Smart NFC Card Application

SRS Document

Vast Expanse (Group 7)

COS 301 - 2019

Wian du Plooy	*u17237263*
Duncan Vodden	*u17037400*
Tjaart Booyens	*u17021775*
Savvas Panagiotou	*u17215286*
Jared O'Reilly	*u17051429*

Table of Contents

Introduction

Vision

Link is a mobile phone application used to simplify and enhance business networking and networking/interaction between companies. Contact information can be shared, and potential clients can be hosted at the company offices, all with the simple tap of a phone. The strong new wave of NFC technologies emerging is the driving force behind Link - enabling easy and effective new ways to communicate and share information, wirelessly.

We hope Link will become the new industry standard for networking between companies. Using this mobile application, companies (and, more importantly, the people within companies) will be able to connect with people in other companies with ease and scale never seen before in business networking. Link is here to link companies together.

Objectives

Link needs to fulfil 2 key objectives to be used effectively as a tool with which to network with employees of other companies:

- Enable easier sharing of company details and employee contact details, as well as make it easier to find the company and where to go to visit the company offices.
- Enable easier hosting of visiting clients at the company offices by allowing simple setup of physical access, guest WiFi access and means to spend money in-house for a visiting client and share it with them.

Link aims to mitigate the time and effort needed for various traditional networking processes:

- The usual hassle of keeping business cards in wallets or purses
- The manual communication of details over WhatsApp or email or related technologies
- The time-consuming organization needed for a client to visit the office premises

Business Needs

Networking between employees in companies is needed to ensure these different companies can get in touch with each other and set up negotiations with each other on how their companies can work together in their common interest. Therefore, effective and easy networking will enhance the ease of initial company-company linking, as well as subsequent company-company interaction and negotiation.

Link fulfils these business needs. With its capability to share vital company and employee information between employees, as well as its capability to setup hosting for visiting clients, Link eases and enhances the business networking process, cementing itself as a valuable tool to allow better company-company interaction.

Scope

Link will be a mobile application, paired with a backend-access system, which will be accessed ideally through a web interface. The mobile application and backend-access system will allow controlled altering and viewing of the data stored on a backend database, with this database being an efficiently and logically designed relational database scheme using an appropriate relational database management system (RDBMS) such as PostgreSQL. To provide the access to this database, it will be securely wrapped and accessible through a web API, implemented using an appropriate server-side language such as NodeJS. The mobile application will be developed using a web framework specifically designed for cross-platform mobile application development, such as the Angular-Ionic-Cordova framework.

Link will allow employees of a company to share their virtual business card as well as their company's location with potential clients who also have the Link app, with some simple setup beforehand, followed by a simple tap of phones to effortlessly share vital contact information. Link will also simplify the process of hosting a potential client at a company's offices, by enabling temporary physical access into the premises, temporary guest WiFi access and temporary cards with which to make in-house purchases to be easily setup using the app and then shared with visiting clients, with a satisfying tap between phones.

All setup of business card and location sharing, temporary physical access into the premises, temporary guest WiFi access and temporary cards for smart payments will be able to be done in the mobile app by employees of the company, and will be able to be shared using the mobile app from employees to potential clients (who may themselves be employees of other companies). Only Link administrators will be able to use the backend-access system, to insert new companies and employees and other details required for when a new company starts using Link, as well as to draw reports on transaction logs for the smart payment cards.

Definitions

Acronym	Definition
BAS	Backend-Access System
VBC	Virtual Business Card: The digital version of a business card that is used by our system instead of a physical business card.
TPA	Temporary Physical Access: A virtual version of an NFC enabled access card that is used by our system to grant access to a client.
TGW	Temporary Guest WiFi: WiFi access given to a guest for a limited period of time.
RDBMS	Relational Database Management System
API	Application Programming Interface
NFC	Near Field Communication
CRUD	Create, Read, Update and Delete

Domain Model

See below for the domain model for the Link system. Note that this UML diagram does <u>not</u> contain any design patterns - this is because the domain model is used only to represent the entities within the system and the relationships between them. Design pattern use will come into the project later when the implementation starts to take place, to ensure good software principles are followed. Software principles should <u>not</u> affect the original description of entities and their relationships.
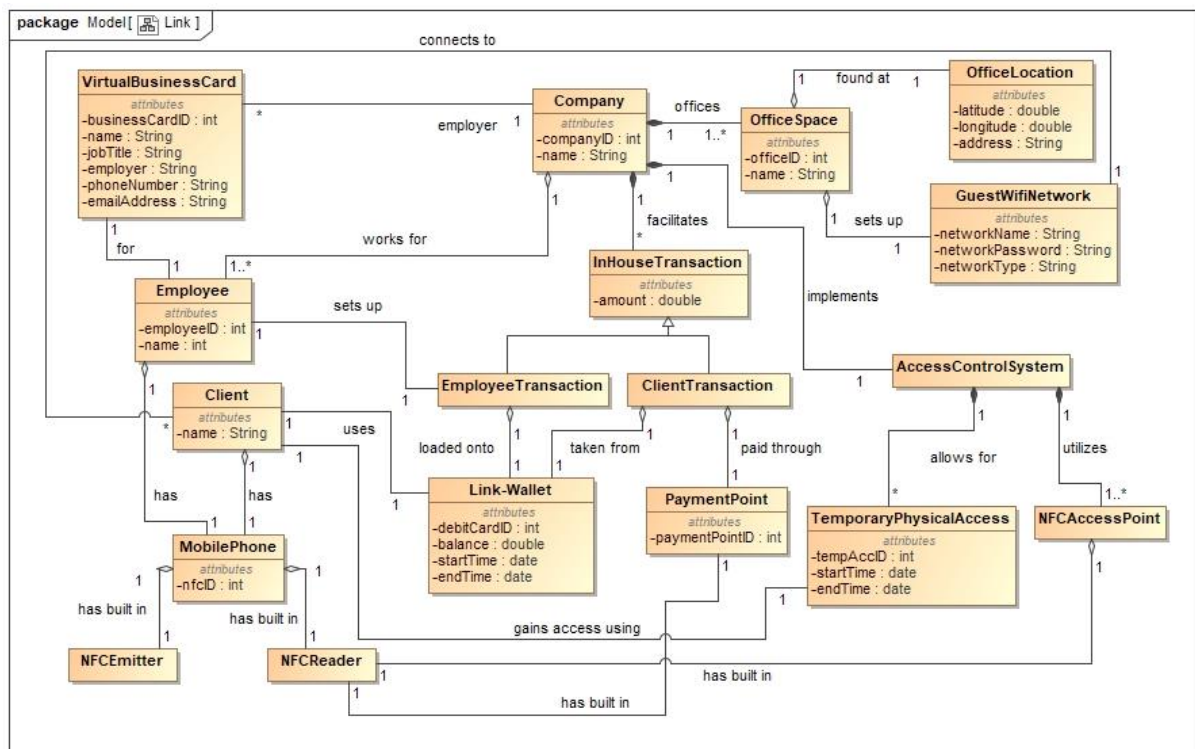
User Characteristics

There are numerous users who are going to interact with the application. All these users have a different use for the application so they will have different requirements.

