CENG3004: Software Engineering

ASTRO

Software Requirements Specification Document

04/05/2025

Team Members:

Barış Berişbek 210709052
Doğukan Taha Tıraş 210709076
Seyfullah Korkmaz 210709004
Yeşim Arslan 220709030

Contents

1 Overview

ASTRO is a mobile application designed as a football match reservation system and a social platform for amateur athletes. It enables users to organize matches, manage teams, and connect with other players. The system integrates venue booking, real-time matchmaking, player rating, and community-building features into a single platform. The system allows users to:

- **Create and Manage Matches**: Team captains can set up matches by selecting venues (e.g., futsal courts, university sports halls), specifying time slots, and inviting players.
- **Search and Join Matches**: Players can find nearby matches using filters like location, date, or skill level.
- **Build Teams**: Users can form teams, invite members, and assign roles (e.g., captain, player).
- **Rate Players and Track Leagues**: Post-match ratings contribute to a dynamic league system, fostering friendly competition.
- **Communicate in Real-Time**: Integrated messaging and push notifications keep participants updated on match changes or invitations.

Why ASTRO?
Traditional methods for organizing amateur matches often rely on fragmented tools like WhatsApp groups or manual venue bookings, leading to inefficiencies and miscommunication. ASTRO addresses these pain points by:

1. Centralizing match organization, team management, and payments into one platform.
2. Reducing administrative overhead for facility operators, who can update availability and pricing seamlessly.
3. Enhancing visibility for amateur players through skill ratings and league standings.

Target Audience

- **Amateur Athletes**: Individuals seeking regular football activities.
- **Team Captains**: Those organizing teams and matches.
- **Facility Operators**: Venue owners managing bookings and pricing.
- **Sports Community Builders**: Organizers aiming to grow local football networks.

2 Requirements

2.1 Functional Requirements

Requirement Identifier	Source	Priority	Description
FR-1	ASTRO Project Documentation	High	Users can register for an account with email, password, and basic profile details.
FR-2	ASTRO Project Documentation	High	Users can log in to the system using their email and password.
FR-3	Derived from Problem Statement	High	Team captains can create teams, invite members, and assign roles (e.g., captain, player).
FR-4	Derived from Problem Statement	High	Users can create matches by selecting venues, dates, and player requirements.
FR-5	ASTRO Project Documentation	High	Users can search for matches using filters (location, date, skill level).
FR-6	Derived from Problem Statement	Medium	Users can join existing matches and confirm participation.
FR-7	ASTRO Project Documentation	High	Users can send and receive private and group messages in real-time.
FR-8	Derived from Problem Statement	High	Users receive push notifications for match invitations, reminders, and updates.

Requirement Identifier	Source	Priority	Description
FR-9	ASTRO Project Documentation	Medium	Facility operators can update venue availability and pricing via a dedicated dashboard.
FR-10	Derived from Problem Statement	Medium	Users can upload profile pictures, team logos, and match-related media.
FR-11	ASTRO Project Documentation	Low	Users can view a history of past matches and participation records.

2.2 Nonfunctional Requirements

Requirement Identifier	Source	Priority	Description
NR-1	ASTRO Project Documentation	High	**Security**: All user data must be encrypted during transmission and storage.
NR-2	ASTRO Project Documentation	High	**Performance**: API response time must be under 200ms for critical operations.
NR-3	Derived from Problem Statement	High	**Scalability**: The system must support up to 10,000 concurrent users.
NR-4	ASTRO Project Documentation	Medium	**Usability**: The interface must be intuitive, requiring minimal training for new users.
NR-5	ASTRO Project Documentation	High	**Reliability**: The system must achieve 99.9% uptime, excluding scheduled maintenance.
NR-6	ASTRO Project Documentation	Medium	**Compatibility**: The application must function seamlessly on Android and iOS platforms.
NR-7	Derived from Problem Statement	Low	**Localization**: Support for Turkish and English languages.
NR-8	Technology Constraint	High	**Technology**: Frontend must be developed using React Native.
NR-9	Technology Constraint	High	**Technology**: Backend must use Node.js/Express.js and PostgreSQL.
NR-10	Technology Constraint	Medium	**Real-time Communication**: Socket.IO must be used for real-time updates.
NR-11	Technical Requirement	Medium	**Recovery**: Daily backups and ability to restore data within 24 hours.
NR-12	Legal Compliance	Low	**GDPR Compliance**: Data storage must adhere to GDPR for EU users.

3 Actors and Roles

1. User

Role:
Represents individuals (amateurs, team captains, sports enthusiasts) who use the mobile application to organize matches, manage teams, and engage with the community.

Key Interactions:

- **Authentication:**
  - Register via email/password.
  - Log in and manage account security (e.g., password reset).
- **Profile Management:**
  - Update personal details (name, contact information).
  - Upload profile pictures (stored in MinIO).
- **Team Management:**
  - Create teams, invite members, assign roles (captain/player).
  - Upload team logos and manage team details.
- **Match Organization:**
  - Create matches (select venue, date, player requirements).
  - Search for matches using filters (location, skill level).
  - Join matches and confirm participation.
- **Messaging:**
  - Send/receive private and group messages in real-time (via Socket.IO).
- **Notifications:**
  - Receive push notifications for invitations, match updates, and reminders.
- **Media Management:**
  - Upload match highlights or photos (stored in MinIO).

2. Facility Operator (Venue Manager)

Role:
Represents venue owners or managers (e.g., futsal court operators) who update availability and pricing for their facilities.

Key Interactions:

- **Update Venue Availability:**
  - Mark reservation slots as "booked" or "available".
- **Pricing Management:**
  - Set or adjust fees for venue bookings.
- **Reservation Confirmation:**
  - View incoming booking requests and approve/deny them.

4 3. Payment Gateway (External System)

Role:
An external service integrated into ASTRO to handle secure transactions.

Key Interactions:

- **Payment Processing:**
  - Process payments for venue bookings.
- **Validation & Confirmation:**
  - Validate transactions and send confirmation receipts.

4. Guest (Unauthenticated User)

Role:
Represents unregistered or logged-out users who can perform limited actions without authentication.

Key Interactions:

- **Browse Matches:**
  - Search and view public match details (without joining).
- **View Team Profiles:**
  - Access basic team information (name, logo, members).
- **Initiate Registration/Login:**
  - Navigate to registration or login screens.
- **Password Reset:**
  - Request a password reset link via email.

5. System (Implicit Actor)

Role:
Manages automated processes such as data validation, payment handling, and machine learning analysis in the background.

Key Functions:

- Store and query data using PostgreSQL.
- Manage media files via MinIO.
- Enable real-time messaging through Socket.IO.
- Detect suspicious voting patterns using machine learning models.

4. Use Casess

UC-1: User Registration

Use Case Identifier	**UC-1**
Use Case Name	User Registration
Participating Actors	Guest, System
Flow of Events	1. Guest selects the "Register" option. 2. System displays the registration form. 3. Guest enters required information (e.g., username, email, password). 4. Guest submits the registration form. 5. **<<Include>>** System validates the input data. 5a. **Extension Point:** If validation fails, **<<Extend>>** System displays an error message (Display Login/Error). 6. **<<Include>>** System verifies the email and password uniqueness/strength. 6a. **Extension Point:** If verification fails (e.g., email already exists), **<<Extend>>** System displays an appropriate error message (Display Login/Error). 7. System creates a new user account in PostgreSQL (NR-9). 8. System encrypts and stores user credentials (NR-1). 9. System potentially sends a verification email (implicit step for email verification). 10. System logs the new user in (or directs them to log in).
Entry Condition	Guest is unauthenticated and accesses the registration interface.
Exit Conditions	- **Success:** Guest becomes a registered User, account created, user may be logged in. - **Failure:** Registration fails, error message displayed, Guest remains unauthenticated.
Related NFRs	NR-1 (Security), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology), NR-12 (GDPR Compliance)

UC-2: Login

Use Case Identifier	**UC-2**
Use Case Name	Login
Participating Actors	Guest/User, System
Flow of Events	1. Guest/User navigates to the Login interface. 2. Guest/User enters their credentials (email/username and password). 3. Guest/User submits the login request. 4. System retrieves stored credentials from PostgreSQL (NR-9). 5. System verifies the provided credentials against the stored, encrypted data (NR-1). 6. **Extension Point:** If credentials are invalid, **<<Extend>>** System displays a login error message (Display Login/Error) and denies access. 7. **Extension Point:** If user forgot password, **<<Extend>>** User initiates the "Reset Password" flow. 8. If credentials are valid, System establishes an authenticated session for the User. 9. System redirects the User to their dashboard or intended page.
Entry Condition	Guest/User is unauthenticated and accesses the login interface.
Exit Conditions	- **Success:** User is authenticated and has an active session. - **Failure:** User remains unauthenticated, error message displayed.
Related NFRs	NR-1 (Security), NR-2 (Performance), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology)

UC-3: Team Creation

Use Case Identifier	**UC-3**
Use Case Name	Team Creation
Participating Actors	User, System
Flow of Events	1. User navigates to the "Create Team" section. 2. System displays the team creation form.

	3. User enters team details (e.g., name, description, logo - *potentially involves Upload Media*). 4. User submits the team creation request. 5. System validates the input. 6. System saves the new team information in PostgreSQL (NR-9). 7. System assigns the creating User as the initial Team Captain. 8. System confirms team creation to the User. 9. **Extension Point:** An approval process may be required **<<Extend>>**(Approve/Deny the Team Request - *implies an Admin/System role not fully detailed*).
Entry Condition	User is authenticated.
Exit Conditions	- **Success:** Team is created, User is assigned as Captain. - **Failure:** Team is not created, error message displayed.
Related NFRs	NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology), NR-11 (Recovery)

UC-4: Create Match

Use Case Identifier	**UC-4**
Use Case Name	Create Match
Participating Actors	User (likely Team Captain/Organizer), System, Payment Gateway (optional), Facility Operator (optional)
Flow of Events	1. User navigates to the "Create Match" section. 2. System displays the match creation form. 3. User enters match details (e.g., date, time, location, opponent team if applicable, rules, potential fees). 4. User submits the match creation request. 5. System validates the input. 6. **Extension Point:** If match requires payment, **<<Extend>>**Initiate "Process Payment" flow. 6a. **<<Include>>** "Online Payment" or "Payment on Facility" use case.

