CS 319 - Object-Oriented Software Engineering

System Design Report

Your Story

An Online Messaging Portal for Text Based RPG Games

Group 22

Ali GÜNEŞ

Cevat Bariş YILMAZ

Erin AVLLAZAGAJ

Ertuğrul AKAY

Course Instructor: Uğur DOĞRUSÖZ

Contents

Introduction

“Your Story” is a game which has completely different gameplay from the other games. is a text-based multiplayer role-playing game, which will be implemented in Java. The game will is designed for desktop computers. There are different scenarios and different kinds of characters in the game. The purpose of the game is having fun by chatting with the other players in order to develop a story and acting like a character you always wanted to be.

The detailed information about the content of the game will be presented in the overview section. This report contains detailed gameplay information and the use cases of the project in the upcoming sections. In the first section we are discussing the purpose of this system and what features will it bring better than any other similar implementation. The second following section is where we describe the goals of our design and all the tradeoffs and our picks. In this section we discuss six goals of our design based on the non-functional requirements. We will also discuss what our choices conflict with and why our decision was better that its counterpart. This section is then followed by the architecture of our software also. In this section we will be discussing about the components and the architecture type we chose as well as detailed description of all the components and their inner classes. All the schematics below are created with Visual Paradigm.

Purpose of the system

This system is created to bring together a group of story writers to develop a story their way and get to know other interesting people in the world. This game’s purpose is to help the players develop their imagination and collaborate to create a great story or even connect with other people sharing the same talent and interest.

This game is different from many similar game because the user is more valued to his writing and creating skills rather than just valued by how fast they click the mouse. Another difference to other games is that the ending of the game is not decided on when the time ends or the “castle is captured”, the ending is let free for the player to decide democratically. The same way is for starting the game or kicking out toxic players.

Design goals

Reliability

Our software is designed by keeping in mind that it shall not be crashed under any user's input. During the design we consider any possible state of our objects such that no conditions can lead to any serious crash. We use exception in any potential buggy piece of code. The exceptions will then reroute the program to normal operating way, sometimes without user's consent. The game will respond in determined time if the internet connection exists, if the objects are instantiated, if a lobby is currently in game, if user doesn't exist in database[1] or any other exceptional case that is determined in Analysis Report for each use case of the software. We determined the bugs that can be occurring and determined how can we handle these bugs. Most of the bugs that can occur are because of the game not actually being in real-time. Since the client will be refreshed every 5 seconds the lobbies might have started the game and still appear in the home page as lobbies waiting for players and many other scenarios. After we finish the project, we will possibly release a beta version to get a better idea of any unexpected bugs that might occur while running the program and fix them, this will increase the reliability of the software.

Robustness

Since our software is going to be written in Java we assume Robustness is a feature we get for free. Since Java runs on all the Operating Systems without crashing we are given a great head-start when choosing this language rather than C or C++. Since the program is online, the program should handle the bugs that occur due to loss of connectivity to the database. The client should be able to handle the connection problem and omit any changes if already logged in. Also, the program should wait for connection to the database to reestablish and disallow any change to the data the user wants to make. When reconnected the client will update its data to server's not vice-versa. If the user is not logged in the client can't let the player see the main view. It will notify the user when connection is established. It will be able to communicate using the JDBC which will take care of the protocol. On top of it is the library Erin had developed to simplify the connection.

Modifiability

Most of the system's functions are easily modifiable when the coupling of the subsystems are kept at minimum. In "Your Story", functions of a class do not enormously affect their sub or parent classes, as the data in the database must not be changed unwillingly. In order to do this, abstract classes will be used in both user interface and in the other layers. The close architecture also makes it easy to modify its parts such that the dependent classes will be easily modified once the layer they depent from is substituted or modified. This architecture type is discussed in more details in the second part of this paper.

Usability

Since the user interface is one of the most crucial factors for players of the Your Story, buttons and other interactable objects will be kept basic. Simple instructions will guide the players to do their aim. Rather than doing chain interactions, (like popping a page after a page in player's' interface, every single step of their actions will mostly be kept in a single page, and all of these steps will have an explanation written on them. With this method players will not be stucked to understand the program.

Portability

Portability is one of problems of software development, since today there are lots of different device types, operating system and versions , having a stable performance between all these environments for the softwares is not that easy. To partially overcome this situation, the development language will be Java which can be executed on different platforms. So that game will be playable for all operating systems that able to run JVM[2].

