CSCI 4390 Senior Project Initial Proposal

Title: Sports Chat+

Team Members:

- **Ruben Aleman (ruben.aleman01@utrgv.edu)**
- **Joe Rivera (joe.rivera02@utrgv.edu)**
- **Abel Victorino (abel.victorino01@utrgv.edu)**
- **Emilio Aguirre (emilio.aguirre01@utrgv.edu)**

Adviser: Dr. Zhixiang Chen

Project Description

Introduction

Sports Chat+ is a database-driven web platform designed to enhance the March Madness experience by integrating real-time chat, fantasy league features, and friendly wagering. The system will efficiently manage teams, players, games, user interactions, and transactions to create a seamless sports engagement experience.

Problem Summary:
While sports fans have access to a wide variety of platforms for watching games or checking scores, there is a lack of unified, interactive environments where users can engage with each other, track teams, and simulate friendly competition in real time. Most existing platforms focus on either static updates or real-money betting, leaving a gap for fans who want a secure, community-driven experience.

Motivation:
SportsChat+ was developed to address this gap by offering a comprehensive platform for sports discussion, virtual betting, and bracket simulation — all in one place. For this version, we utilized the **NCAA March Madness** tournament as the primary dataset to demonstrate functionality and interaction. This event allowed us to design realistic game structures, user roles, and dynamic user engagement features, laying the groundwork for future expansion to other sports.

Comparison of Existing Solutions:

- **ESPN Tournament Challenge:** Offers bracket prediction but lacks real-time chat or betting simulation.
- **Reddit (r/CollegeBasketball):** Active for discussion but lacks structure, moderation, and integration with game data.
- **FanDuel / DraftKings:** Focus on real-money betting and do not support role-based user interaction or gamified tracking.

SportsChat+ fills these gaps by providing an interactive, moderated platform for fans to discuss, predict, and engage — using virtual coins rather than real currency, and structured roles (Admin, Moderator, User) to ensure safety and clarity.

Proposed Solution Summary:
SportsChat+ is a web-based application featuring:

- A secure login and registration system with encrypted passwords and role-based access.
- A relational Azure SQL database supporting team data, player stats, game logs, and betting history.
- Real-time chat rooms linked to specific teams and games.
- A virtual betting system with coin balances and wager tracking.
- Scalable architecture that supports multiple sports beyond March Madness in future versions.

Functional Requirements

SportsChat+ provides users with an interactive platform to engage with sports events through real-time discussions, virtual betting, and bracket tracking. For this implementation, the NCAA March Madness tournament was used as a test case.

The core functional requirements include:

• **User Registration & Authentication**
Users can sign up with a username, email, and password. All passwords are securely hashed using bcrypt. Authentication enables role-based access to features.

• **Role-Based Access**
Three roles are supported: Admin, Moderator, and User.
– Admins manage users and system settings

– Moderators handle chat content
– Users can place bets, chat, and track games

• **Team and Player Data Management**
Team and player data for March Madness is stored and retrievable through SQL queries. Player stats are updated based on game results.

• **Game and Bracket Tracking**
Games are tracked by round, and winners are advanced using bracket logic. The Next_Game and Bracket_Seeding tables support this progression.

• **Real-Time Chat Rooms**
Users can enter chat rooms tied to a team, game, or general discussion. Messages are stored with timestamps and associated with both a user and chat room.

• **Virtual Betting System**
Users can place bets using virtual coins. Each bet is logged with wager amounts, involved users, and game reference. Bets can be updated based on game outcome.

• **Transaction & Payment Logging**
Although no real money is exchanged, all coin transactions are logged in Payments and PaymentTransactions tables for auditability.

• **API Logs and Monitoring**
System interactions and API activity are recorded in API_Logs for tracking external data and debugging integration.

Architecture Overview

The **SportsChat+** platform is built using a modular, full-stack web application architecture with a focus on maintainability, security, and scalability.

Frontend:
• Developed with **HTML, CSS, and JavaScript**
• Uses **EJS templates** for rendering server-side dynamic pages
• Provides user-friendly interfaces for login, chat, betting, and bracket navigation
• Responsive layout supports desktop and mobile access

Backend:
• Built with **Node.js** and the **Express.js** framework
• Routes are defined for user actions including login, registration, betting, chat messaging, and API log interaction
• Uses **bcrypt** for password hashing and validation
• Implements middleware for authentication and error handling

Database:
• Hosted on **Azure SQL Database**
• Structured as a fully relational schema with 17 tables, normalized to reduce redundancy
• Enforces data integrity with foreign keys and constraints
• Includes indexes on frequently accessed fields like UserID, TeamID, and GameID to improve query performance

APIs and Integration:
• Integration layer fetches or simulates data for team matchups and brackets
• All API interactions are logged using the API_Logs table
• Additional routes allow dynamic interaction with team/player data

Deployment:
• The application is fully deployed using **Azure Web Apps**
• GitHub integration allows continuous deployment and version control
• Environment variables and connection strings are managed securely in Azure

Algorithms & Data Structures

The design of SportsChat+ balances relational database principles with application-level logic to ensure consistent, performant interactions across features like bracket updates, chat, and betting.

Normalization Strategy:
• All data is normalized to **3rd Normal Form (3NF)** to avoid redundancy.
• Tables like Players, Teams, Games, and GameStats are linked by foreign keys to preserve data integrity and reduce duplication.
• User-related tables (Users, UserPermissions, Payments, Bets) are structured for modular interaction and future scalability.

Indexing and Performance Optimization:
• Indexes are applied to high-frequency query fields such as UserID, TeamID, GameID, and Email.
• Composite keys are avoided in favor of single-column integer primary keys for faster joins and simplified indexing.
• This indexing strategy improves query speed for leaderboard generation, login lookup, and chat retrieval.

Bracket Progression Logic:
• The Next_Game table maps the flow of games across rounds (e.g., winner of Game A feeds into Game B).
• A backend function checks the WinnerID of a completed game and automatically updates the Team1ID or Team2ID of the next linked game.

Virtual Betting Workflow:
• The Bets table supports tracking wagers between two users on a specific game.
• Logic ensures both users must confirm the wager. After a game is completed, a backend script determines the winner and adjusts coin balances.
• Coin changes are recorded in the PaymentTransactions table, supporting auditability and rollback if needed.

Chat Messaging Storage:
• Chat messages are written in real time to the ChatMessages table, linked to Users and ChatRooms.
• Messages are sorted by timestamp, and pagination is supported for performance.

Security Algorithms:
• User passwords are encrypted using **bcrypt**, which applies salted hashing to prevent plain-text password exposure.
• Role-based access enforcement is handled through the UserPermissions table joined with RoleLookup, restricting access to sensitive routes and operations.

Code Roadmap

The **SportsChat+** web application is organized into modular components across backend and frontend directories, following best practices for scalability and clarity.