Admin

Admin users should fully understand how Link works and how to perform **CRUD** operations as well as how to use the predefined database transactions. Admin will be able to access logs to see what has gone wrong and must be able to rectify issues detected using the logs.
These users are the gateway for a new company to start using Link, as admin will give certain rights to a representative from the company, who will then handle the company they represent.

Company Representative

This user will communicate with an admin user when the company wants to use Link and will then be given access and rights to the application. There should only be one representative per company. This user is then responsible for the following:

- Entering the company details to be displayed on virtual business cards, including office locations
- Successfully adding the company's employees to the database
- Giving certain access rights to different employees based on their type and security level.

Company Employees

Company Employees will be split up into different types of employees and will each have a security level. The security level will be used to allow or reject certain operations that the employees will can perform (different companies have different types of employees and different security levels; thus, it will not be specified directly what users can or cannot do, it must be generalized).

The security level of an employee will bestow certain rights to that employee, for example, allowing them to change his details or that of another employee.

The following operations can be done by the employees of a company based on the type of employee and corresponding security level:

- Perform **CRUD** operations for the company's information and employees
- Update details on the employee's **VBC**
- Share the **VBC** with potential clients using **NFC**
- Give **TPA** to the company's guests
- Give **TGW** to the company's guests
- Load onto the **Link-Wallets** of the company's guests

Company Guest

The company guest will be a physical client/guest visiting a specific company and will need to be granted **TPA**, **TGW** and perhaps a **Link-Wallet** to buy items in-house. This user will use Link when tapping the phone against an Employee's phone in order to exchange these details and store it on the phone. The user can then use Link and the **TPA**, **TGW** and **Link-Wallet** they have been granted inside of the office premises.

Client

These users are the potential clients for a specific company. Clients will be able to use Link to receive a **VBC** using **NFC** by tapping the phone against an employee of some company. A client can then use Link to view the virtual business cards scanned, favourite cards, delete unwanted cards and view the location of a company.

Functional Requirements

Use Cases

Actors - Admin, Client, Company representative, Company Employee, Company Guest

Business Card Subsystem

UC01. Create a **VBC (virtual business Card)** (Actor: Company Employee, System: Business Card)
UC02. View a **VBC** (Actor: Company Employee, System: Business Card)
UC03. Share a **VBC** (Actor: Company Employee, System: Business Card)
UC04. Log in to Link Application (Actor: Company Employee, System: Business Card)
UC05. Retrieve Company Card (Actor: Company Employee, System: Business Card)
UC06. Receive a **VBC** via NFC (Actor: Company Employee & Client, System: Business Card)
UC07. Display all scanned **VBC** (Actors: Client & Company Employee, System: Business Card)
UC08. Navigate using the location on a **VBC** (Actor: Client & Company Employee, System: Business Card)
UC09. Log in to Link Web Interface (Actor: Company Representative, System: Business Card)
UC10. Assign Company Cards to Employees (Actor: Company Representative, System: Business Card)
UC11. Edit Employee Details (Actor: Company Representative, System: Business Card)
UC12. Edit Company Details (Actor: Company Representative, System: Business Card)
UC13. Create a Company (Actor: Link Admin, System: Business Card)

Data Storage and Organisation

UC14. Modify backend database (Actor: Admin, System: Data Storage and Organisation)
UC15. Generate statements/reports (Actor: Admin, System: Data Storage and Organisation)

Access Control Subsystem

UC16. Setup Temporary Access for Guests (Actor: Company Employee, System: Access Control)
UC17. Acquire Temporary Access from Employee (Actor: Company Guest, System: Access Control)
UC18. Access company parking (Actor: Company Guest, System: Access)
UC19. Access temporarily allowed locations (Actor: Company Guest, System: Access Control)

WiFi Access Subsystem

UC20. Setup Temporary WiFi Access for Guests (Actor: Company Employee, System: WiFi Access)
UC21. Share Temporary WiFi Access via NFC (Actor: Company Employee, System: WiFi Access)
UC22. Acquire Temporary WiFi Access from Employee (Actor: Company Guest, System: WiFi Access)

Smart Payment Subsystem

UC23. Setup/Configure a Link-Wallet (Actor: Company Employee, System: Smart Payment)
UC24. Share a Link-Wallet with a user (Actor: Company Employee, System: Smart Payment)
UC25. Acquire a preconfigured Link-Wallet (Actors: Company Guest & Company Employee, System: Smart Payment)
UC26. Purchase items using a Link-Wallet with sufficient funds (Actors: Company Guest & Company Employee, System: Smart Payment)
UC27. Generate a Statement/Report of a Link-Wallet user (Actor: Company Representative, System: Smart Payment)

Use Case Diagrams

Business Card Subsystem
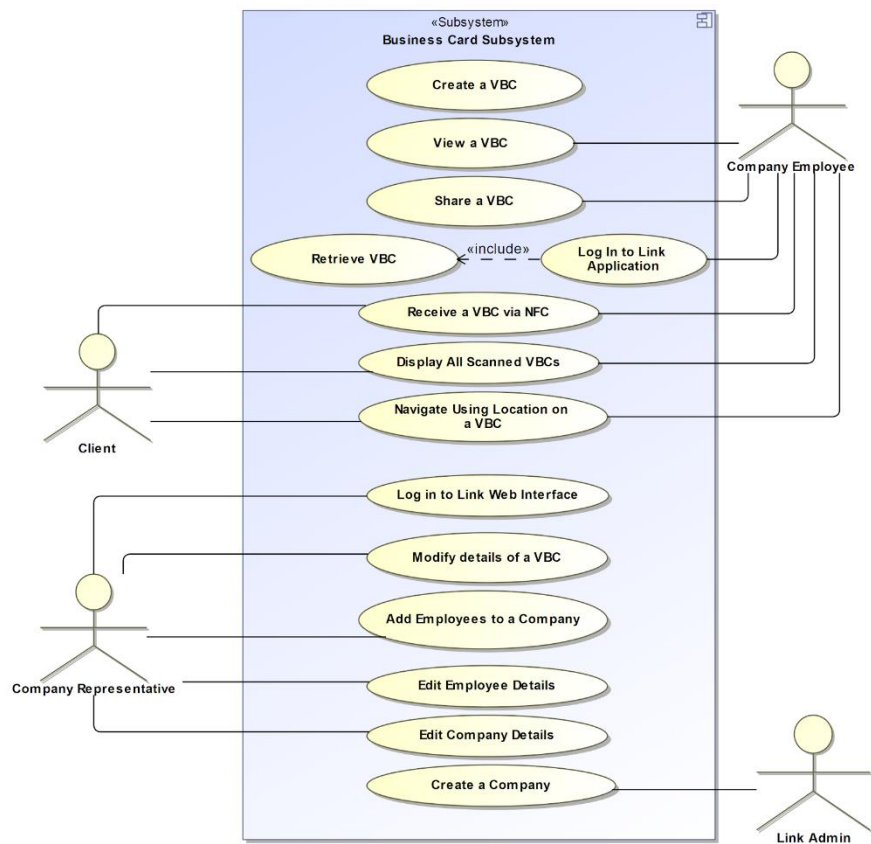
WiFi Access Subsystem