Another advantage of using Java is potential opportunity to port the game for Android environment. Since language used in Android applications is also Java, just by changing the first layer of our architecture, which contains the user interface classes, we can turn the game into an Android app. Moreover, since we using a closed architecture it will not cost us much engineering time.

Finally, using SQL to manage data in our database will let us to migrate any other relational database since it is the standard query language between relational database management systems.

Rapid developments

Rapid development is a key issue to our project. Since we need to release the final product by a specified deadline around beginning of January, before the end of Fall semester of the academic year 2016-2017 we need to be fast at developing and releasing this product. The time to write the code and debug possible bugs is approximately one month, that's why we might release a copy for beta testing by half of the development stage. To help us achieve this rapid development we will be using a MySQL driver for Java called JBDC topped by multi-purpose database communication library that Erin had written before for other applications. This design goal was more as a requirement rather than a choice, because it made us remove the Server written in Java which would be as a middleware between client and database and which would update the client in real time rather than make it query the database every 5 seconds.

Trade-offs

Portability VS Efficiency

To achieve the portability goals, we had to cut some of the functionality of the game. The first trade we made is using Java, which can not be converted to an 'executable-like' shape. Therefore all users have to setup JVM to their computer before playing the game. Also because of existence of a virtual environment between software and operating system, the execution time will be longer than compared to other languages. However since the game won't be that complex, this trade-off won't hurt us much.

The second trade is using a relational database system to manage database. While using a relational database lets us to migrate between different systems because of their standardised structure and language, SQL, we can move between different databases any time. However the downside of relational databases is their efficiency. Since getting the data as we want requires many usage of 'join' command which causes slower performance, relational databases are not fast as object oriented databases.

Rapid Development VS Functionality

By choosing a rapid development we lost a lot in functionality. The game was first decided to have a server and a client sitting in two different machines. The server and the client would communicate and share data with each other. This would make help us drastically reduce traffic in our database and even have the server and database run on different machine which would increase the computational power of both by many times. What made us retreat from this decision was the fact that the server and the client needed a protocol to communicate (so we needed to design a protocol) and also the socket programming is something new to the majority in our group, not to mention handling connection issue with keep-alive[3] packet transmission through time intervals and encrypting the traffic for secure data exchange. So now the client just queries the database every 5 seconds in a different working thread which updates all the dependant objects if any new data comes. The security of the communication is handled by the JDBC driver.

Definitions, acronyms and abbreviations

[1] Database: A system that is efficient for storing or retrieving data
Socket: A term used for all the network setup needed to
[2] JVM: Java Virtual Machine a virtual machine that converts its program instructions into native instruction of the processor it runs in.
[3] Keep-Alive: Network packets sent at a specified interval to let the server know the client is still connected

