**COMP4910 Senior Design Project 1, Fall 2023**

**Advisor: Behçet Melih SARIBATIR**

**AimMaxHit**

**Design Specifications Document**

**Revision 1.0**

**20.01.2025**

**By:**

**Efe Demirtaş, 20070001053**

**Kaan Mert Kozalı, 21070001064**

**Umut Özgür İpin, 20070001037**

**Volkan Ege Kılınç, 20070001058**

# Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Explanation** |
| 1.0 | 20.01.2025 | Initial high level design |

# Table of Contents

[Revision History 2](#_Toc60144428)

[Table of Contents 3](#_Toc60144429)

[1. Introduction 4](#_Toc60144430)

[2. AIMMAXHIT System Design 4](#_Toc60144431)

[3. AIMAXHIT Software Subsystem Design 6](#_Toc60144437)

[3.1. AIMMAXHIT Software System Architecture 6](#_Toc60144438)

[3.2. AIMMAXHIT Software System Structure 7](#_Toc60144439)

[3.3. AIMMAXHIT Software System Environment 7](#_Toc60144440)

[4. AIMMAXHIT Software System Detailed Design: 9](#_Toc60144441)

[4.1. AIMMAXHIT Main Module/Class 8](#_Toc60144442)

[4.2. AIMMAXHIT Subsystem S1 Classes 8](#_Toc60144443)

[4.2.1. Class S1-C1 8](#_Toc60144444)

[4.2.2. Class S1-C2 8](#_Toc60144445)

[4.2.3. Class S1-C3, etc. 9](#_Toc60144446)

[4.3. AIMMAXHIT Subsystem S2 Classes 9](#_Toc60144447)

[4.3.1. Class S2-C1 9](#_Toc60144448)

[4.3.2. Class S2-C2 9](#_Toc60144449)

[4.3.3. Class S2-C3, etc. 10](#_Toc60144450)

[4.4. AIMMAXHIT Subsystem S3 Classes 10](#_Toc60144451)

[4.n. AIMMAXHITSubsystem Sk- Common Infrastructure Classes 10](#_Toc60144452)

[5. Testing Design 10](#_Toc60144453)

[References 10](#_Toc60144454)

# 1. Introduction

AimMaxHit is a gaming analytics platform designed to evaluate and enhance players' skills. The platform combines Unity for the frontend, .NET Core for backend operations, and Firebase for real-time data storage and user authentication. It captures real-time gameplay data, analyzes player performance, and provides actionable insights for improvement.

1. **Login and Registration Operations:**
   * User login
   * User registration
   * Forgot password / password reset
2. **Game Operations:**
   * Start the game
   * Play the game (receive moves from the user)
   * Check the game state
   * End the game
3. **Profile Operations:**
   * View profile
   * Edit profile (name, password, profile picture)
   * View scores
4. **Level and Scoring System:**
   * Level up
   * Scoring system (winners get more points)
   * View levels and scores
5. **Logout and Session Management:**
   * Log out
   * Exit the application
6. **UI Update Operations:**
   * Update the game board
   * Update user information
   * Update scores and levels
7. **Email Confirmation and Security Operations:**
   * Email verification operations
   * Security measures (strong password requirements, use of SSL, etc.)
8. **Error Handling:**
   * Notify users of errors
   * Security checks and error handling

# 2. AIMMAXHIT System Design

AimMaxHit is designed as a modular and scalable gaming analytics platform. Its architecture consists of three main layers:

1. **Presentation Layer**:
   * Built using Unity, providing an interactive and user-friendly interface.
   * Handles user interactions, such as login, gameplay, and viewing analytics.
2. **Application Layer**:
   * Developed with .NET Core to manage the business logic and system processes.
   * Handles real-time data capture, analytics processing, and leaderboard updates.
3. **Data Layer**:
   * Firebase offers secure, scalable, and real-time data storage.
   * Stores user profiles, gameplay data, performance metrics, and system logs.

**3. AIMMAXHIT Software Subsystem Design**

The system comprises the following subsystems:

1. **User Management Subsystem**:
   * Authentication via Firebase.
   * Handles user profiles, password reset, and email verification.
2. **Gameplay Subsystem**:
   * Manages game sessions, including start, play, and end states.
   * Captures real-time performance metrics.
3. **Analytics Subsystem**:
   * Processes gameplay data to generate performance insights.
   * Tracks user progress, levels, and scoring.
4. **Leaderboard Subsystem**:
   * Updates and displays rankings based on player scores.
   * Filters data for global or friend-specific leaderboards.
5. **Notification Subsystem**:
   * Sends real-time updates for achievements, system changes, or reminders.