Access Control Subsystem

«Subsystem»
Access Control Subsystem

- Setup Temporary Access for Guests
- Acquire Temporary Access from Employee
- Access Temporarily Allowed Locations
- Create a Company Building

Company Guest

Company Employee

Company Representative
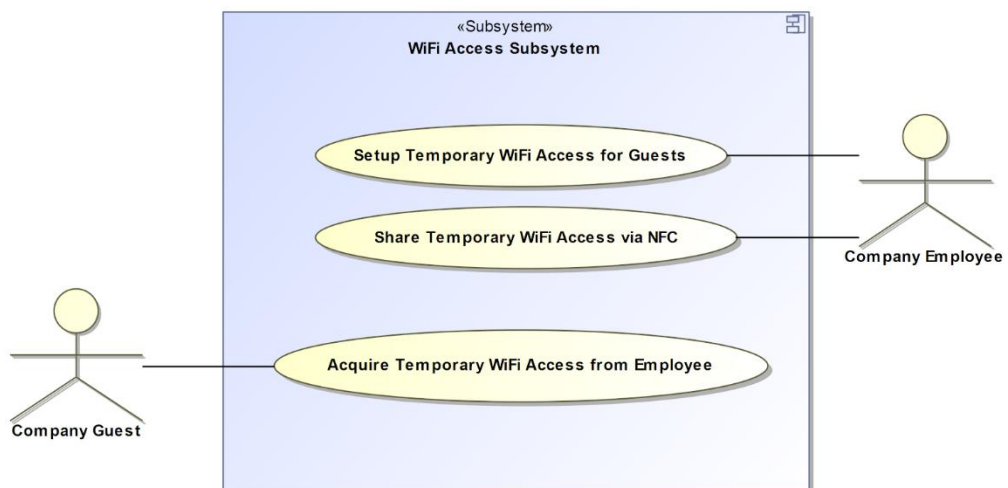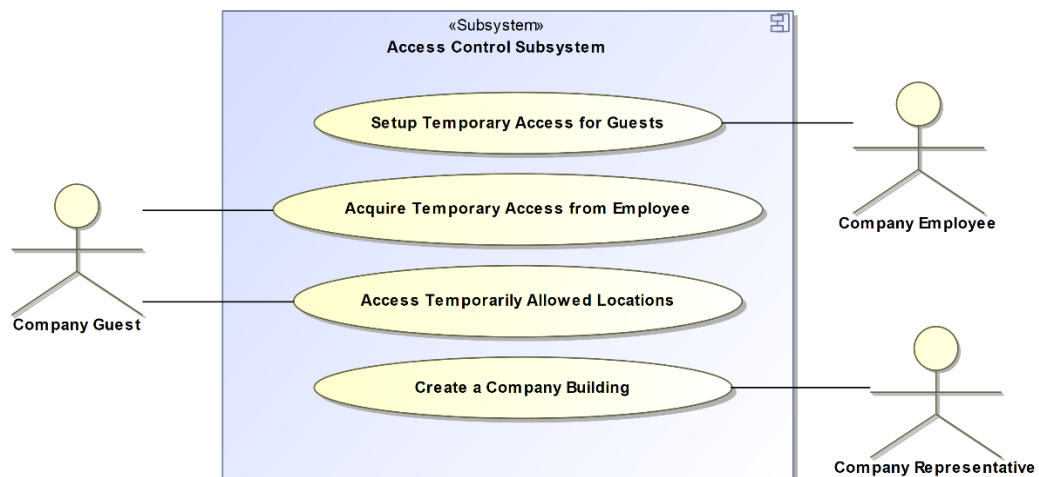
Data Storage Subsystem

«Subsystem»
Data Storage and Organisation Subsystem

- Modify Backend Database
- Generate Statements/Reports

Link Admin

Smart Payment Subsystem

«Subsystem»
Smart Payment Subsystem

- Setup/Configure Link Wallet
- Share a Link Wallet via NFC
- Generate a Statement/Report for Transactions
- Purchase Items Using a Link Wallet
- Generate a Statement/Report for Link Wallet Transactions

Company Guest

Company Employee

Company Representative
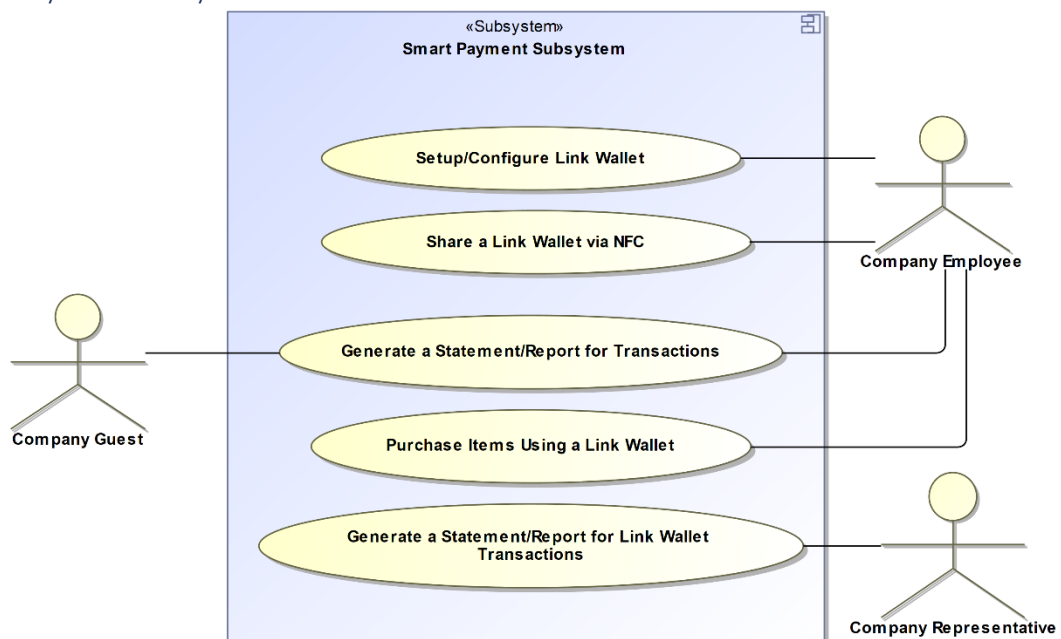
Functional Requirements

R1 Link must provide a backend-access system (**BAS**) for administrators of the Link system   to access and alter the database used in Link