Software Architecture

Subsystem Decomposition

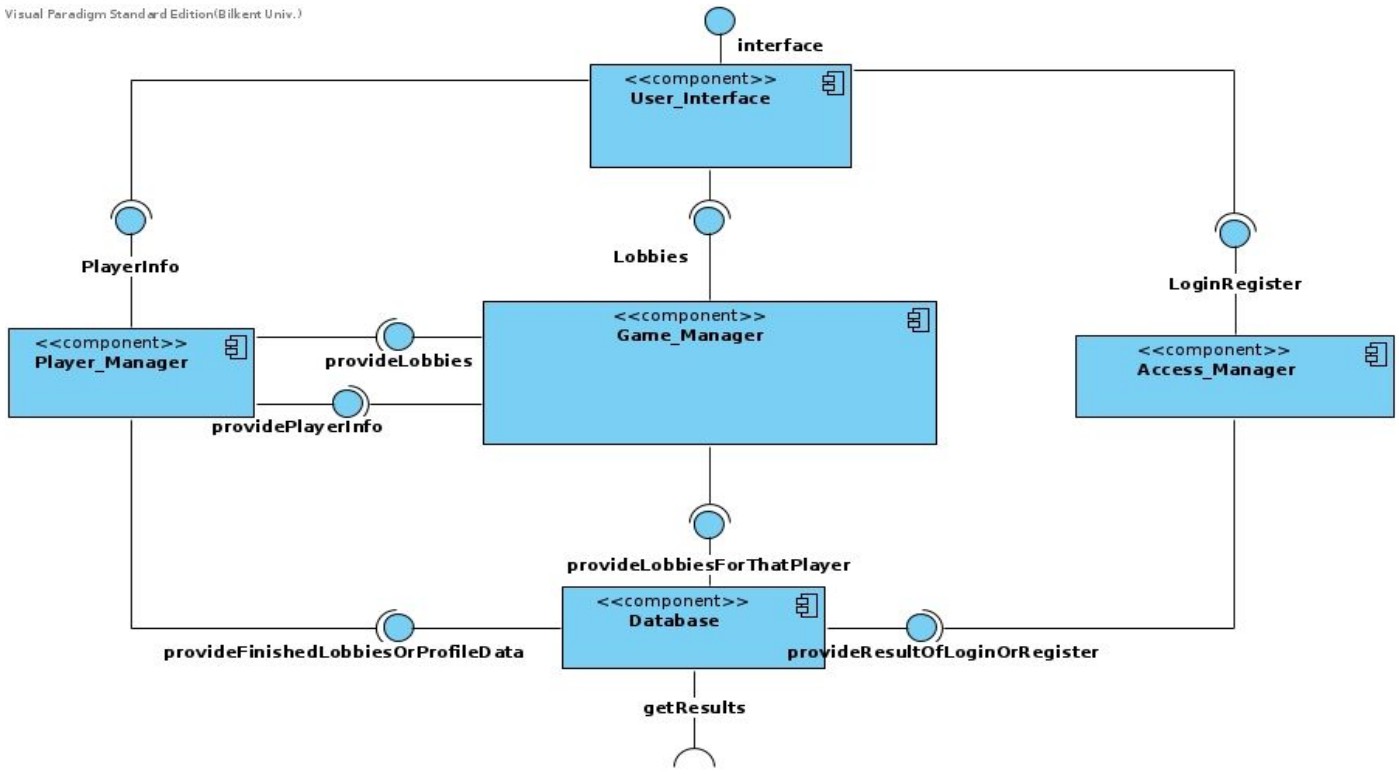

Figure 1.A : Component Diagram of "Your Story".

First layer

The diagram shows all the components and it represents three layered architecture style of our project, "Your Story". The top layer consists of the UI Classes. This layer is responsible for creating the user interface and is designed to offer high usability on the front-end. It takes user input from the keyboard and the mouse, and sends it to managers in second layer which handle the business logic. It will then be updated with any new data the second layer provides. This layer can change in the future to be able to work for Android's UI.

Second layer

In the second layer of architecture, we have the components which handle the business logic. All three of them, are managing and interpreting the data coming from the database in the third layer as well as from the UI components of the layer above. These components are also responsible for UI, when new data comes, or even inserting new data to the database taken from the UI, like chat messages that the user sends.

Third layer

In the third layer, we have the component that handles and simplifies the connection to the database. This one is responsible for feeding new data to the client and provide the required authentication data for the registration and login of a player. It will also insert new data to the database like registering a new user, adding some chat text, updating user's data etc. As mentioned above if there is no internet connection the query(ies) left unprocessed will be considered undone and won't be completed unless the user retriggers such query. The third layer will notify for loss of connection and the second layer will halt and transmit that information to the first layer which will freeze and let the user know the connection is lost and they should wait for it to reestablish.

Reasons for choosing this architecture

In the process of selecting the suitable architectural structure of our project, we considered our design goals and came to the conclusion of using a three layered architecture, for the following reasons:

- It provides a complex structure that properly handles the database communication in regards to the non-functional requirements that were discussed in the analysis report.
- It creates low coupling since all of the classes fit nicely to this architecture and logic of the whole system.
- Since we are able to handle the operations for registration/login of a player and the main game functions in the same level of layer, the data that is required can be stored or taken from a single database, which is in the layer below. This also fits to our design goals (our chosen trade-off), which is Rapid Development over Functionality.

Type of architecture

For this project, we choose the closed architecture over the open one, because one of our design goals is to have the game be modifiable. This will help us to add a server in the near future. So, in that case, we only need to substitute the third layer and modify the second layer, thus the UI layer will stay the same. Or if we need to make this game run in Android devices we only need to modify the UI layer and every business logic will remain intact. This architecture will make it faster to extend the project in the future to enhance it quickly and push the new update quickly to keep up with the market's demands.

Hardware/Software Mapping

Your Story, is a game that is currently being developed in the last version of Java Development Kit (SE 8U112) and can be played across Mac, Linux, Solaris, Windows 7, 8, 8.1 and 10 since these support latest Java updates. As the most basic needs to use our application, one should have a keyboard for user input, mouse for user interface interactions with a computer. On a side note for the keyboard, only English characters will be accepted as an keyboard input, because of programme does not support any other languages.

