Module Interface Specification for ProgNameThe Crazy Tens

Team #25, The Crazy Four

Ruida Chen
Ammar Sharbat
Alvin Qian
Jiaming Li

November 12, 2025

1 Revision History

Date	**Version**	**Notes**
Nov 12th	Rev-1	Module M1-M11

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at SRS

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by ProgName.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of ProgName uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, ProgName uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**	**Level 3 (Leaf Modules)**
Hardware-Hiding Module		M19 (Server OS)
		M20 (Client Runtime)
		M21 (PostgreSQL)
Behaviour-Hiding Module	(Core Domain Logic)	M15
		M16
		M17
		M18
Software Decision Module	Backend (Server)	M1
		M2
		M3
		M4
		M5
		M6
	Frontend (Client)	M7
		M8
		M9
		M10
		M11
		M12
		M13
		M14

Table 1: Module Hierarchy

6 MIS of API Module (M1)

6.1 Module

API

6.2 Uses

- Matchmaking Module M3
- Authentication Module M4
- Repository Module M5

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
POST /api/auth/signup	UserCredentials (JSON)	User (JSON)	AuthError, ValidationError
POST /api/auth/login	UserCredentials (JSON)	AuthToken (JSON)	AuthError, ValidationError
POST /api/game	AuthToken, GameOptions (JSON)	GameSession (JSON)	AuthError
GET /api/profile	AuthToken	UserProfile (JSON)	AuthError, NotFound
PUT /api/profile	AuthToken, UserProfile (JSON)	UserProfile (JSON)	AuthError, ValidationError, NotFound
DELETE /api/profile	AuthToken	SuccessMessage (JSON)	AuthError, NotFound

6.4 Semantics

6.4.1 State Variables

None. This module is stateless.

6.4.2 Environment Variables

- **HTTPRequest**: Represents the incoming HTTP request, containing headers (e.g., Authorization), body (JSON payload), and method (POST, GET, etc.).
- **HTTPResponse**: Represents the outgoing HTTP response, to which the module writes the JSON body and sets HTTP status codes.

6.4.3 Assumptions

- A web server (e.g., Node.js with Express) is running and routing HTTP requests to this module's access programs.
- The modules `Uses` (M3, M4, M5) are available and correctly implemented.
- Incoming `AuthToken` (if required) is expected to be a JWT, verifiable by `M4`.

6.4.4 Access Routine Semantics

POST /api/auth/signup(*UserCredentials*)

- transition: Validates *UserCredentials*. Calls `M4.registerUser(username, password)`. On success, calls `M5.createUser(data)`.
- output: Returns a JSON object of the newly created user.
- exception: `ValidationError` (400) if credentials format is invalid. `AuthError` (e.g., 409 Conflict) if user already exists.

POST /api/auth/login(*UserCredentials*)

- transition: Validates *UserCredentials*. Calls `M4.loginUser(username, password)` to verify credentials and generate a token.
- output: Returns a JSON object containing the `AuthToken` (JWT).
- exception: `ValidationError` (400) if credentials format is invalid. `AuthError` (401/403) if authentication fails.

POST /api/game(*AuthToken, GameOptions*)

- transition: Calls `M4.verifyToken(AuthToken)` to get a UserID. On success, calls `M3.createLobby(UserID)` or a similar game creation service.
- output: Returns a JSON object with the `GameSession` details (e.g., LobbyID).
- exception: `AuthError` (401/403) if *AuthToken* is invalid or missing.

GET /api/profile(*AuthToken*)

- transition: Calls `M4.verifyToken(AuthToken)` to get a UserID. On success, calls `M5.getUserProfile(UserID)`.
- output: Returns the `UserProfile` as a JSON object.
- exception: `AuthError` (401/403) if *AuthToken* is invalid. `NotFound` (404) if the user profile does not exist.

PUT /api/profile(*AuthToken, UserProfile*)

- transition: Calls `M4.verifyToken(AuthToken)` to get a UserID. Validates the *UserProfile* data. On success, calls `M5.updateUserProfile(UserID, data)`.
- output: Returns the updated `UserProfile` as a JSON object.
- exception: `AuthError` (401/403). `ValidationError` (400) if profile data is invalid. `NotFound` (404) if user does not exist.

