Identify the design paradigm you (i.e., the team) chose for your prototype and explain why you chose that paradigm (250-350 words – 5%).

* Top level + Component + Function

After going over the various types of design paradigms available, we decide to use a combination of high-level, component, and function-oriented design paradigms. To start we will use high-level design, which will help us explains the overall architecture that will be used for developing our project 3 and 4 application. This specific design paradigm provides an overview of an entire system, identifying the main components that would be developed for the product and th3eir interfaces. Therefore to get started a top-level design is needed to size the project and to identify those parts of the project that might be risky or time-consuming. As the project proceeds, we need to provide an overview of how the various sub-systems and components of the system fit together. Component design will help us with the basic building blocks for the software application and system being designed as it is portable, replaceable, and reusable. It decomposes the problem into more simple sub-problems represented by procedures, functions, or methods. After defining each component, we will use a function-oriented design as an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function.

For the remainder of the project, we will focus on service-oriented design as it shares many characteristics of component-based software engineering. SOA (service-oriented architecture) is related to the idea of an application programming interface (API), an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software. An API can be thought of as the service, and the SOA the architecture that allows the service to operate.

Within the context of your chosen design paradigm, describe the software architecture of your prototype (250-350 words – 5%).

* Client-Server

As we are designing a phone application for Project 3 and 4 we think that a client-server architecture will be the most efficient as it is widely use to develop phone applications. Client-server architecture is defined as an architecture of a computer network in which many clients (remote processors) request and receive service from a centralized server (host computer). Client-server consisting of two parts, client systems and server systems, both communicating over a computer network or on the same computer. A client-server application is a distributed system made up of both client and server software. Client server application provide a better way to share the workload. The client process always initiates a connection to the server, while the server process always waits for requests from any client. Client-server architecture offers many advantages, the client-server architecture separates hardware, software, and functionality of the system. For instance, if a software adaptation is needed in a particular country, i.e., a change in functionality is necessary. It can be adapted in the system without having to develop a version for phones, tablets, or laptops. Since it separates among hardware, software, and functionality of the system, only the front-end must be adapted to communicate with different devices. As functional and nonfunctional requirements change, modules can be updated without altering the client-server architecture or disrupting service. This architecture also includes some disadvantages, if all the clients simultaneously request data from the server, it may get overloaded and if the server fails for any reason, then no user can use the system.

Use one or more of the UML modeling diagrams to design your prototype (5%).

* Class Diagram

Identify the design patterns you used in your design and explain how you applied them (250-350 words - 5%).

For our creational patters we are choosing to implement a factory method. Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

For our structural patters we are choosing to implement a bridge pattern. Bridge is a structural design pattern that lets us split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other. As we are creating a phone app, when talking about real applications, the abstraction can be represented by a graphical user interface (GUI), and the implementation could be the underlying operating system code (API) which the GUI layer calls in response to user interactions. The abstraction object controls the appearance of the app, delegating the actual work to the linked implementation object. Different implementations are interchangeable as long as they follow a common interface, enabling the same GUI to work under Windows and Linux. As a result, you can change the GUI classes without touching the API-related classes. Moreover, adding support for another operating system only requires creating a subclass in the implementation hierarchy.

For our behavioral patter we are choosing to implement a command patter. Command is a behavioral design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request’s execution, and support undoable operations. We want to use this specific pattern as we want to parametrize objects with operations, queue operations, schedule their execution, or execute them remotely.