	7. **Extension Point:** If match requires approval (e.g., from opponent or facility), **<<Extend>>** System sends request for approval (Approve/Deny The Match Request). 8. System saves the match details in PostgreSQL (NR-9). 9. System confirms match creation (or pending status) to the User. 10. **Extension Point:** If approval is required from another team/organizer, **<<Extend>>** "Send Request to Match Organizer" might be triggered.
Entry Condition	User is authenticated and has necessary permissions (e.g., Team Captain, Organizer).
Exit Conditions	- **Success:** Match is created/scheduled (potentially pending approval or payment). - **Failure:** Match is not created, error message displayed.
Related NFRs	NR-1 (Security - if payment involved), NR-2 (Performance), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology), NR-11 (Recovery)

UC-5: Search and Join Team

Use Case Identifier	**UC-5**
Use Case Name	Search and Join Team
Participating Actors	User, System
Flow of Events	1. User navigates to the "Search Teams" section. 2. User enters search criteria (e.g., team name, location). 3. System queries PostgreSQL (NR-9) and displays matching teams. 4. User selects a team to view details. 5. User selects the "Join Team" or "Send Request to Join" option. 6. **Extension Point:** If team requires approval to join, **<<Extend>>**System sends a request to the Team Captain ("Send Request to Team Captain"). 6a. Team Captain uses "Approve/Deny the Request" functionality (extension of managing team requests). 7. If team is open or request is approved, System adds the User to the team roster in PostgreSQL (NR-9). 8. System confirms the action (joined or request sent) to the User.

Entry Condition	User is authenticated.
Exit Conditions	- **Success:** User has joined the team or sent a request to join. - **Failure:** User fails to join or send request, error message may be displayed.
Related NFRs	NR-2 (Performance - search), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology)

UC-6: Access Previous Matches and Active Matches

Use Case Identifier	**UC-6**
Use Case Name	Access Previous Matches and Active Matches
Participating Actors	User, System
Flow of Events	1. User navigates to their match history or active matches section. 2. System queries PostgreSQL (NR-9) for matches associated with the User or their team(s). 3. System filters matches based on status (e.g., upcoming, ongoing, completed). 4. System displays the list of relevant matches. 5. User can select a match to view details (e.g., score, participants, location, date/time). 6. **<<Include>>** System may retrieve data related to "Performance Analysis with Machine Learning" for completed matches.
Entry Condition	User is authenticated.
Exit Conditions	- **Success:** User views the list of active/previous matches and potentially details of a selected match. - **Failure:** System cannot retrieve match data, error message displayed.
Related NFRs	NR-2 (Performance - data retrieval), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology)

UC-7: Search and Join Match

Use Case Identifier	**UC-7**
Use Case Name	Search and Join Match
Participating Actors	User, System
Flow of Events	1. User navigates to the "Search Matches" or "Find Matches" section. 2. User enters search criteria (e.g., location, date range, skill level, open spots). 3. System queries PostgreSQL (NR-9) and displays matching open matches. 4. User selects a match to view details. 5. User selects the "Join Match" or "Request to Join Match" option. 6. **Extension Point:** If match requires approval, **<<Extend>>** System sends a request to the Match Organizer ("Send Request to Match Organizer"). 6a. Match Organizer uses "Approve/Deny The Match Request" functionality. 7. **Extension Point:** User might be promoted to a participant role **<<Extend>>** ("Promote to Match Participant"). 8. If match is open or request is approved, System adds the User/User's Team to the match roster in PostgreSQL (NR-9). 9. System confirms the action (joined or request sent) to the User.
Entry Condition	User is authenticated.
Exit Conditions	- **Success:** User/User's Team has joined the match or sent a request to join. - **Failure:**User fails to join or send request, error message may be displayed.
Related NFRs	NR-2 (Performance - search), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology)

UC-8: Real-time Messaging

Use Case Identifier	**UC-8**
Use Case Name	Real-time Messaging
Participating Actors	User, System
Flow of Events	1. User accesses a chat interface (e.g., team chat, match chat, direct message).

	2. System establishes a persistent connection using Socket.IO (NR-10). 3. User types and sends a message. 4. System receives the message via Socket.IO. 5. System potentially stores the message in PostgreSQL (NR-9) for history. 6. System broadcasts the message in real-time via Socket.IO to other relevant connected Users in the same chat context. 7. Receiving Users' interfaces display the new message instantly.
Entry Condition	User is authenticated and has accessed a chat context.
Exit Conditions	- **Success:** Message is sent and received by participants in real-time. - **Failure:**Message fails to send/receive, connection error may occur.
Related NFRs	NR-2 (Performance - message delivery), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology), NR-10 (Real-time Communication), NR-11 (Recovery - message history)

UC-9: Upload Media

Use Case Identifier	**UC-9**
Use Case Name	Upload Media
Participating Actors	User, System
Flow of Events	1. User initiates an action requiring media upload (e.g., setting profile picture, team logo, sharing match photos/videos). 2. System presents a file selection interface. 3. User selects a media file (image, video). 4. System uploads the file to a designated storage location (details depend on architecture, could be server filesystem or cloud storage like S3). 5. System validates the file (e.g., type, size). 5a. **Extension Point:** If validation fails, display error message. 6. System potentially processes the media (e.g., create thumbnails). 7. System stores a reference (e.g., URL) to the media in PostgreSQL (NR-9), associated with the relevant entity (User profile, Team, Match). 8. System confirms successful upload.

Entry Condition	User is authenticated and performing an action that allows media upload.
Exit Conditions	- **Success:** Media file is uploaded, stored, and associated with the correct entity. - **Failure:** Upload fails, error message displayed.
Related NFRs	NR-1 (Security - ensure uploaded files are safe), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology), NR-11 (Recovery)

UC-10: Configure Notifications (Provided as Example, included for completeness)

Use Case Identifier	**UC-10**
Use Case Name	Configure Notifications
Participating Actors	User
Flow of Events	1. User navigates to "Settings". 2. User toggles notification preferences (e.g., match reminders, messages). 3. System saves preferences to PostgreSQL (NR-9). 4. Future notifications adhere to configured settings.
Entry Condition	User is authenticated and in the settings menu.
Exit Conditions	- **Success:** Preferences saved. - **Failure:** Invalid configuration or server error.
Related NFRs	NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology)

UC-11: Edit Profile

Use Case Identifier	**UC-11**
Use Case Name	Edit Profile
Participating Actors	User, System

Flow of Events	1. User navigates to their profile section. 2. User selects an "Edit Profile" option. 3. System displays the profile editing form, pre-filled with current data. 4. User modifies their information (e.g., name, bio, profile picture - *potentially involves Upload Media*). 5. User submits the changes. 6. System validates the input data. 6a. **Extension Point:** If validation fails, display error message. 7. System updates the user's data in PostgreSQL (NR-9). 8. System confirms the profile has been updated.
Entry Condition	User is authenticated.
Exit Conditions	- **Success:** User profile information is updated. - **Failure:** Profile update fails, error message displayed.
Related NFRs	NR-1 (Security - if sensitive data is edited), NR-4 (Usability), NR-5 (Reliability), NR-9 (Technology), NR-11 (Recovery), NR-12 (GDPR Compliance)

UC-12: Online Payment

Use Case Identifier	**UC-12**
Use Case Name	Online Payment
Participating Actors	User, System, Payment Gateway
Flow of Events	1. **<<Include>>** This use case is typically included by another (e.g., "Create Match" if fees apply). 2. System determines the amount due. 3. System redirects or presents the User with the Payment Gateway interface. 4. User enters payment details (e.g., credit card info) securely into the Payment Gateway interface. 5. Payment Gateway processes the transaction. 6. Payment Gateway sends a response (success/failure) back to the System. 7. **Extension Point:** If payment fails, System informs the User and potentially cancels the originating action (e.g., match creation). 8. If payment succeeds, System receives confirmation.

	9. System records the successful payment transaction in PostgreSQL (NR-9). 10. System confirms successful payment to the User and proceeds with the originating action.
Entry Condition	User has initiated an action requiring online payment, amount is determined.
Exit Conditions	- **Success:** Payment is processed successfully, transaction recorded. - **Failure:**Payment fails, User is notified.
Related NFRs	NR-1 (Security), NR-2 (Performance), NR-5 (Reliability), NR-9 (Technology), NR-11 (Recovery)

UC-13: Payment on Facility

Use Case Identifier	**UC-13**
Use Case Name	Payment on Facility
Participating Actors	User, Facility Operator, System
Flow of Events	1. **<<Include>>** This use case might be part of another (e.g., "Create Match" or checking in for a match). 2. User indicates intention to pay at the facility (selected during match creation or check-in). 3. User arrives at the facility and interacts with the Facility Operator. 4. Facility Operator uses their interface (part of the ASTRO System) to locate the User/Match requiring payment. 5. Facility Operator collects payment physically from the User. 6. Facility Operator marks the payment as received within the System. 7. System updates the payment status for the User/Match in PostgreSQL (NR-9). 8. System potentially notifies the User (e.g., via app notification) that payment is confirmed.
Entry Condition	User has an outstanding payment due, to be paid physically at the facility. Facility Operator is logged into their system interface.
Exit Conditions	- **Success:** Payment is collected, System status is updated to 'Paid'. - **Failure:** Payment is not collected or not recorded in the system.

Related NFRs	NR-4 (Usability - for Facility Operator), NR-5 (Reliability), NR-9 (Technology), NR-11 (Recovery)