DELETE /api/profile(*AuthToken*)

- transition: Calls `M4.verifyToken(AuthToken)` to get a UserID. On success, calls `M5.deleteUser(UserID)`.
- output: Returns a JSON `SuccessMessage` (e.g., {"status": "deleted"}).
- exception: `AuthError` (401/403). `NotFound` (404) if user does not exist.

6.4.5 Local Functions

None.

6.4.6 Considerations

- This module acts as the primary "firewall" for the backend, enforcing authentication (via M4) before delegating tasks to other modules (M3, M5).
- The stateless nature allows for horizontal scaling (e.g., running multiple instances of the server).
- The secret of this module is the definition of the REST API routes and the JSON data structures (schemas). If the API paths (e.g., `/api/profile`) or the JSON formats change, only this module, M9, and M14 (the clients) should be affected.

7 MIS of Real-time Gateway Module (M2)

7.1 Module

Real-time Gateway

7.2 Uses

- Authentication Module M4
- Game Engine Module M15
- Rules Module M16

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name (Event In)	In	Name (Event Out)	Exceptions
on('connection')	socket	-	SessionError
on('joinGame')	data: { authToken, lobbyID }	emit('gameStateUpdate')	SessionError, NotFound
on('submitMove')	data: { move }	emit('gameStateUpdate')	InvalidMove, NotYourTurn, SessionError

7.4 Semantics

7.4.1 State Variables

- **activeGames**: Map¡GameID, GameSession¿ — A map holding the live `GameSession` objects for all currently active games, keyed by their `GameID`. A `GameSession` includes the current `GameState` and the list of connected sockets (players).

7.4.2 Environment Variables

- **webSocketServer**: The server instance (e.g., Socket.io server) that manages all active client connections, message broadcasting, and room management.
- **clientSocket**: A single, stateful WebSocket connection representing one client.

7.4.3 Assumptions

- The client (M7) connects using the correct WebSocket protocol and endpoint.
- The client (M7) sends valid data structures (JSON) for the `joinGame` and `submitMove` events.
- The `authToken` provided in `joinGame` is a valid JWT, verifiable by `M4`.

7.4.4 Access Routine Semantics

on('connection')(*socket*)

- transition: A new *socket* is registered with the **webSocketServer**. The module attaches listeners (for `joinGame`, `submitMove`, `disconnect`) to this *socket*.
- output: None directly. The server is now ready to receive further events from this client.
- exception: `SessionError` if the connection handshake fails.

on('joinGame')(*data*)

- transition: 1. Calls `M4.verifyToken(data.authToken)` to get a `UserID`. 2. Retrieves the `GameSession` from **activeGames** using a key derived from `data.lobbyID`. 3. Adds the current *socket* to the "room" for that `GameSession`.
- output: Emits `emit('gameStateUpdate', state)` to the joining *socket*, sending the current `GameState` for that session.
- exception: `SessionError` if `authToken` is invalid. `NotFound` if the `GameSession` (lobby) does not exist in **activeGames**.

on('submitMove')(*data*)

- transition: 1. Identifies the `UserID` and `GameID` associated with the *socket* (established during `joinGame`). 2. Retrieves the correct `GameSession` from **activeGames**. 3. Calls `M16.isLegalMove(data.move, gameState)` to validate the move. 4. If legal, calls `M15.applyMove(gameState, data.move)` to get the new `GameState`. 5. Updates the `GameSession` in **activeGames** with the new `GameState`.
- output: Emits `emit('gameStateUpdate', newState)` to **all** sockets in the game's room, broadcasting the updated state.
- exception: `InvalidMove` if M16 returns false. `NotYourTurn` if the `UserID` does not match the `currentTurn` in the `GameState`. `SessionError` if the socket is not authenticated or not in a game.

7.4.5 Local Functions

None.

7.4.6 Considerations

- This module is stateful and server-authoritative. The client (M7) never modifies its own state; it only receives new state from this module via `gameStateUpdate`.
- The secret of this module is the management of stateful connections, serialization of game events, and the "room" logic that maps sockets to specific `GameSession`s.

