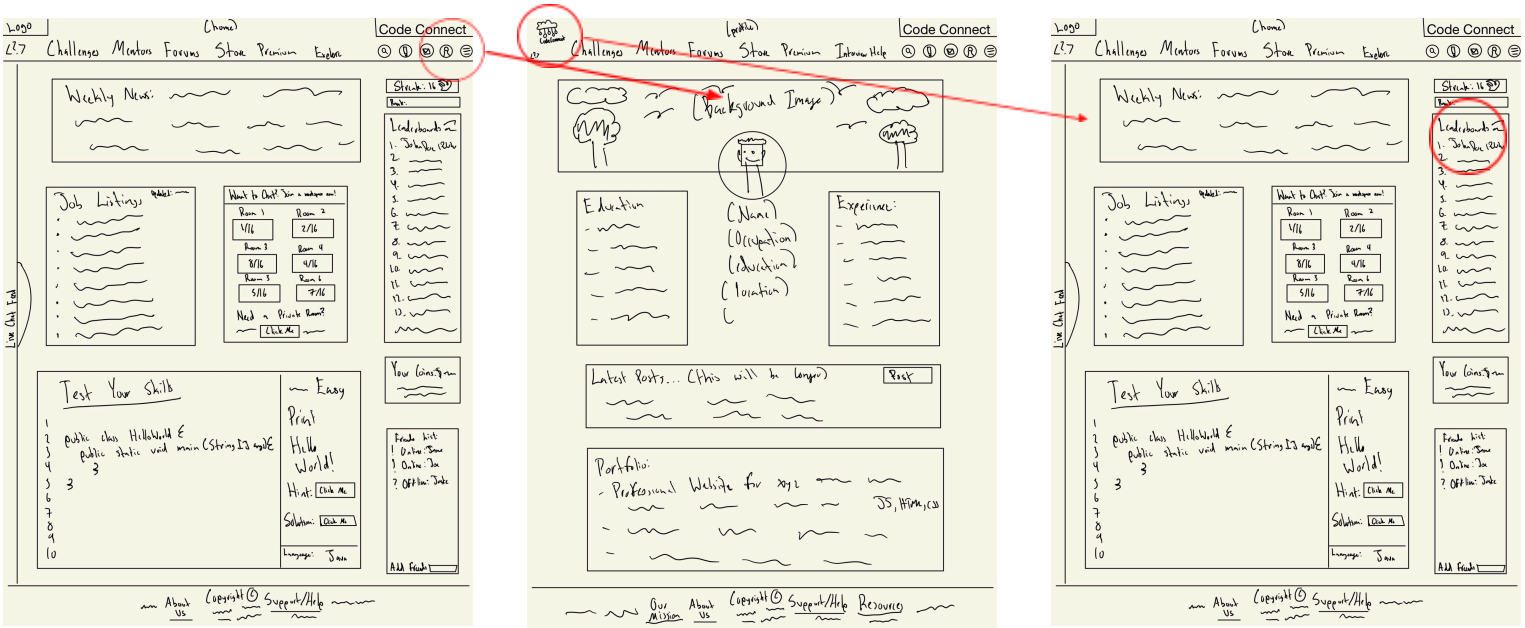




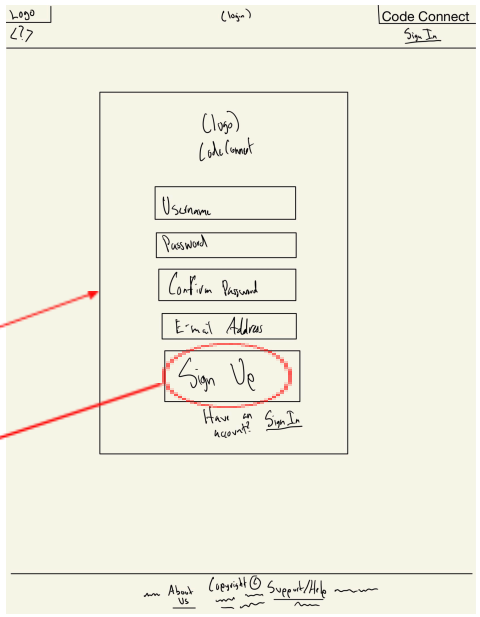
Use Case 3: Josh (Registered User) Changes profile and goes to forums

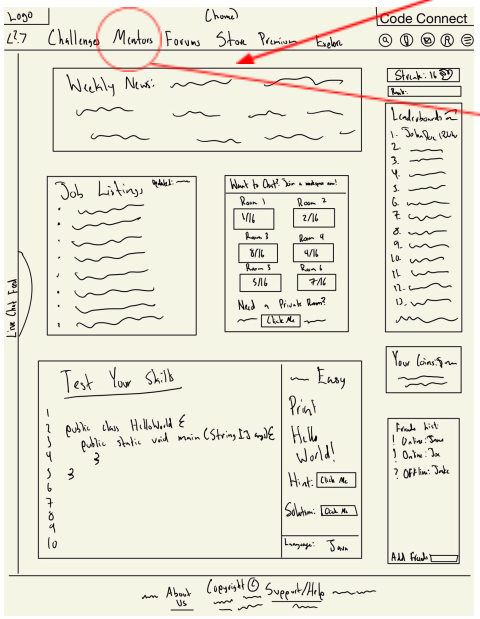


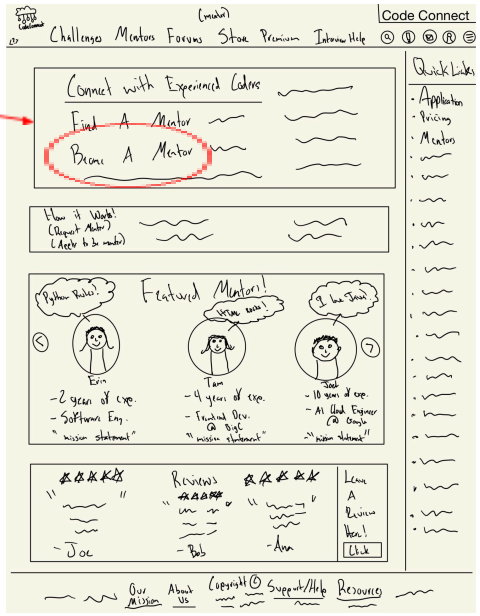
Use Case 4: Caroline (Registered User) Changes Profile and looks at rankings



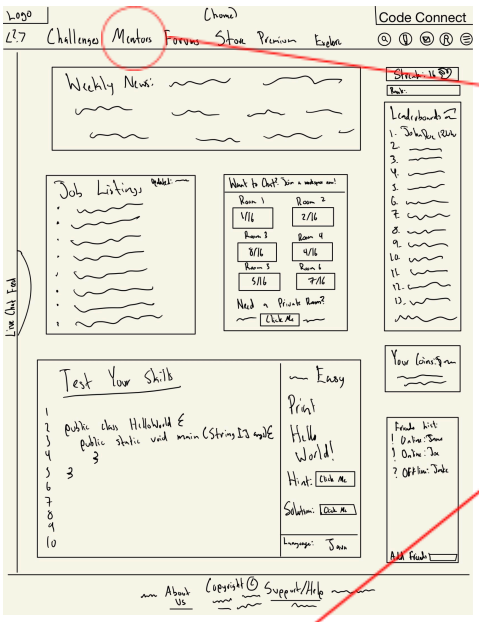
Use Case 5: Heide (Unregistered User) Signs up to be a Mentor

