**1. Introduction**

1. **Purpose**

This document provides a detailed view of the architecture and the user interface design of the CLup system. Building on the RASD document, it gives a more refined technical and functional description of the system, explaining it on a much lower level. To provide the full description of the system, UML diagrams will be used since they are the de facto industry standard. While actual implementation is not part of the document, it outlines the presumed implementation, integration and test plan, to help the software development team in the realization of the application. The purpose of the document is primarily to guide software developers, but it can also provide useful information to end users and investors.

1. **Scope**

CLup is a simple application that helps store managers with handling large crowds inside their store and store customers with planning more efficient and safe grocery shops. The target audience for this application includes every person that shops for groceries in a store, which includes almost all demographics fall into this category. Faced with a worldwide pandemic of the COVID-19 virus countries across the world imposed strict health measures in line with the recommendations of the WHO. To combat the spread of the virus, governments introduced decrees that limited the movement of the population to a certain degree. Only essential movement, such as: going to work, grocery shopping or outdoor exercise, was deemed acceptable. Although successful in the mitigation of the disease, the act put a serious strain on society on many levels. To help reduce the stress and anxiety, many aspects of everyday life involving close contact can be considered and improved upon. This project aims to help with, and resolve the issues surrounding grocery shopping. As we all know, grocery shopping is an essential activity which involves close contact inside the store. Since the COVID-19 virus spreads mainly through airborne particles, this activity plays a key role in its mitigation. To reduce crowding inside the stores, supermarkets need to restrict access to their store and keep the number of people inside below the optimal maximum capacity. The main idea is to enable store customers to enter a queue from home (or wherever they find themselves) through simple interaction with the application. Besides that, the application will give customers the option to "book a visit" to the grocery store. This feature will allow them to view available time slots for their grocery shop, book the most convenient one, and optionally indicate an approximated duration of their visit to further improve the accuracy of the waiting time estimation of the system.

1. **Definitions, Abbreviations, Acronyms**
   1. WHO-World Health Organization
   2. RASD-Requirements Analysis and Specification Document
2. **Revision history**
3. **Reference documents**
4. **Document structure**

**2. ARCHITECTURAL DESIGN**

1. **Overview:** High-level components and their interaction
2. **Component view**
3. **Deployment view**
4. **Runtime view:**You can use sequence diagrams to describe the way components interact to accomplish specific tasks typically related to your use cases
5. **Component interfaces**

The complexity of developing a mobile application encourages the engineers to split the paramount task into smaller subproblems, or components. Each component is then encapsulating a certain part of the applications function, and communicating with other components through interfaces. The use of interfaces enables the developer of a component to be unaware of the concrete implementation of other components, but just know the syntax of their method calls.

This section gives a thorough overview of the systems interfaces, divided by components they belong to. It is important to note that the interfaces and their methods proposed in this section, do not necessarily represent the exact written counterparts in the implementation, but offer a basic guideline of the component communication.

General interfaces:

AndroidAppManager/iPhoneAppManager

-RequestTicket

-EntranceCheck

-GiveTicket

-InformUserToEnter

-CheckTicket

CalculateTime

ManageRequests

StoreSelector

ManageStore

ManageLogin

ManageData

DatabaseManager

GoogleMaps

ManageTickets

ManageBAV

CalculateDistance

ManageSchedule

ManageQueue

StoreManagerLogin

ManageEntrance

ManageExit

TicketChecker

**DODAO SAM JOS TIH INTERFACEA PA POGLEDAJ**

To begin with, the application on the users phone communicates with the system through the AndroidAppManager and IPhoneAppManager interfaces, according to the type of the users OS. They enable communication between the ApplicationServer, or more specifically, the Director component, and the smartphone application. Depending on the type of the user, the interface offers all methods for the interaction with the system.

For example, upon the push of a button in the UI, the application invokes the RequestTicket method on the Director to propagate the request. Similar to that, an application of a store manager can invoke the CheckTicket method on the Director, to propagate the specific request to the ApplicationServer.

The systems main global component, the ApplicationServer, is divided into smaller core components Director, LoginManager, StoreSelectionManager, DBService, GoogleMapsService, StoreManager and RequestManager. The communication between the core components is also done through interfaces.

For the Director to propagate users requests, the RequestManager offers methods contained in the ManageRequests interface. Other than that, the two components also communicate to calculate wait time through the CalculateTime interface.

Furthermore, the Director component communicates with the LoginManager and StoreSelectionManager components. For the LoginManager, it does so through a ManageLogin interface, which enables the propagation of the credential check used when a store manager accesses the application. For the StoreSelectionManager, it uses methods offered by the StoreSelectionManager in the ManageStore interface, so that the store manager can manage information about his store.

To enable users to choose between the stores using the CLup system, the StoreSelectionManager also offers the StoreSelection interface, used by the LoginManager for store managers and the RequestManager for store customers.