8 MIS of Matchmaking Module (M3)

8.1 Module

Matchmaking

8.2 Uses

- Game Engine Module M15
- Real-time Gateway Module M2

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
createLobby	userID: UserID	LobbyID	LobbyError
joinLobby	lobbyID: LobbyID, userID: UserID	void	LobbyFull, LobbyNotFound
startMatch	lobbyID: LobbyID, hostID: UserID	GameID	LobbyNotFound, NotLobbyHost, GameCreationError

8.4 Semantics

8.4.1 State Variables

- **lobbies**: Map¡LobbyID, Lobby¿ — Holds all active, waiting-for-players `Lobby` objects. A `Lobby` object contains at least { `hostID: UserID, players: UserID[], status: string` }.

8.4.2 Environment Variables

None.

8.4.3 Assumptions

- Any `UserID` or `hostID` passed to this module has been authenticated by an upstream module (e.g., M1 or M4).
- Modules M15 and M2 are available and ready when `startMatch` is invoked.

8.4.4 Access Routine Semantics

createLobby(*userID*)

- transition: Generates a unique `LobbyID`. Creates a new `Lobby` object (e.g., { `hostID: userID, players: [userID], status: 'waiting'` }). Adds this new object to the **lobbies** map.
- output: Returns the newly created `LobbyID`.
- exception: `LobbyError` if a new lobby cannot be created (e.g., system limits reached).

joinLobby(*lobbyID, userID*)

- transition: Looks up the `Lobby` in **lobbies** using *lobbyID*. Verifies `Lobby.status == 'waiting'` and `Lobby.players.length < MAX_PLAYERS`. If valid, appends *userID* to the `Lobby.players` array.
- output: `void`.
- exception: `LobbyFull` if the lobby is at capacity. `LobbyNotFound` if *lobbyID* does not exist in **lobbies** or its status is not 'waiting'.

startMatch(*lobbyID, hostID*)

- transition: 1. Looks up the `Lobby` in **lobbies** using *lobbyID*. 2. Verifies that *hostID* matches `Lobby.hostID`. 3. Calls `M15.createGame(lobby.players, gameOptions)` to receive a new `GameState`. 4. Calls `M2.registerGameSession(lobbyID, newGameState)` (or a similar access program) to make the game live. 5. Removes the `Lobby` from the **lobbies** map (or updates `Lobby.status` to 'ingame').
- output: Returns the `GameID` for the newly created match (which may be the `LobbyID`).
- exception: `LobbyNotFound`. `NotLobbyHost` if *hostID* is not the host. `GameCreationError` if M15 or M2 report an error during game creation.

8.4.5 Local Functions

None.

8.4.6 Considerations

- The secret of this module is the data structure of a `Lobby` and the logic for managing the **lobbies** map.
- This module bridges the stateless API (M1) and the stateful game session (M2) by handling the pre-game lobby state.

9 MIS of Authentication Module (M4)

9.1 Module

Authentication

9.2 Uses

- Repository Module M5

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
registerUser	username: string, password: string	User	UserExists, ValidationError
loginUser	username: string, password: string	AuthToken	InvalidCredentials, ValidationError
verifyToken	token: AuthToken	UserID	TokenExpired, InvalidCredentials
manageGuestSession	-	AuthToken	SessionError

9.4 Semantics

9.4.1 State Variables

None. This module is stateless.

9.4.2 Environment Variables

- **CryptoLibrary**: An instance of the password hashing library (e.g., bcrypt).
- **JWT_SECRET_KEY**: The secret key used for signing and verifying JSON Web Tokens (AuthToken), read from a secure environment.

9.4.3 Assumptions

- The **CryptoLibrary** is properly configured.
- The **JWT_SECRET_KEY** is securely provided to the environment.
- Module M5 is available for database operations.