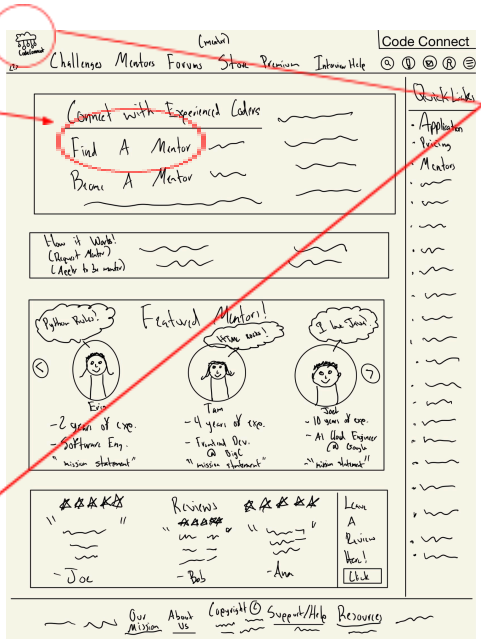


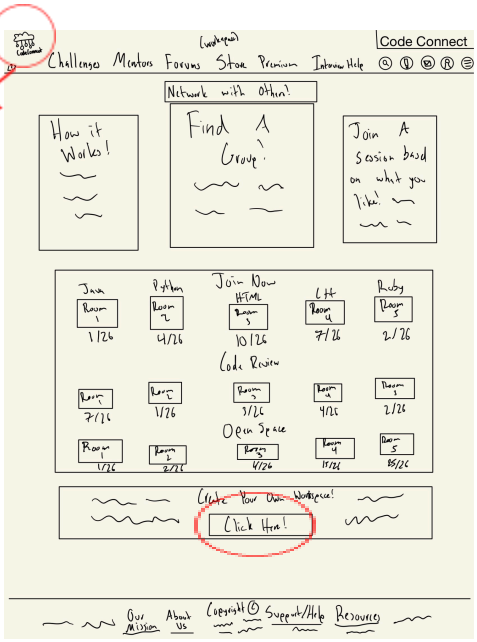


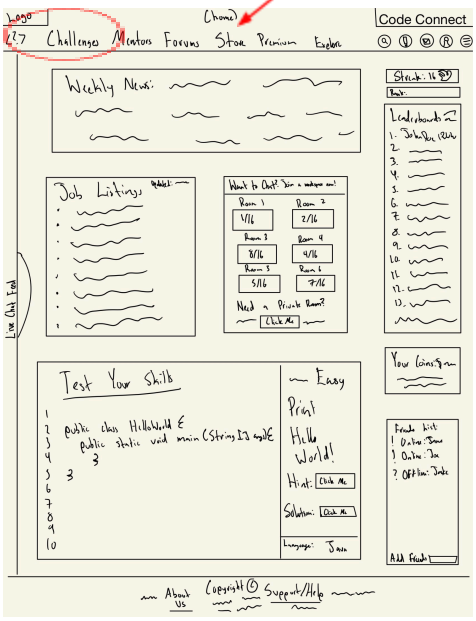


Use Case 6: Jeffrey (Registered User) Finds a mentor, schedules a meeting, and completes the challenges

