Bilkent University

Department of Computer Science

Senior Design Project

KEBAP TYCOON

[www.kebaptycoon.com]

High-Level Design Report

Group Members

Can Akgün
Doğancan Demirtaş
Kıvanç Kamay
Sinem Sav
Ulaş Sert

Supervisor

Hakan Ferhatosmanoğlu

Jury Members

Uğur Güdükbay
Selim Aksoy

Innovation Expert
Armağan Yavuz

Table of Contents

Table of Figures

1. Introduction

1.1 Purpose of the System

Our senior design project is a cross platform application called "Kebap Tycoon" which is a challenging game where players can found their restaurant chains and put effort to be the best in this sector by making decisions on both preferences of sales and locality strategies. The target user group will mostly depend on residences of Turkey as the game will represent instantaneous aspects of Turkish culture.

Its purpose is basically to entertain the players that chose to interact with our game. Moreover, the game itself embodies a purpose of adding a new game that is based on Turkish culture to the global game industry as well.

1.2 Design Goals

Adaptability:

It is necessary for us to create a game which can be played in all platforms and in all operating systems so that the game can reach over more users. We chose LibGDX game engine for writing the code of the game which provides us cross-platform portability.

Reliability

Making a reliable game is very important for us. If the system fails for even a short period of time, it will damage the reputation of the game and most likely reduce the number of the players. Thus mean time between failures should be maximum at all times, especially at prime time.

Modifiability

It is vital to create a system which can be modified easily. We may want to add new features to our game or remove some other. In order to reach that goal we will minimize the coupling of our subsystems. Furthermore using encapsulation and polymorphism in our design will make our system easier to modify.

Ease of Use

We aim to create a game which has a wide range of audience so Kebap Tycoon should be very easy to use. The interface of the game will be very friendly. Menu parts will be not complex but informative. Thus there will be no confusions for the players and it will be easier to play. However, those features will not make the game itself easy since we aim to create hard and fun game.

Ease of Learning

The game will be as much as easy to learn so that players do not have to spend a lot of time trying to understand the game. There will be tutorials as a part of the game where new players can discover how to control and play the game.

1.3 Definitions, Acronyms and Abbreviations

Tycoon Games: Type of games focusing on business simulations including management and economics.

MVC: Model View Controller software architecture.

UI: User Interface.

Parse: Parse is a cloud system that is useful for backend services [1].

REST API: Representational State Transfer that enables us to interact with Parse (from anything that can send an HTTP request) [2].

LibGDX: Game-development application framework written in the Java programming language. It allows for the development of desktop and mobile games by using the same code base [3].

1.4 Overview

Kebap Tycoon will be a game of business simulation and will entertain players by reflecting the humorous sides of Turkish kebap culture. Within this game, there will be a multiplayer logic where players can see their friends' status. This feature will differ Kebap Tycoon from other tycoon games. In addition, our game will be easy to use with user-friendly interfaces. The menus, buttons, images, and simulations will be understandable and attractive.

Making the architecture of this game robust is very important since it will contribute to the implementation part of the project. Decomposition is one of the most important principles in software and we tried to decompose our architecture appropriately. In this report, a detailed description of the proposed architecture including subsystem decomposition, hardware/software mapping, persistent data management, access control and security, global software control, boundary conditions and also subsystem services will be given.

2. Current Software Architecture

There are several tycoon games in the market available; however there is not any open source alternative so the current software architecture cannot be analyzed in depth unfortunately [4].

Additionally, the games on the market do not have any multiplayer feature, so the current systems only consist of client code. The architecture of those games is mainly MVC, because of the game engines impose using it (LibGDX, Unity, Unreal Engine etc.).

3. Proposed Software Architecture

3.1 Overview

Because of our LibGDX Game Engine, we needed to use the MVC architecture, which is applied to all components of the game such as customer, venue, staff and interface elements. Each component in the game contains a Model together with the Controller which updates the component upon either player or in-game action. The View is the display of the component. MVC structure is easy to apply on LibGDX, as LibGDX community highly recommends on building structure of projects with MVC architecture.