UC-14: Update League Standings

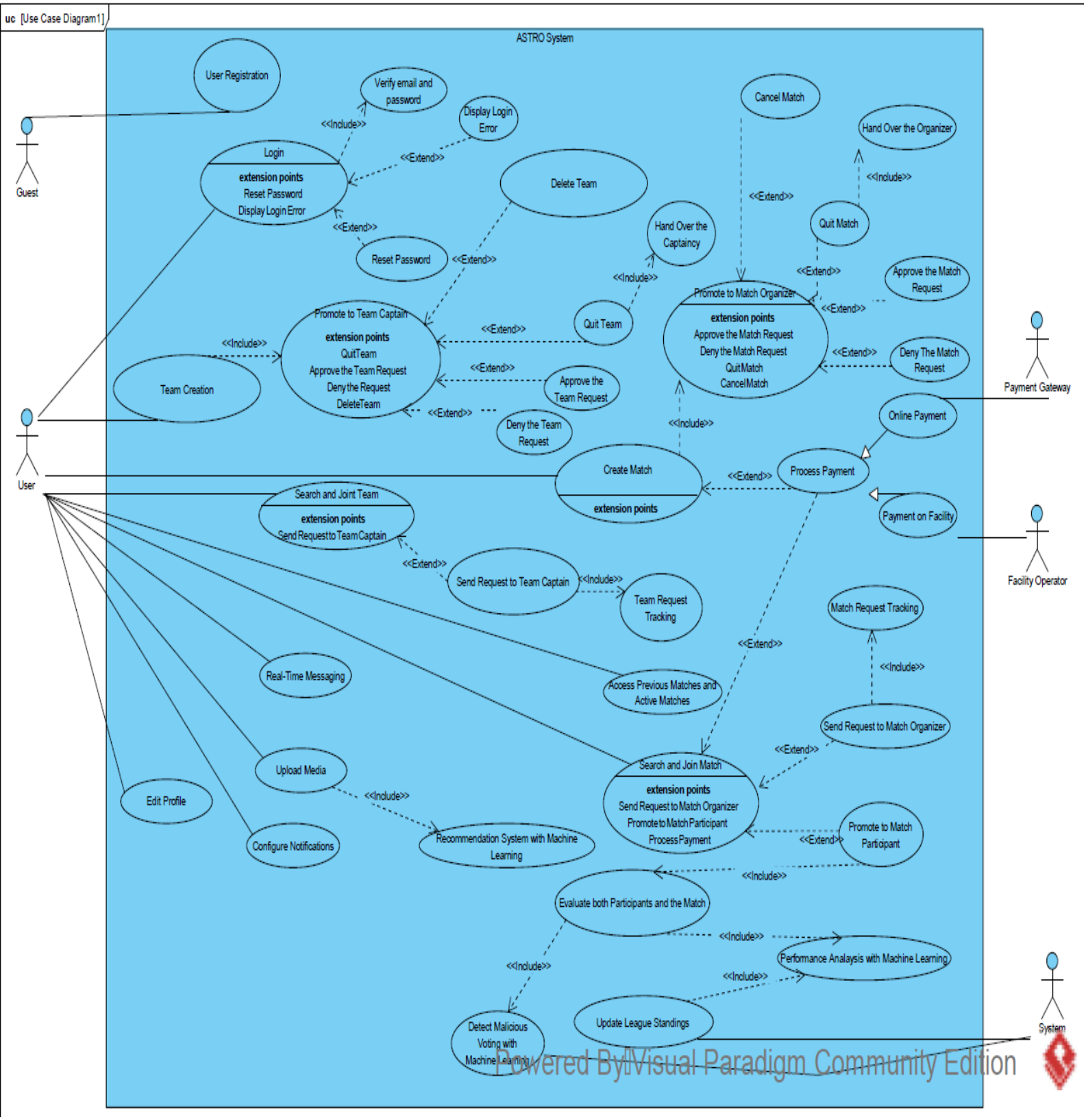
Use Case Identifier	**UC-14**
Use Case Name	Update League Standings
Participating Actors	System (potentially triggered by User/Admin action like reporting scores)
Flow of Events	1. A trigger event occurs (e.g., a match result is entered, a scheduled task runs). 2. System retrieves relevant match results and league rules from PostgreSQL (NR-9). 3. System calculates updated points, rankings, wins/losses/draws, etc., for teams participating in the league. 4. **Extension Point:** If scores seem suspicious or require validation, manual intervention might be flagged. 5. System updates the league standings table(s) in PostgreSQL (NR-9). 6. Updated standings are now available for Users to view.
Entry Condition	New match results relevant to a league are available or a scheduled update is triggered.
Exit Conditions	- **Success:** League standings are recalculated and updated accurately. - **Failure:**Standings fail to update or are updated incorrectly.
Related NFRs	NR-5 (Reliability - crucial for correctness), NR-9 (Technology), NR-11 (Recovery)

UC-15: Detect Malicious Voting with Machine Learning

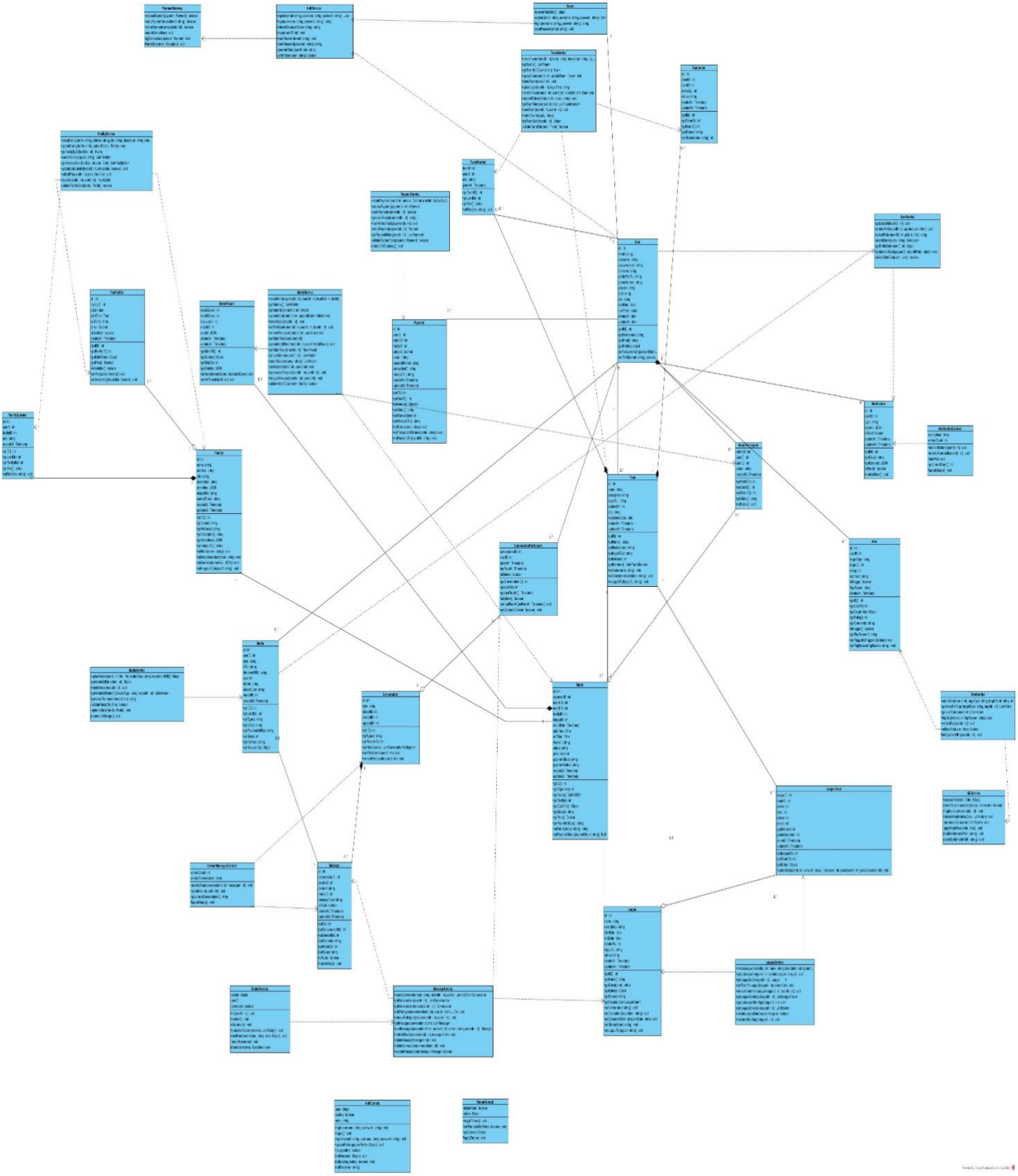
Use Case Identifier	**UC-15**
Use Case Name	Detect Malicious Voting with Machine Learning
Participating Actors	System
Flow of Events	1. A voting event occurs (e.g., post-match rating, MVP vote). 2. System receives the vote data. 3. **<<Include>>** System feeds the vote data (and potentially contextual data like user history, vote timing, IP address) into a pre-trained Machine Learning model. 4. The ML model analyzes the data for patterns indicative of malicious activity (e.g., vote brigading, bot voting, anomalous scores). 5. The ML model outputs a score or flag indicating the likelihood of the vote being malicious. 6. **Extension Point:** If malicious activity is detected above a threshold: 6a. System may automatically discard the vote. 6b. System may flag the vote/user for manual review by an administrator. 6c. System may temporarily suspend voting privileges for the user. 7. If the vote is deemed legitimate, System processes it normally (e.g., includes it in average ratings). 8. System potentially logs the analysis result for monitoring and model retraining.
Entry Condition	A user submits a vote or rating within the system.
Exit Conditions	- **Success:** Vote is analyzed; malicious votes are flagged or handled according to defined rules, legitimate votes are processed. - **Failure:** ML model fails to process the vote, error occurs.
Related NFRs	NR-1 (Security - integrity of ratings), NR-5 (Reliability), NR-9 (Technology - backend/ML integration)

