# SOFTWARE REQUIREMENTS SPECIFICATION [V4]

Software Requirements Specification for a Game Streaming Application called G Stream

NOVEMBER 29, 2023

BY HD SECTOR

# Contents


1. Introduction ..... 3
   1.1 Purpose ..... 3
   1.2 Intended Use ..... 3
   1.3 Scope ..... 3
2. Overall Description ..... 3
   2.1. Product Perspective ..... 3
   2.2. Product Features ..... 3
   2.3. User Classes ..... 4
   2.4. Operating Environment ..... 4
3. Specific Requirements ..... 4
   3.1. Functional Requirements ..... 4
      3.1.1. User Registration ..... 4
      3.1.2. User Authentication ..... 4
      3.1.3. Profile Management ..... 4
      3.1.4. Content Discovery ..... 4
      3.1.5. Live Streaming ..... 5
      3.1.6. Chat and Interaction ..... 5
      3.1.7. Notifications ..... 5
      3.1.8. Monetization ..... 5
      3.1.9. Content Reporting and Moderation ..... 5
   3.2. Non-Functional Requirements ..... 5
      3.2.1. Performance ..... 5
      3.2.2. Usability: ..... 5
      3.2.3. Maintainability ..... 5
      3.2.4. Flexibility ..... 6
      3.2.5. Security ..... 6
      3.2.6. Compatibility ..... 6
      3.2.7. Data Privacy ..... 6
   3.3 Technology Requirements ..... 6
      3.3.1. Programming Language ..... 6
      3.3.2. Data Requirements ..... 6
4. Methodology Required ..... 7
   4.1. SDLC Models ..... 7
      Waterfall Model ..... 7
      RAD Model ..... 8

Spiral Model ..... 8
V-Model ..... 8
Incremental Model ..... 9
Agile Model ..... 9
Iterative Model ..... 9
Big bang model ..... 9
Prototype Model ..... 9
4.2. The suitable SDLC Model for the specific Application Development ..... 10
4.3. Phases of Agile Model ..... 10
4.3.1 Requirements: ..... 10
4.3.2 Design: ..... 11
4.3.3 Development /Iteration: ..... 11
4.3.4 Testing: ..... 11
4.3.5 Deployment: ..... 11
4.3.6 Review ..... 11
4.4. Why choose Agile Model ..... 12
5. System Models/Diagram ..... 13
USE CASE DIAGRAM ..... 13
FLOW CHART ..... 14
UML DIAGRAM ..... 15
6. Test and Validation ..... 16
6.1 Testing Approaches ..... 16
6.1.1 Exploratory Approach ..... 16
6.1.2 Box Approach ..... 16
6.2 Reasons of choosing Grey-Box Testing Approach ..... 17
6.3 Testing Levels ..... 18
6.4 Testing types, techniques and tactics ..... 18
6.5 Testing Process ..... 19
6.6 Hierarchy of Testing Difficulty ..... 20
6.7. Complete Test Plan ..... 21
6.8 Test Cases ..... 23
7. Project Scheduling and Gantt Chart ..... 25
8. Payment Terms ..... 26
9. Contact ..... 26

# 1. Introduction

## 1.1 Purpose

The main purpose of the G Stream Streaming App is to provide gamers with a robust platform to stream their gameplay live and to offer viewers an engaging way to enjoy and interact with gaming content in real time. The app aims to foster a dynamic gaming community and enhance the overall gaming experience for both streamers and viewers.

## 1.2 Intended Use

The Game Streaming App enables gamers to broadcast their gameplay sessions in real time, allowing viewers to watch and interact with the streamers. It facilitates engagement through live chat, comments, and reactions, creating a social and interactive experience around gaming content.

## 1.3 Scope

The Game Streaming App will encompass the following features:

- The User registration and authentication process to ensure secure access.
- User profiles for streamers and viewers with customizable avatar and profile information.
- Live streaming functionality that allows streamers to broadcast their gameplay sessions with live audience.
- Interactive features like live chat, comments, and real-time reactions and donations.
- Search and discovery tools to find streams based on game titles, genres, and streamers profiles.
- Notifications for the followers and subscribers about new streams and interactions.
- Analytics dashboard for streamers to track viewership, engagement, and trends in gaming community.
- Cross-platform compatibility for major operating systems (iOS, Android, Windows & macOS).

# 2. Overall Description

## 2.1. Product Perspective

The G Stream app operates as a standalone live streaming platform, enabling users to discover, watch, and interact with live broadcasts from streamers. It is part of the larger G Stream ecosystem, which includes a web platform and APIs for third-party integrations.

## 2.2. Product Features

Key features of the G Stream game streaming app include:
- User registration and authentication.

- User profile and information management.
- Content discovery through categories, tags and recommendations.
- Live streaming with live chat functionality.
- Real-time notifications for followers and their favorite streamers.
- Monetization options for streamers.
- Content reporting and moderation tools.

### 2.3. User Classes

- Viewers: The Individuals who are interested in watching live streams.
- Streamers: Content creators or streamers who broadcast live.
- Moderators: Trusted users or admins responsible for maintaining platform guidelines.

### 2.4. Operating Environment

The app is designed to run on iOS (above IOS 7) and Android smartphones (version 6.0 and above) and different web browsers (supporting modern standards).

## 3. Specific Requirements

### 3.1. Functional Requirements

#### 3.1.1. User Registration

- Users can sign up using email, phone number, or social media accounts (e.g. twitter, Facebook).
- Validation of user-provided information during registration of profile.
- Account verification via email or SMS by code.

#### 3.1.2. User Authentication

- Secure and safe login mechanism with password encryption.
- Support for two-factor authentication (2FA).
- Session management for the logged-in users.

#### 3.1.3. Profile Management

- Users can create and edit their profiles with their own information.
- Profile includes display name, profile picture, and brief description of user.
- Option to link social media profiles.

#### 3.1.4. Content Discovery

- Browse live streams by category, tags, and popularity of games and streamers.

- Personalized content recommendations based on user preferences and viewing history in profile.

#### 3.1.5. Live Streaming

- Streamers can start and watch live streaming of other players.
- Viewers can watch live streams and interact through chat simultaneously.

#### 3.1.6. Chat and Interaction

- Real-time chat for viewers to interact with streamers and other viewers.
- Moderation tools for streamers and moderators to manage chat.
- Emote and gifting system for viewer engagement.