As our project has a requirement of holding the login credentials of the players and game state information for the saving/loading mechanism on a remote database service, we decompose the game into client and server layers.

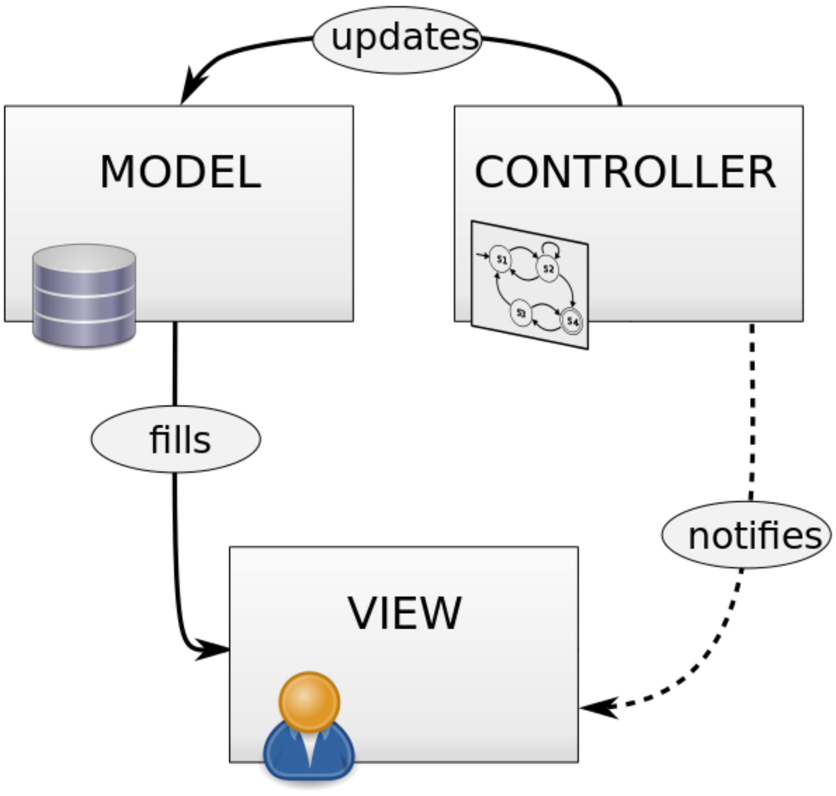
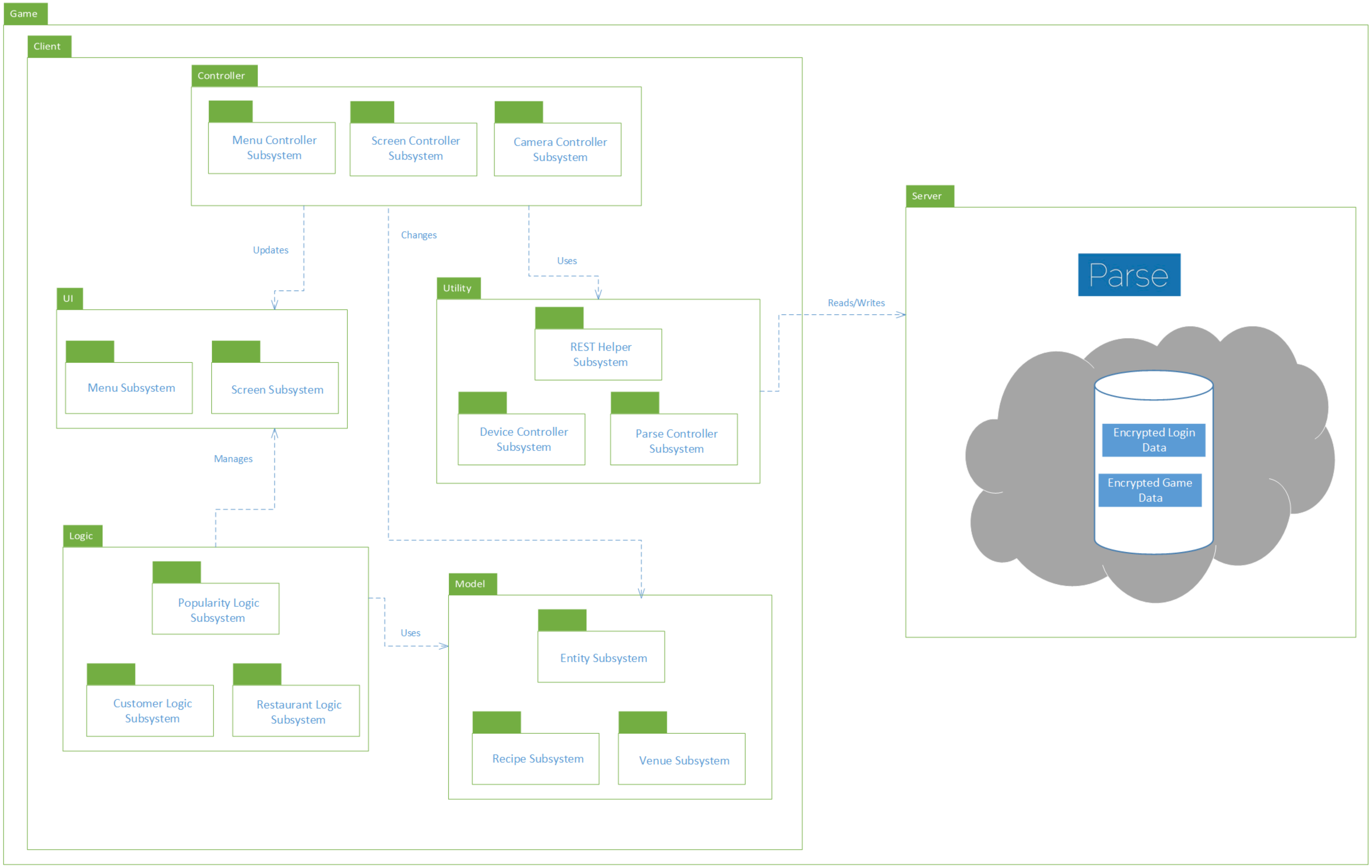