1. Backend – sportschat-backend/

Handles all server-side operations, API routing, and database connectivity.

- server.js: Initializes the Express server, sets up middleware, connects to the database, and defines routing logic.
- fetchNCAAGames.js: Custom script for retrieving NCAA game data (e.g., from an API or web scraping) to populate the database.
- test-server.js: Optional server testing entry point.
- .env: Stores environment variables such as DB connection strings and API keys (excluded from GitHub).
- package.json: Declares server dependencies (e.g., Express, CORS, dotenv).

2. Frontend – sportschat-frontend/src/

Built with **React**, this directory contains user interface components, services, and style files.

a. Pages – /src/pages/

These React components represent individual screens of the app.

- LoginPage.js, SignupPage.js, ForgotPasswordPage.js: User authentication screens.
- HomePage.js: Main hub for accessing features.
- DashboardPage.js: Displays team stats, standings, and possibly brackets.
- TeamsPage.js, StatsPage.js, SimplifiedDatabaseTestPage.js: Specialized views for NCAA team data and testing output.
- Button.js: Reusable UI component for buttons.
- Logo.png: Static image used in headers or auth pages.

b. Services – /src/services/

- api.js: Centralized module for frontend-backend communication using fetch or axios. Handles API calls like login, fetch teams, post messages, etc.

c. Styles –

- App.css, style.css, index.css: CSS files for component-level and global styling.
- logo.svg: Application logo used in headers or footers.

d. Other React Files –

- App.js: Root component defining main app structure and routing.
- App.test.js, setupTests.js, reportWebVitals.js: Boilerplate files for performance monitoring and Jest testing.
- index.js: Entry point for React rendering.

3. Templates – templates/

Store sample HTML, export files, or test page templates used during development.

Table Naming Conventions

To maintain consistency in database structure:

- **Primary Keys (PK)**: The primary key of each table is the unique identifier, named as `<TableName>ID`.
  Example: `TeamID` (PK), `UserID` (PK).

- **Foreign Keys (FK)**: Foreign keys reference related tables, following the pattern `<ReferencedTable>ID`.
  Example: `TeamID` (FK) in Players references `Teams(TeamID)`.

- **Standard Naming**: Table names are plural (e.g., `Teams`, `Users`).
  – Column names use camelCase or underscores for readability (e.g.,

`FavoriteTeamID`, `date_played`).
– Avoid reserved keywords or ambiguous abbreviations.

- **Constraint Naming:**
  Constraints follow a standardized format for clarity:
  – Primary Key: `PK_<TableName>`
  – Foreign Key: `FK_<ChildTable>_<ParentTable>`
  – Unique Constraint: `UQ_<TableName>_<ColumnName>`
  – Check Constraint: `CK_<TableName>_<ConditionDescription>`
  Example: `FK_GameStats_Games`, `CK_Teams_SeedRange`.

-

Data Retrieval Efficiency

To ensure optimal performance, the database will implement:

- **Indexing**: Frequently queried fields like UserID, TeamID, and GameID will be indexed to speed up search operations.
- **Normalization**: Data will be structured to **minimize redundancy**, ensuring efficient storage and retrieval.
- **Optimized Queries**: Joins and transactions will be designed to reduce query execution time, improving real-time data updates.

User Access Control

To manage security and access, user roles will be stored in the **UserPermissions** table:

- **Admin**: Full control over database management, user bans, and system settings.
- **Moderator**: Limited privileges, including chat moderation and managing user-generated content.
- **User**: Standard access, including participating in discussions, viewing games, and placing virtual bets.

User roles will restrict actions like **modifying game data, making payments, or changing permissions** to authorized personnel only.

Security Considerations

To protect user data and transactions, the system will implement:

- **Password Hashing**: User passwords will be stored using bcrypt for encryption and security.

- **Transaction Security**: Virtual coin transactions and game betting records will be verified and encrypted.
- **API Access Control**: Rate limiting and authentication measures will prevent unauthorized API requests and spam.
- **Role-Based Access Control (RBAC)**: Ensures only authorized users perform critical operations.

Database Design Overview

Table Name	Purpose
1. Teams	Stores NCAA team information, including names, coaches, and bracket seeding.
2. RoleLookup	Stores definitions of user roles (e.g., Admin, Moderator, User).
3. Users	Stores user credentials, profile info, and favorite team references.
4. UserPermissions	Links users to their assigned roles using the RoleLookup table.
5. Players	Tracks player details, including position, height, and team assignment.
6. Games	Stores game schedules, teams involved, scores, and round information.
7. GameStats	Logs player performance metrics for individual games.
8. ChatRooms	Defines chat spaces associated with teams, games, or general topics.
9. ChatMessages	Stores user-generated messages within chat rooms.
10. API_Teams	Stores external team data mapped to local team records.
11. API_Games	Stores external game data linked to local game entries.
12. API_Logs	Logs activity related to API transactions and integrations.
13. Payments	Records real-money coin purchases made by users.
14. PaymentTransactions	Tracks detailed coin transactions tied to specific payments.
15. Bets	Stores wagers placed between users on game outcomes.
16. Bracket_Seeding	Assigns official tournament seed numbers to teams.
17. Next_Game	Links current games to their next scheduled matchups in the bracket.

Core Tables

1. Teams Table

Stores NCAA team details participating in March Madness.

```
CREATE TABLE Teams (
   TeamID INT PRIMARY KEY,
   TeamName VARCHAR(100) NOT NULL UNIQUE,
   CoachName VARCHAR(100),
   Conference VARCHAR(100),
   Wins INT DEFAULT 0,
   Losses INT DEFAULT 0,
   Seed INT CHECK (Seed BETWEEN 1 AND 16)
);
```

2. RoleLookup Table

```
CREATE TABLE RoleLookup (
   RoleID INT PRIMARY KEY,
   RoleName VARCHAR(50) NOT NULL UNIQUE
);
```

3. Users Table

Tracks scheduled and completed games.

```
CREATE TABLE Users (
   UserID INT PRIMARY KEY,
   Username VARCHAR(50) UNIQUE NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL,
   PasswordHash VARCHAR(255) NOT NULL,
   FavoriteTeamID INT,
   FOREIGN KEY (FavoriteTeamID) REFERENCES Teams(TeamID)
);
```

4. UserPermissions Table

```
CREATE TABLE UserPermissions (
    UserID INT PRIMARY KEY,
    RoleID INT NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (RoleID) REFERENCES RoleLookup(RoleID)
);
```

5. Players Table

```
CREATE TABLE Players (
    PlayerID INT PRIMARY KEY,
    TeamID INT,
    PlayerName VARCHAR(100) NOT NULL,
    Position VARCHAR(20),
    HeightCM INT,
    WeightKG INT,
    JerseyNumber INT,
    FOREIGN KEY (TeamID) REFERENCES Teams(TeamID)
);
```