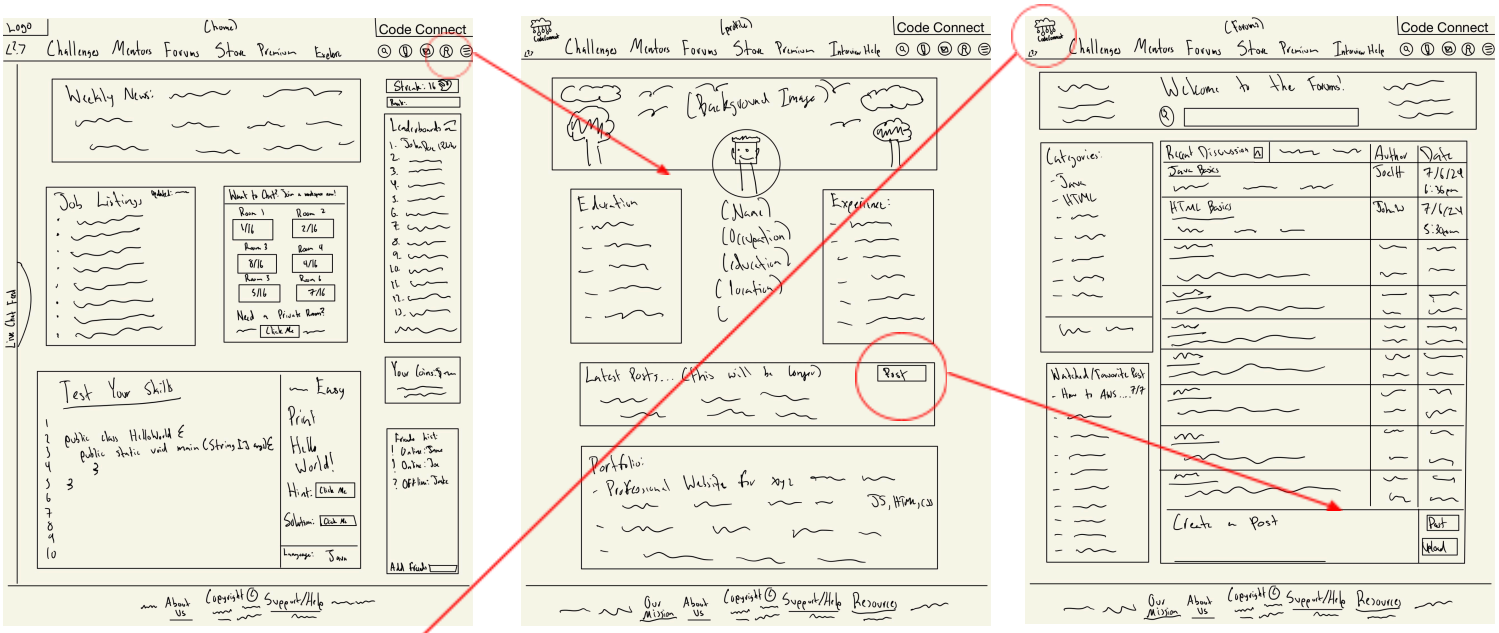


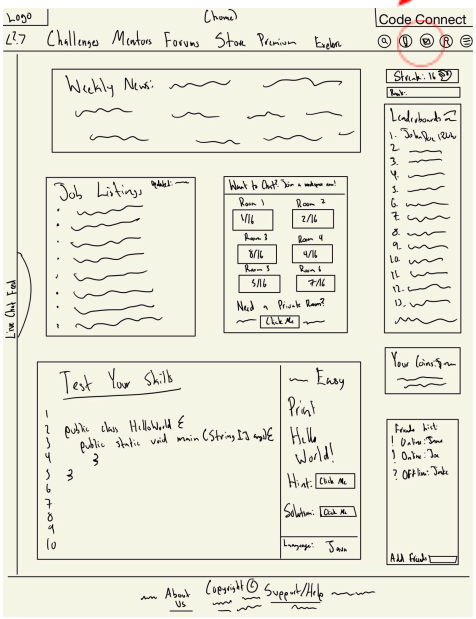




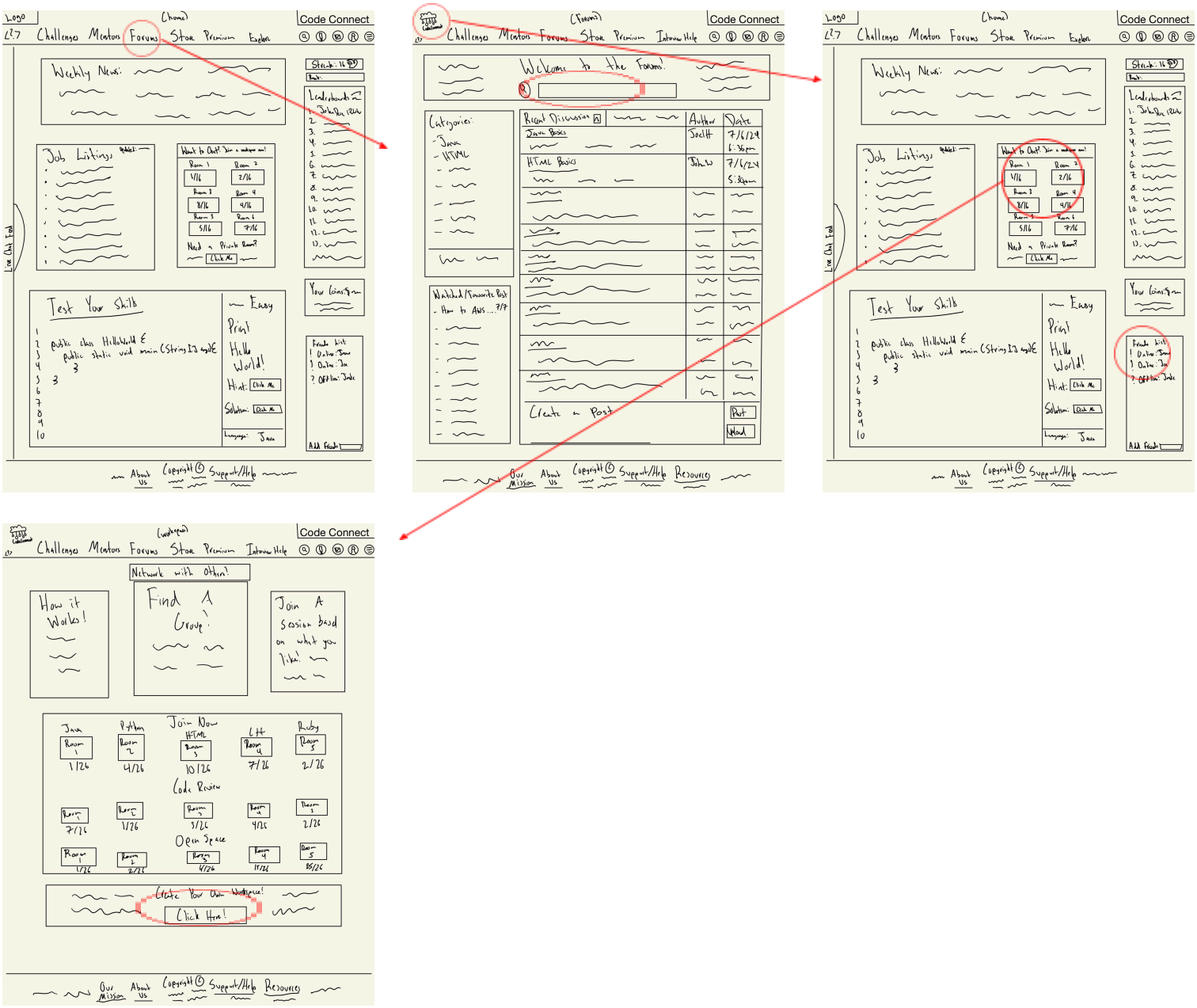


Use Case 7: Tiffany (Registered) Updates profile, Goes into forums, contacts people directly





Use Case 8. Joakim (Registered User) Joins forums, connects with users in a group



4. High level database architecture and organization

1. **Database Requirements**:
    1.1. **First Iteration**: Outline the database requirements derived from functional and non-functional requirements.
        1) User
            a) A user shall receives zero or many notice
            b) A user shall see one leaderboard
            c) A user shall create zero or one profile
            d) A user shall create zero or one portfolio
            e) A user shall able to view zero or many job
            f) A user shall able to join zero or one meeting room
            g) A user shall able to create zero or many post
            h) A user shall able to have zero or many trophy
            i) A user shall able to send message to zero or many user
            j) A user homepage shall show one or zero inbox
            k) A user shall able to solve zero or many coding Challenge
            l) A user shall able to submit zero or many coding challenge submission
            m) A user shall be able to share completed challenge on zero or many posts
            n) A user could be zero or one Premium user
            o) A user shall able to join zero or many group
            p) A user shall have zero or one rank
            q) A user shall have zero or one payment
            r) A user shall become zero or one userHiring
            s) A user shall able to write zero or many supportForm
            t) A user shall able to use one chatbot
        2) Premium User
            a) A premium user are one and only one user
            b) A premium user can become zero or one mentor user
        3) mentor
            a) A mentor user shall have one and only one premium user
            b) A mentor shall be able to give zero or many feedback
            c) A mentor shall join zero or many mentor group to communicate with their mentee
            d) A mentor shall join zero or many mentorGroup
        4) profile
            a) A profile have one and only one user
            b) A profile shall have one externalLinks
            c) A profile shall have zero or one portfolio
        5) External link
            a) A externalLinks shall have one and only one profile
        6) portfolio
            a) A portfolio shall have one and only one user
            b) A portfolio shall have zero or many project

c) A portfolio have one and only one profile

7) project
    a) A project have one and only one portfolio

8) notification
    a) A notice have one or many user

9) group
    a) A group has zero or many user
    b) A group shall have zero or many mentorGroup
    c) A group has one and only one groups
    d) A mentorGroup has one and only one group

10) mentor group
    a) A mentorGroup has zero or many mentor
    b) A mentorGroup has one and only one group
    c) A mentorGroup has one and only one groups
    d) A mentorGroup has one or many mentor

11) groups
    a) A groups has one or many group
    b) A groups has one or many mentorGroup

12) post
    a) A post have one and only one user
    b) A post has one and only one forum Thread

13) forumThread
    a) A forum Thread shall contain zero or many posts

14) forum
    a) A forum shall contain zero or many forumThread
    b) A forum Thread has one and only one forum

15) coding Challenge
    a) A coding challenge have zero or many users
    b) A coding challenges shall contain zero or many challenge submission

16) challenge submission (challengeSub)
    a) A challenge submission have one and only one user
    b) A challenge submission should receive zero or many feedback
    c) A challenge submission has one and only one coding challenge
    d)

17) submissionShare
    a) A share completed challenge should have one and only one use

18) feedback
    a) A feedback should has one and only one mentor
    b) A feedback shall belong to one and only one challenge submission

19) leaderBoard
    a) A leaderboard have zero or many users

20) rank
    a) A rank have one and only one user

b) A rank has zero or one ranks

21) ranks
    a) A ranks shall acquire zero or many ranks

22) trophy
    a) A trophy have one and only one user
    b) A trophy shall have zero or many specific trophy

23) SpecificTrophy
    a) A SpecificTrophy shall have one and only one trophy

24) jobList
    a) A job have one or many user

25) paymentInfo
    a) A payment has one or many users

26) message
    a) A message have one and only one user (Not support one message to multiple users)
    b) A message have zero or many inbox
    c) A message has one thread.

27) MessageThread
    a) A message thread shall contain zero or many message

28) inbox
    a) A inbox have one and only one user
    b) A inbox shall have contain zero or many message

29) supportForm
    a) A supportForm has one and only one user
    b) A supportForm has one and only one chatbot

30) userHiring (company)
    a) A userHiring is one or many user
    b) A userHiring are able to revives zero or many support form
    c) A userHiring has zero or one chatbot

31) mentor
    a) A mentor shall join zero or many mentor group to communicate with their mentee
    b) A mentor shall join zero or many mentorGroup

32) meeting
    a) A meeting room have zero or many users

33) chatbot
    a) A chatbot shall able to support one and only one userHiring(company)
    b) A chatbot shall reply to one and only one user.

1.2. **DBMS Selection**: Briefly (in one or two sentences) explain the chosen DBMS (Database Management System) or associated SQL frameworks and justify why they are the best fit for this project..

Since we use Amazon Ec2 instance for our server, we also use Amazon RDS for mysql database by installing mariadb which will be easy to connect and maintain either by mysql workbench or command line from Ee2 instance.

2. **Database Organization**:

2.1. **Entities and Relationships**: Describe your entities, their attributes, relationships, and domains at a high level.

