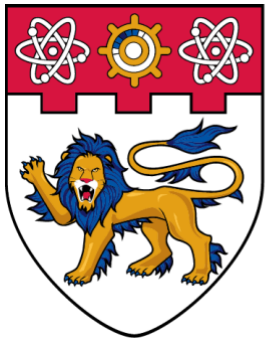

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3003 Software System Analysis & Design

Lab 2 Deliverable: Architecture Design

Good Game Well Played (GGWP)

SS2

Leroy Tang Zi Wei

Alson Kong

Karen Wong

Lim Wei Cong

Sam Jian Shen

Cole Ting-Chun Kuo

Zane Ho

Norman Fung

Jing Herng Pow

Table of Contents

Overview of report

In this report, it will cover the chosen candidate architecture GGWP application will be based on and the rationale for using them. As well as the interface designs for the various subsystems – mainly game client, API server and management subsystems.

The main software quality attributes we took into consideration when deciding on the architecture design includes:

- **Flexibility:** The ability to add/edit new components
- **Integrity:** Ability to protect against unauthorize use/ data loss
- **Reliability:** Probability of system failure
- **Testability:** Ability to verify correct implementation

These were chosen as the main focus of this software because flexibility allows new components to be added easily and maintained through easy reusing, testing and deployment. Integrity and reliability are important to ensure satisfactory user experience by ensuring there is no unauthorized access or data lost as well as service will be available as much as possible.

With those quality attributes in consideration, the candidate architecture for GGWP includes:

- Communicating Processes - **Client & Server** (Overall)
- Event Systems - **Implicit invocation** (Game Client subsystem)
- **Main Program and subroutine** (Management subsystem)
- **Layered System** (API server)

The main architecture GGWP will follow is the client and server structure while individual subsystems within the software will have its own architecture style based on what is more relevant and suitable for its functionality. More elaboration will be provided in the report below.

Rationale for Selected Architecture

As mentioned, GGWP software will consist of the following architecture designs: Client and server, event system, main program and subroutine as well as layered system. The following table give an overview of the pros and cons of the different designs against selected software attributes.

Table 1 - Quality attributes of various architecture design

	Client and Server	Event System	Main Program & Subroutine	Layered System
Flexibility	+	+	-	+
Integrity	+	+	-	-
Reliability	+	-	+	-
Testability	+	+	-	+

Client and Server

Client and Server architecture is the most attractive candidate architecture to use among the identified architecture. Hence, it will be used for the overall communication across our subsystems. The below table will break down the reasoning by quality attributes.

Table 2 - Client and Server reasoning by quality attributes

Quality Attributes	Description
Flexibility	The client and server architecture allow client and server components to be changed independent of one another. The server can be updated to provide new functionalities without affecting the client's availability. This allows the client and server to remain decoupled while allowing for the ability to add capabilities easily.
Integrity	The client and server architecture allow client and server components to maintain integrity by protecting the medium which the data is message passed via. The most common medium is over the HTTP protocol, which can be encrypted easily without much changes on client and server via the encrypted and secure HTTPs protocol.
Reliability	The client and server architecture allow reliability to be achieved by adding proxies to the network messages. The allows messages to be passed successfully as long as the identity of the server is not changed.

	An example of it is the use of load balancing, which client requests can be served by different server instances in a round robin manner and the client is not aware of which instances serves its request.
Testability	The client and server architecture allow the client and server to be tested separately, as both components and independent of one another, except during message sending. This can allow for easy testing on the server by creating messages from a mock client and can easily test clients in a test environment via automated user interaction tests and/or white box testing in code.

Event System

Event System will be used in game client subsystem as the event system offers the most flexibility and does not have significant performance overhead.

Table 2 – Event system reasoning by quality attributes

Quality Attributes	Description
Flexibility	The use of event systems often propagates event signals which contain data about the event that has happened. The use of an implicit invocation event system allows for a publish subscribe mechanism, this allows for flexibility as new subscribers to the event can do so without affecting the other subscribers. Publishers can also add new capabilities to events without affecting subscribers that are listening to existing events.
Integrity	The use of event systems does guarantee integrity. Once an event occurs, the event system will propagate the event to its listeners. It will be difficult to have integrity issues in an event system unless there are data corruption issues in the runtime environment.
Reliability	The use of event systems does not guarantee reliability. As event systems can only be successful if the event handler is available and working properly, once the event handler is not available, the event system would not be able to perform implicit or explicit invocation.
Testability	The implementation of event systems is easy to test as event subscribers and event triggers can easily be changed for use in a testing environment.