9.4.4 Access Routine Semantics

registerUser(*username, password*)

- transition: Validates *username* and *password* formats. Calls `M5.findUserByUsername(username)` to check for existence. Hashes and salts the *password* using **CryptoLibrary**. Calls `M5.createUser(username, hashedPassword)`.
- output: Returns the newly created `User` object.
- exception: `UserExists` if the username is already taken. `ValidationError` if inputs are malformed.

loginUser(*username, password*)

- transition: Calls `M5.findUserByUsername(username)` to retrieve the stored user hash. Compares the plaintext *password* with the stored hash using **CryptoLibrary**. If they match, generates a new `AuthToken` (JWT) signed with **JWT_SECRET_KEY** containing the `UserID`.
- output: Returns the newly generated `AuthToken`.
- exception: `InvalidCredentials` if the user is not found or the password does not match. `ValidationError` if inputs are malformed.

verifyToken(*token*)

- transition: Validates the *token*'s signature and expiration using **JWT_SECRET_KEY**. If valid, parses the `UserID` from the token payload.
- output: Returns the `UserID` extracted from the token.
- exception: `TokenExpired` if the token is past its expiry date. `InvalidCredentials` if the token signature is invalid or the token is malformed.

manageGuestSession( )

- transition: Generates a temporary `AuthToken` (JWT) with a special ”guest” `UserID` or a temporary unique identifier.
- output: Returns the `AuthToken` for the guest session.
- exception: `SessionError` if token generation fails.

9.4.5 Local Functions

None.

9.4.6 Considerations

- The secret of this module is the password hashing algorithm (bcrypt), salt generation, JWT structure, and the **JWT_SECRET_KEY**.
- M1 relies on this module for handling user authentication endpoints.
- M2 relies on `verifyToken` to authenticate WebSocket connections.

10 MIS of Repository Module (M5)

10.1 Module

Repository

10.2 Uses

- Database Module M21 (PostgreSQL)

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
findUserByUsername	username: string	User	RecordNotFound, DatabaseConnectionError
createUser	data: UserData	User	UniqueConstraintViolation, DatabaseConnectionError
saveGameResult	result: GameResult	void	DatabaseConnectionError
getUserProfile	userID: UserID	UserProfile	RecordNotFound, DatabaseConnectionError
updateUserProfile	userID: UserID, data: UserProfile	UserProfile	RecordNotFound, DatabaseConnectionError
deleteUser	userID: UserID	void	RecordNotFound, DatabaseConnectionError

10.4 Semantics

10.4.1 State Variables

- **dbConnectionPool**: A connection pool managing active connections to the M21 database.

10.4.2 Environment Variables

- **DatabaseInstance (M21)**: The instance of the PostgreSQL database software (M21) on which this module executes queries.

10.4.3 Assumptions

- The **DatabaseInstance** is running and accessible.
- A database connection string is securely provided to the environment.
- The database schema (tables, columns, relations) has been initialized and matches the queries defined within this module.

10.4.4 Access Routine Semantics

findUserByUsername(*username*)

- transition: Acquires a connection from **dbConnectionPool**. Executes a SQL `SELECT` query to find the user by *username*.
- output: Returns the `User` object if found.
- exception: `RecordNotFound` if no user with *username* is found. `DatabaseConnectionError` if the query fails.

createUser(*data*)

- transition: Acquires a connection from **dbConnectionPool**. Executes a SQL `INSERT` query to create a new user with *data*.
- output: Returns the newly created `User` object (e.g., with database-generated ID).
- exception: `UniqueConstraintViolation` if the username already exists. `DatabaseConnectionError` if the query fails.

saveGameResult(*result*)

- transition: Acquires a connection from **dbConnectionPool**. Executes a SQL `INSERT` query to store the *result* in the game history table.
- output: `void`.
- exception: `DatabaseConnectionError` if the query fails.

getUserProfile(*userID*)

- transition: Acquires a connection from **dbConnectionPool**. Executes a SQL `SELECT` query to retrieve the user's profile based on *userID*.