1) User : Class - The user that creates the account.
    a) userID (Primary Key) - Number identify each user
    b) firstName - String
    c) lastName - String
    d) userName - String
    e) membershipType - Symbol (FREE, PREMIUM, MENTOR)
    f) email - String
    g) Password - char(60)
    h) points - Number
    i) rankID (Foreign Key):- Rank object : rank that user got
    j) leaderboardID (Foreign Key):- leaderboard the user view
    k) meetingID (Foreign Key):- the meeting user join
    l) challengesCompleted - Array of Numbers(completed challenge IDs)
    m) numChallengesCompleted - Number
    n) allTrophies - Array of SpecificTrophy objects
    o) streakChallengesCompleted - Number
    p) coins - Number
    q) mentees - Array of userIDs
    r) notificationList - Array of Notification objects
    s) bookmarks - Array of postIDs
    t) numPosts - Number : number of posts + number of comments made
    u) groups - Array of Group objects
    v) groupsMentored - Array of MentorGroup objects

w) profile - Profile object
x) isPremium - boolean
y) isMentor - boolean

2) PremiumUser : Class, extends User - A user that has paid the subscription fee for premium features
    a) userID (Primary Key) - Number identify each PremiumUser
3) MentorUser : Class, extends User - A user that has become a mentor to other users. Must be approved by interview.
    a) userID (Primary Key) - Number identify each MentorUser
4) Profile : The user's profile which has information about the user.
    a) Profile ID (Primary Key): number- identifier for profile table
    b) user: User object - the user who owns the profile
    c) isMentor - boolean: used to display if mentor.
    d) biography - String
    e) resume - String : link to user resume
    f) portfolio - Portfolio object (portfolioID)
    g) extLinks - ExternalLinks object (externalLinksID)
5) . ExternalLinks : Class - contains external links to be used by profile
    a) externalLinksID(Primary Key) : number - the externallinks identifiers
    b) xLogo: String - filepath to image
    c) linkedinLogo: String - filepath to image
    d) instagramLogo: String - filepath to image
    e) tiktokLogo: String - filepath to image
    f) facebookLogo: String - filepath to image
    g) xLink: String
    h) linkedinLink: String
    i) instagramLink: String
    j) tiktokLink: String
    k) facebookLink: String
    l) hasX: Boolean
    m) hasLinkedin: Boolean
    n) hasInstagram: Boolean
    o) hasTiktok: Boolean
    p) hasFacebook: Boolean
6) Portfolio: Class - Users will have this to display their projects
    a) portfolioID (Primary Key) : number - identifier for portfolio
    userID: number
    b) user that create portfolio
    c) projects: List of Project objects

    d) visibility: Symbol (public or private)
7) Project : Class - these are data items to be stored in portfolio
    a) project ID (Primary Key) : number - identifier for project
    b) profolioID (Foreign Key): number - a portfolio could have many project
    c) link: String - link to project (projectLink)
    d) desc: String - text description of project (projectDesc)
    e) title: String - title of project (projectTitle)
    f) pictures: String Array - has filepaths to images (projectPics)
8) Notification: Class
    a) notificationID (Primary Key) : number - identifier for project
    b) title: String - title of notification with template
    c) redirectLink: String - clicking notification takes you to link
    d) date: Date object
    e) time: Date object
9) Group : Class - users can join these to bond over commonalities such as being alumni from the same school, or having interest in a certain technology
    a) allMembers: Array of User objects who have access
    b) forum: Forum object
    c) GroupsID: number - every group have one groups
10) MentorGroup : Class extends Group - a group for mentors to communicate with their mentees
    a) mentorMembers: Array of User objects
    b) GroupsID : number - every mentorGroup have one groups
11) Groups : Class - a list of all groups for access
    a) GroupsID: number - identify the group
    b) mentorGroups : Array of MentorGroup objects
    c) groups: Array of Group objects

12) Post : Class - Posts that users can create in order to interact with the larger community
    a) user: User object of user who posts
    b) postID: Number
    c) content: String - the text content
    d) comments: Array of Post objects that are comments/replies to this post
    e) codeBlock: data type?
    f) date: Date object
    g) time: Date object

h) likes: Number - number of likes on the post
i) threadID : number: label thread for post

13) Forum : Class - A collection of posts and data about the forum.
    a) forumID: number: unique identifier for each thread
    b) threadID : number: refer to the thread
    c) threadTitle: String - Title of the forum thread
    d) date: Date object
    e) time: Date object
    f) List of members who have access
    g) threads: Array of ForumThread objects
14) ForumThread: Class - used to contain all posts under a thread topic
    a) threadID: number - label each thread in unique identifier
    b) originalPoster: User object who started the thread
    c) threadTitle: String - title of the forum thread
    d) posts: Array of Post objects which make up the thread
    e) date: Date object
    f) time: Date object
15) CodeChallenge : Class - The coding challenges that each user has access to and can attempt to solve.
    a) userCreator: User object(userID : in userchallenge associated relation)
    b) title: String
    c) challengeID: Number
    d) description: String
    e) language: Symbol
    f) difficulty: Symbol
    g) codingBlock
    h) deadline: Date object
    i) completionPoints: Number - points gained for completing this challenge
    j) solutions: array of ChallengeSubmission objects - record of X successful solutions
    k) codingTests
    l) pseudocodeHint: String
16) ChallengeSubmission: Class - contains submission data
    a) challenge: CodeChallenge object - what this is submitted in response to(challengeSubID)
    b) user: User object - who's submitting the solution(userID)
    c) codingID: int - which coding challenge it's submitted

    d) codSub: string- the code is submitted
    e) date: Date object
    f) time: Date object
    g) verifiedSolution: Boolean - true if the submission was successful
17) SubmissionShare : Class (FR 1.18) - unique class to share with peers when you complete a challenge
    a) submission: ChallengeSubmission object(shareID)
    b) userID : int - weak key for user submission
    c) likes: Number
    d) Comments?
18) Feedback : Class - The review form that mentors use to give feedback to mentees on their coding challenge solutions.
    a) feedbackID : the primary key for identify feedback class
    b) userID : the mentor that review the feedback
    c) challengeSubID : the challenge feedback is reviewed to
    d) solutionSubmission: ChallengeSubmission object the feedback is in response to
    e) organizationFeedback: String - mentor's written feedback on formatting/organization of code
    f) organizationScore: Number - mentor's scoring on scale of 1-5
    g) logicFeedback: String - mentor's written feedback on code logic and efficiency
    h) logicScore: Number - mentor's scoring on scale of 1-5
    i) commentFeedback: String - mentor's written feedback on code comments
    j) commentScore: String - mentor's scoring on scale of 1-5
19) Leaderboard : Class - list of all users organized by ranking points to be displayed as a table
    a) allUsers: Array of User objects - all users in the system
        i) numPoints : Number - Each user's number of points held
        ii) rankTitle: String - Each user's rank title
        iii) rankIcon: String - corresponding icon for user ranking
20) Rank : Class - the different icons and titles that users can acquire as they gain more points
    a) icon : String - path to image file
    b) title : Symbol
    c) rankID: Number - that identify the rank
    d) ranksID: Number - which ranks the rank acquire
    e) pointsRange: function returning true if User collected points fall in range

21) Ranks : Class - list of different rank objects that users can acquire as they gain more points, and functions involving the ranks
    a) ranksID : number - identify what ranks it is
    b) Collection of rank objects
    c) checkRequirements: function to check what rank User has and return the correct rank
22) Trophy : Class - inherited by SpecificTrophy class. Users earn these for specific achievements
    a) name: String
    b) trophyID: Number
    c) description: String - describes requirements to earn trophy
    d) Trophy flag : true if user possesses trophy
    e) userID: number- identifier for user trophy
    f) checkRequirements : abstract method, implemented by SpecificTrophy
23) SpecificTrophy: Class extends Trophy - users can earn trophies for different achievements
    a) InheritTrophy class (trophyID)
    b) hasTrophy : Boolean - used by User, is 1 if checkRequirements returns 1
    c) checkRequirements : function to be implemented