Main program and Subroutine

Main program and Subroutine will be used in management subsystem.

Table 3 – Main program and Subroutine reasoning by quality attributes

Quality Attributes	Description
Flexibility	The main program and subroutine do not offer flexibility as they are tightly coupled components. The main program would have to know explicitly which subroutine to call.
Integrity	The use of main program and subroutine makes it hard for integrity and security issues to occur, as the capability for the main program and subroutine often occurs in the form of a function call in the runtime of the main program. This makes it difficult for data loss and unauthorized changes to happen unless there are inherent security issues or loopholes in the runtime.
Reliability	The use of main program and subroutine makes it easy to achieve reliability of the system, especially if all subroutines are local to the program. This is especially the case if proper code review is done properly to ensure references to subroutines are correct. The use of static-typed languages may be able to help as this allows references to subroutines to be checked during compile time.
Testability	The use of main program and subroutine makes systems hard to test, subroutines often call subroutines and handle the data returned in some forms, but this can be hard to test as the logic is internal to the main program's sequential execution.

Layered System

Layered System will be used in API server.

Table 4 – Layered System reasoning by quality attributes

Quality Attributes	Description
Flexibility	The use of layered architecture allows control logic to be added or updated easily as the control logic for layered architecture is often isolated to the business logic layer or controller layer.
Integrity	The use of layered architecture requires other integrity mechanisms to be set up such that, if the different layers are deployed across different devices/machines, the integrity of data is not compromised. In the case of a local setup, the integrity issue will not be significant and likely will occur only on critical events such as network outage.
Reliability	The use of layered architecture will be reliable if the layers are deployed in the same device/environment. In the case that any of the layers are deployed separately, the reliability of the layered architecture will be similar to that of the client and server.
Testability	The use of layered architecture separates the user interface from control logic and the control logic from data logic which enables testing to be done easier since each layer can be tested independently.