> **R1.1** This **BAS** must allow for the following trivial methods of access into the tables of the database
>
>> **R1.1.1** C - Creation of rows in the various tables of the database, by providing the data to be placed in the row to be inserted
>>
>> **R1.1.2** R - Reading/viewing of the rows in the various tables of the database
>>
>> **R1.1.3** U - Updating of the rows in the various tables of the database, by providing the new/altered data the rows must be updated to
>>
>> **R1.1.4** D - Deletion of rows in the various tables of the database
>
> **R1.2** This BAS must allow for the usage of non-trivial predefined database procedures (database transactions) to perform complex database operations
>
>> 1. These transactions must be executed in such a fashion that ACID principles (Atomicity, Consistency, Isolation, Durability) are held, to guarantee validity even in the event of errors
>
> **R1.3** This **BAS** must only allow authorised access into the database, by administrators of the Link system only, with authorisation being logging in with a username and password
>
> **R1.4** This **BAS** must provide functionality to generate statements to be sent to a company for the payments made using the Link system
>
> These statements should display the following data:
>
>> 1. Logs of money loaded onto **Link-Wallets** (see **R5** below for more information) by company representatives and how much of the money was used by clients
>>
>> 2. Logs of payments made at in-house points of sale by clients using **Link-Wallets** (see **R5** below for more information), including what was bought

R2 Link must provide functionality for an employee of a company to share their virtual business card (**VBC**), along with their company's location, with a potential client using NFC

> **R2.1** From the **Company's** side:
>
>> **R2.1.1** Employees of a company must be able to fill in their personal/contact details which will be displayed on the **VBC**
>>
>> **R2.1.2** A company representative must be able to fill in their company details which will be displayed on all **VBCs** of the employees in their company
>>
>> **R2.1.3** A company must be able to fill in the location of their various offices which will be displayed on all **VBCs** of the employees in their company, and be able to assign employees to offices so the correct office location reflects on their **VBC**
>
> **R2.2** From the **User's** side:
>
>> **R2.2.1** A user of Link must be able to share the **VBC** they have created with another user
>>
>>> 1. This must be done by the user opening Link, selecting 'Share **VBC**, and tapping it against the other user's phone with Link open
>>>
>>> 2. After the sender shares their **VBC**, it must be displayed on the recipient's phone
>>>
>>> 3. As soon as the tap is completed and the data is sent through, the recipient's phone should have the picture of the **VBC** with the relevant details of the sender filled in
>>
>> **R2.2.2** After the sender shares their **VBC**, the **VBC** itself must be stored on the recipient's phone in a list format and be able to be viewed at a later stage
>>
>>> **R2.2.2.1** The recipient should be able to navigate to the list of **VBCs** they have scanned, with the top of the list being more recent **VBCs**

R2.2.2.2 From this list, the recipient should be able to click a **VBC** and be able to view it as if it was just shared with them

R2.2.2.3 The recipient should be able to favourite certain **VBC's**, which will bring them to the top of the **VBC** list

R2.2.2.4 The recipient should be able to delete certain **VBC's**, which will remove them from the **VBC** list

R2.2.3 When the recipient is viewing the **VBC**, they should be able to click on the location on the card and get directions to this location

1. A map application (e.g. Google Maps, Apple Maps) should be opened on the clicking of this location on the **VBC** and take in the location, allowing directions to the location or just general viewing of where the location is

R3 Link must provide functionality for an employee of a company to set up temporary physical access (**TPA**) for the NFC access-control system at their offices for when a potential client comes to visit their offices

R3.1 From the **Company's** side:

R3.1.1 An employee of a company must be able to set-up **TPA** for a potential client using Link by entering the following details for the client:

1. Which office the client is visiting
2. The client's personal details
   a. Name, Contact Details, Company
3. The time frame over which the client is visiting the offices
4. Potential **TPA** to the parking scheme of the offices, entering the number plate of the client's car

R3.1.2 The offices of a company that a client will come to visit must have NFC readers at the entrances to the offices, as well as physical security constraints to ensure only people with correct NFC identification can enter the office premises

R3.2 From the **User's** side:

R3.2.1 On the arrival of the client at the offices, the employee should be able to share the **TPA** they have created with the client

1. This should be done by the representative opening Link on their phone, selecting 'Share **TPA**', and with the client's phone open on Link, tapping their phone against the client's phone, with **NFC** transferring the necessary information across
2. The client can then gain access into the offices by tapping their phone against the **NFC** readers at the entrances, and having the respective security measures allow access into the offices (e.g. turnstiles unlocking)
3. The client should only be able to enter the office premises within the specified time frame

R4 Link must provide functionality for one employee of a company to set up temporary guest WiFi access (**TGW**) at their offices for when a potential client comes to visit their offices

R4.1 From the **Company's** side:

R4.1.1 One employee of a company should be able to setup **TGW** for the offices they are assigned to once, by entering the following information about the guest WiFi network:

1. The name of the network
2. The type of the network

3. The password for the network

R4.1.2 An employee of the company should then be able to, using the **TGW** setup earlier, be able to specify the time limit over which the **TGW** can be used by the visiting client

R4.2 From the **User's** side:

R4.2.1 On the arrival of the client at the offices, the employee should be able to share the **TGW** they have created with the client

1. This should be done by the employee opening Link on their phone, selecting 'Share **TGW**', and with the client's phone open on Link, tapping their phone against the client's phone, with NFC transferring the necessary information across
2. The client can then gain access into the guest WiFi instantly, through Link automatically logging them into the guest WiFi using the details contained in the **TGW** setup by the company employee
3. The client should only be able to use the **TGW** within the specified time frame, through the use of whitelisting or blacklisting on the routers within the offices providing the guest WiFi

R5 Link must provide functionality for an employee of a company to set up a virtual wallet to give to clients coming to visit their offices, which the client can use to make payments within the office premises (e.g. at in-house coffee shops, the company cafeteria, merchandise shops), with these wallets being called **Link-Wallets**

R5.1 From the **Company's** side:

R5.1.1 An employee of a company should be able to setup a **Link-Wallet** for a visiting client by filling in the following details:

1. Which office the client is visiting
2. The client's personal details
   a. Name, Contact Details, Company
3. The time frame over which the client can use the **Link-Wallet**
4. The amount of money to load onto the **Link-Wallet** from their account

R5.1.2 A company should be able to view a report/statement of all transactions completed using **Link-Wallets**

1. This report should show the details of all money loaded onto **Link-Wallets** by employees of the company, including:
   a. When it was loaded
   b. Who it was loaded by
   c. Onto which **Link-Wallet** it was loaded onto, and which client used this **Link-Wallet**
   d. How much of the loaded money was used
2. This report should show the details of all money spent by clients in the offices using **Link-Wallets**, including:
   . When the payment was made
   a. On which point of sale in the offices the payment was made at
   b. What was bought with this payment
   c. Which **Link-Wallet** the payment was made with, and which client used this **Link-Wallet**

3. This report should be able to be generated over any time frame of the company's choosing, as well as weekly/biweekly/monthly reports being sent from Link to the company
4. The company can then choose to subsidise some of these visitor hosting costs, or instruct the employee of the company to pay for the visiting costs, or even a mixture of both
5. A company must have NFC readers at the various points of sale on their premises, to ensure Link can be used to pay for items