- output: Returns the `UserProfile` object.
- exception: `RecordNotFound` if *userID* is not found. `DatabaseConnectionError` if the query fails.

updateUserProfile(*userID, data*)

- transition: Acquires a connection from **dbConnectionPool**. Executes a SQL `UPDATE` query to modify the user's profile matching *userID* with new *data*.
- output: Returns the updated `UserProfile` object.
- exception: `RecordNotFound` if *userID* is not found. `DatabaseConnectionError` if the query fails.

deleteUser(*userID*)

- transition: Acquires a connection from **dbConnectionPool**. Executes a SQL `DELETE` query to remove the user matching *userID*.
- output: `void`.
- exception: `RecordNotFound` if *userID* is not found. `DatabaseConnectionError` if the query fails.

10.4.5 Local Functions

None.

10.4.6 Considerations

- The secret of this module is the database schema, all SQL queries, and connection pooling.
- Other modules (M1, M4) are completely unaware of SQL. They call abstract functions like `getUserProfile`.
- If the database is migrated from PostgreSQL (M21) to another system, only M5 needs to be rewritten; all other modules remain unchanged.

11 MIS of Audit Module (M6)

11.1 Module

Audit

11.2 Uses

- Operating System Module M19 (Server OS)

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
log.info	message: string	void	LogWriteError
log.warn	message: string	void	LogWriteError
log.error	message: string	void	LogWriteError

11.4 Semantics

11.4.1 State Variables

- **loggerInstance**: An instance of the configured logging library (e.g., Winston).

11.4.2 Environment Variables

- **LogStorage**: The destination for log output, typically a file on the M19 filesystem.

11.4.3 Assumptions

- The **loggerInstance** is successfully initialized when the module is loaded.
- The **LogStorage** (filesystem) provided by M19 is writable.

11.4.4 Access Routine Semantics

log.info(*message*)

- transition: Uses the **loggerInstance** to format the *message* as an 'info' level entry (adhering to the hidden log format) and write it to **LogStorage**.

- output: `void`.
- exception: `LogWriteError` if writing to **LogStorage** fails.

log.warn(*message*)

- transition: Uses the **loggerInstance** to format the *message* as a 'warn' level entry and write it to **LogStorage**.
- output: `void`.
- exception: `LogWriteError` if writing to **LogStorage** fails.

log.error(*message*)

- transition: Uses the **loggerInstance** to format the *message* as an 'error' level entry and write it to **LogStorage**.
- output: `void`.
- exception: `LogWriteError` if writing to **LogStorage** fails.

11.4.5 Local Functions

None.

11.4.6 Considerations

- The secret of this module is the log format, the storage location (e.g., file path), and the log retention policy.
- This module is used by other backend modules (M1, M2, M4, M5) to log important system events for debugging and security auditing.

12 MIS of Real-time Client Module (M7)

12.1 Module

Real-time Client

12.2 Uses

- Real-time Gateway Module M2

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
connect	-	void	ConnectionFailed
disconnect	-	void	
on	eventName: 'gameStateUpdate', callback: (state) =¿ void	void	
emit	eventName: 'submitMove', move: Move	void	ConnectionFailed

12.4 Semantics

12.4.1 State Variables

- **socket**: Socket — The `Socket.io-client` instance.
- **isConnected**: bool — Flag indicating the connection status.

12.4.2 Environment Variables

- **BrowserRuntime (M20)**: The client's web browser environment providing WebSocket APIs.

12.4.3 Assumptions

- The M2 server is running and its URL is accessible to the client.
- The browser environment (M20) supports WebSockets.

12.4.4 Access Routine Semantics

connect( )

- transition: Initializes and establishes the WebSocket connection to M2. Sets **socket** to the new instance and **isConnected** to `true` on success.
- output: `void`.
- exception: `ConnectionFailed` if the connection times out or is rejected.

disconnect( )

- transition: Closes the active WebSocket connection. Sets **isConnected** to `false` and **socket** to `null`.
- output: `void`.

on(*eventName, callback*)

- transition: Registers an event listener on the **socket** instance. When M2 emits an event matching *eventName* (e.g., 'gameStateUpdate'), the *callback* is invoked with the data payload.
- output: `void`.

emit(*eventName, move*)

- transition: Serializes and sends the *move* data to the M2 server over the **socket** connection, under the *eventName* (e.g., 'submitMove').
- output: `void`.
- exception: `ConnectionFailed` if **isConnected** is `false`.