Figure 1 - Architecture of Kebap Tycoon Client Layer

3.2. Subsystem Decomposition



Subsystems of the game are created based on two major architectural elements which are Client and Server. With the help of Utility Subsystems these two major elements communicate between each other.

3.2.1 Client

Client layer uses the MVC architecture as described above, but additional components for connecting to remote database are required.

Controller Subsystems

Menu Controller Subsystem:

Responsible for managing the inputs that are coming from any of the menu screens while the game is being played. Each of the menu controllers are created according to the player's choices from the menu options. Thus, since only one controller can be active at a time, the remaining ones are pushed into the controller stack.

Screen Controller Subsystem:

All the game screens such as login screen, register screen, splash screen are handled by screen controller. The main responsibility of a specific screen's controller is managing the initiation of the menus that are covered by this screen.

Camera Controller Subsystem:

The camera controller manages the specific camera angle and the height (z-axis) that is most suitable for a tycoon game scene.

Model Subsystems

Entity Subsystem:

Responsible for storage of entities. An entity is any object within the system that can be rendered on the screen; such as customers, tables, chairs, etc. All properties and methods of entities are contained within here.

Recipe Subsystem:

Responsible for storage of customized recipes for dishes. A recipe of a dish can be modified by the player and customers will react differently to different recipes, such as making a recipe too spicy can cause them to complain.

Venue Subsystem :

Responsible for storage of all of the venue data inside the system; such as popularity, current stock of raw materials, employees etc.

Logic Subsystems

Popularity Logic Subsystem:

Manages the current popularity of any venue in the game based on the feedback customers give. Resulting popularity is viewed indirectly through the determination of the number of customers arriving.