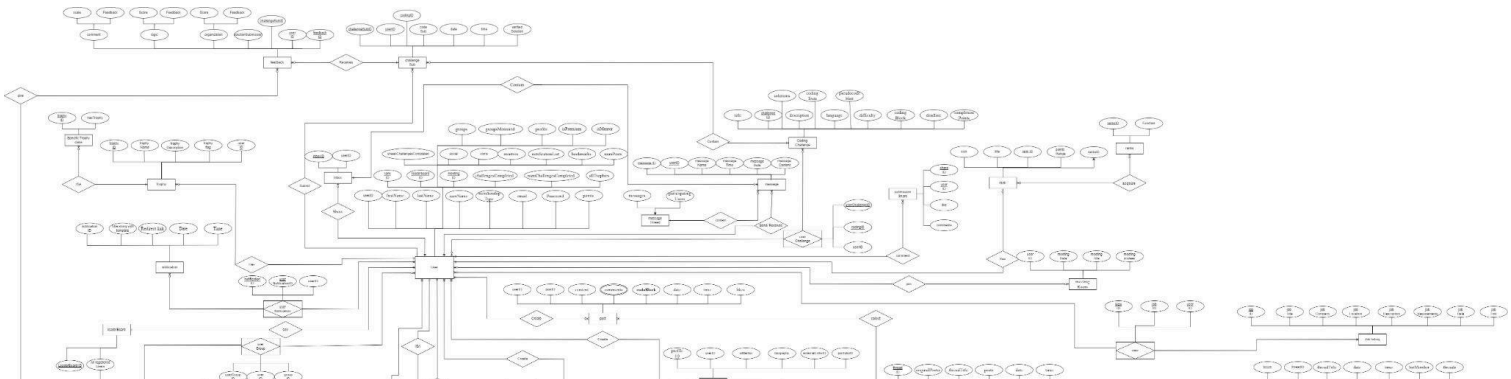
24) JobListing: Class - Job listings that are available for any user to apply for.
    a) user: User object - the user who posts the listing
    b) jobID: Number
    c) title: String
    d) company: String
    e) location: String
    f) description: String
    g) requirements: String
    h) date: Date object
    i) applicationLink: String
25) PaymentInfo: Class - Users will be able to store their payment information for purchases
    a) paymentID: int - primary key for each payment info
    b) cardNumber: Number
    c) cardName: String
    d) zipCode: Number
    e) backCode: Number - CVV/CVC code

26) Message : Class - Sent between users as direct messaging
    a) MessageID: number - identifier for message table
    b) inbox_userID : number - the inbox the message is sending to
    c) messageID: number - the thread the message belong
    d) sendingUser: User object - who's sending the message
    e) receivingUsers: Array of User objects - who's receiving the message
    f) time: Date object
    g) date: Date object
    h) content: String - content of the message
27) MessageThread: Class - includes all past replies to a single message thread
    a) messageThread : number - identifier for each thread
    b) messages: Array of Message objects
    c) participatingUsers: Array of User objects who are in the message thread
28) Inbox : Class - Every user contains an inbox of messages that other users have sent them
    a) inboxID int: primary key for inbox
    b) userID int: weak key for users inbox
    c) messageThreads: array of MessageThread objects
29) SupportForm : Class - All users can use these to communicate with the company
    a) supportFormID : number - unique identifier for supportForm
    b) userID : number - user that send supportForm
    c) userHiring_userID - company(userHiring)that receive the supportForm
    d) name : String - populated by user from UI
    e) date: Date object
    f) time: Date object
    g) message: String - populated by user from UI
30) userHiring: Class extends User - A company user that hiring user
    a) userID : number : to identify company
    b) Company : string - the company that user work for
    c) Position : string - The position of user is in company
31) Meeting : Meeting rooms which users can use to meet with other users (free or premium). (Assuming this relates to workspace idea - what separates this from an inbox thread with multiple recipients?)
    a) meetingID the meeting room user want join
    b) User ID (from user who posted)
    c) Meeting date

d) Meeting title
e) Meeting invitees (list of user_ID's)

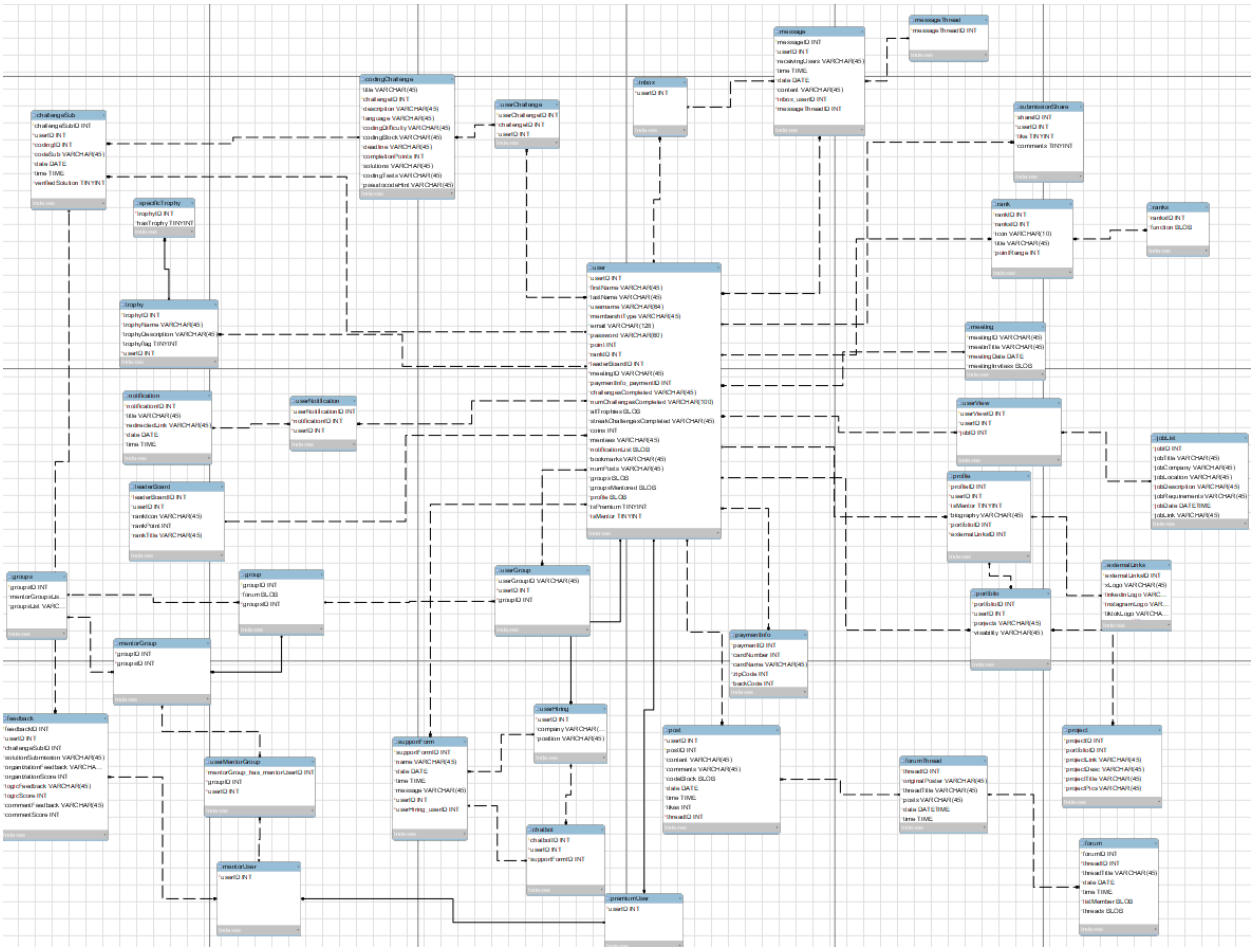
32) Chatbot : alternative to support form for users to communicate with an AI rep for the company
    a) chatbotID
    b) supportFormID
    c) userID
33) userNotification: Associated entity for user and notification since user shall have zero or many notifications and notification has zero or many user.
    a) userNotificationID : number - identify userNotification entity
    b) notificationID: number - reference notification relation
    c) userID: number - reference user relation
34) userView: Associated entity - for user and jobList since a user shall view zero or many job(jobList) and a jobList has zero or many user
    a) userViewID: number - identify userView entity
    b) jobID: number - reference job relation
    c) userID: number - reference user relation
35) userChallenge: Associated entity - for user and codingChallenge since a user could have zero or many codingChallenge and a codingChallenge has zero or many user.
    a) userChallengeID: number - identify userChallenge entity
    b) challengeID: number - reference codingChallenge relation
    c) userID: number - reference user relation
36) userGroup: Associated entity - for user and group since user shall have zero or many group and a group has one or many user.
    a) userGroupID: number - identify userGroup entity
    b) userID: number - reference user relation
    c) groupID: number - reference group relation
37) userMentorGroup: Associated entity - for mentorUser and mentorGroup since a mentor shall have zero or many mentorGroup and a mentorGroup has one or many mentor
    a) groupID: number - reference grouprelation
    b) userID: number - reference user relation

2.2. **ERD Creation**: Develop an Entity Relationship Diagram (ERD) using a software tool like draw.io, focusing on the main functionalities of your system.**(Refer to conceptual design)**



ERD Draw Io link:
https://drive.google.com/file/d/1zurpNjxRKnkSV0DKK_vZ_35LPtuBBGZS/view?usp=sharing

2.3. **EER Diagram**: Create an Entity Establishment Relationship (EER) diagram using MySQLWorkbench or DBeaver.



github link:
https://github.com/sfsu-joseo/csc648-848-05-sw-engineering-su24-T1/blob/backend-dev/application/server/db/High_level_database_architecture/EER.mwb

branch:backend-dev

folder: application/server/db/High_level_database_architecture/ EER.mwb

2.4. **Forward Engineering**: Forward engineer your database model and include it in this section.