6. Games Table

```
CREATE TABLE Games (
    GameID INT PRIMARY KEY,
    Round VARCHAR(50),
    DatePlayed DATE,
    Location VARCHAR(255),
    Team1ID INT,
    Team2ID INT,
    WinnerID INT,
    ScoreTeam1 INT,
    ScoreTeam2 INT,
    FOREIGN KEY (Team1ID) REFERENCES Teams(TeamID),
    FOREIGN KEY (Team2ID) REFERENCES Teams(TeamID),
    FOREIGN KEY (WinnerID) REFERENCES Teams(TeamID)
);
```

7. GameStats Table

```
CREATE TABLE GameStats (
  StatID INT PRIMARY KEY,
  GameID INT,
  PlayerID INT,
  Points INT DEFAULT 0,
  Rebounds INT DEFAULT 0,
  Assists INT DEFAULT 0,
  Steals INT DEFAULT 0,
  Blocks INT DEFAULT 0,
  MinutesPlayed INT DEFAULT 0,
  FOREIGN KEY (GameID) REFERENCES Games(GameID),
  FOREIGN KEY (PlayerID) REFERENCES Players(PlayerID)
);
```

8. ChatRooms Table

```
CREATE TABLE ChatRooms (
  RoomID INT PRIMARY KEY,
  RoomName VARCHAR(255) NOT NULL UNIQUE,
  RoomType VARCHAR(20),
  TeamID INT,
  GameID INT,
  FOREIGN KEY (TeamID) REFERENCES Teams(TeamID),
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
```

9. ChatMessages Table

```
CREATE TABLE ChatMessages (
  MessageID INT PRIMARY KEY,
  RoomID INT,
  UserID INT,
  Message TEXT NOT NULL,
  Timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (RoomID) REFERENCES ChatRooms(RoomID),
  FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

10. API_Teams Table

```
CREATE TABLE API_Teams (
  API_TeamID INT PRIMARY KEY,
  ExternalName VARCHAR(100),
  MappedTeamID INT,
  FOREIGN KEY (MappedTeamID) REFERENCES Teams(TeamID)
);
```

11. API_Games Table

```
CREATE TABLE API_Games (
  API_GameID INT PRIMARY KEY,
  GameID INT,
  SourceName VARCHAR(100),
  FOREIGN KEY (GameID) REFERENCES Games(GameID)
);
```

12. API_Logs Table

```
CREATE TABLE API_Logs (
  LogID INT PRIMARY KEY,
  Description TEXT,
  Timestamp DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

13. Payments Table

```
CREATE TABLE Payments (
  PaymentID INT PRIMARY KEY,
  UserID INT,
  Amount INT NOT NULL,
  PaymentDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

14. PaymentTransactions Table

```
CREATE TABLE PaymentTransactions (
  TransactionID INT PRIMARY KEY,
  PaymentID INT,
  UserID INT,
  CoinChange INT,
  TransactionDate DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID),
  FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

15. Bets Table

```
CREATE TABLE Bets (
  BetID INT PRIMARY KEY,
  GameID INT NOT NULL,
  UserID INT NOT NULL,
  User2ID INT NOT NULL,
  WagerAmount INT NOT NULL,
  BetStatus VARCHAR(20),
  WinnerID INT,
  FOREIGN KEY (GameID) REFERENCES Games(GameID),
  FOREIGN KEY (UserID) REFERENCES Users(UserID),
  FOREIGN KEY (User2ID) REFERENCES Users(UserID),
  FOREIGN KEY (WinnerID) REFERENCES Users(UserID)
);
```

16. Bracket_Seeding Table

```
CREATE TABLE Bracket_Seeding (
  SeedingID INT PRIMARY KEY,
  TeamID INT,
  SeedNumber INT,
  FOREIGN KEY (TeamID) REFERENCES Teams(TeamID)
);
```

17. Next_Game Table

```
CREATE TABLE Next_Game (
  NextGameID INT PRIMARY KEY,
  GameID INT,
  NextGameRefID INT,
  FOREIGN KEY (GameID) REFERENCES Games(GameID),
    FOREIGN KEY (NextGameRefID) REFERENCES Games(GameID)
);
```

Testing and Evaluation

Testing was conducted on both the frontend and backend of the **SportsChat+** application to ensure system reliability, correct behavior, and secure data flow. Below are the structured approaches used for evaluation:

Frontend Testing

• Component Rendering
Each React page component (e.g., LoginPage.js, DashboardPage.js, TeamsPage.js) was tested to ensure proper rendering without runtime errors. Layout integrity was manually verified across different screen sizes.

• Form Validation
User inputs were validated in real-time on the SignupPage.js, LoginPage.js, and ForgotPasswordPage.js. Scenarios tested include empty fields, mismatched passwords, and invalid email formats.

• API Integration Testing
The api.js service file was used to connect frontend forms to backend endpoints. Functional GET and POST requests were tested using Axios to confirm correct data handling:
– Team data loading into TeamsPage.js
– Game statistics display in StatsPage.js
– Chat message submission and retrieval

Backend Testing

• Endpoint Testing
Using Postman and manual fetches from the frontend, all routes in server.js were tested. Routes covered include:
– POST /login, POST /signup
– GET /teams, GET /games, GET /stats
– POST /chat, POST /bet

• Database Integrity Checks
The fetchNCAAGames.js script was used to seed the database. Foreign key constraints were verified, and deletions from tables such as Teams and Users were confirmed to cascade properly to related records like Players and ChatMessages.

• **Betting System Validation**
The virtual coin balance (UserCoins) and bet resolution logic (Bets) were tested for both winning and losing users. Transactions correctly updated balances and recorded winners.

• **Environment Variable Usage**
The .env file was used to manage database credentials and security keys. Environment switching was validated for both local and Azure deployments.

Deployment Testing

• **Azure Web App Validation**
After pushing code to GitHub, the application was deployed using Azure Web Apps. Live deployment was tested for:
– Working routes
– Chat functionality
– Live updates to teams and games
– CORS handling between frontend and backend

Bug Tracking and Fixes

• Layout bugs and minor rendering issues were resolved in files like DashboardPage.js and StatsPage.js.
• Incorrect API error handling was fixed in LoginPage.js and SignupPage.js.
• Coin validation bugs during bet placement were corrected in backend logic.

