1.3 The Design Process

Now that we have our requirements and business concerns - we are ready to start designing the architecture of the application. We will be following the ADD approach to design the application - splitting the process into multiple iterations. During each iteration we will be selecting drivers as our point of focus.

1.3.1 ADD Step 1: Reviewing Inputs

Category	**Details**
Design purpose	The purpose of this design is to make a comprehensive design that meets the requirements of the application as well as sets the application up in a way that it is easily scalable and modifiable.
Primary functional requirements	The primary use cases are: UC-1 UC-2 UC-3 UC-4 UC-5
Constraints	All constraints are used as drivers.
Architectural concerns	All architectural concerns are used as drivers.

Quality attribute scenarios prioritized:

Quality attribute Scenario	**Importance to the customer**	**Difficulty to implement According to the Architect**
QA-1	High	Medium
QA-2	High	High
QA-3	High	High
QA-4	Low	Medium
QA-5	Low	Medium

QA-6	High	Medium

1.3.2 Iteration 1: Establishing an Overall System Structure

The goal of this iteration is to meet architectural concern CRN-4 which is to develop an overarching architecture for the application.

1.3.2.1 Step 2: Establishing Iteration Goal by Selecting Drivers

This iteration is driven by meeting architectural concern CRN-4: Develop an overarching architecture for the application, however we kept in mind all major drivers that could affect the overarching architecture of the application including all major use case scenarios. The following drivers were also used as main drivers during this iteration of the design process.

Quality Attributes:

- QA-2: Performance
- QA-3: Availability
- QA-4: Modifiability

Constraints:

- CON-1: The application must run as intended on Chrome, Firefox, Safari and Edge.
- CON-3: Before processing a payment, the system must be certain that the item/items being purchased are in stock
- CON-4: A relational database must be used

Concerns:

- CRN-1: Designing and implementing a distributed system that can easily scale with a growing user base.
- CRN-2: Leveraging the team's knowledge in multiple domains of web development. Frontend, backend and devops.
- CRN-3: Appropriately dividing up tasks within the development team.

1.3.2.2 Step 3: Choosing Elements of the System to Refine

Since this is the first iteration of our design, the element that we are 'refining' is the EMarketPlace system in it's eternity.

1.3.2.3 Step 4: Choosing Design Concepts That Satisfy the Selected Drivers

Decision 1: Application type

Design Decisions & Locations	Rationale
Create a web application using the client/server model that can run on modern browsers.	This architecture decouples our system into two main sections – the frontend and the backend. By doing this we can make changes to our view without affecting our controllers or models. It also allows us to make changes to our backend without having to make too many changes to our view/frontend. There is also the added benefit of making our system widely accessible. Because this is a web application all you need is a computer/mobile device with a browser and an internet connection in order to be able access it. Both of which are typically readily available nowadays.

Alternative	Reason for Discarding
Mobile application (ios/android)	Although we definitely are planning to create a mobile version of our application, due to a lack of resources we believe that it makes more sense to start with a web application so that our application is accusable by a wide range of devices rather than just one platform.

Desktop application	There are two issues with this architecture. The first is that our application is limited to either Macs OS, Windows or Linux machines. This will eliminate two desktop operating systems from our user base as well as all mobile devices. The second issue is that people will need to download our application to their desktop, which is something that people only do if necessary from our experience. It is much easier to get a user to visit a website rather than download an application. With modern browsers becoming more and more capable each day we don't see a real advantage to desktop applications anymore.

Although we are not making a traditional 'rich client' application - our architecture will closely resemble the architecture of one. This is because we will be using a modern frontend framework that is robust and allows for more processing on the client side. We will still be using a three tier architecture with the client, logic, and the data tiers all present in our architecture. However the lines between the client and logic tiers are more blurred than traditionally seen in our design - this is intentional.

Decision 2: Backend language

Design Decisions & Locations	**Rationale**
Build the backend using Node.js	For the backend we chose Node.js because that is our strongest backend language that is very well documented and tested. It makes working with third part services and SQL easier through frameworks like sequelize and

Alternatives	**Reasons for Discarding**
Python	We considered Python as our backend language of choice, however we decided against it for two main reasons. The first is that most of our expertise lies within Node.js and javascript. The second reason is that we know that python - although easy to use - is not the most performant language out there. Given our priority for high performance we decided to not use python for our backend language.