12.4.5 Local Functions

None.

12.4.6 Considerations

- The secret of this module is the WebSocket connection state and reconnection logic.
- It is the client-side counterpart to M2.
- UI modules (e.g., M11, M12) use this module to receive state updates and send user actions.

13 MIS of Application Shell Module (M8)

13.1 Module

Application Shell

13.2 Uses

- Browser Runtime Module M20 (Client Runtime)
- Authentication Client Module M9
- Lobby View Module M10
- Game Board View Module M11
- Profile View Module M14

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
Render	props: ReactProps	JSX.Element	RouteNotFound

13.4 Semantics

13.4.1 State Variables

- **currentUser**: User — null — Stores the state of the currently logged-in user.
- **currentRoute**: string — The active route from the browser's URL.

13.4.2 Environment Variables

- **BrowserRuntime (M20)**: The browser environment providing the DOM for rendering and the URL History API for routing.

13.4.3 Assumptions

- The React library is loaded in the M20 environment.
- The browser supports the History API.
- Modules M9, M10, M11, and M14 are available to be rendered as children.

13.4.4 Access Routine Semantics

Render(*props*)

- transition: Reads the URL path from the **BrowserRuntime (M20)** to update **currentRoute**. Reads the authentication status to update **currentUser**. Renders the global layout (header, footer). Selectively renders a child module (M9, M10, M11, or M14) based on **currentRoute** and **currentUser**.
- output: Returns a React Element (`JSX.Element`) for the **BrowserRuntime (M20)** to render to the DOM.
- exception: `RouteNotFound` if **currentRoute** does not match any entry in the application's routing table.

13.4.5 Local Functions

None.

13.4.6 Considerations

- The secret of this module is the application routing table and the global layout structure.
- This module acts as a controller view, deciding which page (M10, M11, M14) to display based on URL and authentication state.

14 MIS of Authentication Client Module (M9)

14.1 Module

Authentication Client

14.2 Uses

- API Module M1
- Browser Runtime Module M20 (Client Runtime)

14.3 Syntax

14.3.1 Exported Constants

None.

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
handleLogin	-	void	AuthUIError
handleSignup	-	void	AuthUIError
handleLogout	-	void	
Render	props: ReactProps	JSX.Element	

14.4 Semantics

14.4.1 State Variables

- **username**: string — Stores the value from the username input field.
- **password**: string — Stores the value from the password input field.
- **isLoading**: bool — True if an API request (to M1) is in progress.
- **error**: string — Stores error messages from M1 (e.g., "Invalid credentials").

14.4.2 Environment Variables

- **BrowserRuntime (M20)**: The browser environment providing DOM rendering and storage.
- **AuthStorage**: The client-side storage mechanism (e.g., `localStorage`) used to persist the `AuthToken`.

14.4.3 Assumptions

- Module M1's authentication endpoints (`/api/auth/...`) are available.
- This module is rendered by M8 (Application Shell).

14.4.4 Access Routine Semantics

handleLogin( )

- transition: Sets **isLoading** to `true`. Reads **username** and **password** from state. Calls `M1.POST /api/auth/login`. On success, stores the returned `AuthToken` in **AuthStorage**, sets **isLoading** to `false`, and updates global auth state. On failure, sets **isLoading** to `false` and populates **error**.
- output: `void`.
- exception: `AuthUIError` (represented in the **error** state) if M1 fails.

handleSignup( )

- transition: Sets **isLoading** to `true`. Reads **username** and **password**. Calls `M1.POST /api/auth/signup`. Manages success or failure similar to `handleLogin`.
- output: `void`.
- exception: `AuthUIError` (represented in the **error** state) if M1 fails (e.g., user exists).

handleLogout( )

- transition: Removes the `AuthToken` from **AuthStorage**. Updates global auth state (e.g., sets `currentUser` to `null`).
- output: `void`.

Render(*props*)

- transition: Reads all **State Variables** to determine UI.
- output: Returns a `JSX.Element` containing login/signup forms, inputs, and buttons. UI reflects **isLoading** (e.g., spinner) and **error** (e.g., error message) states.