R5.2 From the **User**'s side:

R5.2.1 On the arrival of the client at the offices, the employee should be able to share the **Link-Wallet** they have created with the client

1. This should be done by the employee opening Link on their phone, selecting 'Share **Link-Wallet**', and with the client's phone open on Link, tapping their phone against the client's phone, with NFC transferring the necessary information across
2. The client can then use this **Link-Wallet** on any in-house point of sale, by tapping their card against the respective NFC readers at the points of sale, with all transactions and payments made being carefully logged
3. The client should only be able to use the **Link-Wallet** within the specified time frame on the card, with attempted payments after or before the time limit being rejected
4. The client should not be able to spend more money than what is loaded onto the **Link-Wallet**, with each payment deducting money from the **Link-Wallet**

Subsystems

S1. Data Storage and Organisation Subsystem

Description

This subsystem is responsible for maintaining the database and performing **CRUD** operations on the database. It will be the interface for database operations for all other subsystems. It will also generate statements from the payments made using the Link system.

Related Functional Requirements

- R1.1.1
- R1.1.2
- R1.1.3
- R1.1.4
- R1.2
- R1.3
- R1.4
- R2.1.1
- R2.1.2
- R2.1.3
- R3.1.1
- R3.2.1
- R4.1.1
- R4.1.2
- R4.2.1
- R5.1.1
- R5.1.2
- R5.2.1

S2. Business Cards Subsystem

Description

This subsystem is responsible for the creating, sharing, receiving and viewing of **VBCs**. A user can create their **VBC** which consists of name, phone number, email address, business name and business location

Related Functional Requirements

- R2.1.1
- R2.1.2
- R2.1.3
- R2.2.1
- R2.2.2.1
- R2.2.2.2
- R2.2.2.3
- R2.2.2.4
- R2.2.3

S3. Access Control Subsystem

Description

This subsystem is responsible for creating and allowing guests **TPA** to a Company's premises. This should be set up by an employee with the appropriate security level.

Related Functional Requirements

- R3.1.1
- R3.2.1

S4. WiFi Access Subsystem

Description

This subsystem is responsible for signing a user into a company's WiFi network with **TGW** and will be set up (if chosen) when the user is given access control (S3.). This should be set up by an employee with the appropriate security level

Related Functional Requirements

- R4.1.1
- R4.1.2
- R4.2.1

S5. Smart Payments Subsystem

Description

This subsystem is responsible for the cashless transactions that will take place within a company using **Link-Wallets**. Employees/Guests can purchase items by using their Link Application.

Related Functional Requirements

- R1.4
- R5.1.1
- R5.1.2
- R5.2.1

Quality Requirements

The requirements in this section provide information about the quality of the application and what the application should be able to achieve.

Q.1 Performance

Q.1.1. Link must be able to handle 200 requests per second.
Q.1.2. Link must be able to respond to an initial request in under 1 second but will depend on internet connection strength and location of the server.

This will ensure that users of Link will not get frustrated and wait in queues to access the application and that they also do not have to wait long for a request to be handled.

Q.2 Reliability & Availability

Link will be hosted on an external server with a contractual agreement with the service provider.

Q.2.1. Availability

Q.2.1.1. Link must be available for 99% throughout its lifetime.
Q.2.1.2. Link must have access to the databases for 99% throughout its lifetime.

This will ensure that all users will be able to use the application as much as possible. It will provide trust between the user and the application as the user will always be able to use it.

Q.2.2. Reliability

Q.2.2.1. Link must behave the same in deployment as it did in testing.

This will ensure that the application behaves in an expected manner. The application must behave the same way in the development stages for every user, as it did when the product owners conducted tests on the application.

Q.3 Extendibility

Q.3.1. Link must be designed using appropriate design patterns to allow for easy extension of functionality.

Allows the application to be extended with more functionality without changing lots of classes or modules. This allows for an easy to maintain and easy to fix, design for the application which is desirable.

Q.4 Usability

Q.4.1. Graphical User Interface

Q.4.1.1. Link must have an easy to navigate user interface to allow all users to understand the application.
Q.4.1.2. Link must be designed in a vertical approach rather than a horizontal design to make navigating through the application easier.

This will ensure that users will find the application easy to use and navigate. It will also be responsive so that it is visually pleasing for the user.

Q.5 Security

Q.5.1. Data Storage

Q.5.1.1. The data of users must be stored in a secure manner and must have controlled access.
Q.5.1.2. All data that conform to the Customer Laws need to be logged for the required amount of time and must be deleted after a certain amount of time.
Q.5.1.3. Audit logs must be stored and must only be accessible to the product owners.
Q.5.1.4. Client passwords must be hashed and salted before storage.

This is to ensure that all personal information that is stored on the database can't be accessed by users of the system. It will also log required information about people as is required by law and will remove these logs after a certain time as is required by law.

Q.5.2. Data Transfer

Q.5.2.1. Data sent over the internet must be encrypted and securely transferred between different locations.
Q.5.2.2. Data sent over **NFC** must be encrypted.
Q.5.2.3. When sending data over **NFC**, a user must be prompted to transfer the data or deny the transfer of data.

This is to ensure that no personal data can be obtained by people in the middle of data communication. This will ensure encrypted data communication to promote privacy. It also allows a form of two-factor authentication in the sense that a user will be prompted to accept data sent over NFC, which adds an additional layer of security.

Q.5.3. Data Access

Q.5.3.1. All data must have clearance levels associated with it, which will give controlled access to data.
Q.5.3.2. All data logs must have controlled access and can only be accessed through an interface (not **API**), only users with desired clearance levels may access the data.
Q.5.3.3. The owner of the product must be able to add privileges or remove privileges from a client.

This will prevent data from being accessed by people without the necessary clearance levels and keep the data private. It will also prevent users to remove any logs and therefore can't hide something by deleting logs.

Q.6 Testability

Q.6.1. All features offered by Link must be testable through unit tests.
Q.6.2. All subsystems of Link must use integration tests to test whether all features it needs from other subsystems are available and if they work.

This will ensure that the application behaves as it should during the testing phase, meaning that you can easily find mistakes in the application.