5 System models

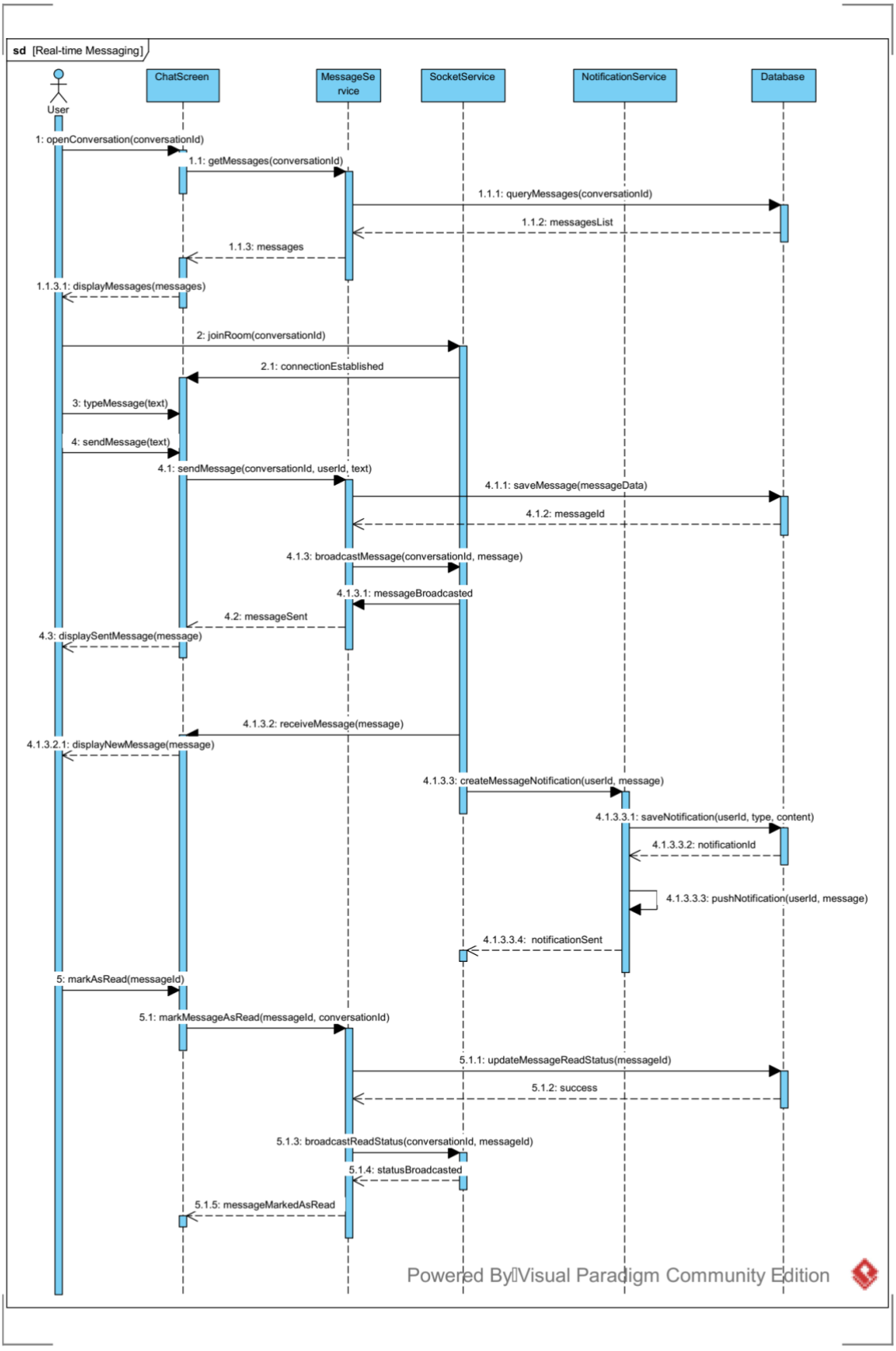
5.1 Use Case Diagrams

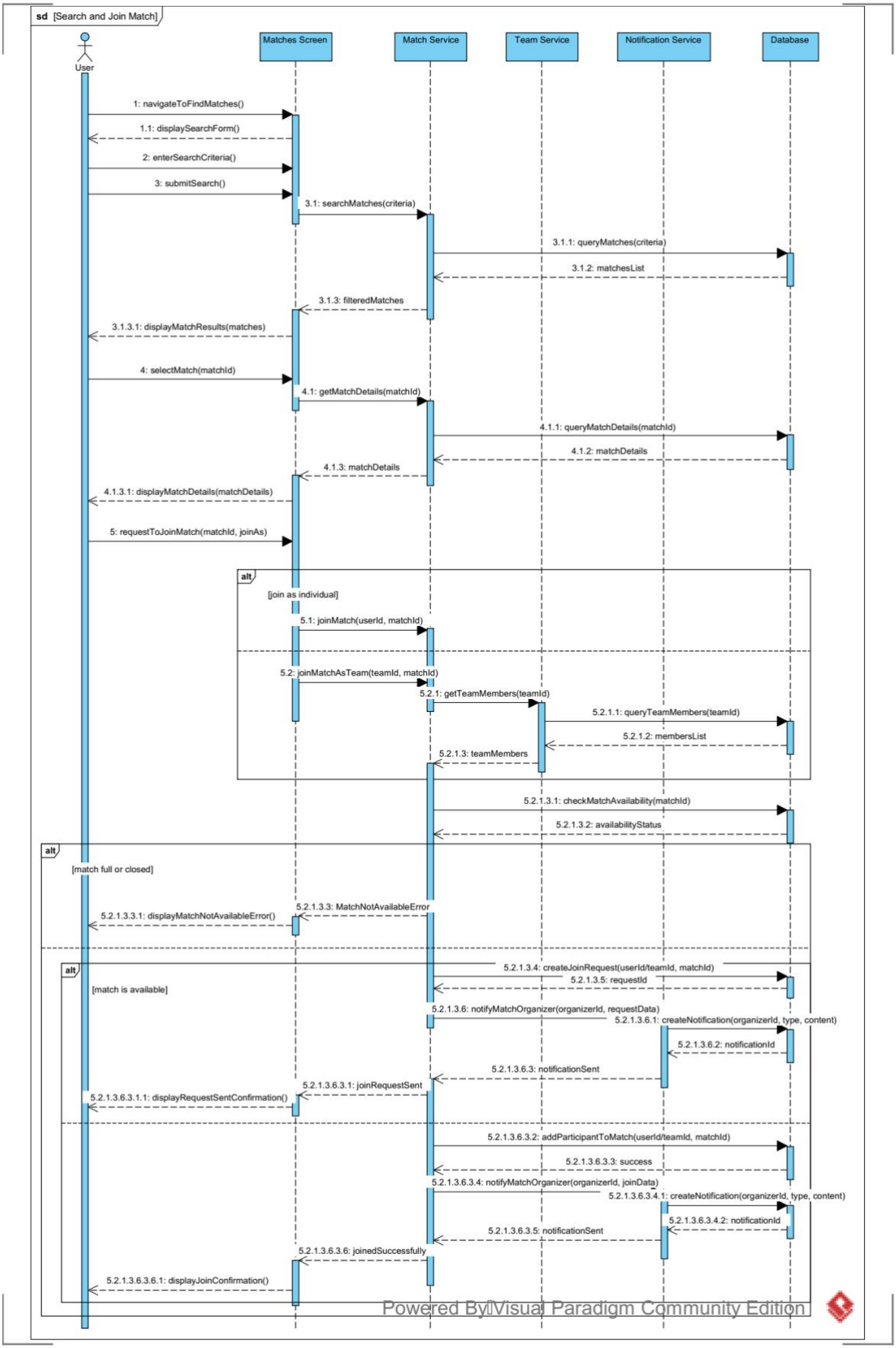


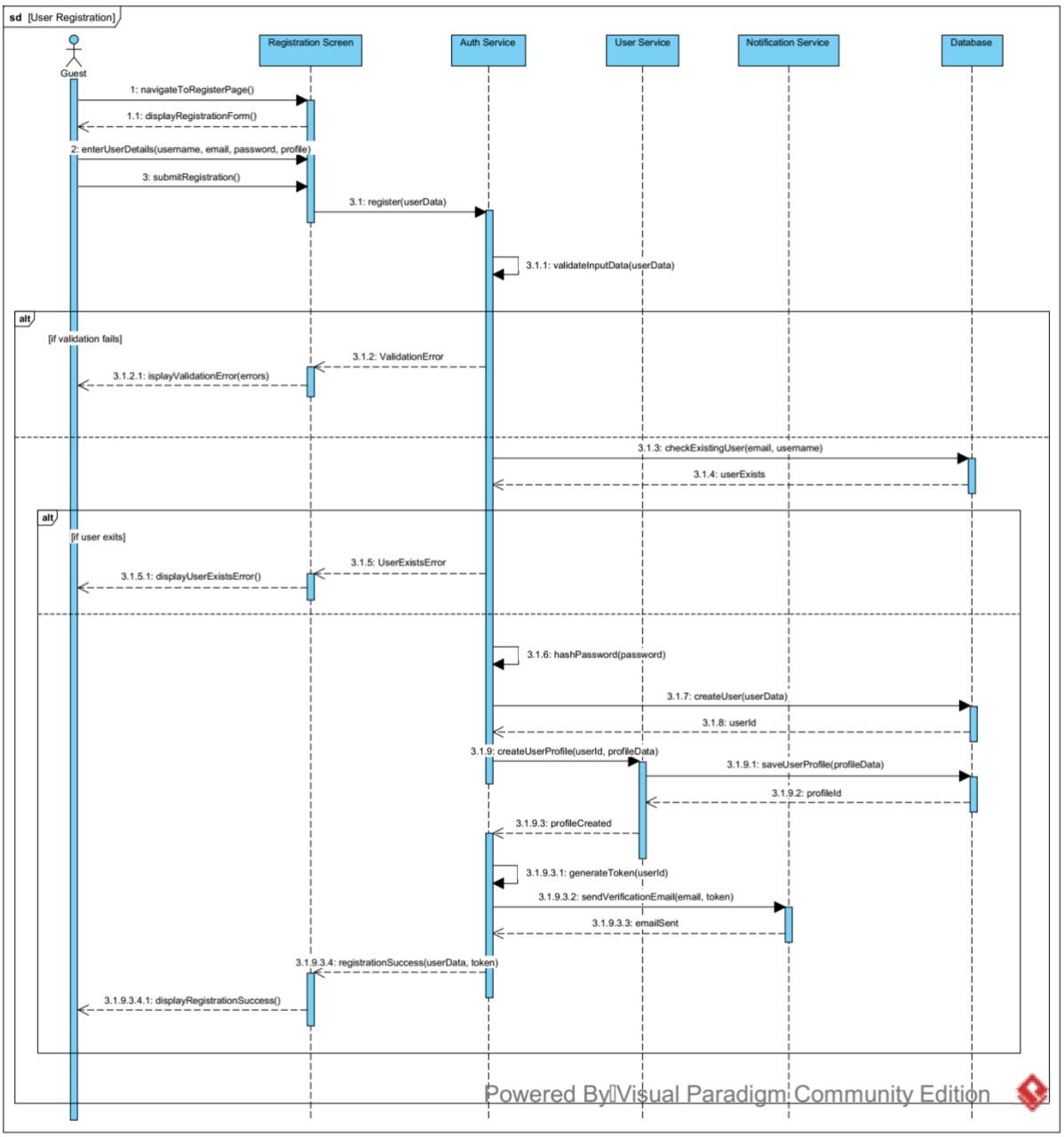
5.2 Class Diagram

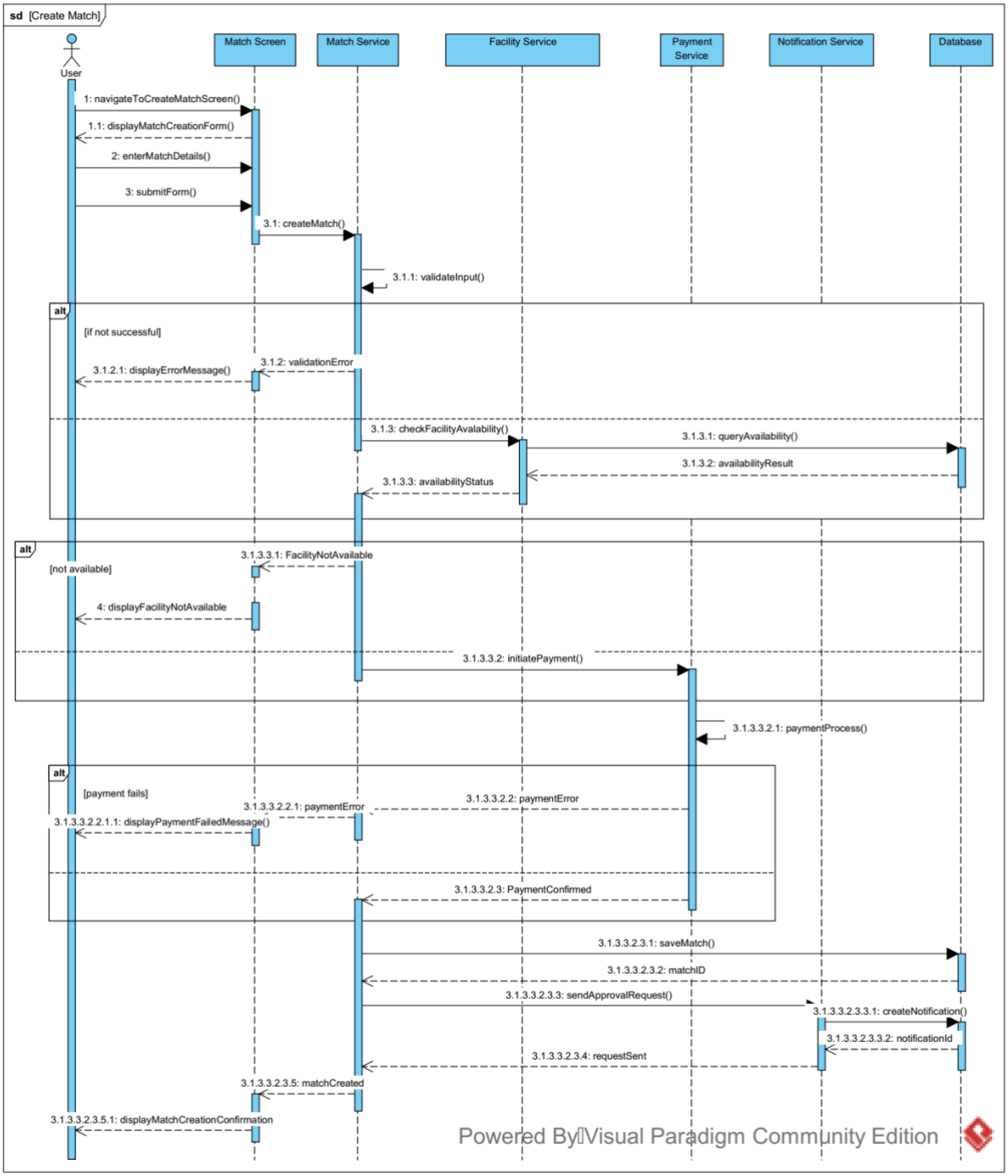


5.3 Sequence Diagrams

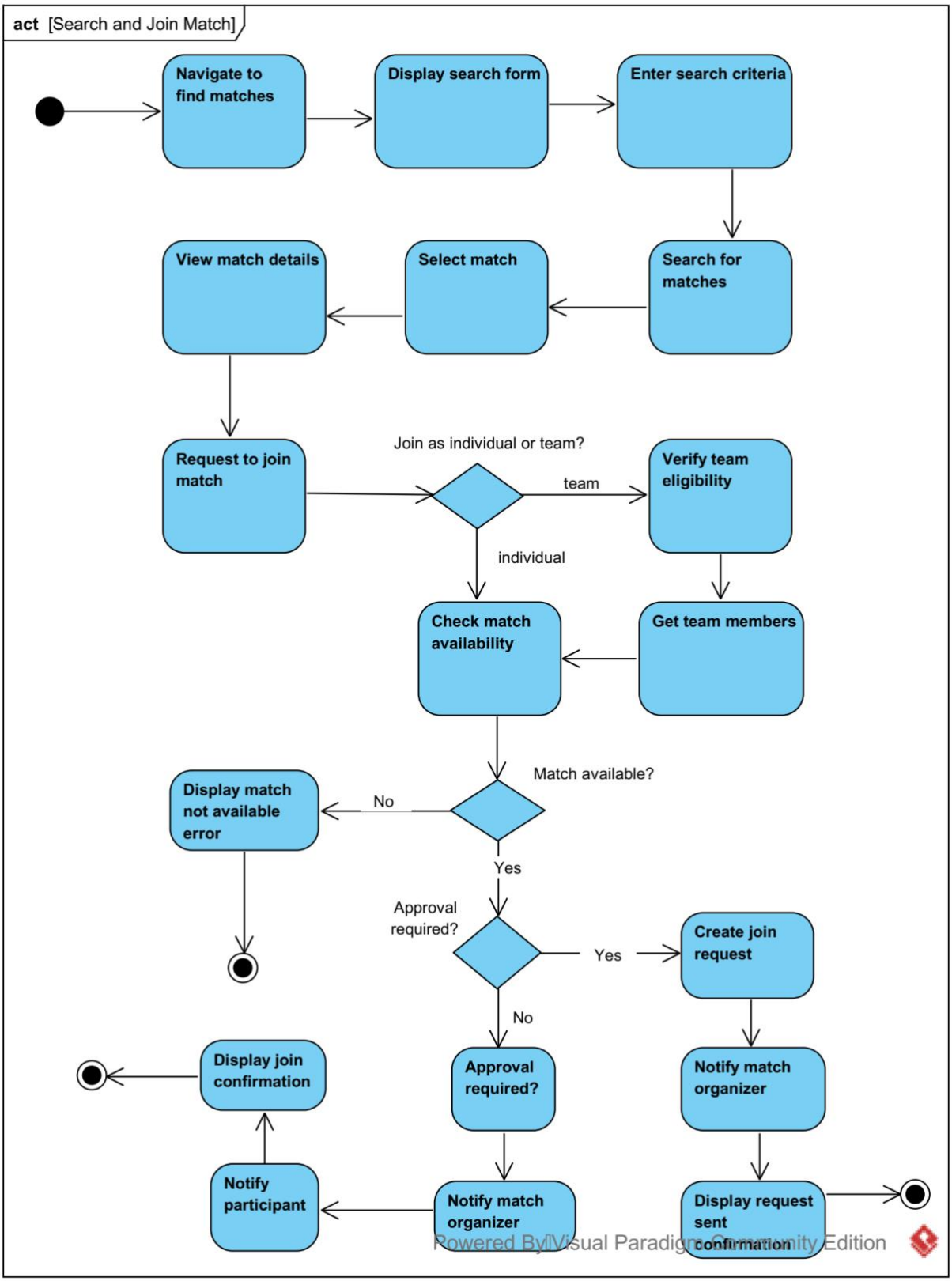


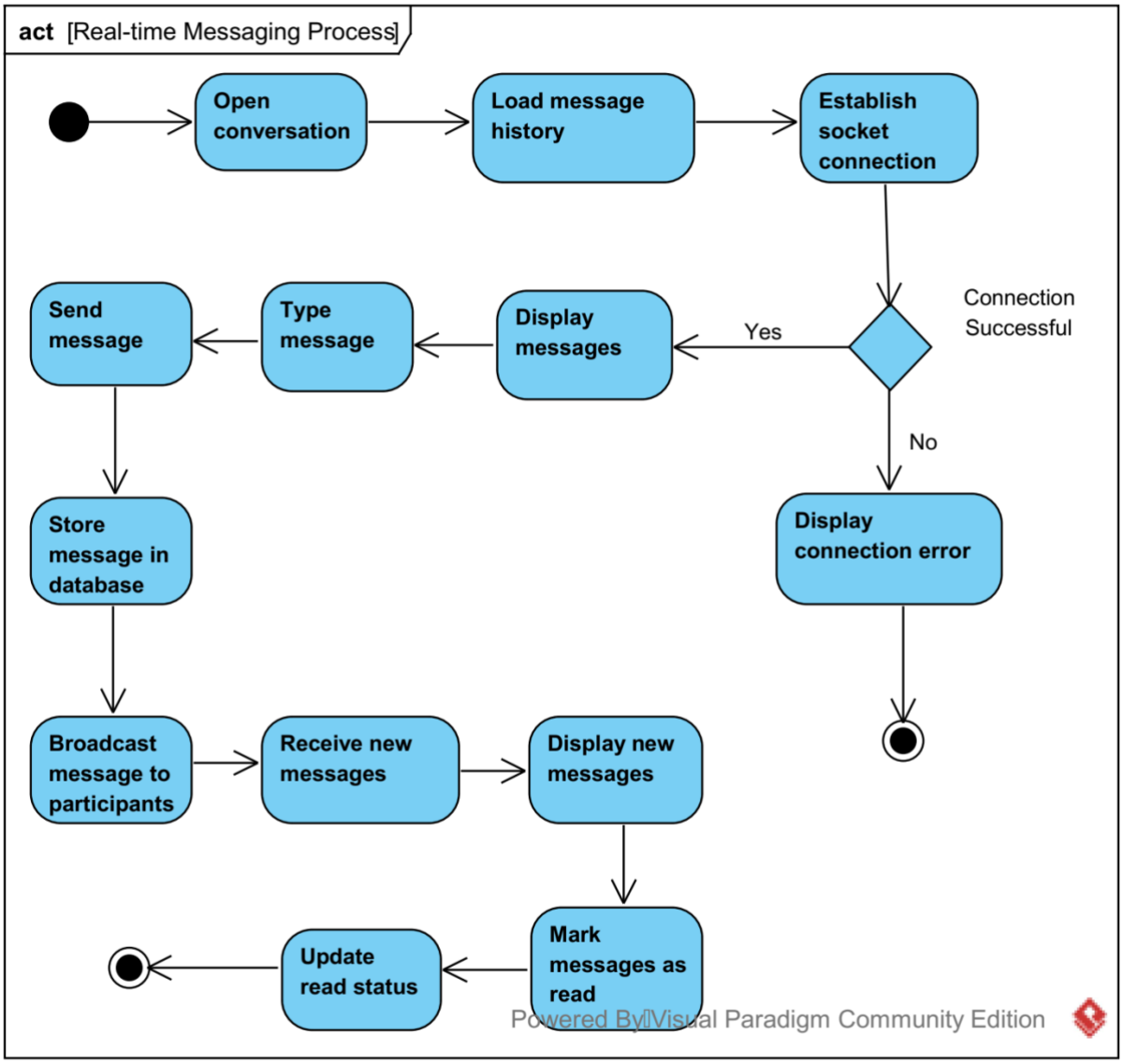

sd [Search and Join Match]
User
Matches Screen
Match Service
Team Service
Notification Service
Database
1: navigateToFindMatches()
1.1: displaySearchForm()
2: enterSearchCriteria()
3: submitSearch()
3.1: searchMatches(criteria)
3.1.1: queryMatches(criteria)
3.1.2: matchesList
3.1.3: filteredMatches
3.1.3.1: displayMatchResults(matches)
4: selectMatch(matchId)
4.1: getMatchDetails(matchId)
4.1.1: queryMatchDetails(matchId)
4.1.2: matchDetails
4.1.3: matchDetails
4.1.3.1: displayMatchDetails(matchDetails)
5: requestToJoinMatch(matchId, joinAs)
alt
[join as individual]
5.1: joinMatch(userId, matchId)
5.2: joinMatchAsTeam(teamId, matchId)
5.2.1: getTeamMembers(teamId)
