UNIVERSITY AT ALBANY

STATE UNIVERSITY OF NEW YORK

ICSI 680: Master's Project

Group Project – Milestone 3

Team A (Alpha):

Alex Bailey

Brandon Williams

Darcy Woodruff

Henry Qiu

Shelley Massinga

Table of Contents

Architectural software design - Game Genome

Project Summary

Game Genome is a Pandora-like recommendation platform for video games. The platform collects user preferences through interactive quizzes and creates personalized stations[1] that generate game recommendations. Each station represents a different gaming taste profile, and users can create multiple stations based on different quiz responses. The system recommends games that match users' preferences, and learns from user feedback (likes/dislikes) to refine recommendations continuously.

Design Pattern Used

Model-View-Controller (MVC)

Software Components

High-Level Components

1. **Authentication System**
    - **Purpose**: Manage user accounts, login/logout functionality, and session management
    - **Components**: LoginController, SignUpController, AccountInformationController, usersAPI

2. **Quiz Module**
    - **Purpose**: Collect user preferences through multiple questions to build a profile for recommendations
    - **Components**: QuizController, QuizService, QuizRepository

3. **Station (Recommendations) Module**
    - **Purpose**: Generate personalized game recommendations based on quiz profiles
    - **Components**: StationController, RecommendationEngine, StationRepository

4. **Feedback Module**
    - **Purpose**: Process user feedback on recommendations to improve future suggestions

[1] This represents a key change since Milestone 1. Originally, the system was designed to provide a single recommendation page based on quiz results. We have now changed the design to support multiple stations (similar to Pandora's music stations), allowing users to create different recommendation feeds for various gaming moods and preferences.

- **Components**: FeedbackController, FeedbackProcessor, FeedbackRepository

5. **Game Data Module**
    - **Purpose**: Provide access to the game database and metadata
    - **Components**: GameRepository, GameDataService

6. **Machine Learning System**
    - **Purpose**: Power the recommendation engine with advanced algorithms
    - **Components**: APIHandler, DataLoader, Preprocessor, ContentBasedRecommender

7. **Favorites Module**
    - **Purpose**: Manage games the user liked
    - **Components**: FavoritesController

Class-Level Components

Authentication System

- **LoginController**
    - **Purpose**: Process login attempts
    - **Methods**: login(email, password)

- **SignUpController**
    - **Purpose**: Handle user registration
    - **Methods**: registerUser(email, password, passwordConfirm)

- **AccountInformationController**
    - **Purpose**: Manage user account settings
    - **Methods**: changeEmail(userId, newEmail), changePassword(userId, oldPassword, newPassword)

Quiz Module

- **QuizController**
    - **Purpose**: Manage user interaction for the quiz process
    - **Methods**: startQuiz(), submitQuizAnswers(answers), validateQuizAnswers(answers), completeQuiz()

- **QuizService**
    - **Purpose**: Process the logic for quiz answers
    - **Methods**: processAnswers(answers), generateProfile(answers), saveQuizResults(profile)

- **QuizRepository**
  - **Purpose**: Store and retrieve quiz results
  - **Methods**: save(profile), retrieve(userId), update(userId, profile)

Station Module

- **StationController**
  - **Purpose**: Handle the creation and management of recommendation stations
  - **Methods**: createStation(profile), getRecommendations(stationId), deleteStation(stationId), getStationsByUser(userId), compareStations(stationId1, stationId2), cloneStation(stationId, newName), switchStation(userId, stationId), mergeStations(stationId1, stationId2, newName)

- **RecommendationEngine**
  - **Purpose**: Algorithmically compute personalized recommendations
  - **Methods**: computeRecommendations(profile), filterByTags(tags), rankGames(profile)

- **StationRepository**
  - **Purpose**: Store user-created stations and associated recommendations
  - **Methods**: saveStation(stationData), getStation(stationId), deleteStation(stationId)

Feedback Module

- **FeedbackController**
  - **Purpose**: Handle user interactions with game recommendations
  - **Methods**: submitRating(userId, gameId, rating), skipGame(userId, gameId), favoriteGame(userId, gameId), tagGame(userId, gameId, tags[])
  -
- **FeedbackProcessor**
  - **Purpose**: Analyze user feedback to refine station preferences
  - **Methods**: processRating(stationId, gameId, rating), analyzeUserBehavior(userId, stationId), adjustStationPreferences(stationId, adjustments), refineRecommendations(stationId)

- **FeedbackRepository**
  - **Purpose**: Store user feedback on recommendations

    - **Methods**: saveRating(userId, stationId, gameId, rating), saveBehavior(userId, gameId, behaviorType), getFeedbackHistory(stationId), getUserPreferencePattern(userId)