ER Diagram

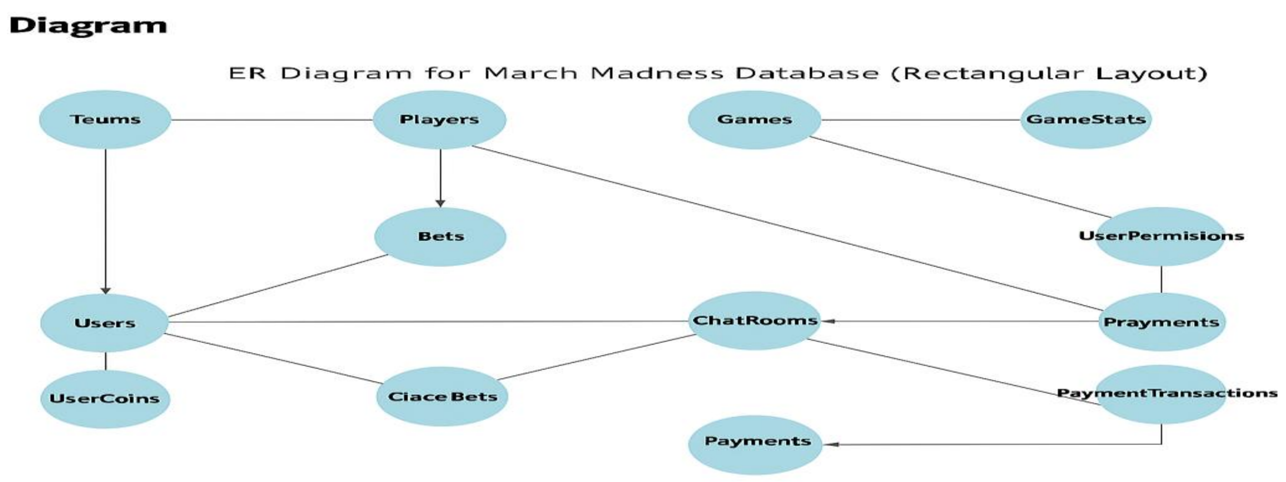
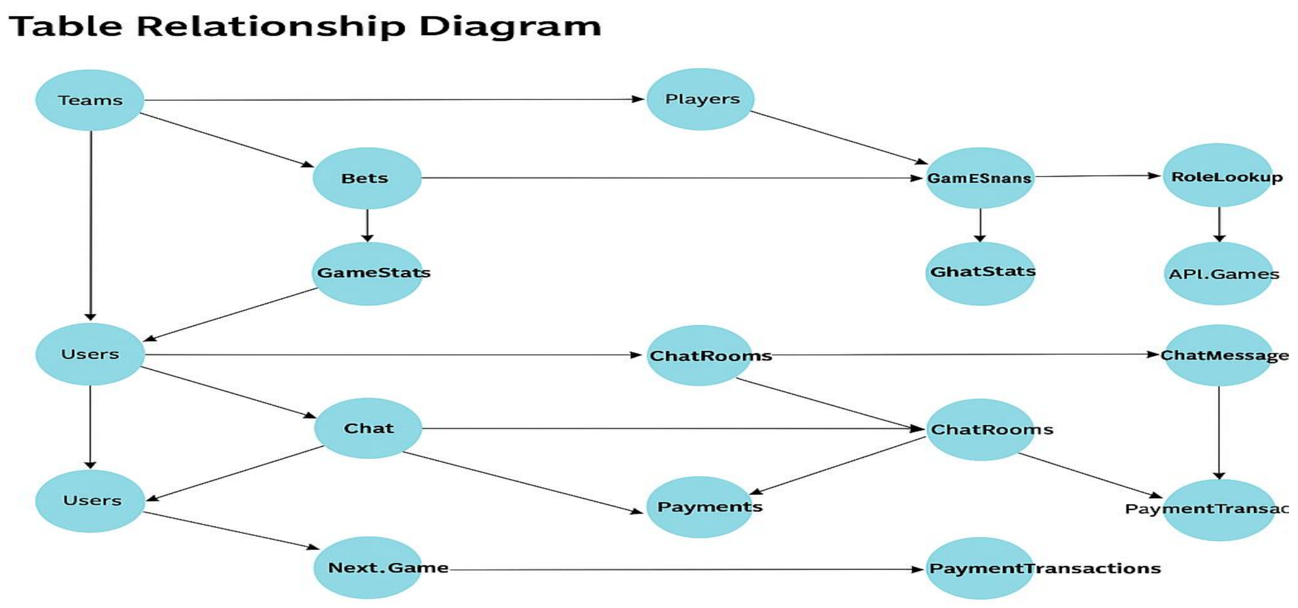
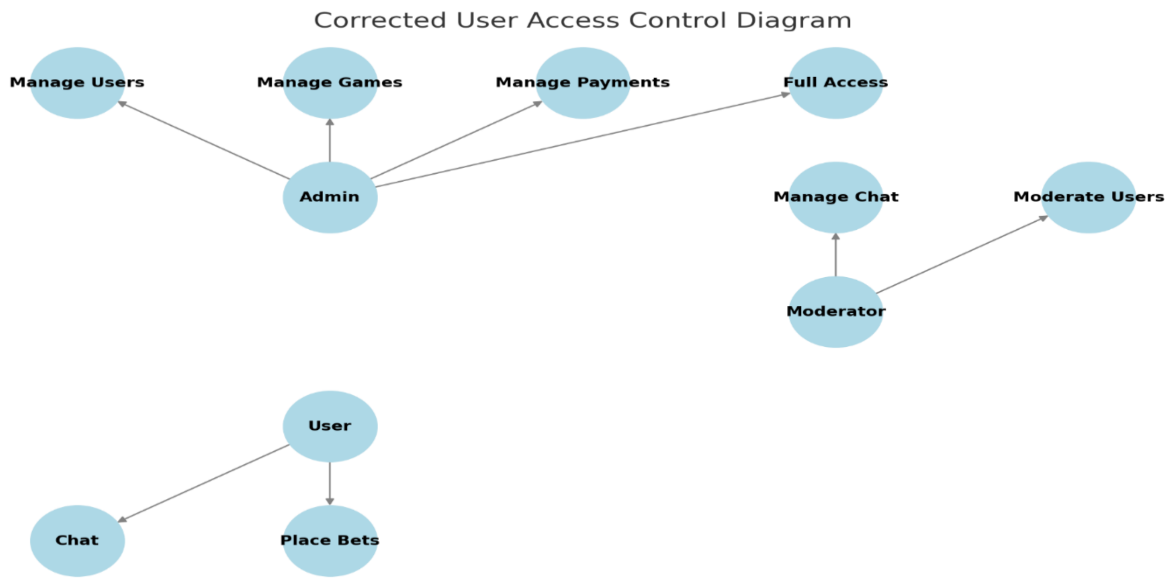