5.2.1.1: queryTeamMembers(teamId)
5.2.1.2: membersList
5.2.1.3: teamMembers
5.2.1.3.1: checkMatchAvailability(matchId)
5.2.1.3.2: availabilityStatus
alt
[match full or closed]
5.2.1.3.3: MatchNotAvailableError
5.2.1.3.3.1: displayMatchNotAvailableError()
alt
[match is available]
5.2.1.3.4: createJoinRequest(userId/teamId, matchId)
5.2.1.3.5: requestId
5.2.1.3.6: notifyMatchOrganizer(organizerId, requestData)
5.2.1.3.6.1: createNotification(organizerId, type, content)
5.2.1.3.6.2: notificationId
5.2.1.3.6.3: notificationSent
5.2.1.3.6.3.1: joinRequestSent
5.2.1.3.6.3.1.1: displayRequestSentConfirmation()
5.2.1.3.6.3.2: addParticipantToMatch(userId/teamId, matchId)
5.2.1.3.6.3.3: success
5.2.1.3.6.3.4: notifyMatchOrganizer(organizerId, joinData)
5.2.1.3.6.3.4.1: createNotification(organizerId, type, content)
5.2.1.3.6.3.4.2: notificationId
5.2.1.3.6.3.5: notificationSent
5.2.1.3.6.3.6: joinedSuccessfully
5.2.1.3.6.3.6.1: displayJoinConfirmation()
Powered By Visual Paradigm Community Edition