Decision 3: Deployment Service

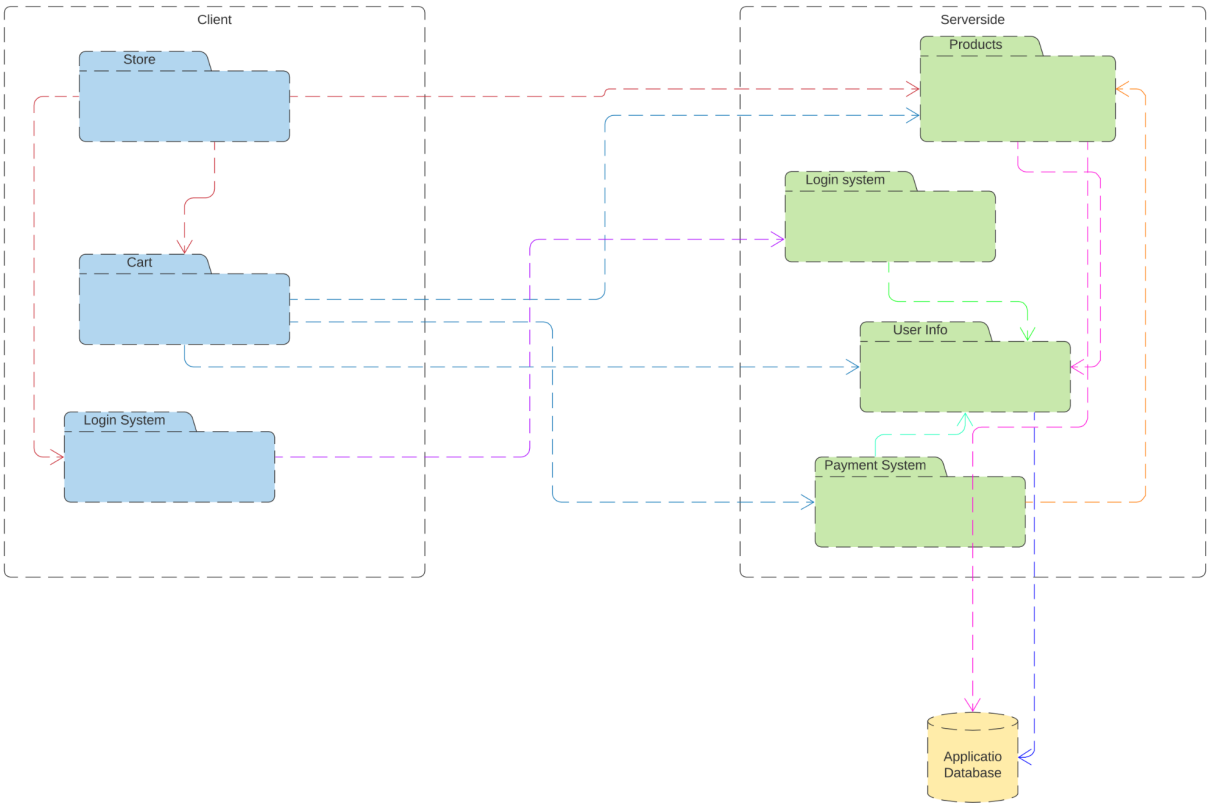
Design Decisions & Locations	**Rationale**
Deploy the system using Firebase	We decided to use firebase to deploy our application because this makes it very easy for us to scale our system appropriately.

Alternatives	**Reasons for Discarding**
Azure, AWS	We chose firebase over these two platforms because firebase is very well documented and we can easily switch over to google cloud platform if that fits our needs more at any time.

Decision 4: APIs

Design Decisions & Locations	**Rationale**
Rest API	We decided to use rest APIs to communicate with our frontend - returning JSON data - as this will make our system highly modifiable and decoupled from our view. This also means that we can expand our application to other platforms (IOS/Android) without having to make too many changes to our backend.