#### 3.1.7. Notifications

- Push notifications for follower activity (e.g., new followers, live stream alerts).
- Option to customize notification preferences.

#### 3.1.8. Monetization

- Streamers can monetize their content through ads, donations, and subscription models.
- Integration with payment gateways (e.g. PayPal, Google Pay) for transactions and donations.
- Revenue tracking and withdrawal options for the streamers.

#### 3.1.9. Content Reporting and Moderation

- Users can report any inappropriate content if they want.
- Moderators can review and take action on reported content and users.
- Automated content flagging for potential violations.

### 3.2. Non-Functional Requirements

#### 3.2.1. Performance

- App should load quickly and stream smoothly on various network conditions.

#### 3.2.2. Usability:

Usability of software will be easy so that e-learner can use it without any difficulty.

#### 3.2.3. Maintainability

Software would build up in such a way that classifications of errors and maintenance of mechanism become easy.

#### 3.2.4. Flexibility

Software would be flexible so that it can easily accept all changes at low cost, time and experience.

#### 3.2.5. Security

- Secure storage and transmission of user data.
- Regular security audits and vulnerability assessments.

#### 3.2.6. Compatibility

- Support for major web browsers like Chrome, Firefox, Safari and mobile platforms (iOS, Android).
- Responsive design for various screen sizes and resolutions.

#### 3.2.7. Data Privacy

- User data encryption at rest and in transit.
- Ensure transparent privacy policy.

## 3.3 Technology Requirements

#### 3.3.1. Programming Language

**Backend:** Choose a backend programming language for server-side development. Common choices include Python (Django, Flask), Node.js (Express.js), Ruby (Ruby on Rails), or Java (Spring Boot).
**Frontend:** Use HTML, CSS, and JavaScript (React, Angular, or Vue.js) for building the best user interface and client-side functionality.

#### 3.3.2. Data Requirements

- Use of a relational database (e.g., MySQL) for user profiles and interactions.
- Use of NoSQL database (e.g., MongoDB) for real-time chat and notifications.

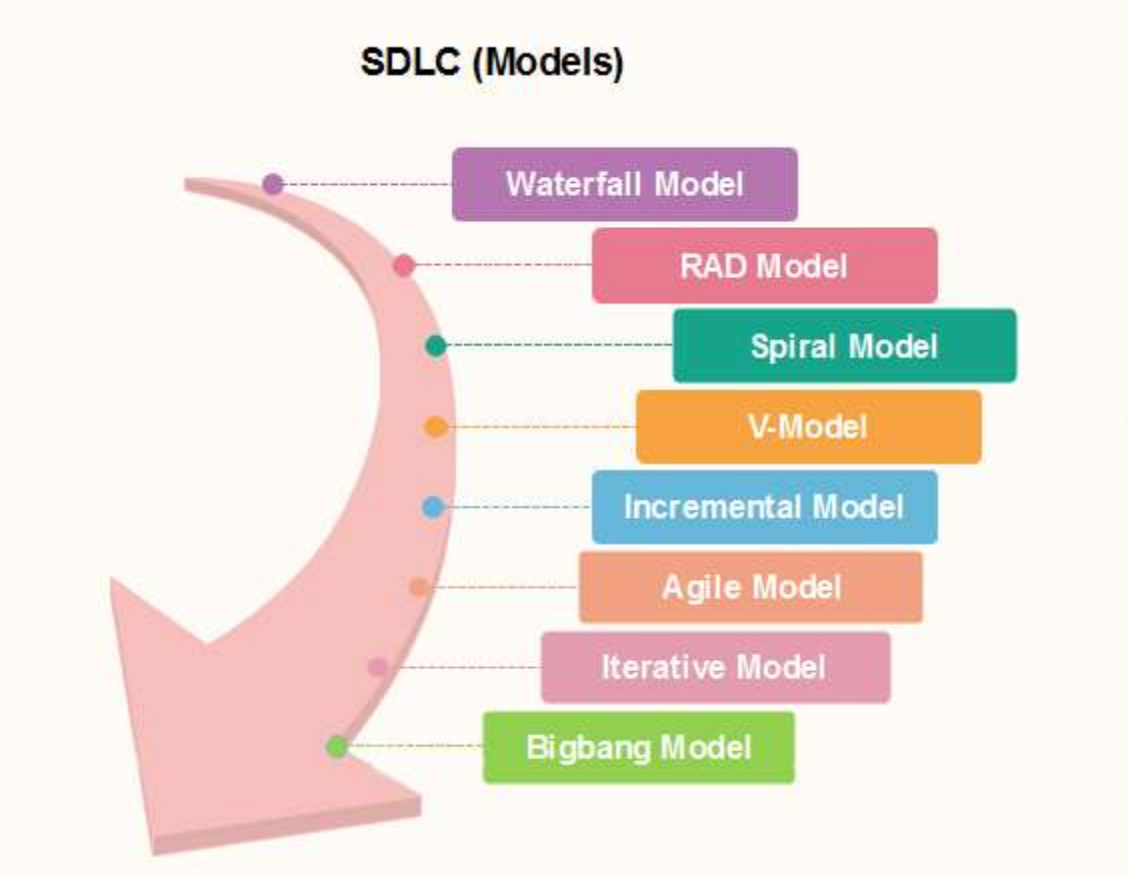
# 4. Methodology Required

## 4.1. SDLC Models

Software Development life cycle (SDLC) is a model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

There are different software development life cycle models. These models are also called "**Software Development Process Models**." Each SDLC process model follows a series of phase unique to its type for software development.

Here, are some well-known and important models of SDLC life cycle:




**Waterfall Model**

The waterfall is a universally accepted SDLC model. In this method, the whole process of software development is divided into many phases. The waterfall model is a continuous software

development model in which development is seen as flowing steadily downwards like waterfall through the steps of requirements

- analysis
- design
- implementation
- testing integration
- and maintenance.

In waterfall model the project is divided into different phases from upward to downward. In this model each phase is completely wrapped up before the next phase.

**RAD Model**

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period of time. The RAD model ensures great system development in lesser time with focused team and implementation.

The phases of RAD model are:

- Data Modelling
- Process Modelling
- Application Generation
- Testing and Turnover

**Spiral Model**

The spiral model is a risk-driven process model. It is used for risk management that uses elements of one or more process models like a waterfall, incremental etc.

Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the goals, and the constraints that exist. This is the first step of the cycle.

The next step is to develop strategies that solve uncertainties and risks. Which involves activities such as benchmarking, simulation, and prototyping.