14.4.5 Local Functions

None.

14.4.6 Considerations

- The secret of this module is how and where the `AuthToken` is stored on the client (e.g., `localStorage` vs. cookie).
- This module is responsible for both the UI of the forms and the client-side logic of communicating with M1.

15 MIS of Lobby View Module (M10)

15.1 Module

Lobby View

15.2 Uses

- API Module M1
- Real-time Client Module M7
- Browser Runtime Module M20 (Client Runtime)

15.3 Syntax

15.3.1 Exported Constants

None.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
Render	props: ReactProps	JSX.Element	
handleCreateGame	-	void	CreateGameError
handleJoinGame	lobbyID: LobbyID	void	JoinGameError

15.4 Semantics

15.4.1 State Variables

- **lobbiesList**: Lobby[] — An array of available game lobbies.
- **selectedLobby**: LobbyID — null — The ID of the lobby currently selected in the UI.
- **isLoading**: bool — True if a create or join operation is in progress.

15.4.2 Environment Variables

- **BrowserRuntime (M20)**: The browser environment providing DOM rendering.

15.4.3 Assumptions

- Modules M1 and M7 are available and configured.
- This module is rendered by M8 (Application Shell).

15.4.4 Access Routine Semantics

Render(*props*)

- transition: Reads **lobbiesList**, **selectedLobby**, and **isLoading** from state.
- output: Returns a `JSX.Element` that renders the UI for listing, creating, and joining game lobbies. Renders a loading indicator if **isLoading** is true.

handleCreateGame( )

- transition: Sets **isLoading** to `true`. Calls `M1.POST /api/game` to create a new lobby. On success, receives a `newLobbyID` and calls `handleJoinGame(newLobbyID)`.
- output: `void`.
- exception: `CreateGameError` (displayed in UI) if the M1 call fails.

handleJoinGame(*lobbyID*)

- transition: Sets **isLoading** to `true`. Calls `M7.emit('joinGame', { lobbyID: lobbyID, ... })`. On success, the M7/M2 connection will trigger a state change that M8 will use to render M11.
- output: `void`.
- exception: `JoinGameError` (displayed in UI) if M7 fails to join.

15.4.5 Local Functions

None.

15.4.6 Considerations

- The secret of this module is the UI layout for displaying, creating, and joining games.
- It coordinates user actions, calling M1 for lobby creation and M7 for joining a real-time session.

16 MIS of Game Board View Module (M11)

16.1 Module

Game Board View

16.2 Uses

- Browser Runtime Module M20 (Client Runtime)
- Move Controller Module M12

16.3 Syntax

16.3.1 Exported Constants

None.

16.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
Render	props: ReactProps	JSX.Element	None

16.4 Semantics

16.4.1 State Variables

- **clientGameState**: GameState — The current game state object (hands, deck, discard pile).
- **validMoves**: Card[] — An array of cards in the player's hand that are legal to play.

16.4.2 Environment Variables

- **BrowserRuntime (M20)**: The browser environment providing DOM rendering and CSS.

16.4.3 Assumptions

- This module is rendered by M8 when a game is active.
- The **clientGameState** and **validMoves** are provided (likely as props).
- Event handlers from M12 are attached to the rendered elements.

16.4.4 Access Routine Semantics

Render(*props*)

- transition: Reads **clientGameState** and **validMoves** from state/props.
- output: Returns a `JSX.Element` that renders the main game interface, including the player's hand, the discard pile, and the deck. It visually highlights any cards in the hand that are also present in the **validMoves** list.

16.4.5 Local Functions

None.

16.4.6 Considerations

- The secret of this module is the DOM/CSS structure and animation logic used to render the game board.
- This is primarily a "dumb" rendering component; it displays state and delegates user input handling to M12.