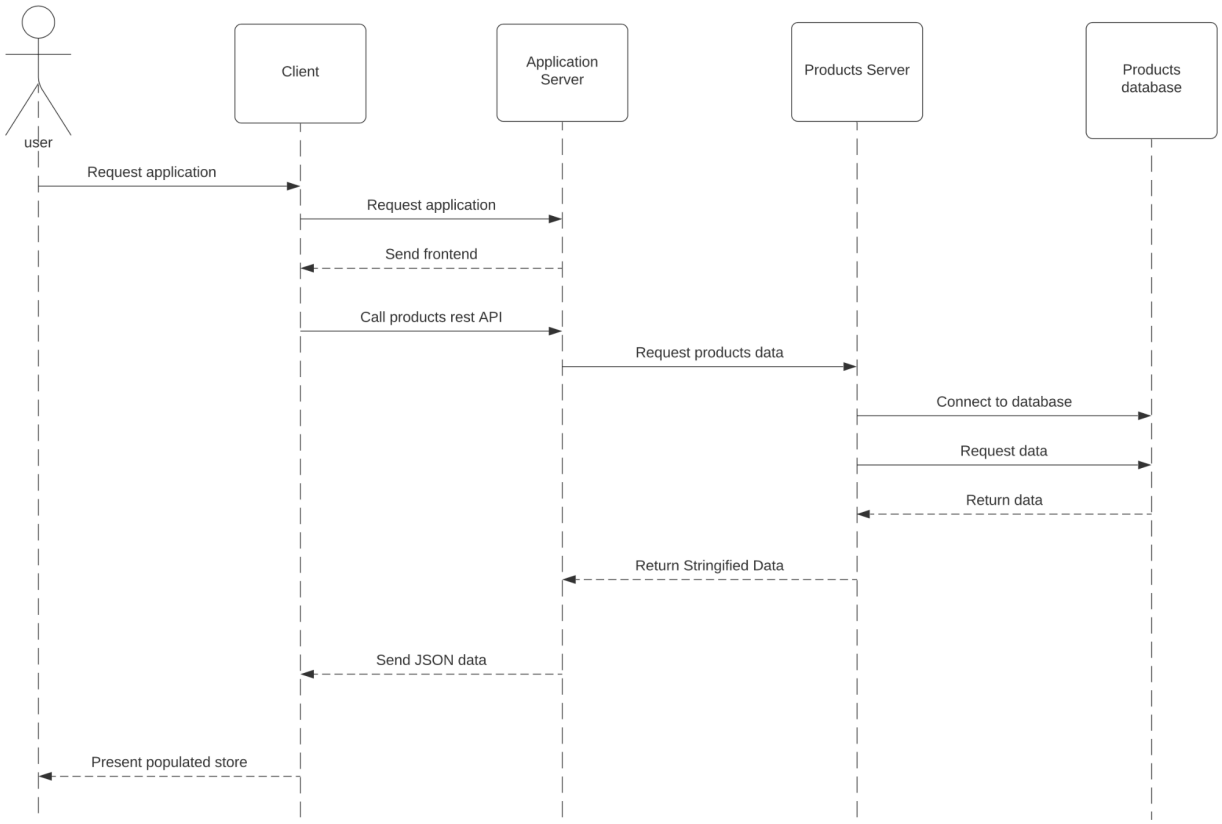
1.3.2.4 Step 5: Sketch Views and record Design Decisions