Since there is an online text messaging and login system with a huge database, full internet connection is required throughout the usage of the application. We do not connect this database to a server since there is not enough time for development, but updating it every 5 seconds so that the player can see the new messages on their monitor. That's why, we have another machine to host the database of Your Story. This machine will be active 7/24, to handle the operations that the players need to complete before and when playing.

Persistent Data Management

An external database will be used to maintain data flow between clients fluently. All clients will be updating its data every 5 seconds and will be able to modify the data in the database by themselves. Data passed between clients through the database will include dynamic contents like chat messages, lobby informations and profile infos. Other static contents like background pictures will be stored in client-side.

Additionally all clients will be able to directly reach the database, there won't be any server-side

script to handle requests coming from clients. Therefore while having a faster connection to the database, we won't be doing any extra work for server-side scripts. However our trade-off here will be uncontrolled accesses to the database.

Finally, database will have MySQL database management system and since MySQL is a relational database, SQL language will be the used to query the data and do insertion, update, deletion, etc.

Access Control and Security

"Your Story" will require a connection to the database. The username and password of the database will be hardcoded inside the core of the software. Anyone who can decompile "Your Story" can view its source code and thus get the username and password that server for connecting to the database. This will allow remote code execution in our database from anyone which is a huge issue and we are aware of that. However we will do our best to hide it to the large public, and possibly implement a recognition algorithm that will make the database unresponsive to remote connection other that from the official verified client. This issue happened because we omitted the java server as an intermediate of the database and client due to having rapid development as a design goal. Other than that the connection to the database is secure and allows no eavesdropping. This is implemented by JDBC driver. Once the client is connected to the database and authenticated he can get all the personal data about himself, but no personal data, like password, of other players will be disclosed. The only data a player can view is other players' profile.

Boundary Conditions

"Your Story" is an online program, so it can be problematic to lose the connection to the database while playing the game. Database connection loss blocks sending messages/votes and receiving messages in the game, and participating to lobby out of the game. While in the lobby, if you lose the connection, you have been kicked out of the lobby. If it happens while in the game, you are just given the warning about loss of the Internet connection, but the game will still continue without you. Also, your messages cannot be saved to database, so if you reload the page, your message will be disappear.

We do not have any data loss problems because our database will be saved in another device. If the program is closed during the game, you can open the program and connect the lobby again without any loss of data. The problems that can be occur while running the game is all about database connection problems and we have these boundaries to be able to gracefully handle such problems.

Subsystem Services

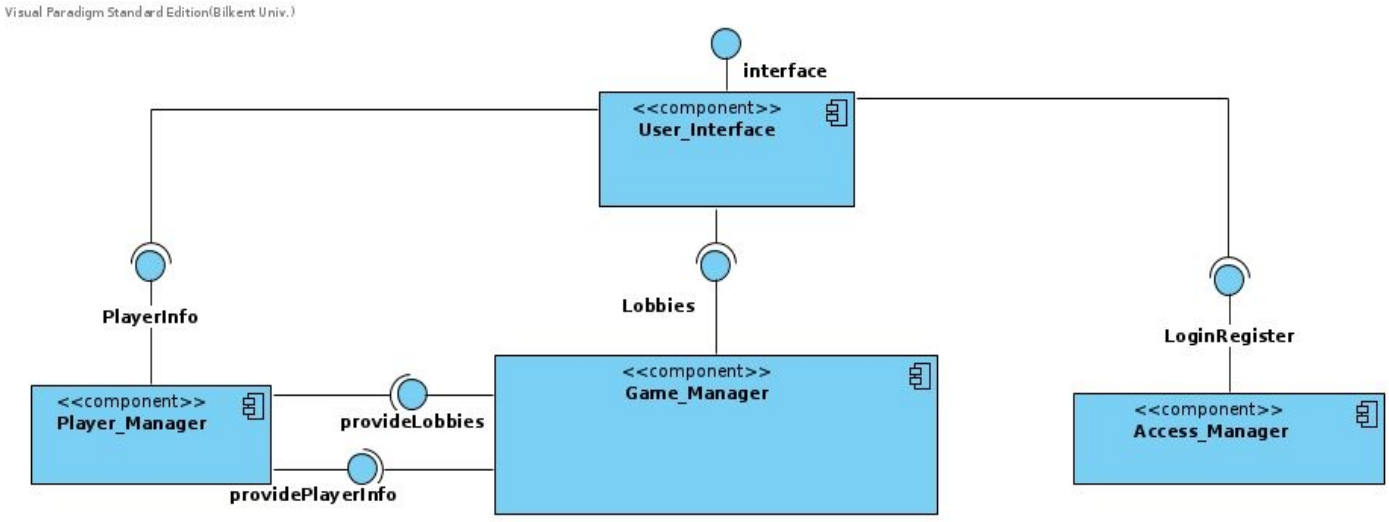

Figure 2.A : Component Diagram Layer 1-2 of "Your Story".