Githlink:https://github.com/sfsu-joseo/csc648-848-05-sw-engineering-su24-T1/blob/backend-dev/application/server/db/High_level_database_architecture/forward_engineer_process.sq

branch:backend-dev

folder: application/server/db/High_level_database_architecture/ forward_engineer_process.sql

3. **Media Storage Decision**:

3.1. Decide if images and video/audio will be stored in file systems or in DB BLOBs (Binary Large Objects).

After discussion we will keep the image in the instance and the metadata for image and video link will be stored in mysql, so the user shall search for the component they need.

3.2. Describe any other special data format requirements like for video/audio/GPS, etc.

The user profile entity contains another externalLinks which link to another entity that has components for the user to fill out. For example xLink,linkedinLink, ,instagramLink,tiktokLink and facebookLink. For coding challenges because it is plain text it will also be stored in mysql as an object.

5. High Level APIs and Main Algorithms

APIs List

- api/auth
  - api/auth/login
    Authenticates users and sets authentication cookies (token).
  - api/auth/signup
    Verifies if a user already exists, if not create a new user and send verification email
  - api/auth/verify/:token
    Decode JWT token and verify related email

    - api/auth/reset/:token
      Decode JWT Token and allow user to reset password

- api/user
    - api/user/:user_id
    - api/user/:user_id/rank
    - api/user/:user_id/trophy
    - api/user/:user_id/portfolio
    - api/user/:user_id/mentorship
    - api/user/:user_id/mentorship?mentor_id=
    - api/user/:user_id/mentorship/:mentorship_id
    - api/user/:user_id/payment
    - api/user/:user_id/payment/:payment_id
    - api/user/:user_id/feed
    - api/user/:user_id/thread
    - api/user/:user_id/thread/:thread_id
    - api/user/:user_id/thread/:thread_id/message
    - api/user/:user_id/thread/:thread_id/message/:message_id

- api/portfolio
    - api/portfolio/:portfolio_id
    - api/portfolio/:portfolio_id/project

- api/leaderboard
  Get leaderboard details

- api/project
    - api/project/:project_id

- api/meeting
    - api/meeting/:meeting_id

- api/group

    - api/group/:group_id
    - api/group/:group_id/member
    - api/group/:group_id/join

- api/challenge
    - api/challenge/:challenge_id
    - api/challenge/:challenge_id/submission

        - api/challenge/:challenge_id/submission/:submission_id
        - api/challenge/:challenge_id/submission/:submission_id/run
        - api/challenge/:challenge_id/submission/:submission_id/feedback
        - api/challenge/:challenge_id/submission/:submission_id/feedback/:feedback_id
        - api/challenge/:challenge_id/feedback
        - api/challenge/:challenge_id/feedback/:feedback_id

- api/forum
    - api/forum/:forum_id

- api/post
    - api/post/:post_id
    - api/post/:post_id/comment
    - api/post/:post_id/comment/:comment_id

- api/job
    - api/job/:job_id
    - api/job/:job_id/apply
    - api/job/:job_id/application
    - api/job/:job_id/application/:application_id

Significant Algorithms or Processes

1. Verification Emails
   It will send emails to verify users and reset passwords. It will use a URL and short lifetime json web token (JWT). If the user opens the URL in browser (Get Request) within the timeframe, it will perform the required operation like verify email or reset password. This functionality utilizes SMTP to send the emails to users.

2. Login
   Authenticate users. It retrieves the password salt, hashes the user input with HMAC SHA-256, and compares it with the user hashed password in the database. It also generates a json web token and responds with the HTTP header Set-Cookies. It will support resetting passwords by email.

3. Signup
   It will make sure no other user exists with the same email. It will verify password strength, generate random salt, and hash the password with HMAC SHA-256. It stores the values in the database. It will also verify user email with a verification link.

4. Resource Authorization
   It will ensure the request user is privileged to perform the resource operation. It will check if the user is the owner of said record or it is public.

5. Resource Data Operations
   CRUD: create, read, update, delete with sequelize ORM. It will also sync required search fields to the Opensearch instance.
   List: It will rank, sort data according to resource type and user properties. For example, if the user is browsing challenges, it will make a request to the search instance with the user fields and sort according to relevance. If the user is listing solutions, it will rank them according to run metrics like resource utilization and time.
   Filter: It will handle users requests to filter records with any combination values of the resource properties.
   Search: It will make a request to the search engine with a user query. It will also pass additional fields to boost and ranking results according to user preference/filters. If a user requests a search on a previous search result, this API will just include the previous query with an and parameter for the new one. If no results are found, it will omit the user query and pass the additional fields to find relevant ones.

6. Challenge Creation
   This API will accept required data to create a challenge and ensure the challenge can be solved. It will require a base code snippet, expected stdout, test snippet, programming language, and required environment constraints (memory, CPU, time, network).

7. Challenge Submission
   It will run a submission with the appropriate programming language and get run metrics like resource usage and time. It will combine the user, test, base codes, make a request to Judge0 endpoint with a generated callback URI for this specific submission. Judge0 will make a PUT request to the callback URL and the API will update the submission result accordingly.

8. Submissions Ranking
   The API will sort submissions according to their run metrics like resource utilization and execution time. Judge0 will execute the successful submissions multiple times to gather accurate metrics.

Microservices and tools

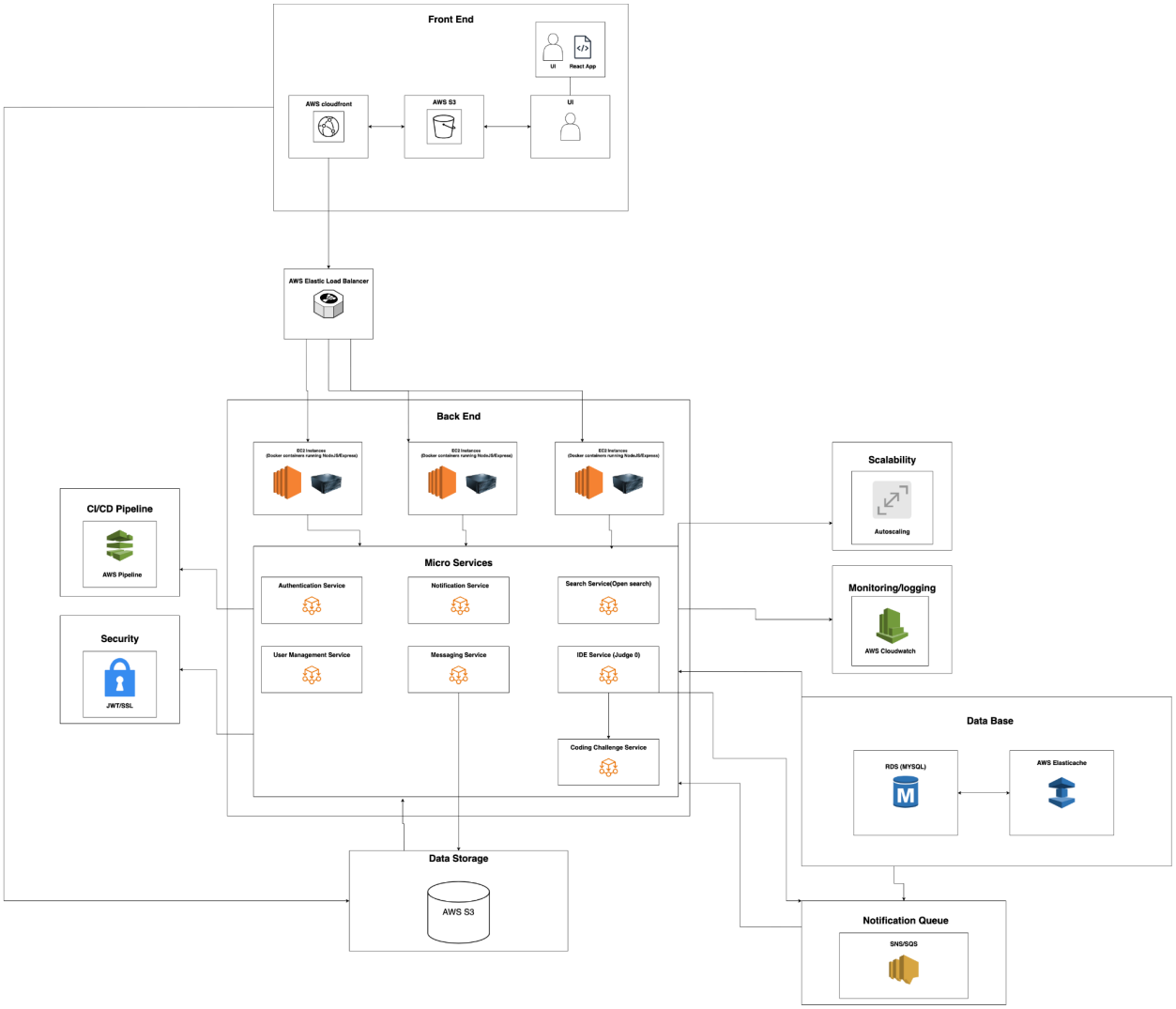
1. Judge0 (Still in discussion)