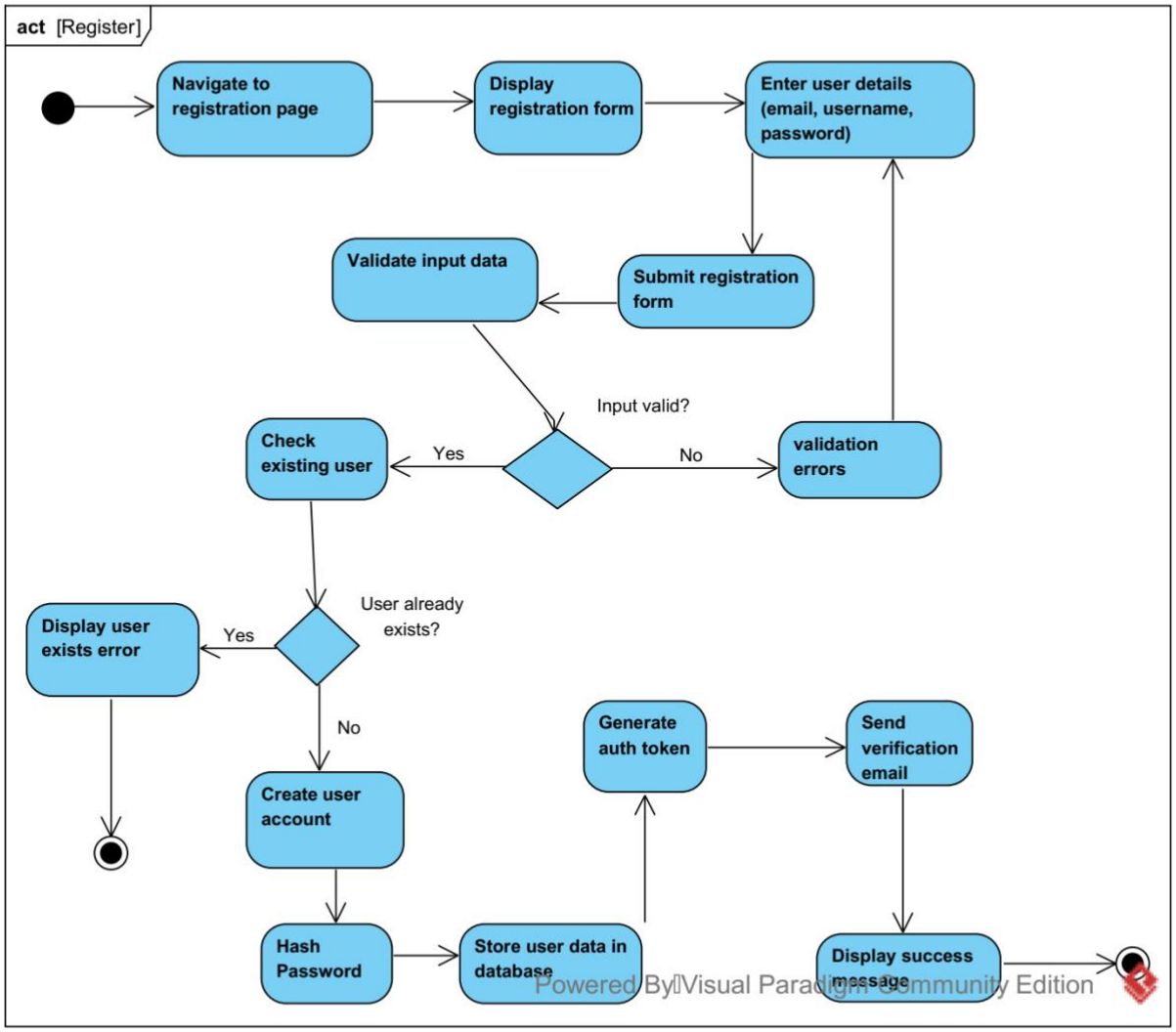

sd [User Registration]
Guest
Registration Screen
Auth Service
User Service
Notification Service
Database
1: navigateToRegisterPage()
1.1: displayRegistrationForm()
2: enterUserDetails(username, email, password, profile)
3: submitRegistration()
3.1: register(userData)
3.1.1: validateInputData(userData)
alt
[if validation fails]
3.1.2: ValidationError
3.1.2.1: isplayValidationError(errors)
3.1.3: checkExistingUser(email, username)
3.1.4: userExists
alt
[if user exits]
3.1.5: UserExistsError
3.1.5.1: displayUserExistsError()
3.1.6: hashPassword(password)
3.1.7: createUser(userData)
3.1.8: userId
3.1.9: createUserProfile(userId, profileData)
3.1.9.1: saveUserProfile(profileData)
3.1.9.2: profileId
3.1.9.3: profileCreated
3.1.9.3.1: generateToken(userId)
3.1.9.3.2: sendVerificationEmail(email, token)
3.1.9.3.3: emailSent
3.1.9.3.4: registrationSuccess(userData, token)
3.1.9.3.4.1: displayRegistrationSuccess()
Powered By Visual Paradigm Community Edition