Overall Architecture Design

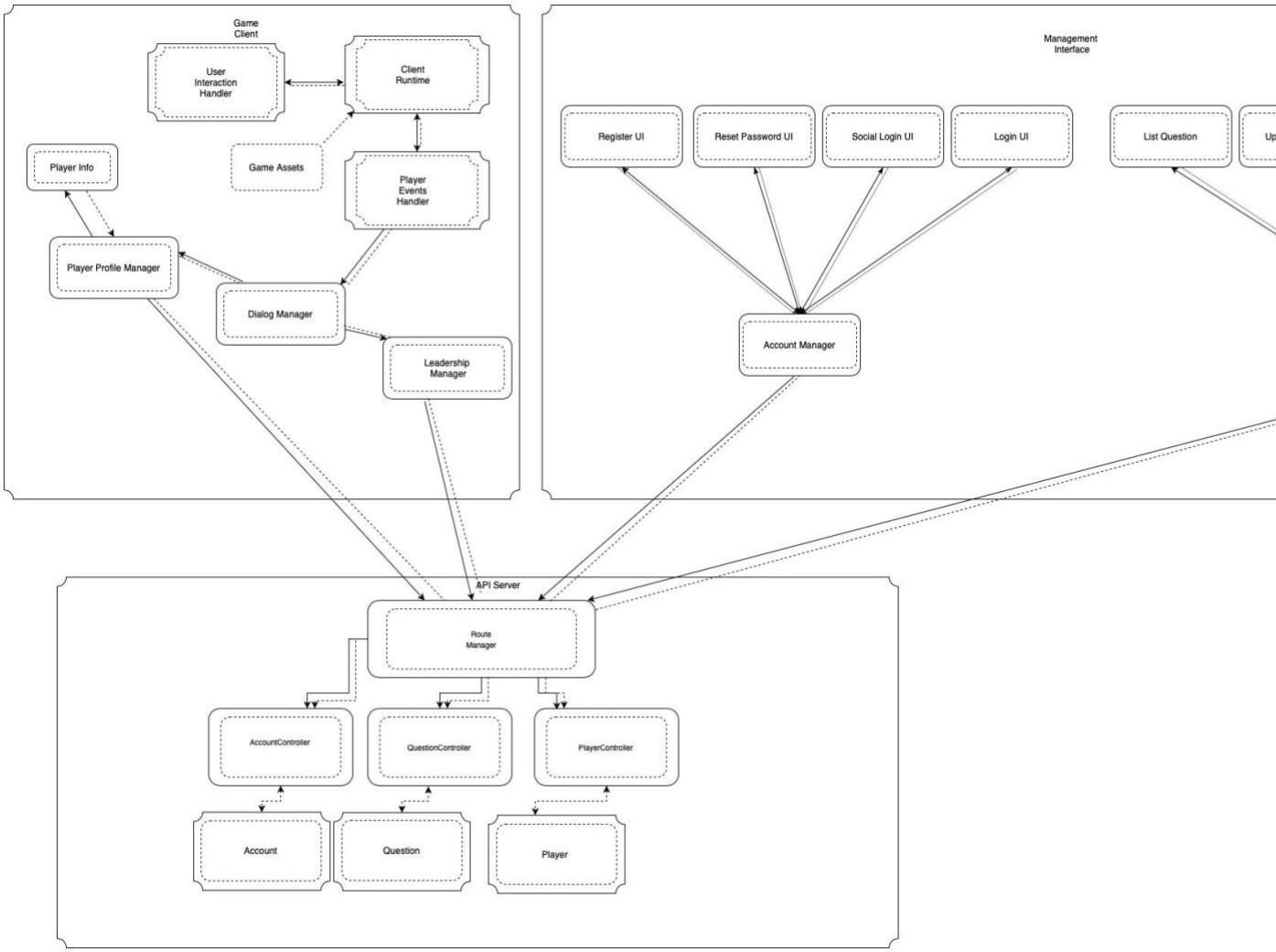

Figure 1 - GGWP overall architecture design

Individual subsystem interface design

In this section below, it will cover the public methods for individual subsystem as well as the architecture diagram.