Trace-ability Matrix

For notation used in Trace-ability Matrix please refer to:

- Functional Requirements (pg. 10)
- Subsystems (pg. 14)
- Quality Requirements (pg.16)

Functional Requirement No.	Subsystems				
	S1	S2	S3	S4	S5
R1.1.1	x				
R1.1.2	x				
R1.1.3	x				
R1.1.4	x				
R1.2	x				
R1.3	x				
R1.4	x				x
R2.1.1	x	x			
R2.1.2	x	x			
R2.1.3	x	x			
R2.2.1		x			
R2.2.2.1		x			
R2.2.2.2		x			
R2.2.2.3		x			
R2.2.2.4		x			
R2.2.3		x			
R3.1.1	x		x		
R3.2.1	x		x		
R4.1.1	x			x	
R4.1.2	x			x	
R4.2.1	x			x	
R5.1.1	x				x
R5.1.2	x				x
R5.2.1	x				x

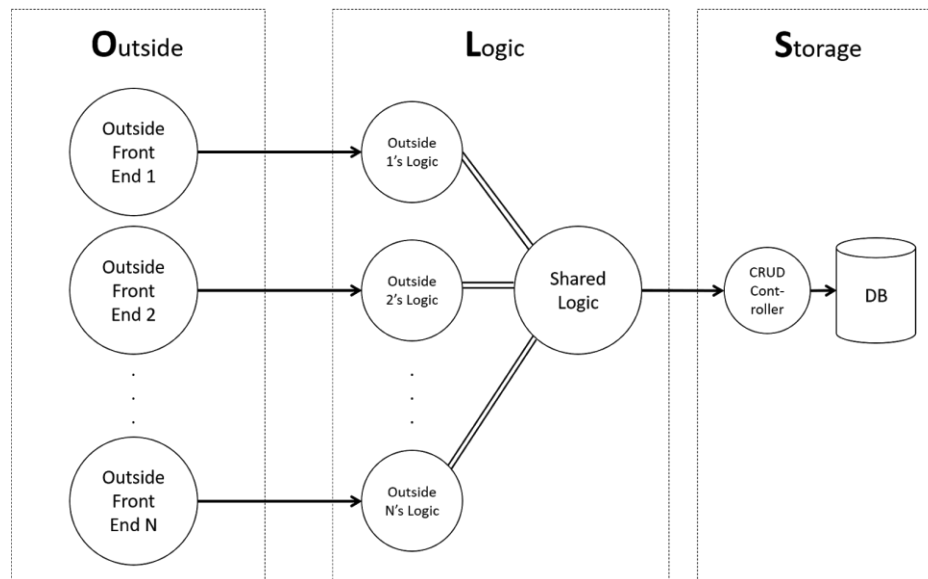|	Subsystems				
Quality Requirement No.	S1	S2	S3	S4	S5
Q1	x				
Q2.1	x	x	x	x	x
Q2.2	x	x	x	x	x
Q3	x	x	x	x	x
Q4.1		x	x	x	x
Q5.1	x				
Q5.2	x	x	x	x	x
Q5.3	x				
Q6	x	x	x	x	x

Deployment Diagram
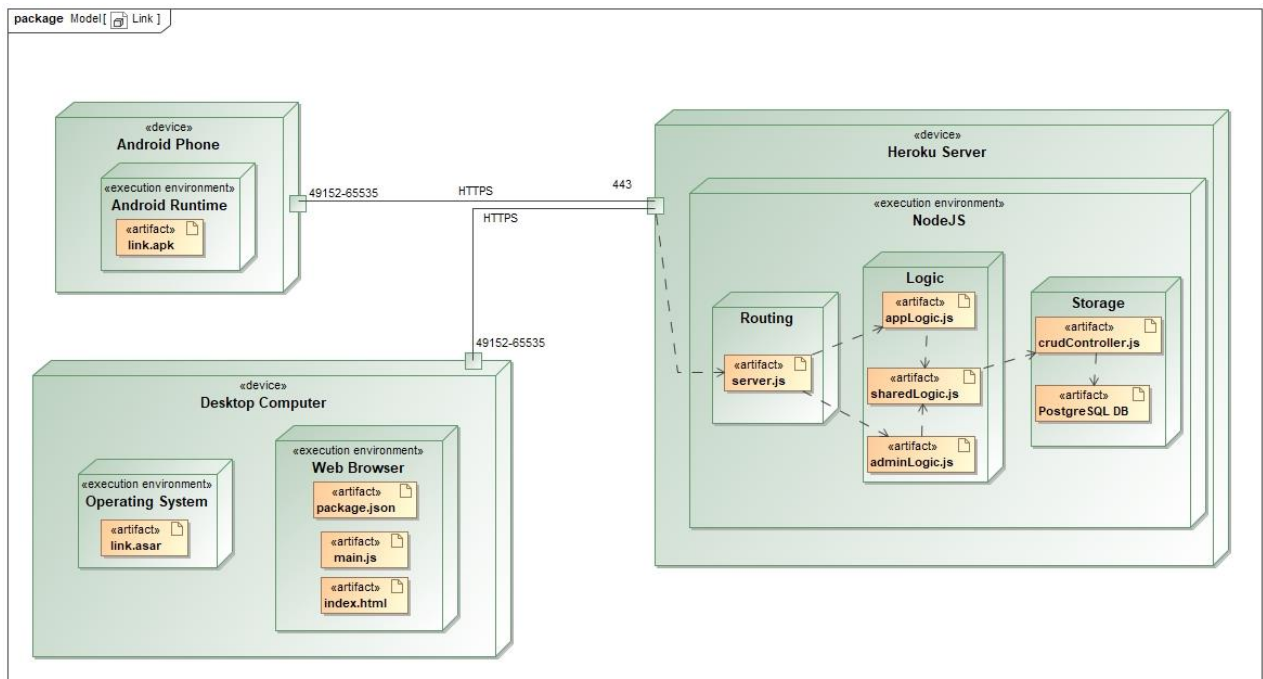
Architectural Designs and Requirements

The OLS Architecture Design Pattern

For the Link project, we have created our own architectural design pattern, which we believe is not only useful for the Link project but for many currently existing software applications and for future software applications. Our custom architectural design pattern is called **OLS**, which stands for **O**utside, **L**ogic, **S**torage. **OLS** consists of 3 components (O, L and S) and each of these components uses an architectural design pattern within themselves. The below diagram illustrates the core components of **OLS**:

The 3 distinct components of **OLS** (O, L and S) can be described in the following way:

Outside

Represents:
The Outside component represents the various front-end subsystems, designed for the users of the project. Each outside front-end is designed for a specific user type and may be hosted on a different type of device.
The choice of front-end framework/languages/model is dependent on the project and which framework would suit the project well. The only thing required is the ability to make requests to an endpoint, e.g. an HTTP request.

Communication:
These outside front-ends will only know of their associated Logic sub-component (located in the greater Logic component), which means they will only need to communicate with this one sub-component and use the API functions defined in that sub-component, without worrying about the rest of the architecture.
This one location of communication greatly simplifies requests sent out by this outside front-end, as it only must know the API functions of one sub-component and the location of that sub-component.

Architectural Design Pattern:
The architectural design pattern used here can vary, allowing a software architect to make a good problem-dependent choice.
For example, a variant of the **MVC architectural design pattern** could be used here, as it enables Views to be created with appropriate Controllers (as this is the 'front-end' component, this is

important) and the Model could be the connection to the Logic sub-component, fetching relevant data in storage and making appropriate changes to the data in storage.

Logic

Represents:

The Logic component represents the API's used by the various front-end subsystems, with each front-end interface having an associated sub-component (API) in the greater Logic component (which all their requests sent are sent to).