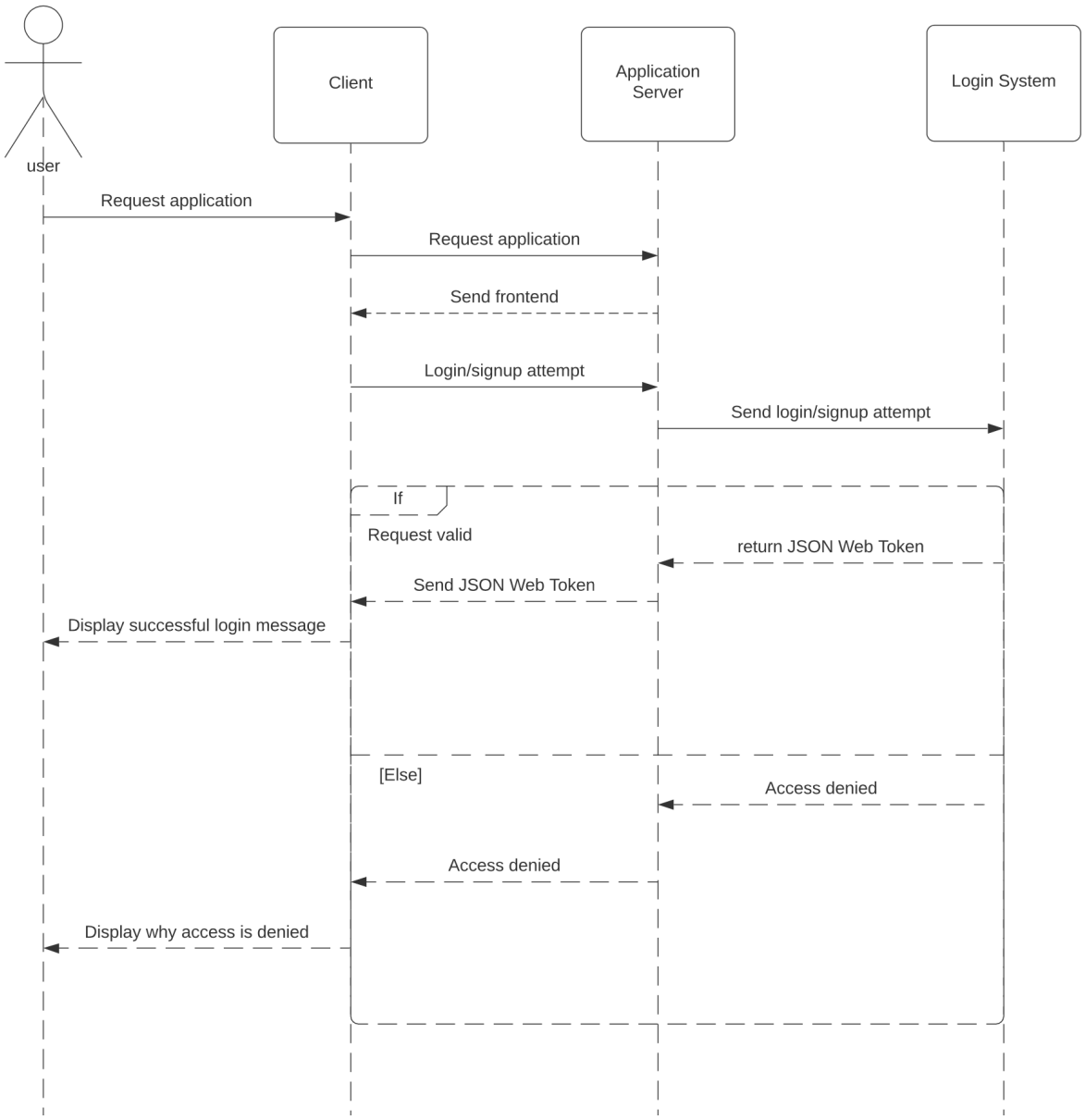
Element	Responsibility
Products	All of our products are stored here along with their associated metadata. Our rest APIs that fetch and return data from the database are in this package as well. The database will be a real-time database in order to have accurate data about product availability at all times in our store.
Login System	This is where login requests are processed. Account information is also stored here.
Payment System	This is where payments that our users make are processed as well as payments we make to our sellers on the site.
User Info	This is where we will have APIs that are associated with the users. This includes if they are vendors or customers. As well as other information like session specific

	information like what is in their cart.
Store	The store is the main screen for our application. This is where the user can see all items on offer on our platform. The store will fetch all of our products from the products package on the backend.
Cart	This is where the user adds items from the store and checkouts out to pay for them - if available.

Sequence diagram for requesting the website and the products that populate the site



Sequence diagram for login system



1.3.2.5 Review iteration

In this iteration we set out to establish an overarching architecture for our application. We believe that the results of this iteration reflect that. We came up with an application type - web application - that will help drive the rest of the decisions we make during the design process. The main elements of the application were also determined along with how they will communicate with each other.