**V-Model**

V model also known as Verification and Validation Model. This model works in sequential manner of V shape. Model In this type of SDLC model testing and the development, the step is planned in parallel. So, there are verification phases on one side and the validation phase on the other side. V-Model joins by Coding phase.

## Incremental Model

The incremental model is necessarily a series of waterfall cycles. At the start of the project the requirements are divided into groups. The SDLC process is repeated, with each release adding more functionality also known as “increment” until all requirements are met. The incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

The incremental model can be used for complex and bigger projects.

## Agile Model

The Agile Model is combination of incremental and iterative process of software development. It defines each iteration’s number, duration, and scope in advance. In this process the project is divided into multiple iterations and each of them lasts about 3–8 weeks. The division of the entire project into small parts helps to complete the project in less time and with less risk.

Agile Model divides tasks into time boxes to provide specific functionality for the release. Each build is incremental in terms of functionality, with the final build containing all the attributes. Agile Model is perfect for bigger projects and software developments.

## Iterative Model

It is a particular implementation of a software development life cycle works based on the initial requirements which are clearly defined and necessary features are added through iterations until the final system is developed.

The process in this model works by multiple iterations that’s why it is called iterative model.

## Big bang model

In this model, there are no specific defined process and a very little planning is required. Developers do not follow any specific process. The development starts with the required and necessary funds and efforts in the form of inputs. The customer requirements are also not defined in this model as a result the outcome may or may not be as per the customer's requirement.

Big bang model suitable for smaller and less complex projects.

## Prototype Model

The prototype model works by building the prototype first with the requirements gathering, then testing it and again making changes to the prototype until the final and acceptable outcome is achieved. The developer and the user meet and define the purpose of the software, identify the needs, etc.

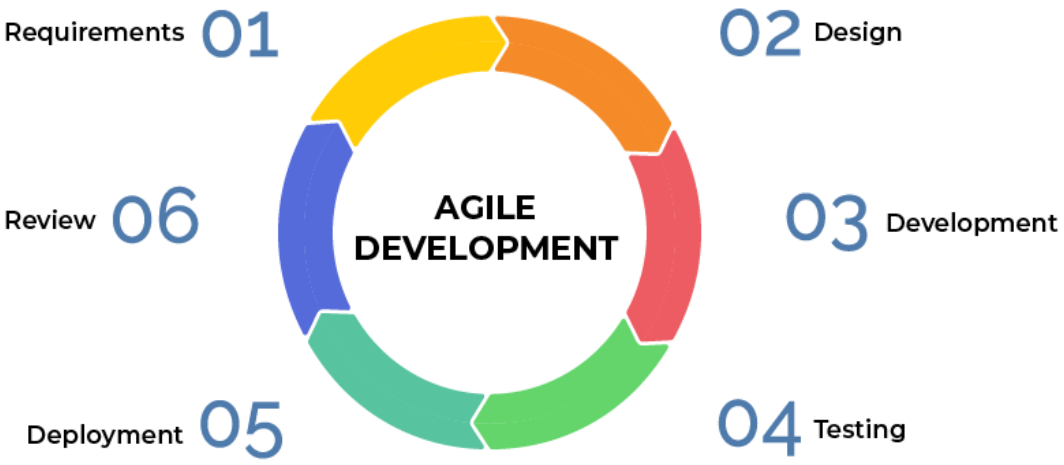
First a quick design or a prototype is created. This design focuses on those aspects of the software that will be visible to the user. It then leads to the development of a prototype. The customer then checks the prototype, and any modifications or changes that are needed are made to the prototype. These prototypes are created using loops and keep getting updates for better outcome.

Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

## 4.2. The suitable SDLC Model for the specific Application Development

**Agile** is based on the adaptive software development methods, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. The Agile Model is combination of incremental and iterative process of software development. In this process the project is divided into multiple iterations and each of them lasts about 3–8 weeks. The division of the entire project into small parts helps minimize delivery time and overall risk.

Users/Customers have the opportunity to make modifications throughout project development phases




## 4.3. Phases of Agile Model

### 4.3.1 Requirements:

The requirements or the features are directly taken from the customers or users. Which is written in form of stories or features in collaboration with the stakeholders.

### 4.3.2 Design:

In Agile, design is an ongoing activity that happens in parallel with development. It focuses on creating just enough design to support the current set of user stories. The designing process in not detailed designing to avoid over-engineering.

### 4.3.3 Development /Iteration:

The third phase of the agile development process is development or iteration.

In the development or iteration phase, the following steps are accomplished:

- The association with clients.
- Iterations and functionalities are prioritized and implemented.
- Each iteration should be closely examined.
- Delivering regular working software releases.
- Ensuring product quality by testing at regular intervals.

### 4.3.4 Testing:

The Testing phase in Agile is continuous and integrated throughout the development process. Here automated testing is used to ensure that code changes don't introduce new bugs. For exploratory and usability testing manual testing is also performed.

### 4.3.5 Deployment:

Continuous Deployment is often a key practice in Agile. It means that code changes are automatically deployed to production as soon as they pass automated tests. This ensures that working software is consistently delivered to users.

### 4.3.6 Review

Reviews in Agile are part of the sprint cadence. At the end of each iteration/sprint, the team holds a sprint review meeting to demonstrate the completed user stories to stakeholders. This is an opportunity for stakeholders to provide feedback on the delivered features.

In Agile, these phases are not strictly linear, and they often overlap. The key is to continuously iterate and refine the product. Additionally, Agile encourages adaptability and responsiveness to changing requirements, so adjustments can be made at any point in the process.

## 4.4. Why choose Agile Model

G Stream is a live streaming platform focused on gaming and esports content. For such an application, a flexible and iterative approach like Agile, with specific attention to Continuous Deployment, would be well-suited.
The reason Agile is best for G Stream application development because Agile development methodology can help reduce the risks associated with complex projects and it ensures that the development team completes the project on given time and with the given budget.

Here's some more reasons:

**1. Frequent Feature Delivery:** Agile emphasizes delivering small, incremental updates frequently. This aligns with the need to continuously introduce new features, enhancements, and improvements in real-time to keep viewers and streamers engaged on platforms like G Stream.

**2. Suitable for Bigger and Complex Project:** This is particularly important for large-scale projects, which often involve different priorities and needs. Because Agile methodologies allow teams to act quickly and easily to changes and new requirements.