Table Relationship Diagram



User Access Diagram



DATA Flow Diagram

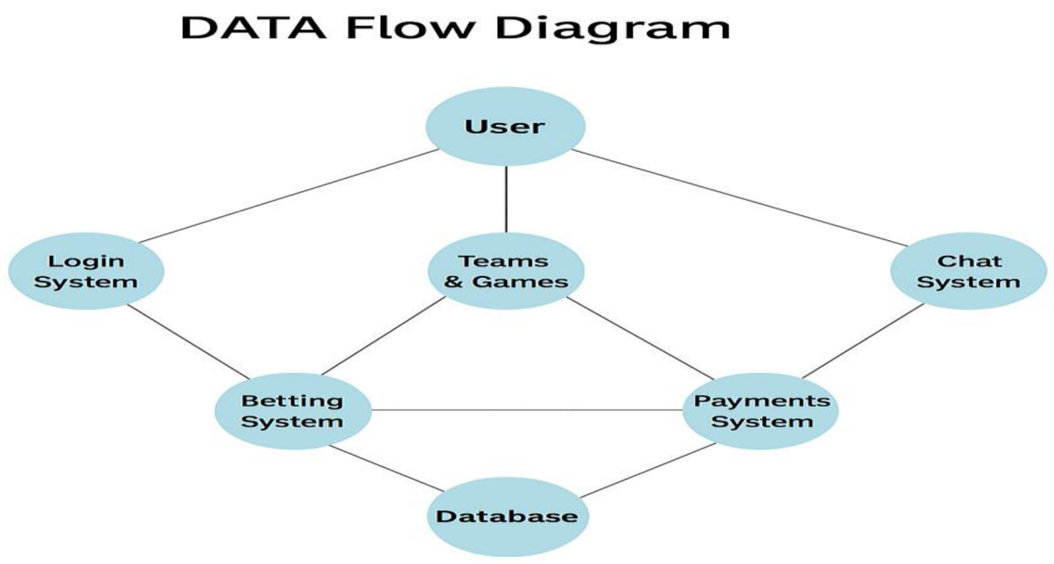
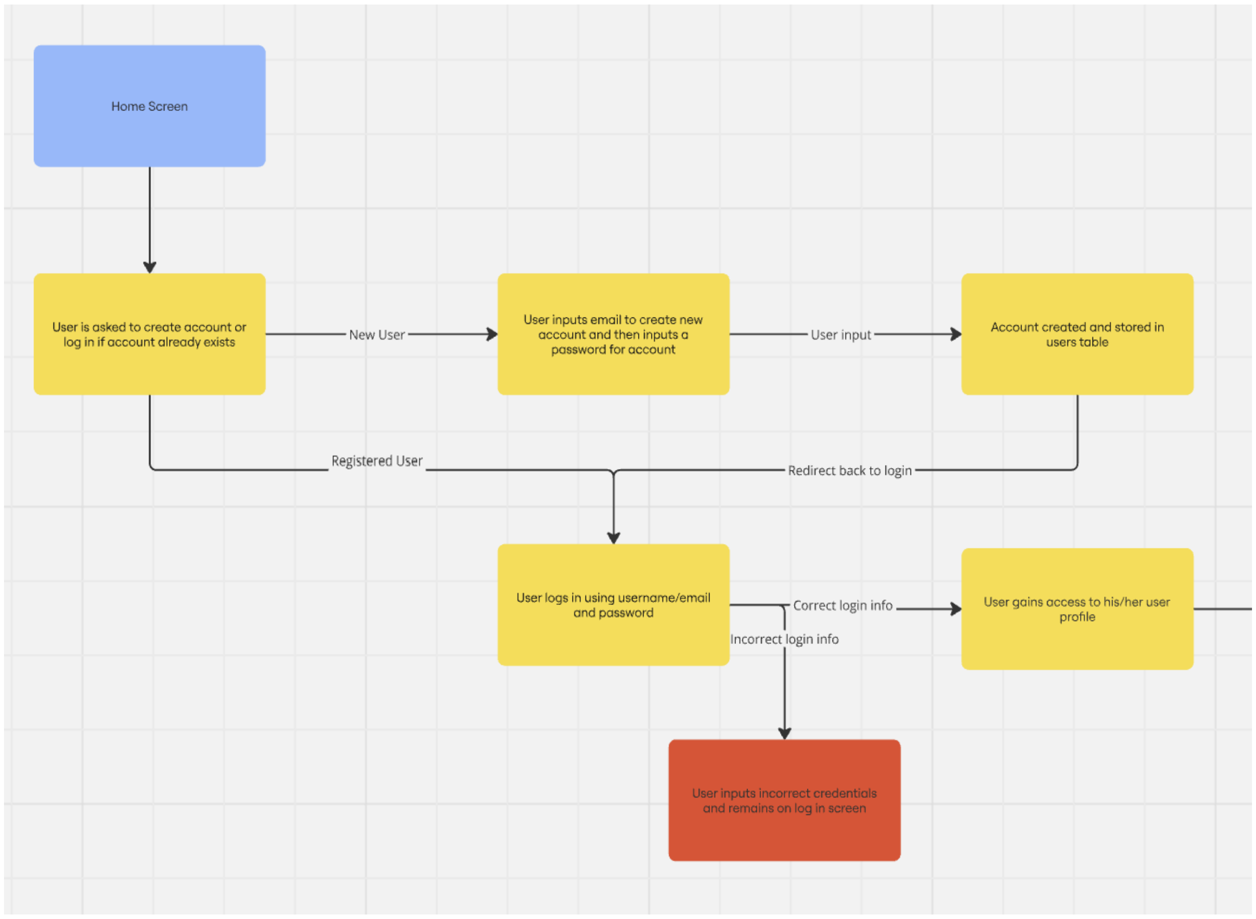


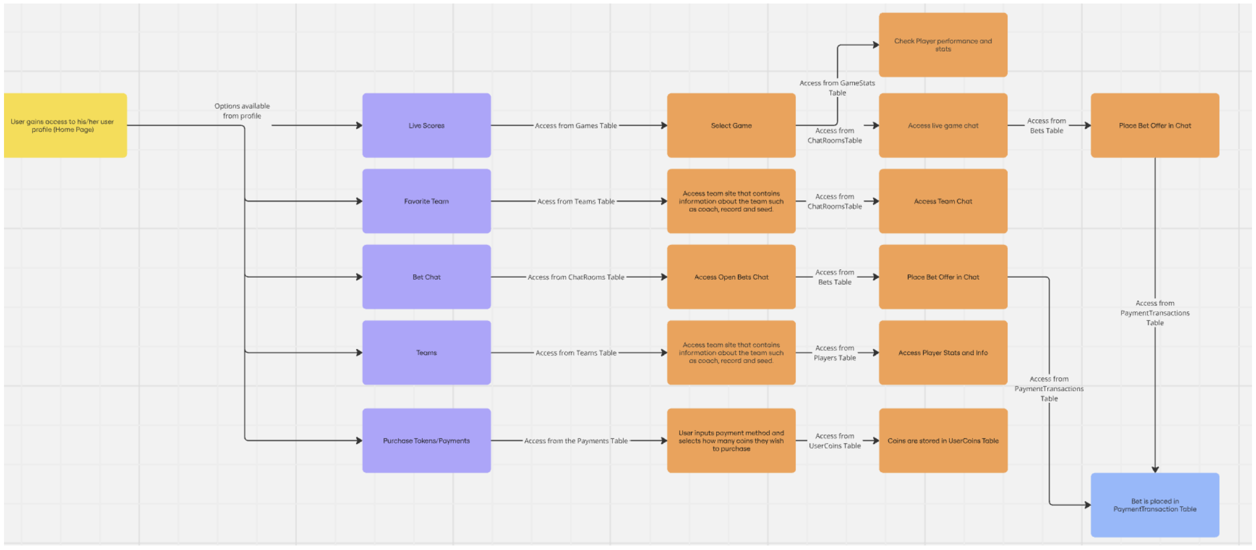
Table Name	**Purpose**
Teams	Stores NCAA teams
Players	Tracks team rosters
Games	Manages game schedule and results
Users	Stores user information
GameStats	Tracks player performance
Chat	Supports real-time messaging
UserPermissions	Controls user access levels