# 3.1. AIMMAXHIT Software System Architecture

The architecture of AimMaxHit is designed to be modular, scalable, and maintainable, supporting real-time gameplay analytics and user interactions. The system architecture consists of three primary layers:

**1. Presentation Layer**

* **Technology Used**: Unity
* **Description**:
  + The presentation layer provides an interactive and visually engaging interface for the users.
  + It handles user interactions, such as login, registration, gameplay initiation, and data visualization.
  + Responsible for real-time updates to the user interface, such as gameplay data, leaderboard rankings, and notifications.
* **Key Components**:
  + **Login and Registration Interface**: Enables secure authentication and onboarding of users.
  + **Game Dashboard**: Displays key metrics, performance insights, and options for initiating gameplay.
  + **Game Board**: The core interface for gameplay, capturing user moves and providing visual feedback.

**2. Application Layer**

* **Technology Used**: .NET Core
* **Description**:
  + The application layer acts as the business logic engine, processing user requests and managing system operations.
  + It connects the frontend (Unity) with the backend (Firebase) through APIs.
  + Real-time gameplay data is processed here for analytics and leaderboard updates.
* **Key Responsibilities**:
  + **Game Session Management**: Manages game lifecycle, including initiation, real-time data capture, and conclusion.
  + **Performance Analytics**: Processes and analyzes gameplay data to generate actionable insights.
  + **Leaderboard Management**: Updates player rankings and provides data to the frontend for display.

**3. Data Layer**

* **Technology Used**: Firebase
* **Description**:
  + The data layer is responsible for storing and retrieving application data in a secure and efficient manner.
  + It ensures real-time synchronization across devices and users.
  + Designed to handle large-scale data storage and rapid retrieval for analytics.
* **Key Responsibilities**:
  + **User Data Management**: Stores user profiles, authentication data, and preferences.
  + **Game Data Storage**: Captures and stores gameplay metrics, session logs, and performance insights.
  + **Leaderboard Data**: Maintains global and friend-specific leaderboards.

**System Communication Flow**

1. **User Action**: Users interact with the Unity interface to log in, initiate a game, or view analytics.
2. **Frontend Request**: Unity sends requests to the .NET Core application layer via RESTful APIs.
3. **Backend Processing**: The application layer processes requests, interacts with Firebase for data retrieval or updates, and applies business logic.
4. **Real-Time Updates**: Firebase ensures real-time synchronization, pushing data updates to the Unity interface.
5. **Feedback to User**: Processed data is visualized in the Unity interface, providing users with immediate feedback.

**Security Measures**

* **Authentication**: Firebase Authentication ensures secure user access.
* **Data Encryption**: All communication between layers is encrypted using HTTPS.
* **Role-Based Access Control (RBAC)**: Limits access to administrative functionalities and sensitive data.

**Scalability Features**

* Firebase supports horizontal scaling for data storage and retrieval.
* .NET Core is optimized for high-concurrency scenarios, ensuring efficient request handling.
* Unity's modular design allows for easy updates and feature additions.

3.2. AIMMAXHIT Software System Structure

**3.2.1. Unity and C# Components**

*Package/Component Structure:*

* Unity Package: Represents the front-end environment for designing and implementing the user interface.
* C# Scripts Package: Contains scripts to manage user interactions, gameplay logic, and client-side calculations.

Discussion and Justification:

* Unity provides a robust framework for building an interactive and visually appealing gaming interface.
* The modular C# scripts ensure maintainability and scalability of gameplay mechanics.

**3.2.2. .NET Core Components**

*Package/Component Structure:*

* **Controllers Package: Handles API requests and business logic for AimMaxHit.**
* **Services Package**: Encapsulates reusable logic such as data processing and analytics generation.
* **Data Access Package**: Manages communication with Firebase for data storage and retrieval.