**3. User-Centric Development:** Agile model development is mostly a user-centric development. Regular feedback from streamers and viewers can help shape the platform to better meet their needs and preferences, which is crucial for the success of a live streaming service.

**4. Flexibility and Adaptability:** Agile model is flexible and has great adaptability because allows you to adapt quickly to market shifts and emerging technologies, ensuring that the platform remains competitive.

**5. Continuous Improvements:** Agile encourages continuous improvement through retrospectives. The development team can reflect on their processes and practices to make ongoing refinements and optimizations.

**6. Continuous Integration and Deployment:** Agile enable automated testing and deployment, reducing manual errors and speeding up the release process. For a platform that needs to be constantly updated and improved, continuous integration and deployment is critical.

**7. Time-to-Market**: Agile, along with, Continuous Integration and Deployment allows for faster time-to-market. New features and bug fixes can be deployed quickly and efficiently.

**8. Risk Management:** Agile ensures early detection of risks through iterative development and continuous testing. This is important for a platform like G Stream, which requires high levels of reliability and stability.

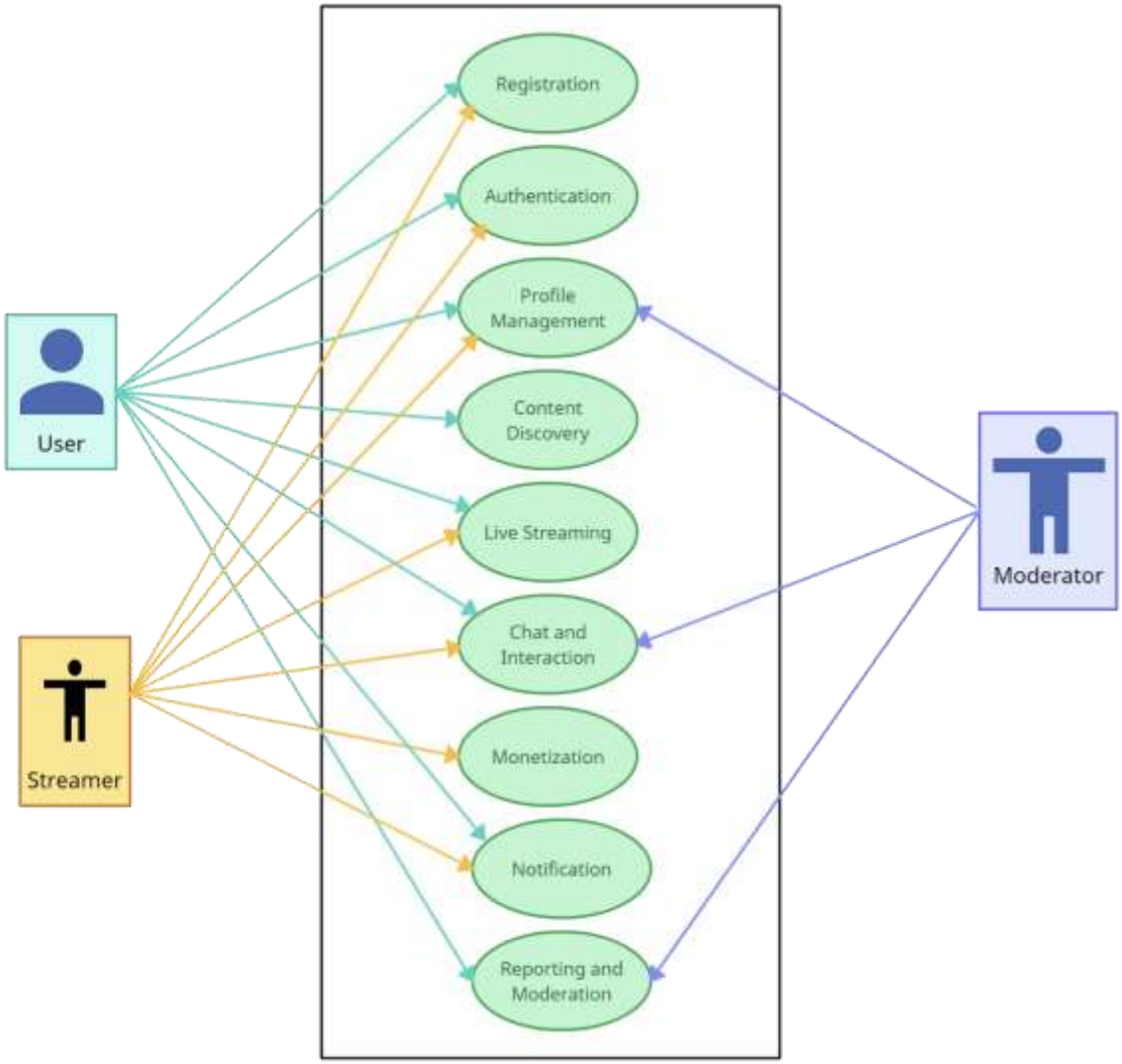
**9. Transparency and Communication:** Agile methodology ensures great transparency and communication with customers. It promotes transparency through regular ceremonies like stand-up meetings, sprint reviews, and retrospectives.

**10. Cross-Functional Teams:** Agile teams are typically cross-functional, including members with skills in development, testing, design, and more.