17 MIS of Move Controller Module (M12)

17.1 Module

17.2 Uses

17.3 Syntax

17.3.1 Exported Constants

17.3.2 Exported Access Programs

17.4 Semantics

17.4.1 State Variables

17.4.2 Environment Variables

17.4.3 Assumptions

17.4.4 Access Routine Semantics

17.4.5 Local Functions

17.4.6 Considerations

18 MIS of Scoreboard View Module (M13)

18.1 Module

18.2 Uses

18.3 Syntax

18.3.1 Exported Constants

18.3.2 Exported Access Programs

18.4 Semantics

18.4.1 State Variables

18.4.2 Environment Variables

18.4.3 Assumptions

18.4.4 Access Routine Semantics

18.4.5 Local Functions

18.4.6 Considerations

19 MIS of Profile View Module (M14)

19.1 Module

19.2 Uses

19.3 Syntax

19.3.1 Exported Constants

19.3.2 Exported Access Programs

19.4 Semantics

19.4.1 State Variables

19.4.2 Environment Variables

19.4.3 Assumptions

19.4.4 Access Routine Semantics

19.4.5 Local Functions

19.4.6 Considerations

20 MIS of Game Engine Module (M15)

20.1 Module

20.2 Uses

20.3 Syntax

20.3.1 Exported Constants

20.3.2 Exported Access Programs

20.4 Semantics

20.4.1 State Variables

20.4.2 Environment Variables

20.4.3 Assumptions

20.4.4 Access Routine Semantics

20.4.5 Local Functions

20.4.6 Considerations

21 MIS of Rules Module (M16)

21.1 Module

21.2 Uses

21.3 Syntax

21.3.1 Exported Constants

21.3.2 Exported Access Programs

21.4 Semantics

21.4.1 State Variables

21.4.2 Environment Variables

21.4.3 Assumptions

21.4.4 Access Routine Semantics

21.4.5 Local Functions

21.4.6 Considerations

22 MIS of Scoring Module (M17)

22.1 Module

22.2 Uses

22.3 Syntax

22.3.1 Exported Constants

22.3.2 Exported Access Programs

22.4 Semantics

22.4.1 State Variables

22.4.2 Environment Variables

22.4.3 Assumptions

22.4.4 Access Routine Semantics

22.4.5 Local Functions

22.4.6 Considerations

23 MIS of Base Conversion Module (M18)

23.1 Module

23.2 Uses

23.3 Syntax

23.3.1 Exported Constants

23.3.2 Exported Access Programs

23.4 Semantics

23.4.1 State Variables

23.4.2 Environment Variables

23.4.3 Assumptions

23.4.4 Access Routine Semantics

23.4.5 Local Functions

23.4.6 Considerations

24 MIS of Operating System Module (M19)

24.1 Module

24.2 Uses

24.3 Syntax

24.3.1 Exported Constants

24.3.2 Exported Access Programs

24.4 Semantics

24.4.1 State Variables

24.4.2 Environment Variables

24.4.3 Assumptions

24.4.4 Access Routine Semantics

24.4.5 Local Functions

24.4.6 Considerations

25 MIS of Browser Runtime Module (M20)

25.1 Module

25.2 Uses

25.3 Syntax

25.3.1 Exported Constants

25.3.2 Exported Access Programs

25.4 Semantics

25.4.1 State Variables

25.4.2 Environment Variables

25.4.3 Assumptions

25.4.4 Access Routine Semantics

25.4.5 Local Functions

25.4.6 Considerations

26 MIS of Database Module (M21)

26.1 Module

26.2 Uses

26.3 Syntax

26.3.1 Exported Constants

26.3.2 Exported Access Programs

26.4 Semantics

26.4.1 State Variables

26.4.2 Environment Variables

26.4.3 Assumptions

26.4.4 Access Routine Semantics

26.4.5 Local Functions

26.4.6 Considerations

27 MIS of [Module Name —SS]

[Use labels for cross-referencing —SS]

[You can reference SRS labels, such as R??. —SS]

[It is also possible to use LaTeXfor hypperlinks to external documents. —SS]

27.1 Module

[Short name for the module —SS]

27.2 Uses

27.3 Syntax

27.3.1 Exported Constants

27.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

27.4 Semantics

27.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

27.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

27.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

27.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]

- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

27.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

28 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
2. What pain points did you experience during this deliverable, and how did you resolve them?
3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?
4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?
5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)
6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)