

**Low-Level ARCHITECTURE and Data Models**

**P04: Dairy Farm Management System**

****

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| 22100125 | Abdullah Saleem |
| 22100072 | Khawaja Junaid |
| 22100070 | Furqan Athar |
| 22100209 | Saad Qadeer |

**Submitted to: Sir Waqar Ahmad**

**Table of Contents**

[1. Introduction 3](#_Toc89110578)

[2. System Architecture 4](#_Toc89110579)

[2.1 Architecture Diagram—As it is in the prototype code 4](#_Toc89110580)

[2.2 Architecture Diagram—As it should-be 5](#_Toc89110581)

[3. Data Models 6](#_Toc89110582)

[4. Tools and Technologies 9](#_Toc89110583)

[5. Who Did What? 13](#_Toc89110584)

[6. Review checklist 13](#_Toc89110585)

# Introduction

In attempts to modernize the dairy industry in Pakistan, which is one of the greatest producers and consumers of milk and other dairy products in the region, our proposed and developed system will present a solution to better manage the dairy farms and optimize sales and help in better record keeping.

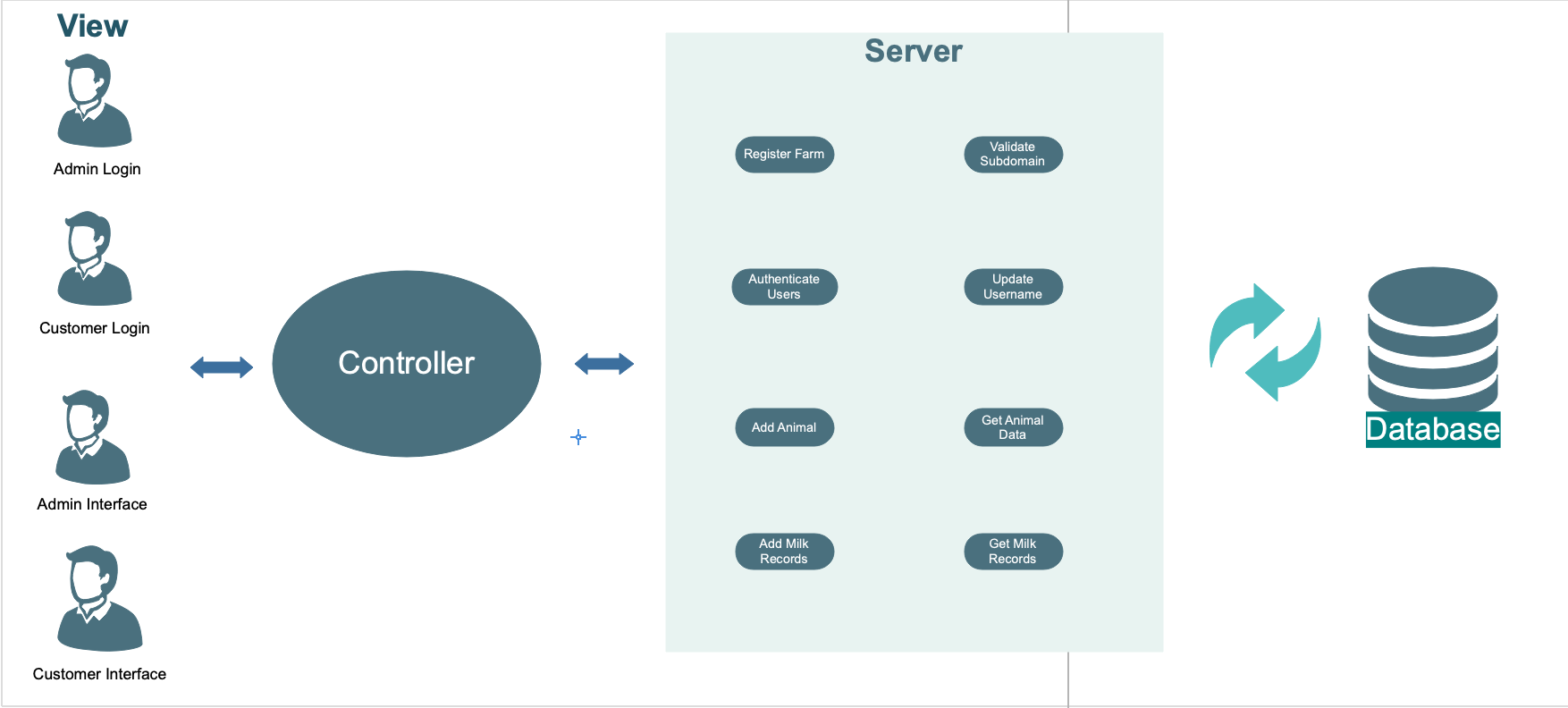
Furthermore, multiple dairy farm owners will have the opportunity to set up their accounts on our management system, as we aim to develop a multi-tenant software, which will aid them in getting more customers and for the better management and record keeping of their farms. The system will allow the customer to keep track of daily milk production, sales, expenses and maintenance. Moreover, the system will also provide an interface for the customers of the dairy farm to check their delivery and monthly invoice.