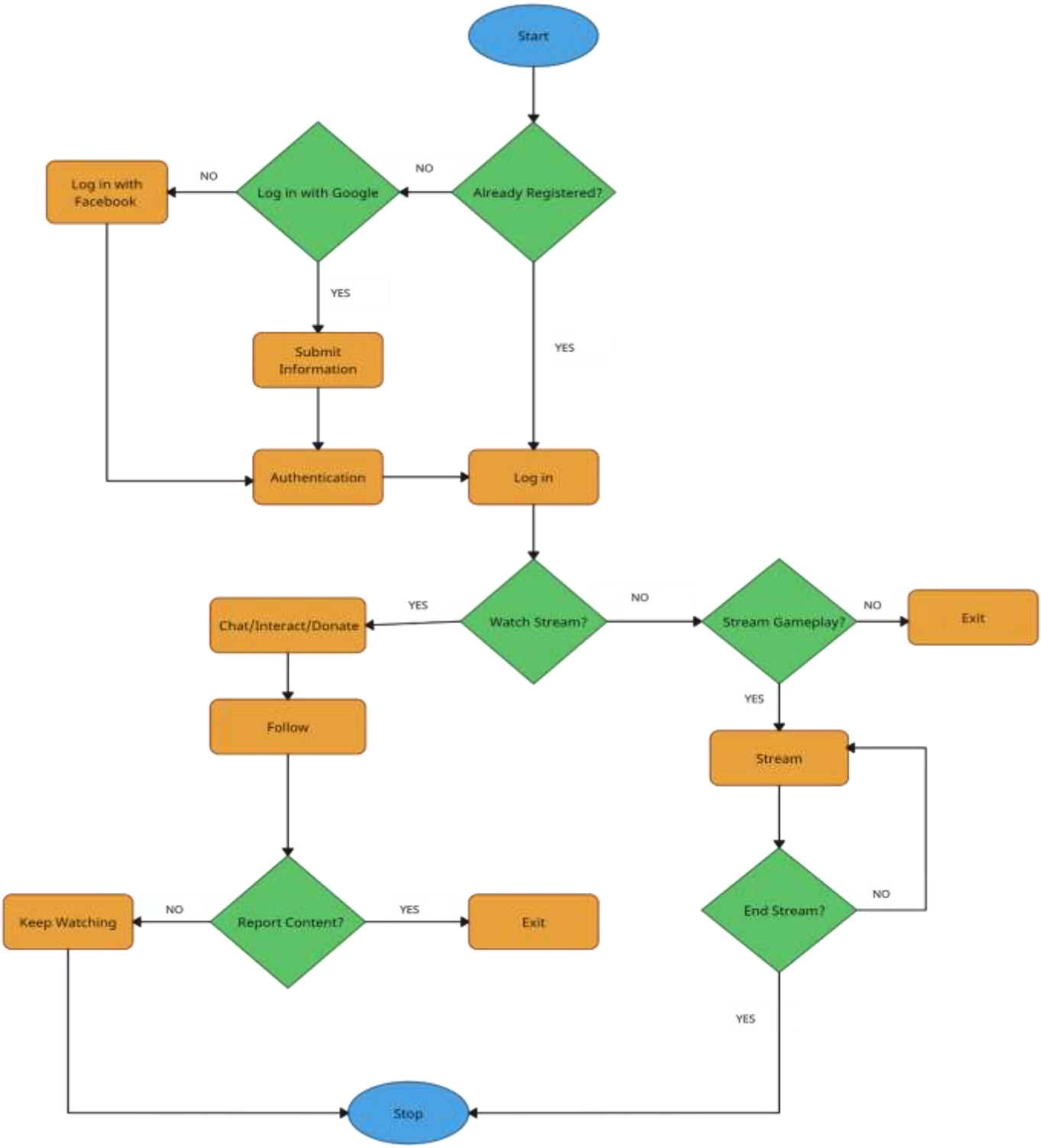
## 5. System Models/Diagram

Use Case Diagrams, UML diagram and Entity-Relationship Diagrams are displayed in this section.

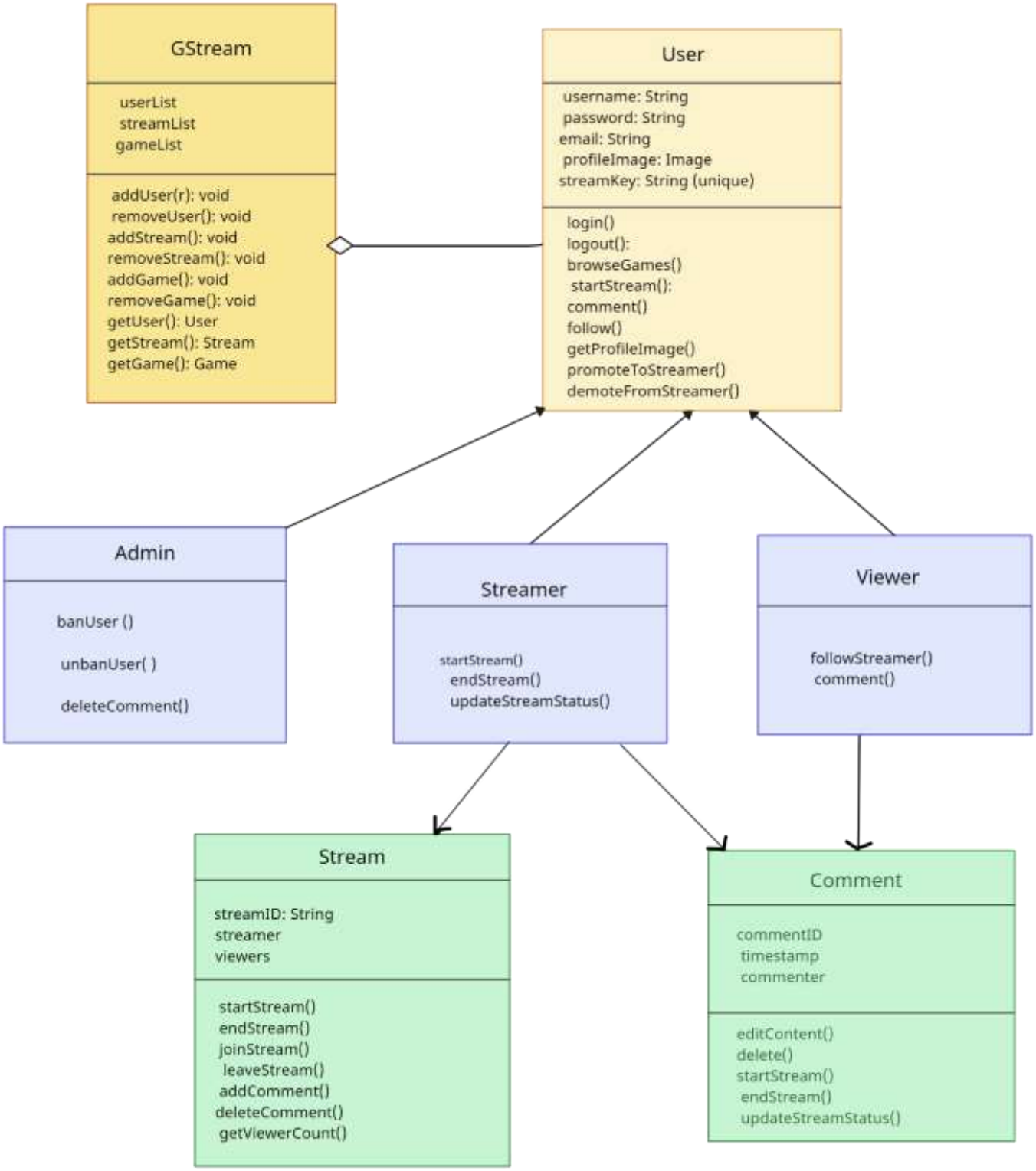
**USE CASE DIAGRAM**



# FLOW CHART


Start
Already Registered?
NO
Log in with Google
NO
Log in with Facebook
YES
Submit Information
YES
Authentication
Log in
Watch Stream?
YES
Chat/Interact/Donate
NO
Stream Gameplay?
NO
Exit
YES
Follow
Stream
Report Content?
NO
Keep Watching
YES
Exit
End Stream?
NO
YES
Stop