The calculation of the distance between the user and the store is done with the help of Google Maps. To implement that, the GoogleMapsService offers a GoogleMaps interface to the RequestManager. Upon gathering information, RequestManagers subcomponents BookAVisitService and DistanceService perform the distance calculation through the CalculateDistance interface.

Depending on the type of the user request, the RequestManager further branches into different subcomponents. The TicketService and QueueService encapsulate functions needed to perform the get in a virtual line function, and the BookAVisitService and ScheduleService serve to carry out the book a visit function of the system. For communication between the Director and the RequestManager, the former uses the ManageTickets and ManageQueue interfaces, whereas the latter uses the ManageBAV and ManageSchedule interfaces.

To physically manage store entrances and exits, the CLup system also offers a specific interface for the store managers. The logic behind that interface is contained in the StoreManager component. The component communicates with the LoginManager through the StoreManagerLogin interface, to execute logins of the store managers. This design choice improves upgradeability for future requests and different users. Furthermore, the StoreManager component also communicates with the DBService to persist data through the ManageData interface. Before persisting, the data is handled by the subcomponents EnterService and ExitService through the ManageEntrance and ManageExit interfaces. To connect the full circle of the system function, through the TicketChecker interface, offered by the RequestManager, the StoreManager can control the influx of the customers.

Lastly, in order to persist the applications data, the DatabaseManager interface is used. The interface enables the Application server, or more precisely, the DBService component, to invoke methods to write data to, or read data from, the database. The DBService itself, offers the ManageData interface to all other subcomponents of the ApplicationServer to connect them to the database.

1. **Selected architectural styles and patterns:**

Three-tier architecture image

To develop the full system, a three-tier client-server architecture will be used.  The choice is encouraged by the popularity of the model, and its valuable aspects such as modularity, scalability or abstraction. The three layers in the three-tier architecture are: presentation layer, application layer, and data layer. By dividing system artifacts in three different layers and enabling mutual communication with abstract interfaces, we hide unnecessary information and improve testability of each single layer. To communicate, the server waits for the clients requests, and upon a request, it extracts data from the database, processes it, and serves it to the client.

Depending on the needs of the system and the work done locally on the client, the client can be thin or thick. For the CLup system, a thin client design is a much better fit, since the calculations can easily be done on the server, and the application has to be usable by all smartphones, even the older ones. The data processed by the server can then be presented to the client device.

Basic software design practices such as modularity and the use interfaces will help greatly with the development, testing and further improvement of the application. Dividing the code in concise modules encourages reusability, and enables extensive upgradeability. Additionally, having functional modules makes component testing possible and quick. Abstraction in code, done through the use of interfaces enhances all the aforementioned effects and relieves the developer of the knowledge and complexity of all components outside his scope.

MVC image

We would also like to take advantage of the MVC(Model-View-Controller) global design pattern. The pattern is widespread in the software engineering community because of its scalability and simplicity, and provides a perfect base for our application. The Model component is the main component of the pattern in charge of the business logic. The View component controls the user interface, and the way in which information is represented. And the Controller component acts as a bridge between the Model and the View. It accepts data from, and creates instructions for the Model or the View.

To persist data like store enter and exit times, the system must be able to communicate with a database. DAO (Data Access Object) design pattern acts as an interface and makes that possible. For example, if written in the Java language, JDBC API will be used as DAO.

Main communication protocol used in the system is HTTPS. It provides a simple, popular, and secure connection for message exchange. It is also important to note that to use third-party software, we need to use specific protocols. In the case of GoogleMaps, we need to use the REST API.

In the end, the main software design patterns will be mentioned. Firstly, Observer/Listener behavioral pattern enables the system to subscribe to the updates of the client devices location and upon change execute new calculations. The pattern defines a one-to-many connection between objects and updates them automatically upon notification. And lastly, Bridge structural pattern is used in various places to decouple an abstraction from the implementation. In that way, components can be changed of substituted by completely new ones completely independently. Through aggregation and encapsulation the Bridge separates responsibilities into different classes.

1. **Other design decisions**

The CLup system relies on two main third party components to function properly: the database and the GoogleMaps API. Since most popular database vendors nowadays offer very similar functionalities, the database choice is left to the development team. For the needs of the system, even a NoSQL database system, like Firebase, would work. Firebase is a platform for creating applications, both web and mobile. It was founded in 2011 and is now the primary database option for application development offered by Google. It offers a real-time database, simple interfaces, and high-level security, which makes it well suited for our needs.

To calculate the expected time for a customer to get to the store, the best choice is GoogleMaps API. Through the years, Google polished their map system offer and today they have a widespread, highly accurate, and easy-to-use satellite imagery and street maps. It also offers, real-time traffic conditions and route planning for traveling by foot, car, and public transport which makes it a perfect mach for the needs of the CLup system.

In the end, one important design decision should be noted: to avoid security concerns with storing sensible user data in the database, the first version of the system evades user registration altogether. The scope of the system, so far, does not require the system to store such data. However, if the scope of the system grows in such a way that user registration becomes imminent, upgrade of the system built in line with this design document should not be a problem, especially since a form of user registration should already be implemented on the store side.