Game client interface design

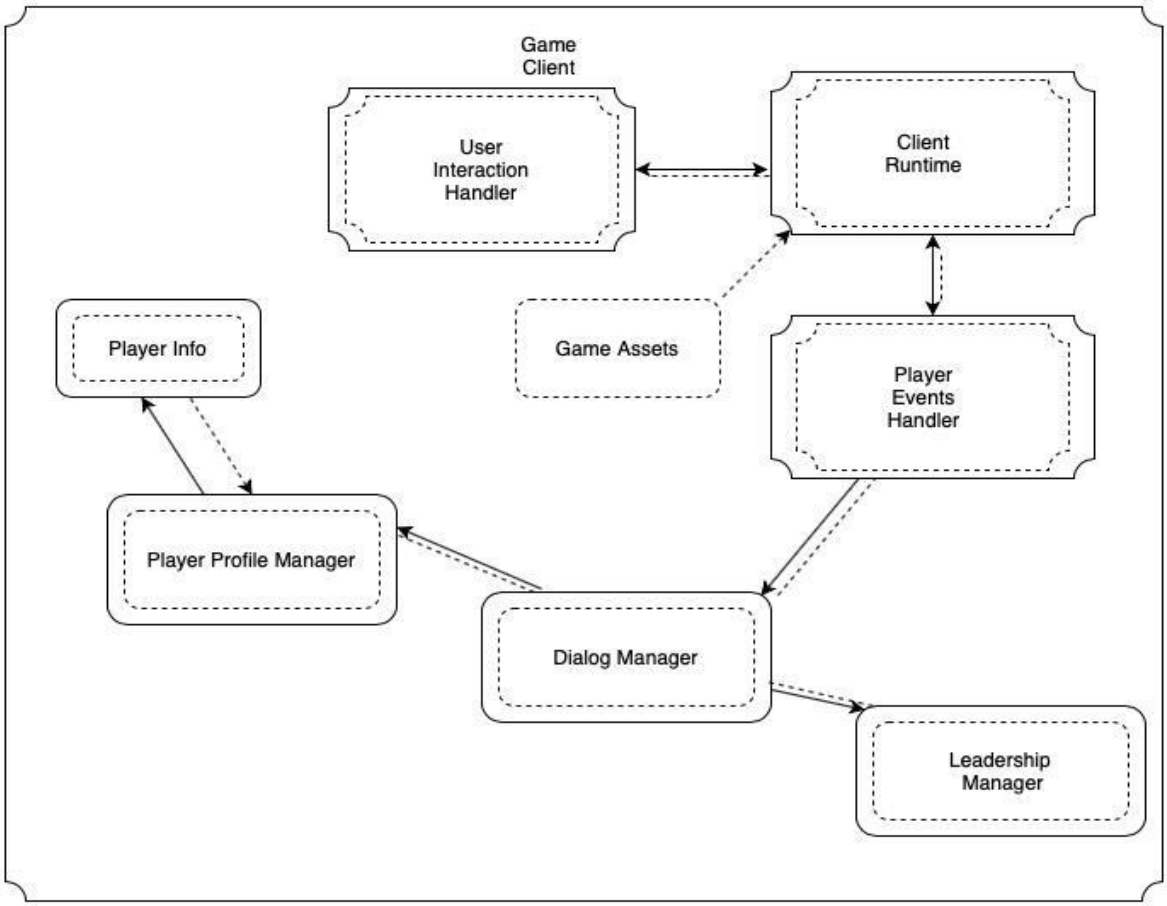

Figure 2 - game client's software architecture visual notation

As the game client subsystem does not have an incoming data flow, the game client subsystem will not have any public methods. The game client subsystem is a client in our client and server architecture.

The game client subsystem will be calling the following public methods on the API Server:

- **Player**
  - All
- **Question**
  - Get questions by subject and difficulty

Management Subsystem

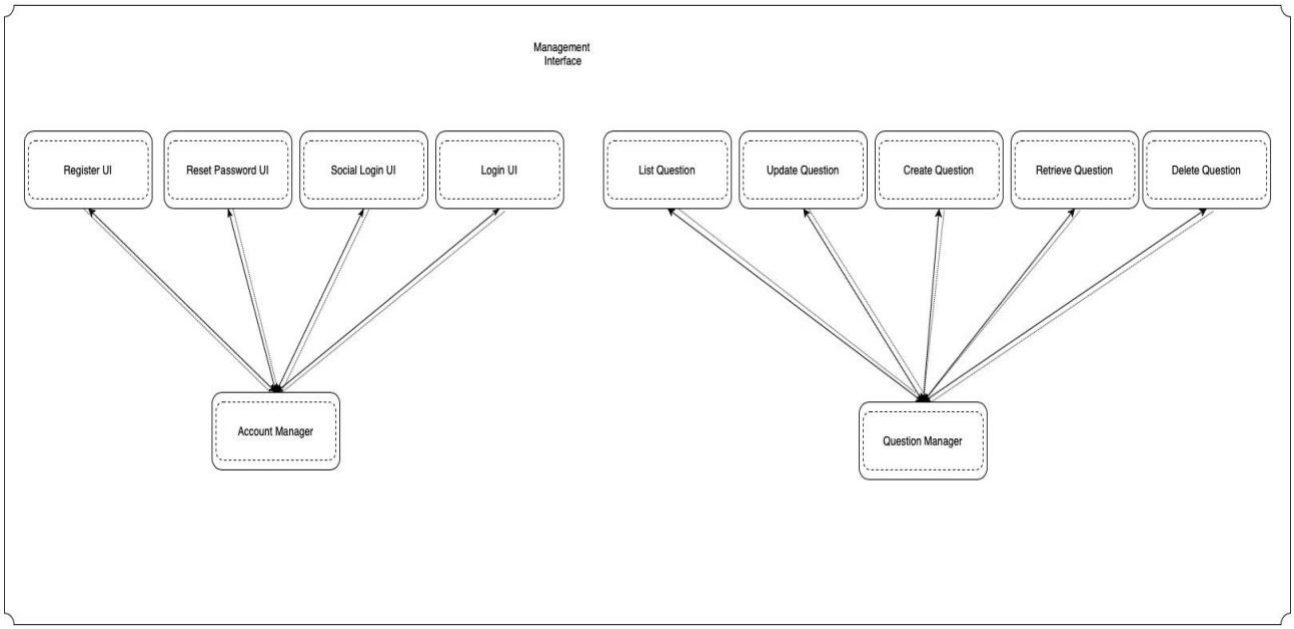

Figure 3 - Management Subsystem's software architecture visual notation

As the management subsystem does not have any incoming data flow, the management subsystem does not have any public methods. The management subsystem is a client in our client and server architecture.