# UML DIAGRAM


GStream
userList
streamList
gameList
addUser(r): void
removeUser(): void
addStream(): void
removeStream(): void
addGame(): void
removeGame(): void
getUser(): User
getStream(): Stream
getGame(): Game
User
username: String
password: String
email: String
profileImage: Image
streamKey: String (unique)
login()
logout():
browseGames()
startStream():
comment()
follow()
getProfileImage()
promoteToStreamer()
demoteFromStreamer()
Admin
banUser ()
unbanUser( )
deleteComment()
Streamer
startStream()
endStream()
updateStreamStatus()
Viewer
followStreamer()
comment()
Stream
streamID: String
streamer
viewers
startStream()
endStream()
joinStream()
leaveStream()
addComment()
deleteComment()
getViewerCount()
Comment
commentID
timestamp
commenter
editContent()
delete()
startStream()
endStream()
updateStreamStatus()

# 6. Test and Validation

## 6.1 Testing Approaches

There are 2 resting approaches **Manual** and **Automated.**

## Manual Testing

### 6.1.1 Exploratory Approach

Exploratory Testing is an unscripted, manual software testing type where testers examine the system with no pre-established test cases and no previous exposure to the system. Instead of following a strict test plan, they jump straight to testing and make spontaneous decisions about what to test on the fly.
During exploratory testing, testers explore the software by interacting with it, trying out different features, then observing its behavior. They may even intentionally break the software, input unexpected data, or explore edge cases to uncover potential issues.
As long as testers get to understand the system's workings and can suggest strategies to test the system more methodically, they have accomplished the task.
Exploratory testing plays a crucial role in software testing, especially Agile projects, where automation testing has taken precedence. Automation testing admittedly brings a wide range of benefits to QA teams, but it is impossible to ignore exploratory testing and more broadly manual testing.

### 6.1.2 Box Approach

Box testing approach usually has 2 divisions black-box and white-box. A hybrid approach called grey-box testing may also be applied to software testing methodology

#### Black-Box Testing

In black box testing, the testing team analyzes the workings of an application without first having an extensive understanding of its internal structure and design. During testing, the input value is simply compared with the output value. Due to its nature, black box testing is sometimes called specification-based testing, closed box testing, or opaque box testing.

Black box testing mainly focuses on the comprehensive examination of application functionality. It is closely related to behavioral testing; however, behavioral testers may have limited knowledge of internal application workings.

#### White-Box Testing

The software application's internal coding, design, and structure are examined in white box testing to verify data flow from input to output. White box testing is leveraged to improve design, usability,

and application security. The other names for this methodology include code-based testing, glass box testing, open box testing, clear box testing, and transparent box testing.

**Grey-Box Testing**

The gray box testing is a combination of both white and black box testing approach. It allows testers to approach a software product from the point of view of a user, while also allowing them to access its internal code.
The gray-box testing involves inputs and outputs of a program for the testing purpose but test design is tested by using the information about the code. The tester aims to find all the possible code and functioning issues by using this method. At this stage a specialist is able to test the end-to-end functions.

## 6.2 Reasons of choosing Grey-Box Testing Approach

For a game streaming app, a combination of black box, white box, and grey box testing is often beneficial and Grey-box testing uses properties of both black and white box testing. Grey box testing provides flexibility, allowing testers to strike a balance between user-centric testing and understanding internal workings where needed.

In **Black box** testing we don't have to know the internal integration and we need only a little knowledge about the programming code implementation. As a result, there is limited scope – it may not include all of the potential defects, it will only test the parts of the software that are visible to the tester. There's also lack of technical insight and testers may not have the technical insight needed to identify certain types of defects.
On the other hand, **White-Box** testing time-consuming because it requires an understanding of the software's internal workings. It is expensive, as it requires the use of highly skilled testers that can understand the software's internal logic. It has limited scope and it may not include all potential defects. It Can be overly technical where it requires a high degree of technical expertise, which might be challenging for non-technical stakeholders to understand.

As **Grey-Box** testing is a combination of black box and white box testing, it provides the best of both worlds i.e. benefits of both the testing techniques. It is less time consuming and less technical than white-Box testing. It also gives more technical insight than Black-Box testing which helps to identify certain types of defects and errors. Some other advantage of using Grey-Box testing are,

i) For grey box testing, functional specifications and other design documents are used. It does not need the use of the source code which helps in keeping the source code safe from any disruptive changes.

ii) It helps in keeping testers and developers separate, which reduces any disagreement between them.

iii) Even with a partial understanding of the code, testers conduct grey box testing from the end user's perspective. This helps in identifying any issues that the developers might have missed during unit testing.

iv) It results in the instant fixing of the issues as a tester can change the partially available code to check for the results.

v) Even without high-level programming skills, the testers can perform this testing.

vi) It is platform and language-independent.

## 6.3 Testing Levels

There are 4 testing levels in software testing.


**i) Unit Testing:** Unit testing is done by the developer himself. After he has written code for a feature, he will ensure it is working fine. Here every module of the application gets tested respectively.

**ii) Integration Testing:** Integration testing is done by the developer. Integration testing is a testing technique where two or more independent components are tested together. Here test cases are written to ensure the data flowing between them is correct.

**iii) System Testing:** System testing is done by the tester where the entire application is tested as a single unit. Hence, system testing test cases are also performance test cases, load testing, and stress testing test cases.

**iv) Acceptance Testing:** Acceptance testing is done by the client where he evaluates whether the product is made by the requirement he listed out. Acceptance testing is done at the UAT server where a well-tested product is deployed by the team for the client's reference so he can track ongoing changes in the project

## 6.4 Testing types, techniques and tactics

- **Installation testing:** Ensures the software installs, upgrades, and uninstalls correctly, validating the installation process.
- **Compatibility testing**: Verifies that the software functions as intended across different environments, devices, and configurations.