***Discussion and Justification:***

* *.NET Core's structured approach separates concerns, enhancing code readability and maintainability.*
* *The service-oriented architecture facilitates scalability and modular updates.*
* *Robust backend operations ensure high performance and reliability during gameplay.*

**3.2.3. Firebase Components**

Package/Component Structure:

* Authentication Module: Manages user login, registration, and session handling.
* Firebase Module: Stores real-time data such as gameplay stats, user profiles, and leaderboards.
* Cloud Functions Module: Executes serverless backend logic for tasks like email verification and notifications.

***Discussion and Justification:***

* *Firebase’s real-time database supports instant updates to the front-end.*
* *Its integrated authentication system enhances security and simplifies user management.*
* *Cloud Functions provide scalability and reduce server-side workload.*

**3.2.4. Python and Data Analysis Components**

Package/Component Structure:

* Data Processing Package: Implements Python scripts for analyzing gameplay metrics.
* Visualization Package: Generates insights and graphs for user performance.
* Machine Learning Module: (Future) Applies ML algorithms for advanced player recommendations.

***Discussion and Justification:***

* *Python’s simplicity accelerates data analysis and insight generation.*
* *Visualization tools enhance the user experience by presenting actionable feedback.*
* *Modular structure supports future enhancements, such as AI-based analysis.*

**3.2.5. API Component**

*Package/Component Structure:*

* **RESTful API Package:** Defines endpoints for client-server communication, such as login, game session data, and leaderboard updates.

*Discussion and Justification:*

* RESTful APIs enable efficient and stateless communication between the Unity front-end and .NET Core backend.
* This architecture supports scalability and integration with additional services like analytics or notifications.

# 3.3. AIMMAXHIT Software System Environment

**3.3.1. Hardware Environment**

***Description:*** *The AimMaxHit Application is tailored to operate seamlessly on conventional computer hardware, including desktops, laptops, and servers. It is optimized to support gaming peripherals and ensure low-latency operations for a smooth user experience.*

**3.3.2. System Software Environment**

***Description:*** *AimMaxHit is compatible with major operating systems, including:*

* *Windows (10 and later versions).*

**3.3.3. Middleware**

*Middleware integration enhances the application’s functionalities, enabling seamless communication between the front-end (Unity) and back-end (Firebase and .NET Core).*

**3.3.4. Programming Language and Frameworks**

* ***Programming Languages:***
  + ***C#:*** *Used for developing the Unity front-end.*
  + ***.NET Core:*** *Handles the back-end logic and APIs.*
  + ***Python:*** *Utilized for data analysis and performance insights.*
* ***Frameworks:***
  + ***Unity Framework:*** *Creates the interactive user interface and handles gameplay mechanics.*
  + ***Firebase Frameworks:*** *Manages real-time data storage and authentication.*
  + ***.NET Core Framework:*** *Ensures robust server-side operations and API management.*

**3.3.5. Database**

***Database Management System (DBMS):*** *Firebase is chosen for its:*

* *Real-time data synchronization.*
* *Scalability to handle large datasets.*
* *Cloud-based infrastructure ensuring global accessibility.*
* *Seamless integration with other platforms and applications.*

**3.3.6. Software Tools**

* ***Development Environment:***
  + *Unity Editor for front-end development.*
  + *Visual Studio for back-end .NET Core development.*
  + *Python IDEs (PyCharm or Jupyter or Google Colab) for data analysis.*
* ***Version Control:*** *Unity Collab and Git is utilized for collaborative development, efficient tracking of code changes, and maintaining a robust version history.*

**3.3.7. UML Deployment Diagram**

*Discussion and Justification:*

* The outlined hardware and software environment ensures broad compatibility across a variety of user devices and operating systems.
* Middleware incorporation facilitates real-time communication and enhances overall application functionality.
* The selection of C#, .NET Core, and Python aligns with modern development practices, promoting efficiency and ease of maintenance.
* Firebase is chosen for its reliability, scalability, and suitability for real-time gaming applications.

# 4. AIMMAXHIT Software System Detailed Design:

Subsequent parts of this section is kept here for completeness and to give you an idea on what you will be doing in terms of detailed design in the context of COMP 4920 course next semester.

For the course COMP 4910,Section 5 will be almost empty, it is sufficient that you write “detailed design will be carried out in the context of COMP 4920 course next semester”.

The detailed design section should be organized in terms of classes of packages/components for a complex software system, for example Section 5.1 will contain all classes/methods of package/component 1, that is subsystem 1, Section 5.2 all classes/methods of package/component 2, that is subsystem 2, etc.