Game Data Module

- **GameRepository**
    - **Purpose**: Provide access to the game database
    - **Methods**: getGameById(gameId), searchGames(criteria), getGamesByGenre(genres[]), getGamesByTags(tags[]), getSimilarGames(gameId)
- **GameDataService**
    - **Purpose**: Process and enrich game data for the recommendation engine
    - **Methods**: formatGameData(gameId), extractGameFeatures(game), compareGames(gameId1, gameId2), categorizeGame(gameId)

Machine Learning System

- **APIHandler**
    - **Purpose**: Gateway between frontend and machine learning system
    - **Methods**: receive_POST(), send_response()

- **DataLoader**
    - **Purpose**: Handle data-related operations
    - **Methods**: get_user_data(), get_all_game_data(), get_game_name()

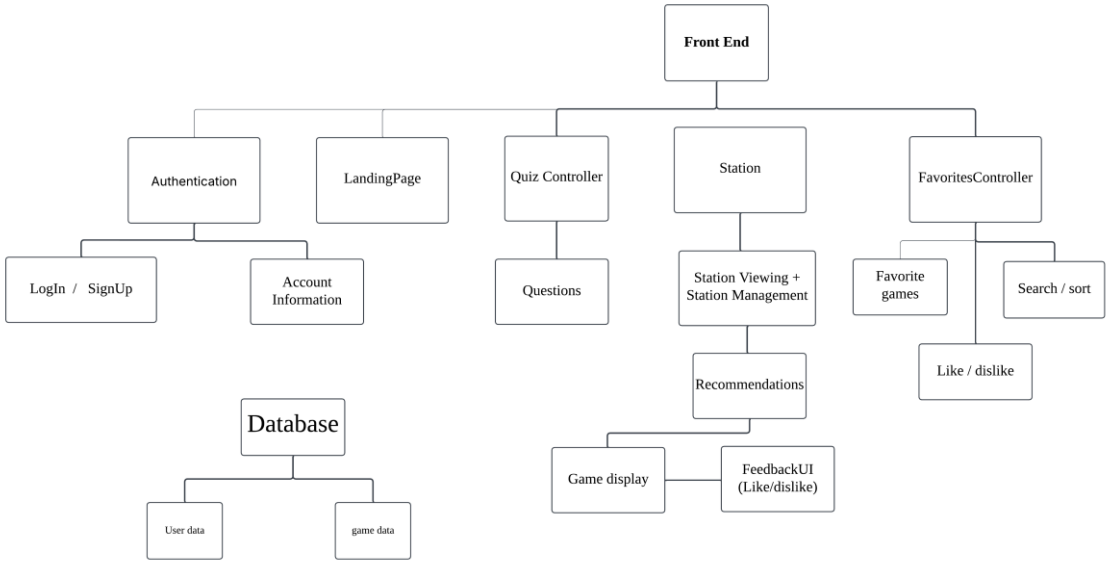
- **Preprocessor**
    - **Purpose**: Clean and transform raw data for machine learning algorithms
    - **Methods**: clean_data(), vectorize_data()

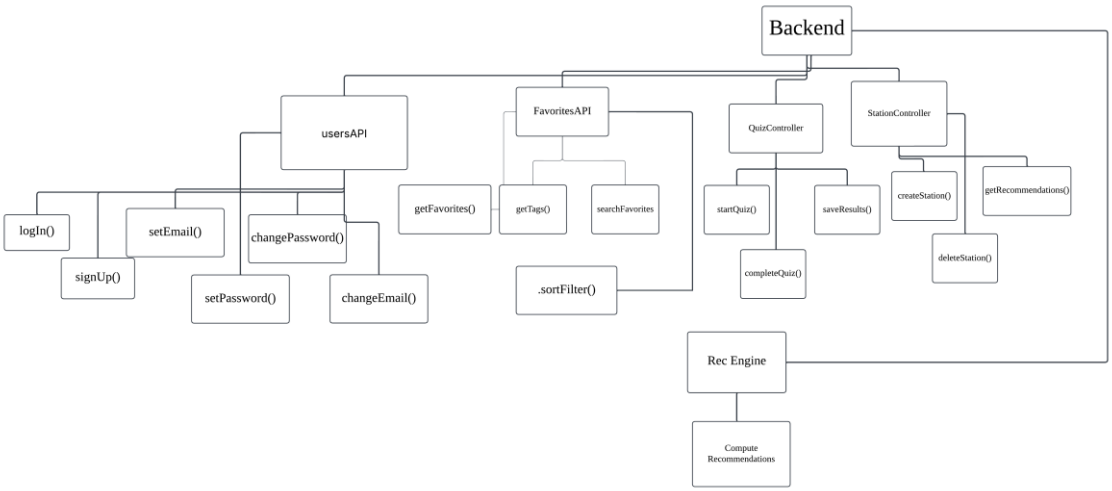
- **ContentBasedRecommender**
    - **Purpose**: Apply content-based filtering algorithms
    - **Methods**: compute_similarity(), recommend_top_games()