The management interface will be calling the following public methods:

- **Account Manager**
  - Login
  - Register
  - Reset Password
  - Social Login
- **Question Manager**
  - Post Question
  - Update Question
  - Delete Question
  - Get Question
  - Get Question by ID

API Server

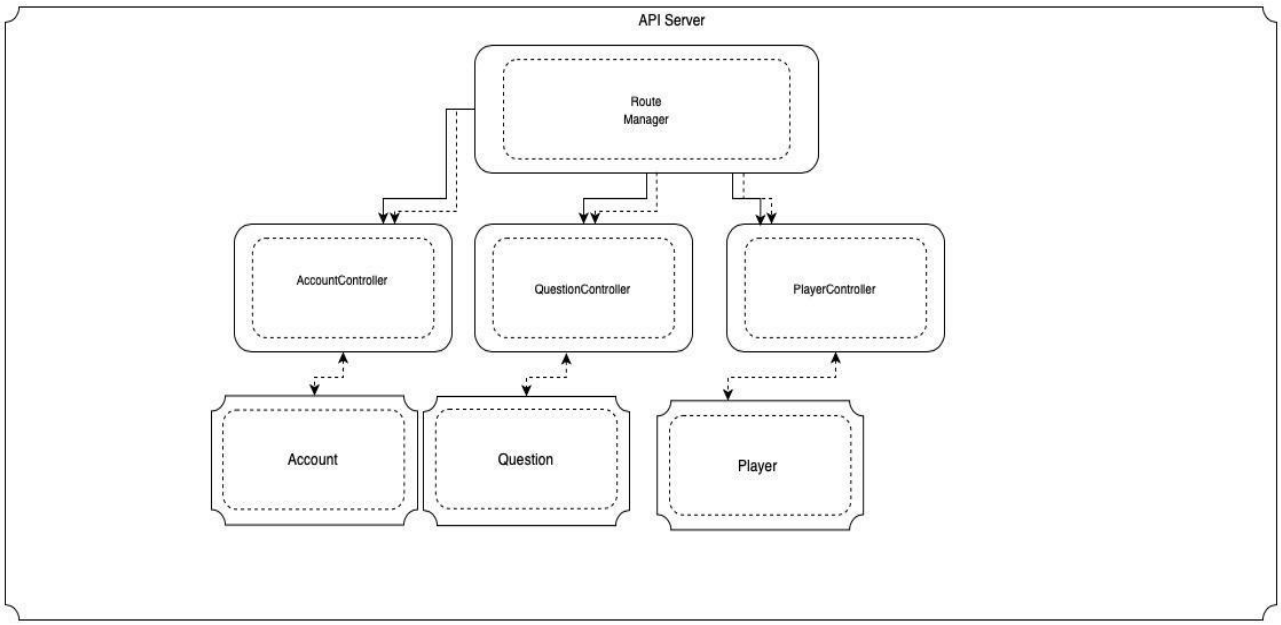

Figure 4 - API server's software architecture visual notation

API Server exposes the following public methods to the game client and management subsystem. This is a description of the public methods, grouped by the different entities for the API Server.

- **Player**
    - Get leadership board by subject
        - Parameters: subject name
        - Usage Scenarios: When the user enters the classroom and talks to the NPC.
    - Get global leadership board
        - Usage Scenarios: When the user enters the main map and talks to the NPC.
    - Get player's profile
        - Parameters: username
        - Usage Scenarios: When the user intends to check his/her profile.
    - Update player's profile
        - Parameters: Updated level, updated progress, subject name, player's name
        - Usage Scenarios: When the user intends to update his/her profile.

- **Account**
  - Login
    - Parameters: username, password
    - Usage Scenarios: When the user intends to login to the system.
  - Register
    - Parameters: username, password, email address
    - Usage Scenarios: When the user enters the system for the first time and intends to register an account for the game.
  - Reset Password
    - Parameters: username
    - Usage Scenarios: When the user forgets his/her password and intends to reset the password.
  - Social Login
    - Parameters: email address, password for social account
    - Usage Scenarios: When the user intends to login with a social account.
- **Question**
  - Post Question
    - Parameters: question, answer, correct answer, additional information, question difficulty
    - Usage Scenarios: When the user intends to create a new question and submit to the system.
  - Update Question
    - Parameters: question ID, question, answer, correct answer, additional information, question difficulty
    - Usage Scenarios: When the user intends to update the content of an existing question.
  - Delete Question
    - Parameters: question ID
    - Usage Scenarios: When the user intends to delete an existing question.
  - Get Question Lists
    - Parameters: subject
    - Usage Scenarios: When the user (teacher) intends to view the questions of a subject.

- Get Question by ID
  - Parameters: question ID
  - Usage Scenarios: When the user (teacher) intends to view the content of a specific question in the question list.
- Get Questions by Subject and Difficulty
  - Parameters: subject, difficulty
  - Usage Scenarios: When the user enters a classroom, this function is called to generate the question list for this user according to the subject and the user level in this subject.