It is an open source online IDE to run different code snippets, manage sandbox environments, resources, and queues. It comes with the following features:

- It has scalable architecture in which it can scale vertically in increased threads and horizontally in multiple instances. It has a queue and creates multiple workers. The application API can decide the submission instance according to availability.
- Sandboxed compilation and execution where user untrusted codes are run in a managed sandbox with support of setting multiple options like network access, memory limit, CPU limit, timeout, and isolate concurrent submissions.
- Support for custom user-defined compiler options, command-line arguments, and time and memory limits.
- Detailed execution results like stdout of execution, memory/cpu utilizations, and time. It allows even running the code multiple times to get accurate results and enable better ranking results.
- Webhooks (HTTP callbacks) in which the application API will not have to wait for the submission to end or pool for results and just receive the call back request once the submission is done.
- Open Source and can be hosted in an EC2 instance but we will use a shared cloud in rapid apis and migrate once scaling is required.

2. Opensearch Instance
   It is an open-source search and indexing engine. It enables advanced search and indexing functionality.
   - Fuzzy matching to handle misspellings
   - Result ranking and filtering
   - Multi field queries and field boosting
   - K-NN relevance matches
   - Realtime results
   - Scale out to accommodate large data

6. System Design



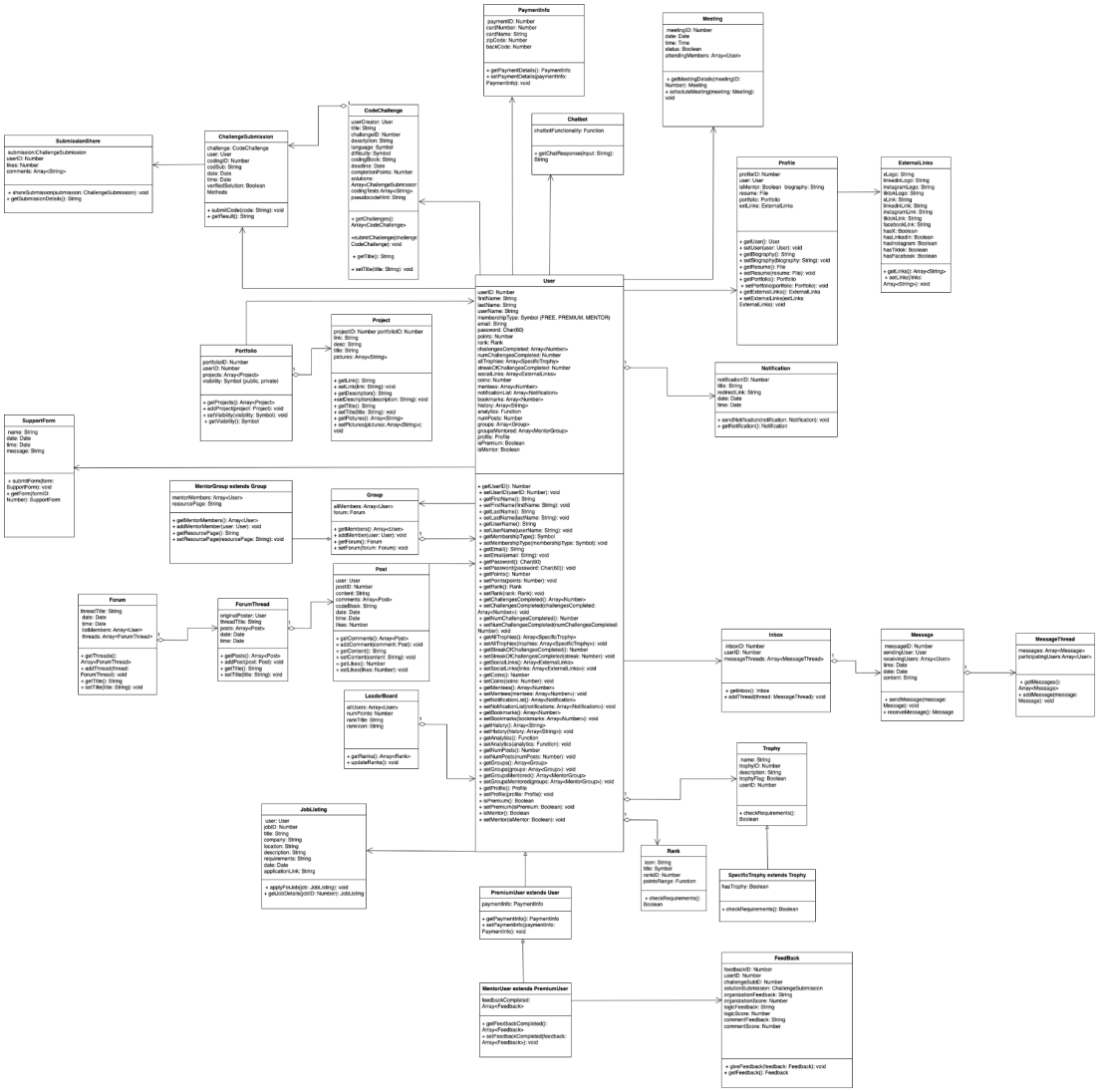
https://drive.google.com/file/d/1doonMbyxpfyKRlWUUELP12gjqkFxR1u_/view?usp=sharing

For our system design we use a microservices approach, where independent services like authentication, user management, coding challenges, notifications, messaging, search, and an integrated development environment communicate with one another to form the product. This improves the manageability, scalability, and flexibility of our product. For high availability and dependability, a lot of EC2 instances performing backend services are spread out with traffic using Amazon Elastic Load Balancer. Data that is frequently needed is cached using ElastiCache, which reduces the load on the main database and speeds up finding data.

Scalability is achieved through auto scaling groups, which change the number of EC2 instances according to traffic load for fault tolerance and reliability. AWS S3 stores huge files and backups with built in reliability, while the database is hosted in a multiple availability zone architecture for high availability and data security. Docker containers deployed on EC2 instances contain backend services, which assure consistency and make deployment and scaling easier. ElastiCache improves performance by caching data, while RDS (MySQL) guarantees data replication across different availability zones.