SportsChat+ Control Flow Table

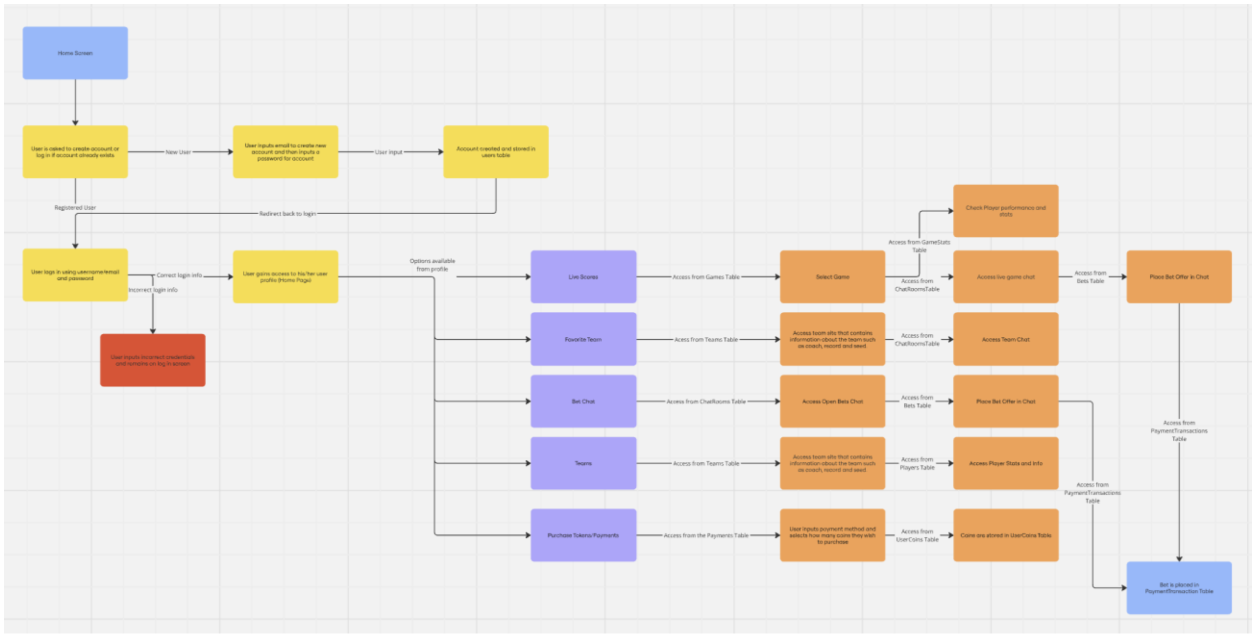
Login Procedures


Home Screen
User is asked to create account or log in if account already exists
New User
User inputs email to create new account and then inputs a password for account
User input
Account created and stored in users table
Registered User
Redirect back to login
User logs in using username/email and password
Correct login info
Incorrect login info
User gains access to his/her user profile
User inputs incorrect credentials and remains on log in screen

Website Flow


User gains access to his/her user profile (Home Page)
Options available from profile
Live Scores
Favorite Team
Bet Chat
Teams
Purchase Tokens/Payments
Access from Games Table
Acess from Teams Table
Access from ChatRooms Table
Access from Teams Table
Access from the Payments Table
Select Game
Access team site that contains information about the team such as coach, record and seed.
Access Open Bets Chat
Access team site that contains information about the team such as coach, record and seed.
User inputs payment method and selects how many coins they wish to purchase
Access from GameStats Table
Access from ChatRoomsTable
Access from ChatRoomsTable
Access from Bets Table
Access from Players Table
Access from UserCoins Table
Check Player performance and stats
Access live game chat
Access Team Chat
Place Bet Offer in Chat
Access Player Stats and Info
Coins are stored in UserCoins Table
Access from Bets Table
Place Bet Offer in Chat
Access from PaymentTransactions Table
Access from PaymentTransactions Table
Bet is placed in PaymentTransaction Table

Overview


Home Screen
User is asked to create account or log in if account already exists
New User
User inputs email to create new account and then inputs a password for account
User input
Account created and stored in users table
Registered User
Redirect back to login
User logs in using username/email and password
Correct login info
Incorrect login info
User gains access to his/her user profile (Home Page)
User inputs incorrect credentials and remains on log in screen
Options available from profile
Live Scores
Favorite Team
Bet Chat
Teams
Purchase Tokens/Payments
Access from Games Table
Acess from Teams Table
Access from ChatRooms Table
Access from Teams Table
Access from the Payments Table
Select Game
Access team site that contains information about the team such as coach, record and seed.
Access Open Bets Chat
Access team site that contains information about the team such as coach, record and seed.
User inputs payment method and selects how many coins they wish to purchase
Access from GameStats Table
Access from ChatRoomsTable
Access from ChatRoomsTable
Access from Bets Table
Access from Players Table
Access from UserCoins Table
Check Player performance and stats
Access live game chat
Access Team Chat
Place Bet Offer in Chat
Access Player Stats and Info
Coins are stored in UserCoins Table
Access from Bets Table
Place Bet Offer in Chat
Access from PaymentTransactions Table
Access from PaymentTransactions Table
Bet is placed in PaymentTransaction Table