User_Interface Component

The User_Interface component provides a user friendly environment for players. This component updates the view for player everytime there is an interaction occurring between layers. It also gets hardware input from the user, and sends it to the managing layer. For register and login process, when it gets the username and password, it will send these information to access manager to validate it. After the validation process is done, the UI will change depending on the validity of the user credentials. After that, if it is valid, it will call the player manager to generate the home page.

Player_Manager Component

Player_Manager component requests the data of that particular logged in account. The data is something like the profile picture, description, finished games, unfinished games and all the visible ongoing lobbies where the player can participate. It interacts with the Game_Manager component to create lobbies which will be ascribed to that player. It will also update the view, any time it gets new information.

Game_Manager Component

Game_Manager component handles all the features of the game. It is used to create lobbies and send that data to the database. It will handle everything happening in the lobby like voting, choosing a character and seats to the database. It also sends the data of newly created like the name, story chosen again to the database. This component also responsible sync all data between clients for each player action in lobby.

Access_Handler Component

Access_Handler takes the input from UI and validates that the credentials are correct and then returns the result to the UI. To validate these credentials it connects to the database component and verifies the credential by querying the users' table if the input exists. Then if the table replies with an empty set then that username and password combination doesn't exist. If validation was successful it will get the id of user.

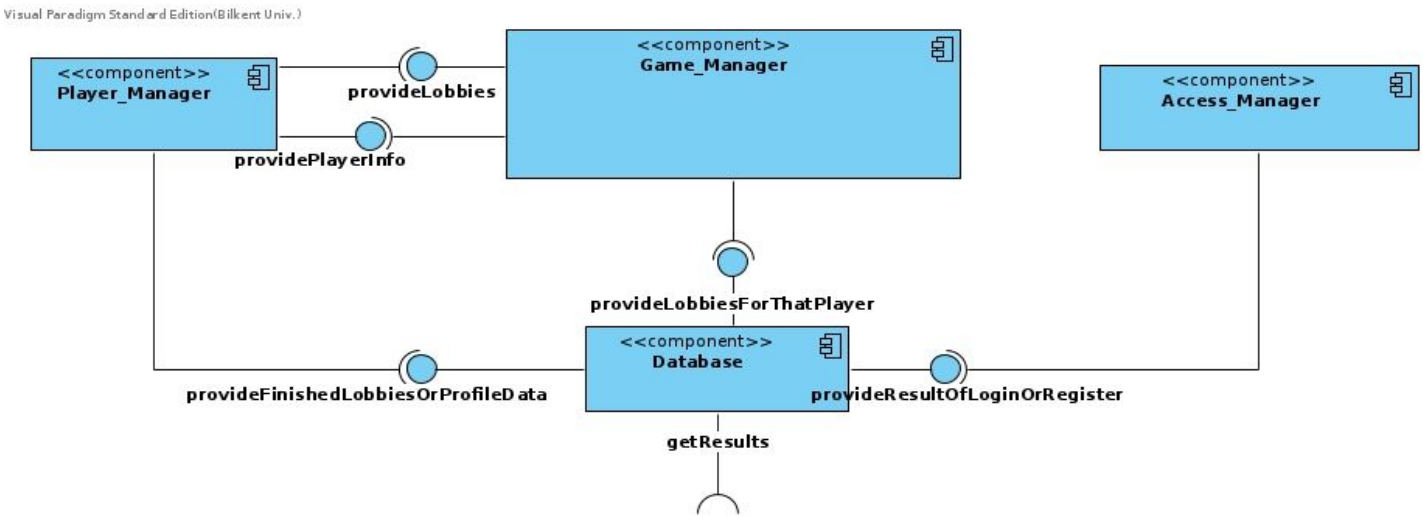

Figure 2.B : Component Diagram Layer 2-3 of "Your Story".

Database Component

Database component handles the connection between client and database securely. It will be able to change or add data to database while also provide the needed data for the usage of second layer. This is going to be achieved by sending queries to the database and receiving the result set from it. Second layer depends on the information retrieved from the database component and the UI layer depends on the data of the second layer. So this again demonstrates how we implemented the close architecture and its dependencies. Such component might change in the future to connect to a server through sockets, so even its naming might change to something more suitable at that time.