These sub-components utilize a single sub-component, called Shared Logic, which stores a collection of useful functions to be used (e.g. regex validation functions, password hashing functions) as well as a collection of functions which deal with, strip and extract requests coming in and also formulate the responses to these requests. Shared Logic also provides the connection to the Storage component. This grouping of functions into Shared Logic allows each sub-component to only have to implement the required functionality it needs to present to its associated outside front-end and not all the necessary pre and post-processing tasks, simplifying the coding of these sub-components. Shared Logic also allows certain standards all sub-components should follow to be defined in one place, in the Shared Logic sub-component, simplifying standards updates and increasing maintainability and updating.

Communication:

These sub-components in the Logic component only need to communicate with their respective outside front-end and with the Storage component, and both of these communications is done through the Shared Logic sub-component, meaning these sub-components can provide functionality regardless of the location of the outside front-end and the Storage component - they only need to know where Shared Logic is.

This one location of communication greatly simplifies the responses that the sub-components need to create, as well as how they would use the Storage component, as it is done through one channel.

Architectural Design Pattern:

The architectural design pattern used here is the **N-tier/Layered architectural design pattern**. There are approximately 3 layers used in this Logic component: a *Routing* layer (which is implicit as well as trivial, simply directing an outside request to an endpoint on a sub-component to the appropriate sub-component, so it can deal with the request), the *Sub-Logic* layer (which contains the sub-components that deal with requests routed to them by the *Routing* layer), and the *Shared-Logic* layer (which contains the Shared Logic sub-component, providing simplifying functionality to the sub-components in the *Sub-Logic* layer, as well as receiving the responses from the *Sub-Logic* and sending them back to the relevant outside front-end, and finally also receives requests to fetch/update data in the Storage component from the *Sub-Logic* layer and forwards it to the Storage component).

Storage

Represents:

The Storage component represents the persistent storage used by the project, in the form of a database and an API wrapper (called CRUD Controller) which facilitates controlled access into the database and controlled modification of the database.

The database can use any DBMS appropriate to the application, and CRUD Controller must provide functions to create, view, update and delete the data in the database, with parameters of the logical data that needs to be used. This way, only the CRUD Controller needs to know the exact structure of the database and which language to use to query the data, decoupling the rest of the project from the DBMS used. All communication into the database is therefore received by the CRUD Controller and a response returned by CRUD Controller.

Communication:
The CRUD Controller provides an interface to the sub-components of the Logic layer (through the Shared Logic sub-component) to access the database, and directly accesses the data in the database using the correct query language. It does not need to know where the Logic layer and its sub-components are, only where the database is located.

Having one entry point to the Storage component (the CRUD Controller) simplifies the access into the database and allows the DBMS and database scheme to be changed with only CRUD Controller having to update its code. This logical CRUD of the data in the database allows the rest of the project to communicate logically with the Storage component, simplifying the querying of the database, only through one point of communication.

Architectural Design Pattern:
The architectural design pattern used here is the **Persistent Storage architectural design pattern**. The database is where we store our data persistently, and the CRUD Controller is how we access this persistently stored database. This is the standard approach to using a database in this type of application and has been tried and tested with good results. Having more than one database (distributed) or changing the scheme of the database is also possible, all that has to update is the CRUD Controller (which 'translates' the logical requests for data into the explicit queries for data and returns the result).

Advantages of using OLS

The core components of OLS were described above, however, this general description hides the reason why we created and chose to use this architectural design pattern.

OLS Integrates Easily

The Link application started off with a simple description - a mobile app and an admin portal, paired to a backend server. However, we soon discovered that our application would have to integrate with many other systems and interfaces, not just an admin interface and the mobile app.

For example, we need to integrate our system with the access control system of the EPI-USE offices and other access control systems of other companies. We also needed to integrate our system with WiFi routers of the companies in their buildings. It would be difficult to understand the inner workings of each system to be integrated into, so instead, for every system to integrate with (i.e. each Outside Front-End), we can define a new sub-component in the Logic component, which that Outside system would use to gain access into our data in a controlled fashion.

This approach means that the outside system to integrate with only requires one additional API, meaning we don't have to change any existing ones and this API will solely serve that outside system. A new integration will therefore add as minimal code as possible.

OLS Prevents Re-Definition and Standardizes

With the use of the Shared Logic, any and all general functions which all the Sub-Logic components need can be defined in one place (in the class inside of Shared Logic). This means they don't have to redefine the methods in their component, and this also means that if we need to change what one of those general functions do, it can be changed in one place and all other components will be able to continue using the functions without any change.

Not only does using OLS prevent re-definition (as described above), it also allows for certain standards for the API's to be specified once and implemented once (in Shared Logic). These standards can include the format of JSON to be sent in the body of a POST request, the variables to use in the classes to access the body of the POST request after extraction and parsing, the formatting of responses to requests, the way we validate websites, cellphones, emails, passwords, etc. The entire system, using OLS, can therefore utilize these standards with no exception, and they only must be defined once.

This standardization avoids different software engineers doing their own formatting and following their own standards, differing from the team's approach and eventually leading to conflict and code mismatch errors. In OLS, standardization and definition are defined once, in one place - as semantically they should be.

OLS Separates the Database

As our data grows bigger and bigger, new types of databases have been developed to improve the throughput of our database systems (like NoSQL). However, many of the applications present today started off with Relational Databases and had to change much of their code in a very non-obvious and non-trivial fashion, resulting in much time and money spent on effectively only upgrading the database system.

OLS, due to the use of CRUD Controller, allows an interface to the database to be used by the other components, with 'logical' queries being executed (for example, not 'SELECT * FROM tblEmployees WHERE….', instead, one would call getEmployee(..)). This means that the structure and scheme and type of the database is completely decoupled from the rest of the system, allowing the database to be completely switched out to a new type/scheme or the structure to change, with the only changes needing to take place in the CRUD Controller.

This means that updating our code to use a new database system is easy - we only need to look in one component and update that, maintaining the same interface. So, when, in the future, new database systems come out that handle our data in new and innovative ways, to update the code of

an OLS-architecture based system would be easier than other architectures and save both time and money.

OLS is in Demand
The usage of mobile applications and admin interfaces, paired with a backend API, is very prevalent in the field of software engineering is high. These types of projects are common and in demand, and these created systems needing to be integrated with other systems is happening more and more in present-day software society, as all our various systems get bigger and talk to each other more and more (e.g. the Internet of Things). OLS provides an easy way to conceptualize, design and implement these types of applications, providing lower coupling and better maintainability than other traditional architectural design patterns.