- **Smoke and sanity testing:** Smoke testing checks basic functionalities to ensure stability, while sanity testing verifies specific features or components after changes.
- **Regression testing:** Detects unintended side effects of code changes by verifying that existing functionalities still work as expected.
- **Acceptance testing**: Validates that the software meets specified requirements and is ready for release based on user acceptance criteria.
- **Alpha testing**: Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site.
- **Beta testing**: Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Involves external users testing the software in a real-world environment before the official release.
- **Functional vs non-functional testing**: Functional testing assesses specific features, while non-functional testing evaluates aspects like performance, security, and usability.
- **Continuous testing:** Continuous testing includes the validation of both functional requirements and non-functional requirements. Incorporates automated testing into the development pipeline to ensure code quality throughout the software development lifecycle.
- **Destructive testing:** Assesses how the system behaves under extreme conditions, often by intentionally stressing or breaking it.
- **Software performance testing:** Evaluates the responsiveness, scalability, and stability of the software under varying conditions.
- **Usability testing:** Examines the user interface and overall user experience to ensure the software is easy to use.
- **Accessibility testing:** Verifies that the software is usable by individuals with disabilities and complies with accessibility standards.
- **Security testing:** Identifies vulnerabilities and ensures the software is resistant to unauthorized access, attacks, and data breaches.
- **Development testing:** Performed by developers during the development process to catch and fix issues early.
- **Concurrent testing**: Assesses the software's ability to handle multiple users or transactions simultaneously.
- **Conformance testing or type testing**: Ensures that the software complies with specified standards or regulations.
- **Output comparison testing**: Compares the actual output of the software with the expected output.
- **Property testing:** Focuses on verifying specific properties or characteristics of the software.
- **VCR testing:** A form of automated testing that records and replays HTTP interactions to ensure consistent behavior over time.

## 6.5 Testing Process

### Agile

Agile testing is a software testing methodology aligned with the principles of agile software development. Agile development emphasizes collaboration, flexibility, and continuous iteration, and it is designed to support these principles by providing a flexible and adaptable approach to testing.

It focuses on early and continuous testing throughout the development process, emphasizing automated testing and collaboration between testers and developers. Agile testing aims to quickly identify and address defects to deliver high-quality software that meets the customer's needs. Agile testing process emphasis on collaboration and communication may be advantageous for grey box testing.

**The main testing activities in agile are:**

**Requirement analysis:** Testers work with business analysts and product owners to understand the requirements of the new feature.

**Test design:** Testers design the test cases for the new feature.

**Test execution**: Testers execute the test cases and report any bugs.

**Defect management**: Testers work with developers to fix the defects.

**Release management:** Testers help to plan and execute the release of the new feature.

## 6.6 Hierarchy of Testing Difficulty

The hierarchy of testing difficulty classes based on the number of test cases required to construct a complete test suite reflects different levels of complexity and challenges in achieving thorough testing. Here's how each class is typically used and their hierarchy in terms of testing difficulty:

**1. Class I:** There exists a finite complete test suite.

**Use Case:** Class I is applicable when it is feasible to define a finite set of test cases that collectively provide complete coverage of the system under test. It is commonly employed in scenarios where system behavior can be thoroughly captured with a manageable number of test cases.

**Hierarchy:** This is the least challenging class in terms of testing difficulty because it suggests that a finite set of test cases can ensure complete coverage, simplifying test planning and execution.

**2. Class II:** Any partial distinguishing rate can be reached with a finite test suite.

**Use Case**: Class II is useful when it is acceptable to have an incomplete capability to distinguish correct systems from incorrect systems, as long as it is achieved with a finite set of test cases. It is often applied in situations where achieving 100% coverage is impractical, and partial coverage is deemed sufficient.

**Hierarchy:** Slightly more challenging than Class I as it allows for incomplete coverage but still requires a finite set of test cases, demanding careful consideration of which scenarios to prioritize.

**3. Class III**: There exists a countable complete test suite.

**Use Case:** Class III is relevant when a complete test suite exists, but this suite may involve a countable (potentially infinite) number of test cases. It is frequently employed in systems where exhaustive coverage is necessary, but a finite countable set is more practical than an uncountable one.

**Hierarchy:** Higher in difficulty compared to Class I and Class II due to the potential countability of the test suite, requiring meticulous management of an extensive but countable set of test cases.

**4. Class IV:** There exists a complete test suite.

**Use Case:** Class IV is used when a complete test suite exists, but the nature of completeness (finite or countable) is not specified. It is often chosen when the nature of the system under test allows for exhaustive testing, regardless of the potential infiniteness of test cases.

**Hierarchy:** Higher in difficulty compared to Class III as it does not specify whether the complete test suite is finite or countable, introducing additional uncertainty and complexity in test planning.

**5. Class V:** All cases.

**Use Case:** Class V is the most demanding and is applicable when testing all possible cases is necessary to achieve completeness. It is typically reserved for critical systems where every conceivable scenario must be evaluated.

**Hierarchy:** Highest in difficulty, as it requires testing every possible case, potentially implying an infinite number of test cases. Achieving and managing such comprehensive coverage is a monumental challenge, often requiring advanced testing strategies and tools.

## 6.7. Complete Test Plan

Test plan is a document that describes the scope, approach, resources, and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, the degree of tester independence, the test environment, the test design techniques, and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.

Agile Test Plan Outline:

**1. Testing Strategy:**

- Agile methodologies being used (e.g., Scrum, Kanban).
- Iterative testing approach.
- Collaboration with development and other teams.

**2. Test Levels:**

- Unit testing, integration testing, system testing, and acceptance testing.

**3. Test Types:**

- Functional testing, performance testing, usability testing, security testing, and compatibility testing.

**4. Testing Environment:**

- Platforms (iOS, Android, Web).
- Supported devices and browsers.
- Network conditions (3G, 4G, Wi-Fi).

**5. Test Data:**

- Types of games to be tested.
- Sample streaming content.
- User profiles for testing.

**6. Performance Testing:**

- Load testing to simulate multiple users.
- Bandwidth testing for streaming quality.
- Stress testing under varying network conditions.

**7. Usability Testing:**

- Evaluate the user interface for intuitiveness.
- Test user interactions during live streaming.

**8. Security Testing:**

- Secure transmission of gaming data.
- Authentication and authorization testing.
- Encryption of sensitive information.