JWT is used for safe authentication of users, data encryption for protecting sensitive information, and SSL/TLS encryption for data in transit. Network traffic to EC2 instances is managed by security groups. The user interface (React App), AWS CloudFront for distributing static components, and AWS S3 for storing data make up the frontend layer. EC2 instances running Docker containers with NodeJS/Express services and other microservices form the backend layer. RDS (MySQL) and ElastiCache are features of the database layer. SNS/SQS manages notifications, AWS CloudWatch handles logging and monitoring, and AWS CodePipeline handles CI/CD. Overall our architecture ensures a safe, scalable, and adaptable system that can manage changing traffic and offer a good user experience

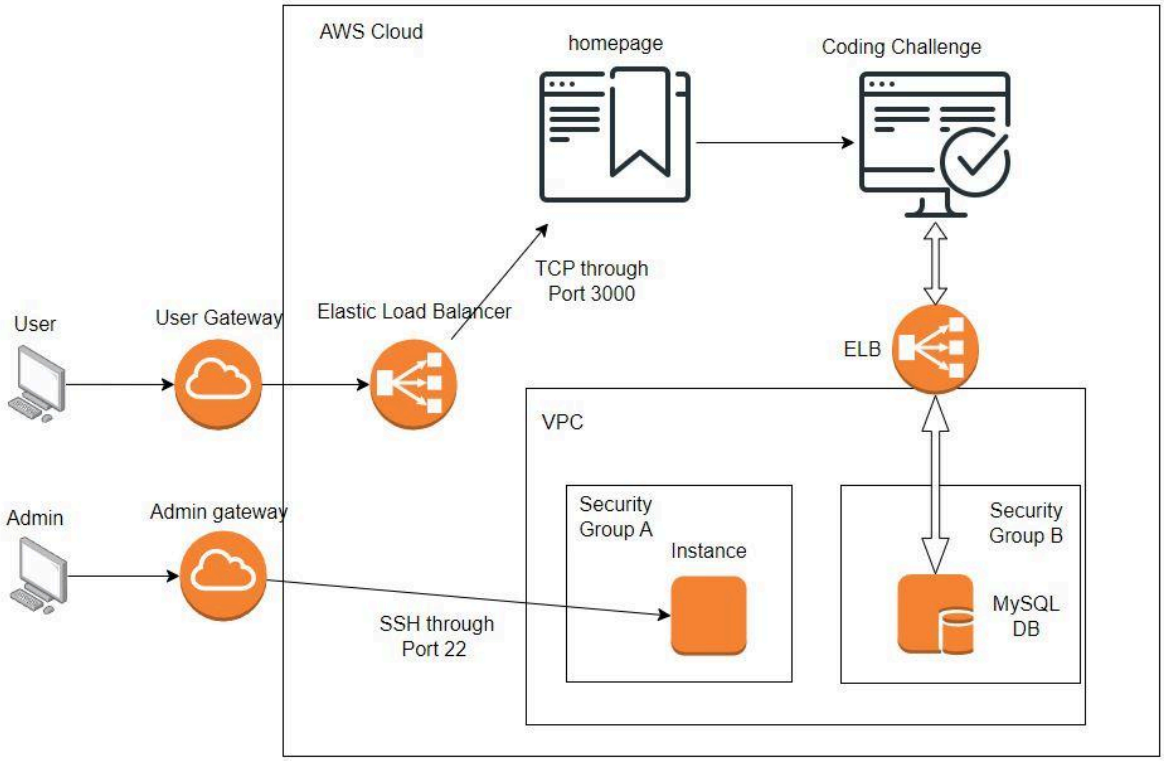
UML Diagram



https://drive.google.com/file/d/1hG_XbO-yDG5hdWcWTPN0i2EP4WwMEPFY/view?usp=sharing

A number of patterns are used in the UML class diagram for our app to improve maintainability, scalability, adaptability, and reusability. For data consistency and to reduce cost, the Leaderboard class follows a pattern that guarantees a single instance. The creation process is encapsulated for flexibility and manageability in the pattern used to build ChallengeSubmission objects. Subscribed users can get updates based on events from the NotificationService thanks to our implementation, which separates the sender and recipient. We provide an ongoing combination of behaviors by adding premium functionalities to User profiles without changing the base class. Our design for the ranking system included the possibility to use various ranking algorithms, which improved scalability and flexibility. Finally, by creating a single interface, we simplify user related tasks and increase the readability and maintenance of the code.

7. High Level Application Network and Deployment Design



8. Identify actual key risks for your project at this time

1) **Skills Risk (do you have the right skills):**
    a) Most of our group has not taken a database course yet, so we rely on only one team member to keep the backend team up to speed.
    b) Previous networking experience is limited, so our network diagram is very simple.
2) **Schedule Risk (can you make it given what you committed and the resources):**
    a) People have jobs and scheduling meetings with the whole team doesn't always line up with everyone's schedule.
    b) People sometimes have things come up(emergency, etc) and they're not in the loop of our plan
    c) We didn't plan for the potential for a bottle neck, which delayed some of our internal deadlines longer than expected.
    d) Recent heat advisories may affect internet access with blackouts or brownouts from overloaded electric grids
3) **Technical Risks (any technical unknowns to solve):**

    a) Not everyone is familiar with the backend technologies/micro services, thus we needed to take extra time to learn them and implement them.

4) **Teamwork Risks (any issues related to teamwork):**
    a) Notion is a new tool, thus our team isn't used to updating their tasks through it yet.
    b) With people having work, and thus unable to make every meeting, we rely on recordings to keep everyone on the same page.
    c) Potential communication outside of team channel may cause disarray unless everything is relayed back to the team channel
5) **Legal/Content Risks (can you obtain content/SW you need legally with proper licensing, copyright):**
    a) We do not have a collection of challenges to populate the coding challenges section of the product.
    b) We would have to seek some kind of legal advice for the use of social iconography.
    c) Would have to get permission from legal teams to use their problems and solutions
    d) Would have to get permission from videographers to use their solutions
    e) Company permission to be represented within our application

9. Project management

After reflecting on Milestone 1, it was clear that we needed to be more organized in our approach to Milestone 2. First, we created our Milestone 2 template, with each section of the document being pulled directly from the Milestone 2 document. Our team got together in a meeting and we reviewed the document in full going through each goal listed on the document and, using Notion, created a rough idea of who might be the owner of each task. Through our meetings, we were able to refine some of the tasks down and organize the larger goals into more manageable chunks. Notion was also a huge help when we needed to organize tasks, and I also wished we had used it for Milestone 1 because we were able to assign tasks and update the group as to the status easier than simply using Discord. We also were able to silo some of the work between team members that were more focused on front-end or back-end development. We still looked over the work and were able to critique each deliverable as it was added to the M2 document. Moving forward we will use this approach, as it made the M2 sprints easier to manage.

10. Detailed list of contributions

- **Max Shigeyoshi (6/10)**
    - Created M2 Document
    - Project Management Write-up
    - Network and Distributions Diagram
    - Identifying Key Risks
    - Created Notion Board
    - Reviewed Data Definitions
    - Search Functionality
- **Aaron Rayray (8/10)**
    - High level mock ups
    - Splash,Sign In,Sign-Up,Home
    - Mentor,Profile,Workspace,Forums
    - Body portion of the webpages
    - Sign In Page(frontend)
    - Sign Up Page(frontend)
    - Forgot Password Page(frontend)
    - Identified key risks when going through frontend
- **Noah Hai (7/10)**
    - System Design
    - Created ERD
    - Created UML
- **Shez Rahman (9/10)**
    - Data Definitions
    - Prioritized functional reqs.
    - Reviewed Data Definitions
    - Sign in function of the backend
    - Sign up function of the backend
- **Ghadeer Al-Badani (9/10)**
    - High level api and algorithms
    - Created sign in functionality
    - Sign in function of the backend
    - Sign up function of the backend
    - Created README to instruct on how to
- **Majd Alnajjar (7/10)**
    - Headers design
    - Footers design

    - CodeConnect icon design
    - Forum page Design
    - About me Design
    - About me page
    - Forum Page
- **William Pan (8/10)**
    - High Level database design
    - Reviewed Data Definitions
    - Created ERD
    - Created EER
    - Forward Engineering
    - High level mockups
    - Search Functionality
    - Created Test DB
- **Phillip Ma (7/10)**
    - Created all use case diagrams
    - Created body sections for webpages
    - High level mockups
    - Identified key risks
    - Distributed front end tasks