Customer Logic Subsystem:

Manages the generation of new customers and customers themselves as; pathfinding, desired food, expectations, giving feedback, etc. The view of customers and their actions are lead through the UI over game logic.

Employee Logic Subsystem:

Manages the employees as; pathfinding, responsibilities, action queue, etc. The view of employees and their actions are lead through the UI over game logic.

UI Subsystems

Menu Subsystem:

The Menu Subsystem is responsible for showing the menus to the user and creation of their controllers.

Screen Subsystem:

The Screen Subsystem is responsible for showing the game screens to the user and creation of their controllers.

Utility Subsystems

Device Controller Subsystem:

The game uses device features such as Internet connection, media player and apps installed on device (Facebook). Also device state, screen orientation and language need to be retrieved by the game. That’s why a device controller subsystem will be created to get native or physical device information and to make changes.

Parse Controller Subsystem:

The game stores the user login data and the game state in a cloud database service, namely Parse. Parse has Android, iOS, and Windows Phone SDKs however there is not any pure Java SDK. In order to keep the codebase compact and not to write any platform specific code, base Parse functions will be implemented in Parse Controller Subsystem.

REST Helper Subsystem:

Parse REST API will be used in order to make information exchange between client and server. For ease of use, first a generic REST Helper subsystem will be created and then it will be used in Parse Controller Subsystem.

3.2.2 Server

In order not to implement a whole back-end service from scratch, a popular DBaaS, Parse, will be used in this project. Parse provides 30 requests per second, 20GB file storage, 20GB database storage and 2TB file transfer as well as one million unique push notifications and unlimited analytics tools.

3.3 Hardware – Software Mapping

Kebap Tycoon mainly targets mobile device users. In order to be played in Android devices, a Gradle build must be done and the "apk" file should be deployed to the device.

For iOS devices, the Xcode project should be built and linked with a developer account. Then the artifact must be deployed to the device (An Apple developer account is needed in that case).

In above cases, the Parse database service is used for information exchange and user authentication. The connection with Parse REST API will be done with HTTP requests.

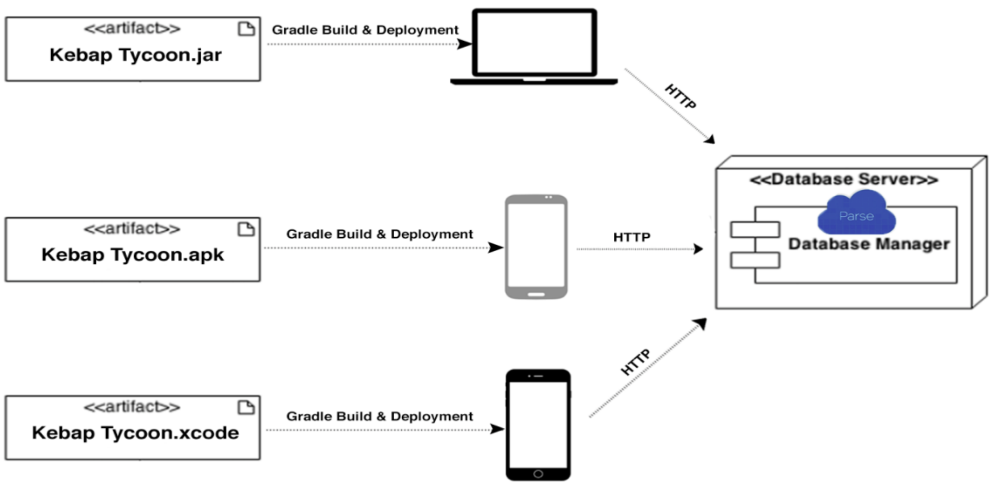


Figure 3 - Hardware-Software Mapping

3.4 Persistent Data Management

In our system, we will use Parse to provide mechanisms for backend services that include managing and storing data. We will use database for storing a player’s data i.e. amount of money, number of friends, item stocks, number of restaurants etc. Moreover, in our system, the players will not have the opportunity to save manually, rather than this, we have agreed to provide auto-save function. With this approach, we guarantee to have a persistent system and permanent data. However, if the system confronts errors due to the frameworks or services that we use (LibGDX or Parse), the current game progress can likely be interrupted and lost.