Our Use of OLS

After the above description of OLS, we can now illustrate how we are currently using the OLS architectural design pattern in our project:
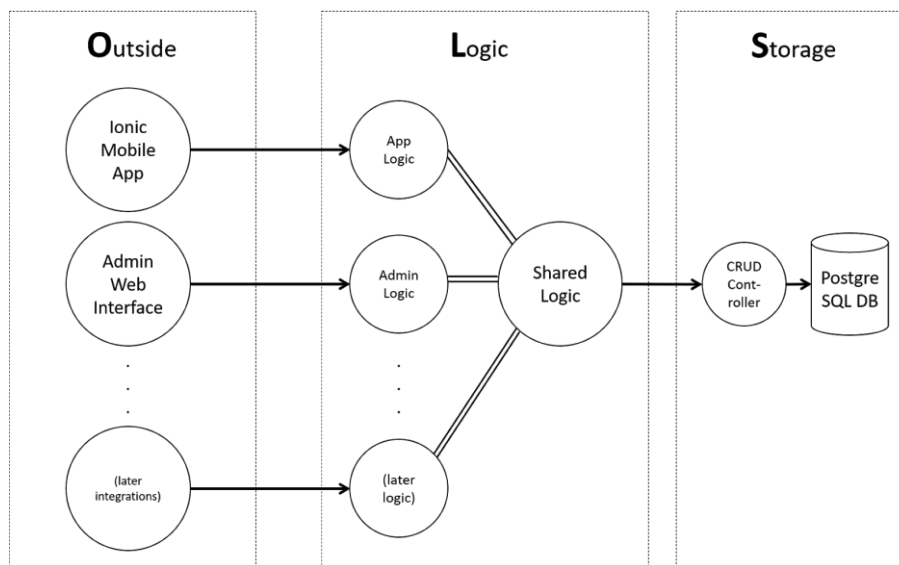
Our Outside component consists of:
- Our Ionic mobile application, to be tested with Android devices initially
- A web interface for Link admins and company admins to access the system
- (possible further systems that integrate with us and our system)

Our Logic component consists of:
- App Logic, the API that the Ionic mobile app will request to
- Admin Logic, the API that the Admin web interface will request to
- Shared Logic, a class that extracts and parses the JSON of a certain standard in HTTP POST request bodies, provides API Key validation and Login functionality, provides some general validation functions (e.g. to determine valid emails), provides functions to format the response to a request in the set standard and provides access, through the CRUD Controller, to the PostgreSQL database.

Our Storage component consists of:
- CRUD Controller, the function API that all other sub-components in the Logic component will use to logically query the database
- A PostgreSQL database and DBMS, enabling us to use the entity-relationship capabilities of a relational database to simplify and optimize our data storage

Constraints

The architectural constraints are separated based on the three major components of the **OLS** architectural design:

Outside (O) Constraints

- Each outside component must communicate with its respective Logical Controller responsible for the server-side logic
- Each outside component must comply with the data transfer requirements set by its Logical Controller, this is set by the producers of the individual controller and is not necessarily standard between Logic Controllers.
- Each outside component must have an end point to communicate to its corresponding Logic Controller which can differ based on the different types of outside components

- ✓ **Disadvantages:** High coupling between the Outside Components and their respective controllers
- ✓ **Advantages:** Separation of Concerns

Logic (L) Constraints

- A single point of failure exists in the Shared Logic controller in the Logic component. All logical controllers that make use of this dependency have the potential to crash if the Shared Logic controller fails.
- All access to data in the Storage Component must be sent to a CRUD Controller, no direct database access is permitted.

- ✓ **Disadvantages:** Multiple logical layers increases integration complexity and can produce increased communication overhead, however this allows for Separation of Concerns.
- ✓ **Advantages:** Separation of Concerns between the logic of different outside components, decreased coupling within the Logic Component.

Storage (S) Constraints

- **Input Constraints**
  - All communication with the Storage component must enter through the single point of entry known as the CRUD Controller.
  - All input data must comply to the standard data transfer format of the architecture's implementation, which is the JSON format. This standard format can change in other implementations of the architecture.

- **Output Constraints**
  - All responses from the CRUD Controller must comply with the specified standard JSON format.
  - If the implementation makes use of multiple (distributed) CRUD Controllers, then these controllers must have a protocol in place which keeps the database in a stable state. Thus, the output of two identical queries should not be different unless in between these two queries there was a modification to the database.

- ✓ **Disadvantages:** As it currently stands, there is a single point of failure for data access to/from the persistent storage. This is rectified by the ability for the Storage Component to have multiple distributed controllers with the same functionality which will provide a near-

perfect distributed access for the Logic Component. This solution however leads to the need to update multiple instances of the CRUD Controller in multiple places which could be a daunting task. Orchestrated deployment tools such as Kubernetes can be used to solve this auxiliary problem.

✓ **Advantages:** These input constraints allow for increased security as the CRUD Controller acts as a proxy between the Logic Layer and the actual database. This proxy ensures that all queries that reach the database follow the specific constraints imposed by the database management system (DBMS). The CRUD Controller is responsible for sanitizing all queries and constraining the number and type of queries that may be performed on the database by the Logical Layer (L) requester.

Technological Decisions

Version Control: Git – Hosted on GitHub

We will be using git and GitHub as it gives us all the required functionality for version control for our project. It allows us to track changes in our code across various versions such as who did what, when and where those files are stored.

Project Management: ZenHub

ZenHub allows for seamless integration with GitHub, allowing us to have our project management tool in the same location as our repository. It has a very easy-to-use interface for creating new features as well as moving features across different stages of its lifecycle.

Continuous Integration: Travis CI

Travis CI is a hosted continuous integration service which is used to build, and test software projects hosted on GitHub. It is easy to use, and we don't have to think about Travis CI once it is set up. When we commit a change, a build will be triggered and Travis CI will test the integration, if it passes the test, then the integration went fine, if not then Travis will let us know that something went wrong.

Testing: Jasmine, Karma

Jasmine and Karma are open source testing frameworks for JavaScript. They allow for simple integration with our Continuous Integration tool (Travis CI). These testing frameworks allow us to easily execute tests locally during development in order to ensure our code is working and producing the correct outputs.

Front End Framework: Angular 7 and Ionic

We have decided to use Angular and Ionic for our Front-End Framework as it is a cross-platform framework, allowing us to create a single app which will work across various platforms such as PC/Android/iOS (some issues surrounding iOS payments however) without any additional development time for each platform. Using Angular makes it possible to build fast and scalable applications, coupled with Ionic allows for the creation of a powerful and feature-rich application.

Hosting: Heroku

We will be utilizing the free hosting service provided by Heroku as it allows for easy integration with GitHub allowing us to auto deploy our Master Branch (located on GitHub) to Heroku.

Data Storage: Postgres Database

Heroku allows us to utilize a Postgres Database provided the hosting provider. This allows for easy integration with our application, as well as easy setup of the database itself.

Server: NodeJS

We decided on using NodeJS as our back-end server it has been proven to be a robust language allowing for increased speed and performance with comparison to other languages. There are a vast number of tools that we can utilize within NodeJS in order to complete our project. As a plus side our team is experienced and comfortable in using NodeJS as our server language

Mobile Technology: NFC – enabled Devices

For the purpose of our project we require NFC-enabled devices. This will be used for the core functionalities of our project for sharing data and information between mobile devices.