# 4.1 AIMMAXHIT Main Module/Class

The main module acts as the entry point for the system, coordinating initialization, module integration, and high-level operations.

# 4.2. AIMMAXHIT Subsystem S1 Classes

**4.2. AIMMAXHIT Application Class Structures**

**4.2.1. Class S1-C1: User Management**

* **Responsibilities**: Handles user registration, login, password reset, and profile management.
* **Methods**:
  + registerUser(details): Registers a new user with the provided details.
  + login(credentials): Authenticates a user with email and password.
  + resetPassword(email): Sends a password reset link to the user’s email.

**4.2.2. Class S1-C2: Game Mechanics**

* **Responsibilities**: Manages game logic, including bot and multiplayer modes.
* **Methods**:
  + initializeGame(gameType): Starts a new game of the specified type (bot or multiplayer).
  + executeMove(player, position): Processes a player’s move and updates the game state.

**4.2.3. Class S1-C3: Player Profile and Level System**

* **Responsibilities**: Manages player profiles, avatars, and level progression.
* **Methods**:
  + updateProfile(profileDetails): Updates user information.
  + uploadAvatar(image): Uploads a new profile avatar.
  + gainExperience(points): Increments user experience points for level progression.

**4.2.4. Class S1-C4: Game Board and Status Management**

* **Responsibilities**: Manages the game board state and determines game outcomes.
* **Methods**:
  + placeMarker(player, position): Places the player’s marker on the board.
  + checkWinner(): Checks if a player has won or if the game is a draw.

**4.2.5. Class S1-C5: Difficulty Settings and AI**

* **Responsibilities**: Adjusts the difficulty level and manages AI behavior.
* **Methods**:
  + setDifficulty(level): Sets the difficulty level for the game.
  + aiMove(): Calculates and executes the AI’s next move.

**4.2.6. Class S1-C6: Communication and Notifications**

* **Responsibilities**: Manages in-game communication and user notifications.
* **Methods**:
  + sendMessage(message, recipient): Sends a message to another user.
  + notifyUser(event): Sends a notification about a specific game event.

**4.2.7. Class S1-C7: Game Statistics and Analysis**

* **Responsibilities**: Collects and analyzes gameplay data.
* **Methods**:
  + collectStats(gameData): Collects statistics from a completed game.
  + generateReport(): Produces a detailed performance report for the user.

**4.3. AIMMAXHIT Subsystem S2 Classes**

**4.3.1. Class S2-C1: Game Progress Management**

* **Responsibilities**: Manages online game sessions and player matchmaking.
* **Methods:**
  + createOnlineGame(sessionDetails): Sets up a new online game session.
  + matchPlayers(): Matches players for an online game.

**4.3.2. Class S2-C2: Score and Statistics Management**

* **Responsibilities**: Tracks and manages player scores and statistics.
* **Methods**:
  + recordGameResult(gameData): Records the results of a completed game.
  + generateStatistics(): Generates aggregated statistics for players.

**4.3.3. Class S2-C3: Customization and Settings**

* **Responsibilities**: Handles UI customization and user preferences.
* **Methods**:
  + customizeInterface(options): Applies user-defined UI customization settings.
  + updateSettings(settings): Updates application settings based on user preferences.

# 5. Testing Design

**General Test Remarks:**

* **Unit Testing**: Each module, method, and class in the AIMMAXHIT application should undergo unit testing to validate individual functionalities.
* **Integration Testing**: Following unit testing, integration tests should be conducted to ensure that different parts of the application work together seamlessly.
* **User Acceptance Testing**: Critical for assessing the application’s usability and ensuring it meets the end-user requirements.
* **Performance Testing**: To ensure the application performs well under expected load conditions.
* **Security Testing**: Especially crucial for the user management module to safeguard personal and sensitive data.

**References**

During the development of the AIMMAXHIT application, the following references were consulted to ensure best practices, security measures, and optimal coding standards:

1. **Unity Documentation** - [Unity Documentation](https://docs.unity.com)
2. **Firebase Documentation** - [Firebase Documentation](https://firebase.google.com/docs?hl=tr)
3. **.NET Core Documentation** - [.NET Core Documentation](https://learn.microsoft.com/en-us/dotnet/core/)
4. **Python Documentation** - [Python Documentation](https://docs.python.org/3/)