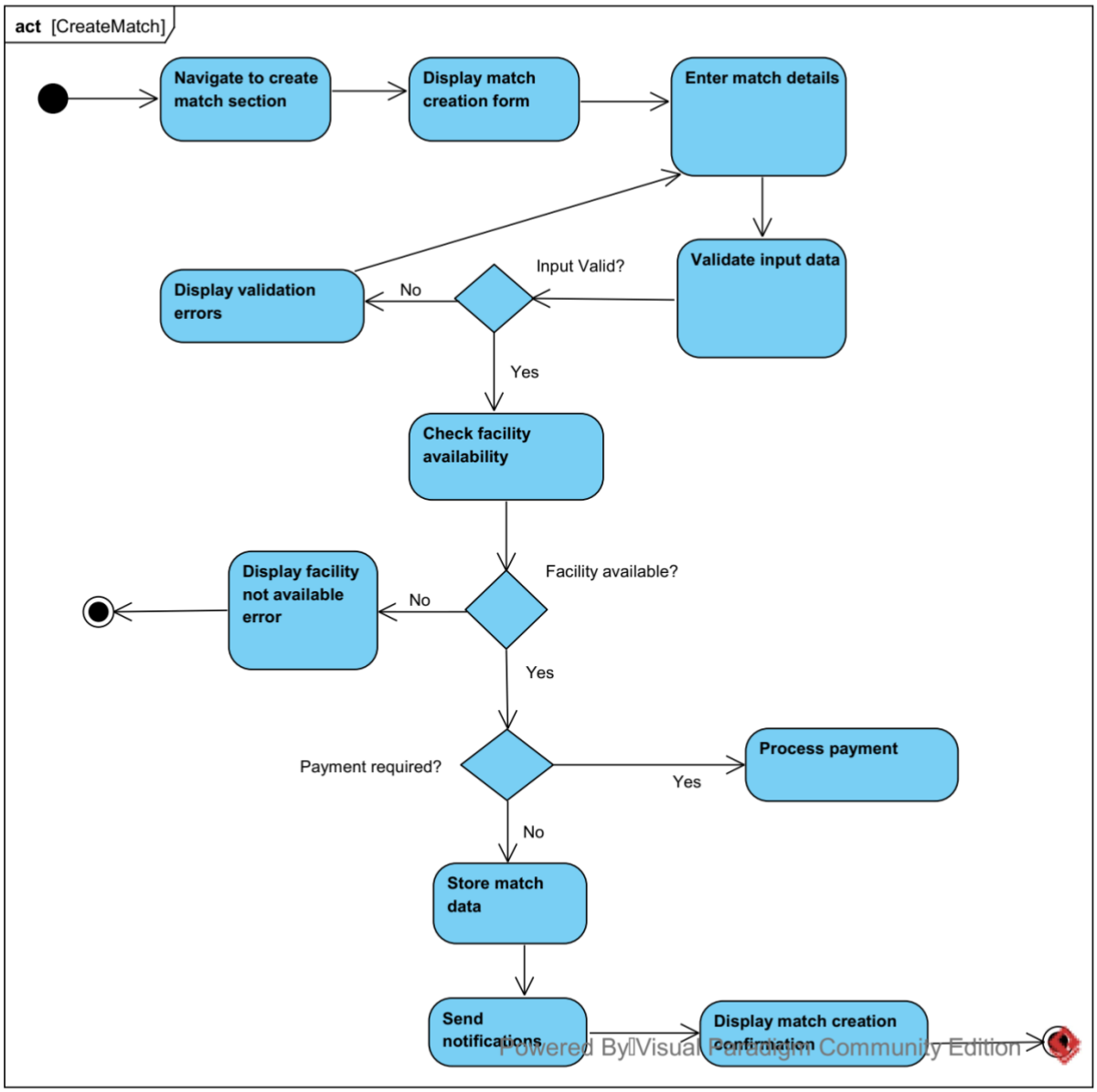

sd [Create Match]
User
Match Screen
Match Service
Facility Service
Payment Service
Notification Service
Database
1: navigateToCreateMatchScreen()
1.1: displayMatchCreationForm()
2: enterMatchDetails()
3: submitForm()
3.1: createMatch()
3.1.1: validateInput()
alt
[if not successful]
3.1.2.1: displayErrorMessage()
3.1.2: validationError
3.1.3: checkFacilityAvalability()
3.1.3.1: queryAvailability()
3.1.3.2: availabilityResult
3.1.3.3: availabilityStatus
alt
[not available]
3.1.3.3.1: FacilityNotAvailable
4: displayFacilityNotAvailable
3.1.3.3.2: initiatePayment()
3.1.3.3.2.1: paymentProcess()
alt
[payment fails]
3.1.3.3.2.2.1: paymentError
3.1.3.3.2.2: paymentError
3.1.3.3.2.1.1: displayPaymentFailedMessage()
3.1.3.3.2.3: PaymentConfirmed
3.1.3.3.2.3.1: saveMatch()
3.1.3.3.2.3.2: matchID
3.1.3.3.2.3.3: sendApprovalRequest()
3.1.3.3.2.3.3.1: createNotification()
3.1.3.3.2.3.3.2: notificationId
3.1.3.3.2.3.4: requestSent
3.1.3.3.2.3.5: matchCreated
3.1.3.3.2.3.5.1: displayMatchCreationConfirmation
Powered By Visual Paradigm Community Edition

6 Activity Diagrams

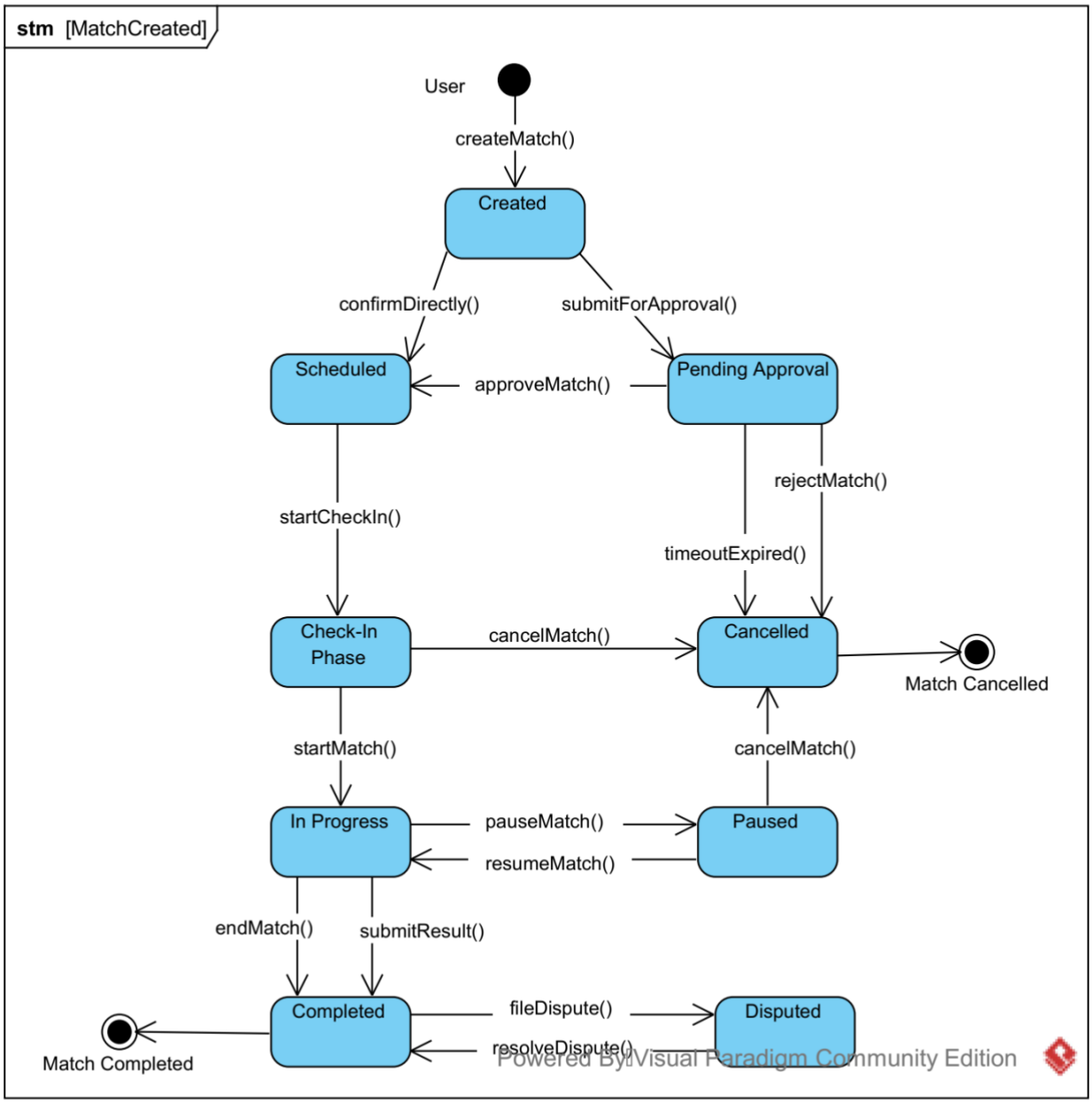


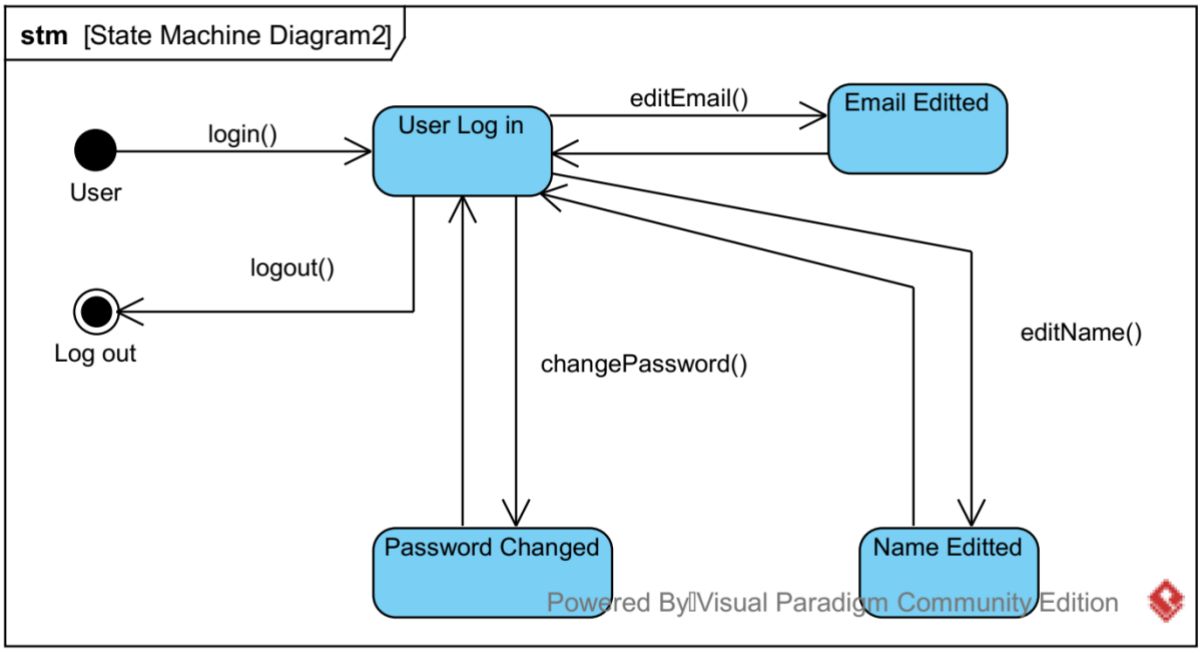

act [Real-time Messaging Process]
Open conversation
Load message history
Establish socket connection
Connection Successful
Yes
No
Display messages
Type message
Send message
Store message in database
Display connection error
Broadcast message to participants
Receive new messages
Display new messages
Mark messages as read
Update read status
Powered By Visual Paradigm Community Edition


act [Register]
Navigate to registration page
Display registration form
Enter user details (email, username, password)
Validate input data
Submit registration form
Input valid?
Check existing user
Yes
No
validation errors
User already exists?
Display user exists error
Yes
No
Create user account
Generate auth token
Send verification email
Hash Password
Store user data in database
Display success message
Powered By Visual Paradigm Community Edition


act [CreateMatch]
Navigate to create match section
Display match creation form
Enter match details
Validate input data
Input Valid?
No
Display validation errors
Yes
Check facility availability
Facility available?
No
Display facility not available error
Yes
Payment required?
Yes
Process payment
No
Store match data
Send notifications
Display match creation confirmation
Powered By Visual Paradigm Community Edition

7 Statechart Diagrams




stm [State Machine Diagram2]
User
login()
User Log in
editEmail()
Email Editted
logout()
Log out
changePassword()
editName()
Password Changed
Name Editted
Powered By Visual Paradigm Community Edition

8

9 User Interface Diagram


9:19
Ana Sayfa
Hoş Geldin
ASTRO
Hızlı Erişim
Maç Oluştur
Yeni bir maç organize et
Maç Bul
Mevcut maçlara katıl
Takımlarım
Takımlarını yönet
Maçlarım
Maçlarını takip et
Hakkımızda
Biz Kimiz?
Merhaba! Biz, futbol tutkusunu dijital dünyaya taşıyan yenilikçi bir ekibiz. ASTRO, futbolseverlerin bir araya gelip yeteneklerini sergileyebilecekleri, yeni insanlarla tanışabilecekleri ve spor keyfini sistemli bir şekilde yaşayabilecekleri bir
Ana Sayfa
Keşfet
Oluştur
Maçlarım
Takımlarım
Profil

10 Glossary

The glossary is a collection of terms pertaining to the ASTRO project, ensuring that readers, including technical and non-technical stakeholders, can understand the terminology used. The following terms were included, based on the project's focus and standard software engineering definitions:

- **ASTRO**: A mobile application designed as a football match reservation system and a social platform for amateur athletes. It aims to centralize match organization, team management, and communication.
- **Actors**: Individuals or systems that interact with the ASTRO application. In this project, actors include User, Facility Operator, Payment Gateway, Guest, and System.
- **Amateur Athletes**: The primary target users of the ASTRO application, who are looking for regular football activities.
- **Class Diagram**: A type of static structure diagram in software engineering that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- **Community Building**: Features within ASTRO that aim to connect players and grow local football networks.
- **Facility Operator (Venue Manager)**: Individuals or entities who own or manage sports facilities (e.g., futsal courts) and use ASTRO to update venue availability and pricing.
- **Functional Requirements (FR)**: Specific functions or behaviors that the ASTRO system must perform. Examples include user registration, team creation, and match searching.
- **GDPR (General Data Protection Regulation)**: A regulation in EU law on data protection and privacy for all individuals within the European Union and the European Economic Area. The ASTRO system aims for GDPR compliance for EU users.
- **Guest**: An unauthenticated user who can perform limited actions within the ASTRO app, such as browsing matches or viewing team profiles, without logging in.
- **Machine Learning (ML)**: Used in ASTRO to detect suspicious or malicious voting patterns for player ratings.
- **Matchmaking**: A feature of ASTRO that helps connect players or teams for matches, potentially based on skill level or location.
- **MinIO**: An open-source object storage server, mentioned in the document as the storage solution for media files like profile pictures and team logos.
- **Node.js/Express.js**: The specified backend technology for the ASTRO application.
- **Nonfunctional Requirements (NFR)**: Qualities or constraints of the ASTRO system, such as security, performance, scalability, and usability.