Interface Mockups

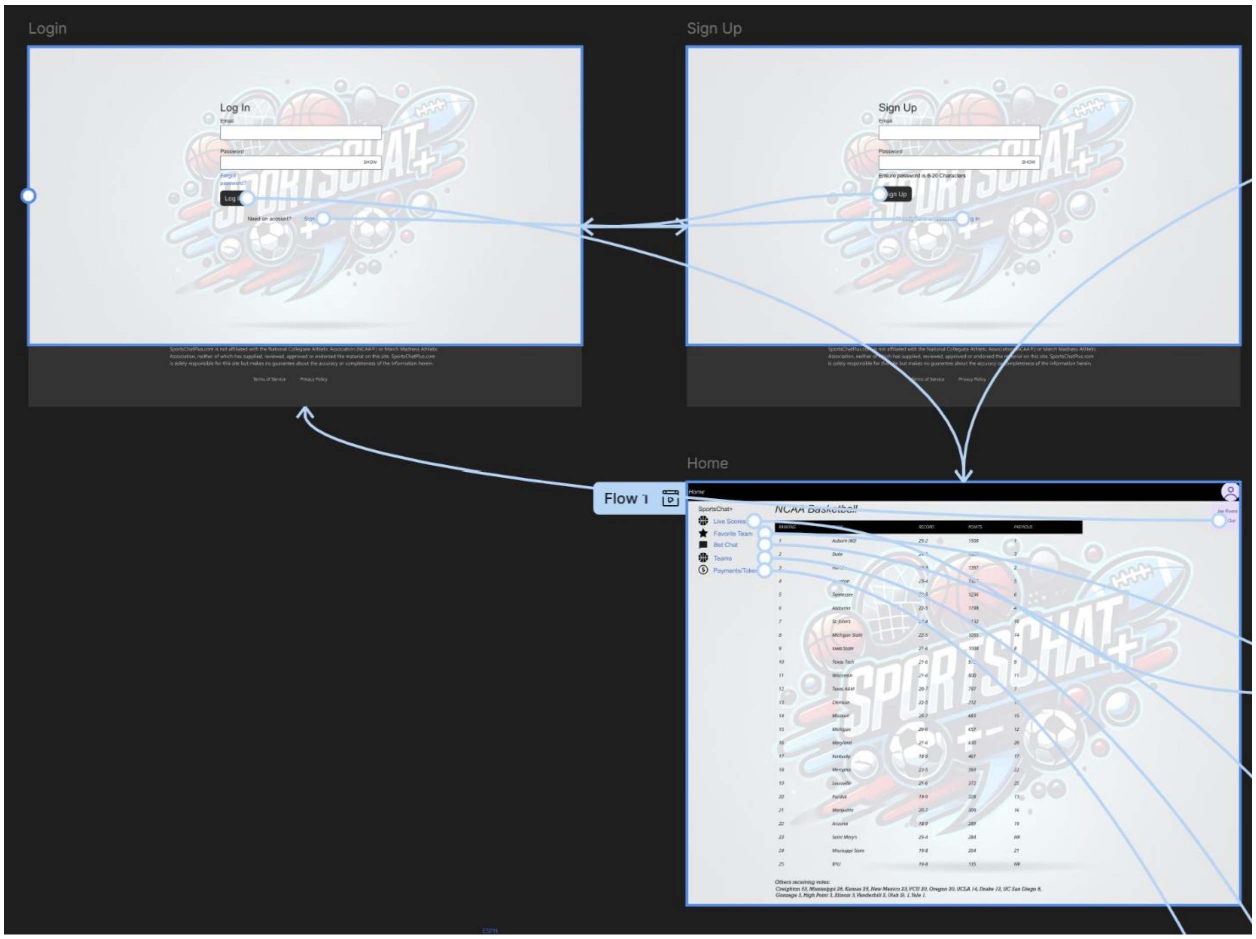
The following Figma mockups were designed during the planning phase to visualize the user experience.

- **Login Screen:** Allows users to sign in securely using their credentials.
- **Dashboard / Bracket View:** Displays upcoming games, teams, and tournament progression. Interactive features include bracket selection, virtual bets, and chat windows.

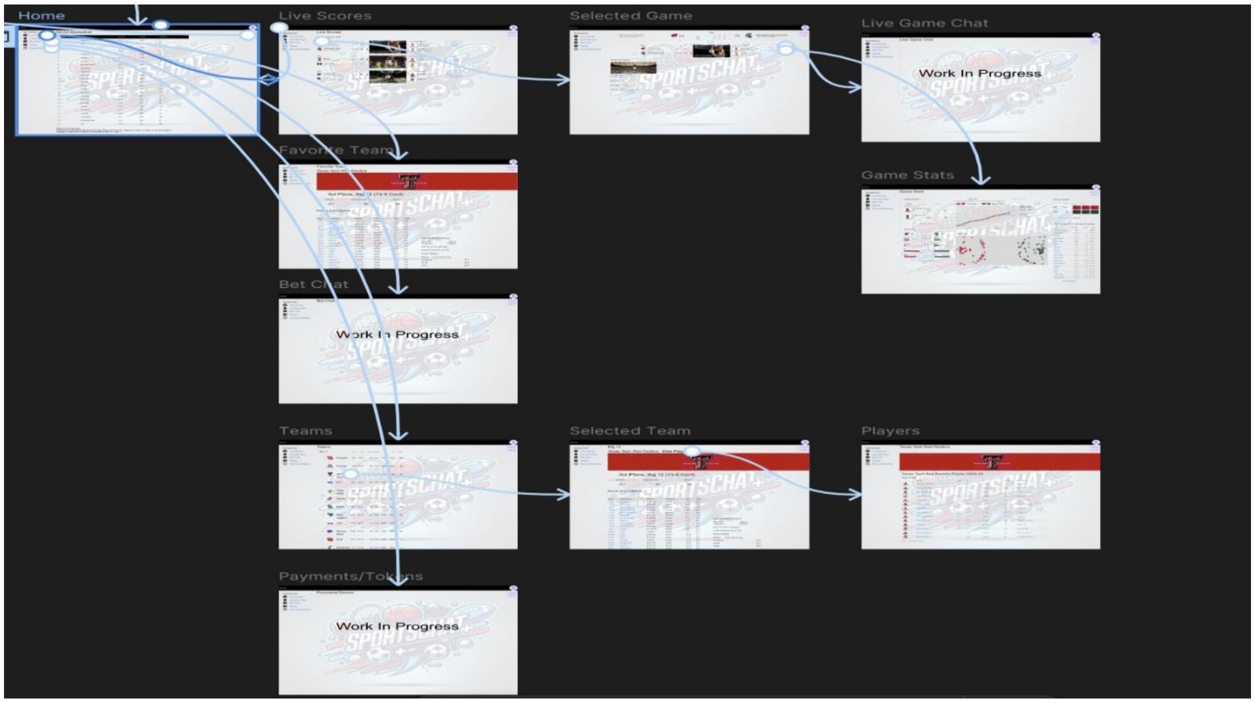
This follows the exact same flow design as the provided flow chart to give a visual representation of what we are intending our site to look like. This design symbolizes our minimum working product, we have intentions to add more features and expand our site as the semester progresses.