**9. Compatibility Testing:**

- Test on different devices (phones, tablets, computers).
- Ensure compatibility with popular browsers.

**10. Acceptance Criteria:**

- Define criteria for each user story or feature.
- Align acceptance criteria with user expectations
-

## 6.8 Test Cases

Test cases define how to test a system, software or an application. A test case is a singular set of actions or instructions for a tester to perform that validates a specific aspect of a product or application functionality. If the test fails, the result might be a software defect that the organization can triage. These the test cases for G stream application.

**1. User Authentication Test Case:**

- Create a new account with valid credentials.
- Check that users receive an error message if they try to create an account with an existing email address.
- Confirm that users can log in using valid credentials.
- Verify that users are logged out successfully when they choose to log out.

**2. Game Selection Test Case:**

- Ensure that the app displays a list of available games.
- Confirm that users can search for specific games.
- Verify that game information (title, description, cover image) is accurate.

**3. Streaming Functionality Test Case:**

- Check that users can select a game to start streaming.
- Verify that the selected game starts streaming within an acceptable time frame.
- Test the ability to pause and resume a streaming session.

**4. Quality of Streaming Test Case:**

- Check and evaluate the streaming quality under optimal network conditions.
- Test the app's behavior and streaming quality under low bandwidth conditions.

**5. User Interface Test Case:**

- Confirm that the user interface is intuitive for navigating through the list of games.
- Verify that the controls for pausing, resuming, and navigating within a game are user-friendly.

**6. Performance Test Case:**

- Test the app's performance under various load conditions, simulating multiple users accessing the platform simultaneously.
- Test the app's responsiveness during gameplay and streaming.

**7. Social Integration Test Case:**

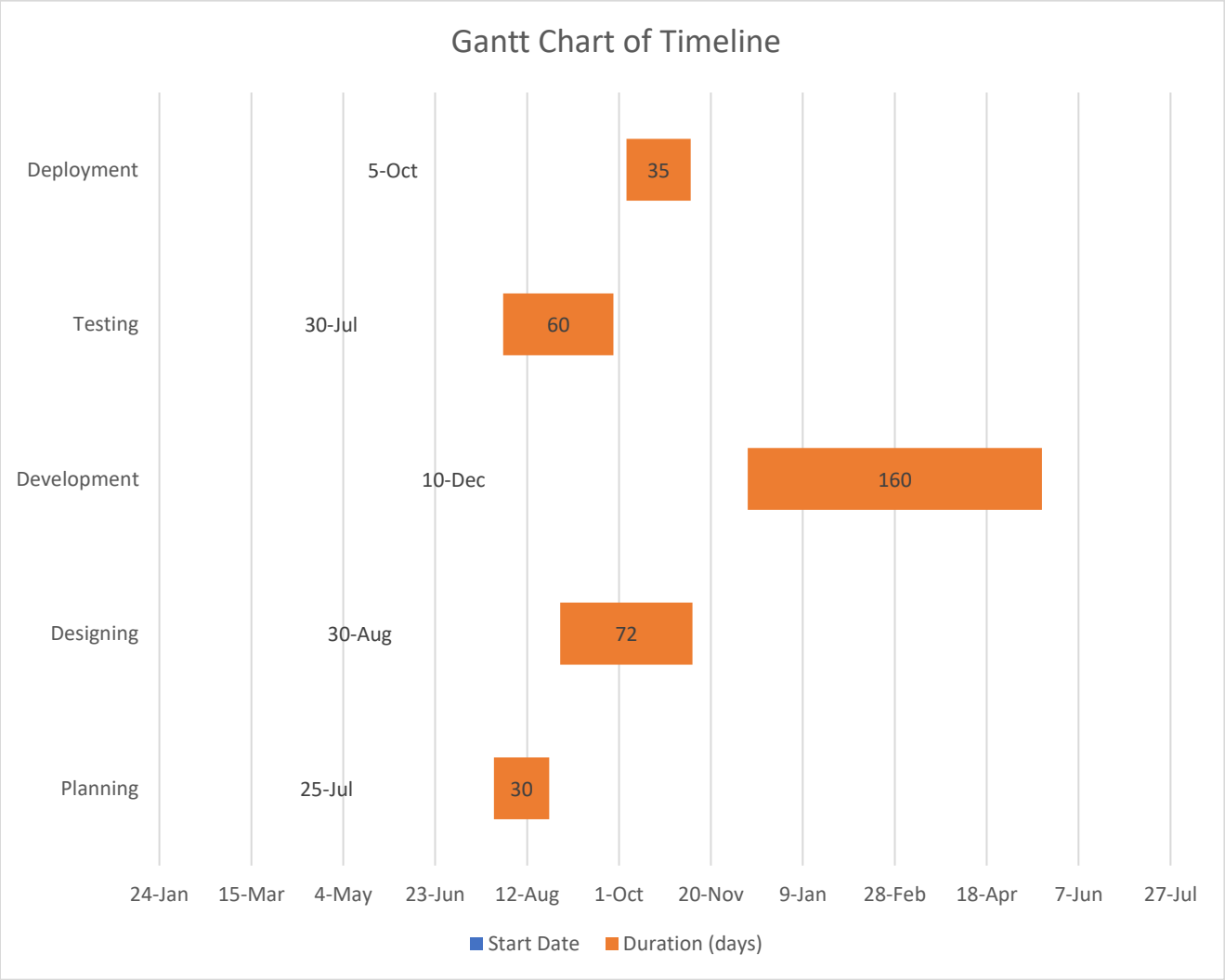
- Test the integration with social media platforms for sharing gaming activities.

**8. Notifications Test Case:**

- Confirm that users receive appropriate notifications for important events such as new game release.

# 7. Project Scheduling and Gantt Chart

The project will be divided into phases: Planning, Design, Development, Testing, and Deployment.



## 8. Payment Terms

We propose the following payment terms:

| | |
|---|---|
| Paid on acceptance of this proposal. | 10% (10%) |
| Paid on signing of our Application development agreement. | 40% (50%) |
| Paid at 70% Application Demonstration. | 25% (75%) |
| Paid at completion the Application. | 25% (100%) |

## 9. Contact

By Phone:
**+8801828867270**

Contact Us by Email
**HDsector@gmail.com**

On our website
**www.hdsector.bd**

**Agreement Signed By:**

…… ……

Client Signature

Authority Signature
Hirankur Dewan
Chief Executive Officer
HD SECTOR