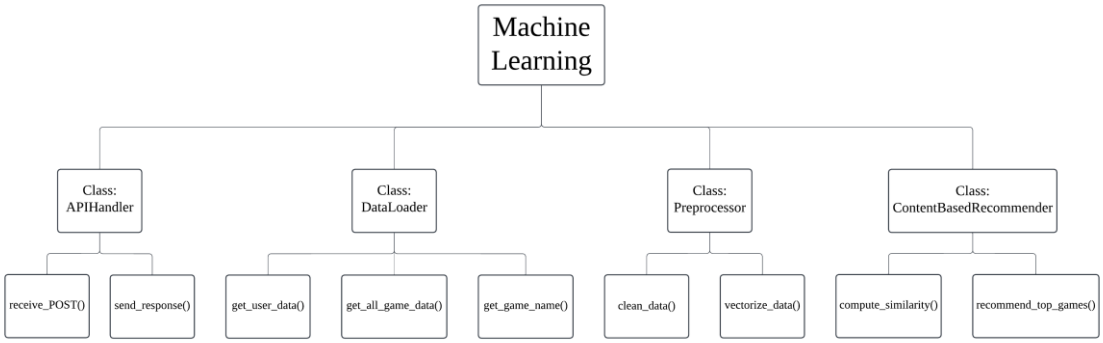
Favorites Module

- **FavoriteController**
    - Methods: loadFavorites(), likeGame(gameId), unlikeGame(gameId)

Wiring Diagram







Persistence

Data to be Persisted

1. **User Authentication Data**:
    - User credentials (email, password hash)
    - Account information
    - Session tokens

2. **User Quiz Profiles**:
    - Game preferences (favorite games, genres, playing style, gaming goals)
    - Station identifiers
    - Timestamp of quiz taken

3. **Stations/Recommendations**:
    - Station ID and metadata
    - User ID
    - Generated recommendations (game IDs)
    - User actions (favorites, likes, tags)

4. **Game Data**:
    - Game metadata (title, description, release date, etc.)
    - Game features (genres, tags, platforms)
    - User ratings and reviews

5. **Feedback Data**:
    - User ratings of recommendations
    - Interaction history
    - Custom tags

Storage Method

- **Database**: MongoDB (MongoDB Atlas)

Collections:

- **users:** UserID, Username, Email, PasswordHash, CreatedAt, UpdatedAt
- **quiz_profiles:** ProfileID, UserID, FavoriteGames[], Genres[], PlayingStyle[], GamingGoals[], CreatedAt
- **stations:** StationID, UserID, ProfileID, StationName, CreatedAt, LastPlayedAt

- **recommendations**: RecommendationID, StationID, GameID, Rank, Tags[]
- **game_feedback:** FeedbackID, UserID, StationID, GameID, RatingType, RatingValue, Timestamp
- **games:** GameID, Title, ReleaseDate, Developer, Publisher, Description, ImageURL
- **game_metadata:** GameID, Genres[], Tags[], Platforms[], Features[]
- **steam:** AppID, Name, ReleaseDate, DeveloperID, PublisherID, RequiredAge, Achievements, PositiveRatings, NegativeRatings, AveragePlatytime, MedianPlaytime, OwnersMin, OwnersMax, Price

Languages Used

Frontend

- JavaScript/TypeScript with React

Backend

- JavaScript/TypeScript with Node.js/Express

Machine Learning Components

- **Python**
  - Used for the recommendation engine's core algorithms
  - Powers the content-based filtering and similarity calculations
  - Connects to the Node.js backend through API endpoints

Decision Rationale

- React was chosen for the frontend due to its component-based architecture, which makes it ideal for creating reusable UI elements like game cards and stations.
- Node.js/Express was selected for the backend to maintain language consistency between frontend and backend, simplifying development and deployment.
- Python was chosen for the machine learning components due to its robust libraries for data processing and machine learning (NumPy, Pandas, scikit-learn).

Deployment

Web Deployment

The application will be deployed as a web application on AWS.