- **Payment Gateway**: An external service integrated with ASTRO to securely process payments for venue bookings.
- **PostgreSQL**: The specified relational database system for storing ASTRO application data.
- **Push Notifications**: Real-time alerts sent to users for match invitations, reminders, and updates.
- **React Native**: The specified frontend technology for developing the ASTRO mobile application.
- **Real-Time Communication/Messaging**: Instantaneous exchange of messages between users, facilitated by Socket.IO in the ASTRO application.
- **Scalability**: The ability of the ASTRO system to handle a growing number of users, with a target of supporting up to 10,000 concurrent users.
- **Sequence Diagram**: A type of interaction diagram in software engineering that shows how processes operate with one another and in what order.
- **Socket.IO**: A JavaScript library used for real-time web applications, specified for enabling real-time updates and messaging in ASTRO.
- **Software Requirements Specification (SRS) Document**: A document that describes the functional and nonfunctional requirements of a software system. The provided document is an SRS for ASTRO.
- **Statechart Diagram**: A type of diagram used in computer science and related fields to describe the behavior of systems. It depicts states and transitions between states.
- **System (Implicit Actor)**: An actor that manages automated processes within ASTRO, such as data validation, payment handling, and machine learning analysis.
- **Team Captain**: A user role in ASTRO responsible for organizing teams, creating matches, and inviting players.
- **Use Case**: A description of a specific way a user interacts with the ASTRO system to achieve a particular goal. Examples include "User Registration" and "Create Match."
- **Use Case Diagram**: A graphical depiction of the interactions among the elements of a system, showing the relationships between actors and use cases.
- **User**: An individual (amateur athlete, team captain, sports enthusiast) who uses the ASTRO mobile application.
- **User Interface (UI) Diagrams**: Visual representations of the screens and interactive elements of the ASTRO application.
- **Venue Booking**: A core feature of ASTRO allowing users to reserve sports facilities like futsal courts.

These definitions ensure that all stakeholders can understand the technical jargon and project-specific terms, enhancing the document's accessibility.

11 References

The references section lists sources that informed the design decisions for the ASTRO project, including official documentation for technologies used and standard software engineering texts. The following references were included, based on the project's technical stack and standard practices:

1. Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.
2. Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.
3. React Native Documentation. (2024). Retrieved from React Native Docs
4. Node.js Documentation. (2024). Retrieved from Node.js Docs
5. PostgreSQL Documentation. (2024). Retrieved from PostgreSQL Docs
6. Socket.IO Documentation. (2024). Retrieved from Socket.IO Docs
7. MinIO Documentation. (2024). Retrieved from MinIO Docs
8. International Organization for Standardization. (2011). *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*.
9. General Data Protection Regulation (GDPR). (2016). *Regulation (EU) 2016/679 of the European Parliament and of the Council*. Retrieved from GDPR Info
10. Ambler, S. W. (2004). *The Object Primer: Agile Model-Driven Development with UML 2.0* (3rd ed.). Cambridge University Press.
11. Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd ed.). Addison-Wesley Professional.
12. Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufmann.
13. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

These references ensure that the design document is grounded in established practices and provides a basis for further reading and validation.

12 Appendix

The appendix provides detailed technical information that supports the design of the ASTRO system, ensuring that all supplementary materials are accessible for stakeholders. The following subsections were included, based on standard practices and the project's requirements:

- **A.1 Technical Architecture Overview**

The ASTRO application employs a modern technology stack designed for scalability, performance, and cross-platform compatibility:

- **Frontend Architecture**:
    - Framework: React Native
    - Key Libraries: Navigation (React Navigation), State Management (Redux/Context API), UI Components (Native Base), Maps Integration (React Native Maps)
    - Client-side Storage: AsyncStorage for user preferences, Redux Persist for offline data access
- **Backend Architecture**:
    - Server Framework: Node.js with Express.js
    - Database: PostgreSQL, with schema optimized for relational data (users, teams, matches) and indexes on frequently queried fields
    - Real-time Communication: Socket.IO
    - Media Storage: MinIO (S3-compatible object storage)
    - Authentication: JWT-based with refresh token mechanism
    - Payment Processing: Integration with external payment gateway
- **System Deployment**:
    - API Hosting: Docker containers on AWS ECS
    - Database: AWS RDS for PostgreSQL
    - Media Storage: MinIO on dedicated storage servers
    - CI/CD Pipeline: GitHub Actions for automated testing and deployment
- **A.2 Data Dictionary**

The data dictionary defines the structure of key entities in the ASTRO system, ensuring clarity for database design and implementation:

Entity	Attribute	Data Type	Description	Constraints
User	user_id	UUID	Unique identifier for users	Primary Key
User	email	VARCHAR(255)	User's email address	Unique, Not Null
User	password_hash	VARCHAR(255)	Hashed user password	Not Null

Entity	Attribute	Data Type	Description	Constraints
User	username	VARCHAR(50)	User's display name	Unique, Not Null
User	profile_picture_url	VARCHAR(255)	URL to profile image	Nullable
User	skill_level	INT	Player's skill rating (1-10)	Default 5
Team	team_id	UUID	Unique identifier for teams	Primary Key
Team	name	VARCHAR(100)	Team name	Unique, Not Null
Team	description	TEXT	Team description	Nullable
Team	logo_url	VARCHAR(255)	URL to team logo	Nullable
Team	created_at	TIMESTAMP	Team creation date	Not Null, Default CURRENT_TIMESTAMP
Match	match_id	UUID	Unique identifier for matches	Primary Key
Match	venue_id	UUID	Reference to venue	Foreign Key, Not Null
Match	creator_id	UUID	User who created the match	Foreign Key, Not Null
Match	start_time	TIMESTAMP	Match start time	Not Null
Match	end_time	TIMESTAMP	Match end time	Not Null
Match	status	ENUM	Status of match (open, full, canceled, completed)	Not Null, Default 'open'
Venue	venue_id	UUID	Unique identifier for venues	Primary Key
Venue	name	VARCHAR(100)	Venue name	Not Null
Venue	address	TEXT	Physical address	Not Null

Entity Attribute	Data Type	Description	Constraints
Venue geo_location	POINT	GPS coordinates	Not Null
Venue price_per_hour	DECIMAL(10,2)	Hourly rental cost	Not Null

- **A.3 API Endpoints Documentation**

The ASTRO system exposes the following API endpoints, ensuring clear communication between the frontend and backend:

- **Authentication Endpoints**:
    - POST /api/auth/register - Register new user
    - POST /api/auth/login - User login
    - POST /api/auth/refresh - Refresh authentication token
    - POST /api/auth/password-reset - Request password reset
- **User Endpoints**:
    - GET /api/users/:id - Get user profile
    - PUT /api/users/:id - Update user profile
    - GET /api/users/:id/teams - Get teams for user
    - GET /api/users/:id/matches - Get matches for user
- **Team Endpoints**:
    - POST /api/teams - Create new team
    - GET /api/teams/:id - Get team details
    - PUT /api/teams/:id - Update team details
    - DELETE /api/teams/:id - Delete team
    - POST /api/teams/:id/members - Add member to team
    - DELETE /api/teams/:id/members/:userId - Remove member from team
- **Match Endpoints**:
    - POST /api/matches - Create new match
    - GET /api/matches - Search for matches
    - GET /api/matches/:id - Get match details
    - PUT /api/matches/:id - Update match details
    - DELETE /api/matches/:id - Cancel match
    - POST /api/matches/:id/join - Join a match

    - POST /api/matches/:id/leave - Leave a match
- **Venue Endpoints**:
    - GET /api/venues - Search for venues
    - GET /api/venues/:id - Get venue details
    - GET /api/venues/:id/availability - Check venue availability
    - POST /api/venues/:id/book - Book venue (Facility Operator only)
- **A.4 Security Considerations**

Security is paramount for the ASTRO system, with the following measures implemented:

- **Data Protection**: All transmitted data is encrypted using TLS 1.3, passwords are hashed with bcrypt, personal data complies with GDPR, and regular security audits are conducted.
- **Authentication & Authorization**: Role-based access control, JWT tokens with expiration, rate limiting on authentication endpoints, and IP-based anomaly detection for suspicious logins.
- **Payment Security**: PCI-DSS compliant processing, no storage of credit card details on ASTRO servers, and delegation to trusted third-party gateways.
- **A.5 Localization Strategy**

The ASTRO application supports multiple languages to cater to a global audience:

- String externalization using i18n, right-to-left (RTL) layout support, date/time formatting appropriate to locale, and currency display based on user's region.
- Initial languages supported: Turkish and English.
- **A.6 Testing Strategy**

A comprehensive testing strategy ensures the system's reliability and performance:

- **Unit Testing**: Jest for JavaScript/React Native components, with a minimum 80% code coverage target.
- **Integration Testing**: API endpoint testing with Supertest, database integration tests.
- **UI Testing**: React Native Testing Library for component tests, Detox for end-to-end mobile app testing.
- **Performance Testing**: Load testing with Artillery, response time benchmarking against a 200ms requirement.
- **User Acceptance Testing**: Beta testing program with selected amateur football teams, feedback collection, and issue tracking.