SportsChat+ Prototype

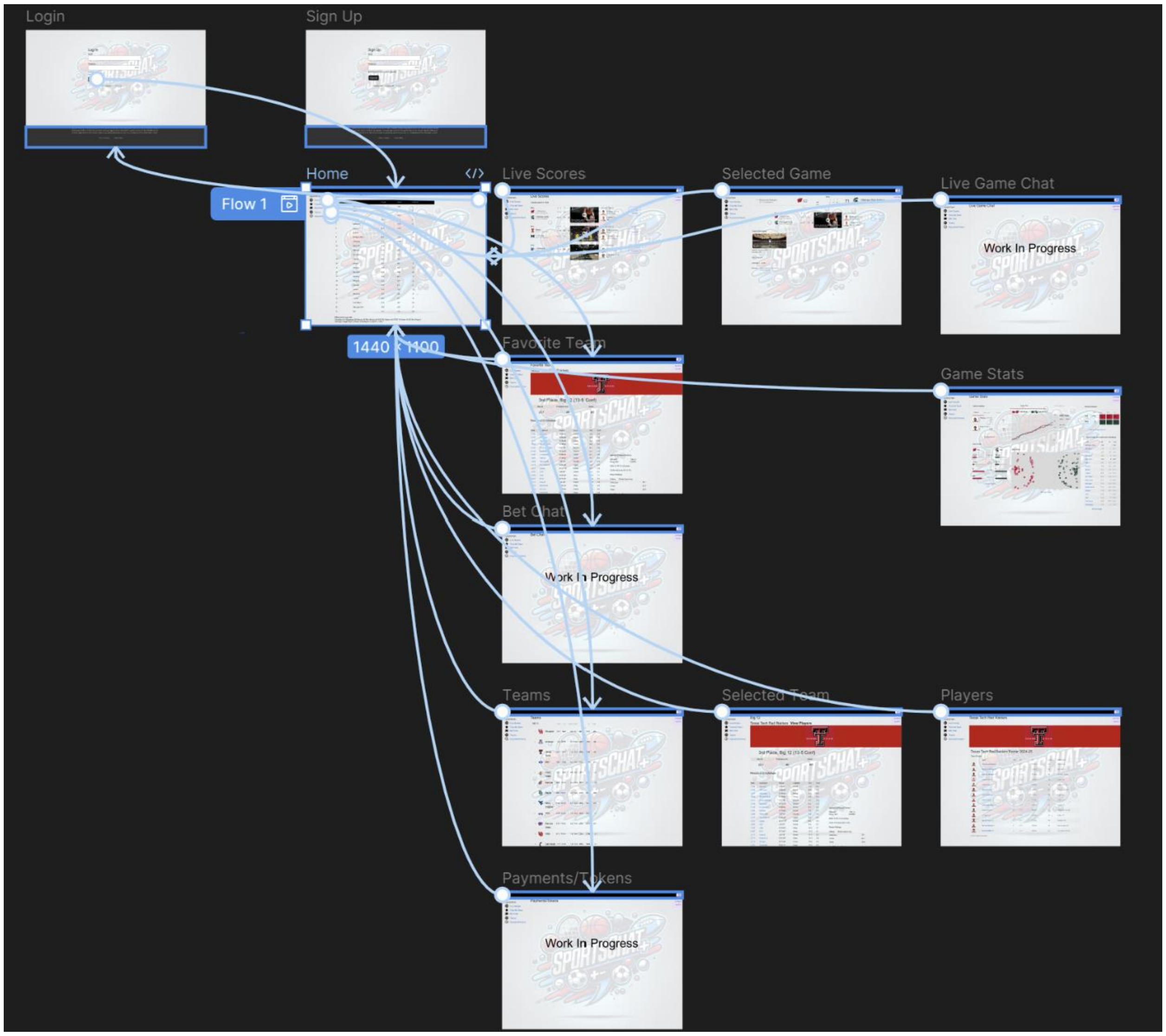
Login Procedures



Website Flow



Overview



The link for the prototype:

https://www.figma.com/design/VL292phxTjEam8au1gGGr6/SportsChat%2B?node-id=16-2344&t=yBMXGsD8UrhRSgRp-1

Project Management

The development of **SportsChat+** followed an iterative and collaborative workflow, focused on practical deliverables and consistent progress. Although it was built as a solo project, industry-standard tools and practices were used to simulate team development.

• **Version Control:**
All project files were managed using **Git** and hosted on **GitHub**. Feature-specific

branches were used (e.g., dashboard-ui, api-auth, sql-schema) to ensure modular progress and reduce merge conflicts. Clear commit messages tracked each stage of development.

• **Task Tracking (Jira):**
A **Jira Kanban board** was used to manage tasks and milestones throughout the project. Tasks were categorized by phase (e.g., frontend, backend, database) and labeled as "To Do," "In Progress," or "Done."
Epics included:
– Authentication System
– Chat & Betting Features
– Azure Deployment
– Testing & Documentation

• **Milestone-Based Development:**
The project followed a milestone-driven timeline:
- **Week 1:** Project setup, database schema finalized, user auth created
- **Week 2:** Game tracking, betting logic, and database interactions
- **Week 3:** Chat system and dashboard UI built
- **Week 4:** Testing, deployment to Azure Web Apps, final report

• **Challenges & Resolutions:**
- Azure SQL connection configuration was a key obstacle early on
- CORS issues were resolved by configuring middleware in Express
- Bracket logic required refactoring to properly reference Next_Game links
- Frontend alignment and mobile responsiveness took additional iteration

Lessons Learned & Evaluation

Developing **SportsChat+** offered hands-on experience with full-stack development, cloud deployment, and database design under realistic project constraints. The process not only reinforced technical skills but also highlighted the importance of design planning, modular structure, and testing discipline.

• **Key Takeaways:**
– Working with **Azure SQL** taught the importance of managing remote database permissions and secure connections.
– Designing a **normalized relational schema** helped improve understanding of data integrity, foreign keys, and JOIN queries.
– Using **React with Express** improved experience in routing, component reuse, and frontend-backend separation.

– **Jira task management** helped track progress and simulate a collaborative workflow with accountability.
– Implementing **role-based access control** solidified backend logic security.

• **What Worked Well:**
– Deployment to **Azure Web Apps** was successful, enabling real-time testing of production behavior.
– The chat and betting systems were implemented cleanly with minimal bugs.
– SQL indexing and normalization made data retrieval fast and reliable.

• **What Could Be Improved:**
– A stronger initial design for **state management** in React (e.g., use of useContext or Redux) would have improved scalability.
– Unit and integration tests could have been automated to ensure reliability after updates.
– More detailed **error handling** and user feedback would improve UX, especially for failed bets or login errors.

• **Future Enhancements:**
– Implement OAuth (e.g., Google login) for easier authentication
– Add user profile customization and avatars
– Integrate a live API for NCAA scores and automate bracket updates
– Expand chat with WebSockets for real-time interaction