3.5 Access Control and Security

When Kebap Tycoon is opened in the very first beginning, the user should identify himself. Signing up a new user differs from creating a generic object in that the username and password fields are required. The password field is handled differently than the others; it is encrypted with bcrypt when stored in the Parse Cloud and never returned to any client request.

Moreover, when a registered user is trying to play the game, he needs to make himself recognizable for the system by providing the correct data for his account. The query for the database which is about this authentication needs to securitized via encryption as well.

3.6 Global Software Control

Our architecture has client and server sides(client-server system) and as there will be a game logic that enables time and status change in the game without expecting user input, the game is not event-driven but close to process-driven applications. Thus, we have logic subsystem in the client side of the architecture. In some cases, the game starts with Facebook login in which we have to use Facebook API. In other cases, we use form registration. User login data then stored in the Parse. Any status change during game will also be stored in Parse and will be read from there (with the help of implemented functions in Parse Controller Subsystem). For this data transfer between client and server, Parse REST API will be used (REST Helper Subsystem). Lastly, Internet connection is needed to play this game and this connection will be checked by Device Controller Subsystem.

3.7 Boundary Conditions

- **Initialization:** Kebap Tycoon will be available for Android, iOS and desktop platforms. Thus, players can play this game after they download the application. For desktop application, Java should be installed. The game starts with either form registration or Facebook/Google registration. Thus, for the initialization of the game, Internet connection and true username/password combination is required.
- **Termination:** The game will be terminated by clicking on Exit Game button and to prevent mistakenly exits (user can click on Exit Game mistakenly), confirmation from user will be requested with a question before exiting the game. This case will be valid for Android, iOS and desktop applications.
- **Failure:** Memory or CPU related errors, Internet connection loss and power loss would cause a failure. In these cases, the game will be terminated without player's confirmation.

4. Subsystem Services

We have used MVC architecture mixture with Client-Server architecture and here we will describe the services provided by main subsystem components.

- **Model:** Storage of the every and anything that affects the game state such as venues, in game objects, recipes.
- **User Interface:** Views of the view components of each object in the game, as being responsible for their update and output. Displays game graphics, menus, and everything that is displayed to the player
- **Controller:** Views of the view components of each object in the game, as being responsible for input. Processes every tap or click done by the players.
- **Utility:** Provides utility support to the system such as communicating with Facebook servers via REST API, communicating with the game server for loading and saving.
- **Logic:** Provides the set of rules that the ingame objects follow such as how customers rate food or how ingame people find the path between two points.

5. Glossary

LibGDX: Game-development application framework written in the Java programming language. It allows for the development of desktop and mobile games by using the same code base [3].

Parse: It is a mobile backend as a service (MBaaS) and used for providing web and mobile application developers with a way to link their applications to backend cloud storages. It has features such as user management and push notifications [1].

REST API: Representational State Transfer that enables us to interact with Parse (from anything that can send an HTTP request) [2].

6. References

[1] Parse.com, "Parse", 2015. [Online]. Available: https://parse.com/products. [Accessed: 19 Dec 2015].

[2] Parse.com, "Parse", 2015. [Online]. Available: https://parse.com/docs/rest/guide. [Accessed: 19 Dec 2015].

[3] Libgdx.badlogicgames.com, "libgdx", 2015. [Online]. Available: http://libgdx.badlogicgames.com. [Accessed: 21 Dec 2015].

[4] Store.steampowered.com, “Steam”, 2015. [Online]. Available: http://store.steampowered.com/tag/en/Management/#p=0&tab=NewReleases. [Accessed: 23 Dec 2015].