Potential users of our management system include dairy farm owners (who wish to digitize their records and keep an updated track of their farms) and the customers who wish to order dairy products, be it for domestic or commercial use.

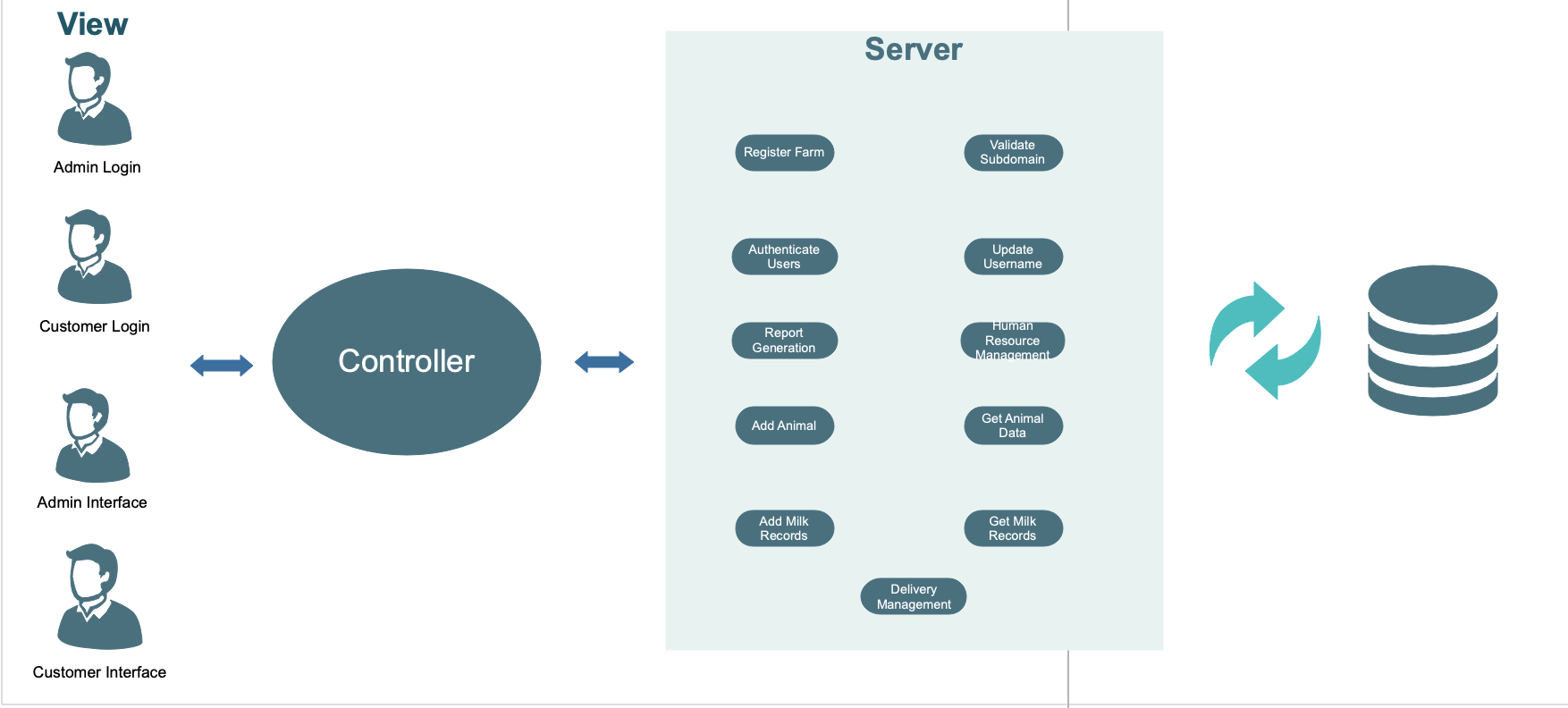
The main purpose of this product will be to provide multiple tenants a way to see various trends in their milk production, expenses and income streams and to make better decisions for the future.

# System Architecture

## Architecture Diagram—As it is in the prototype code



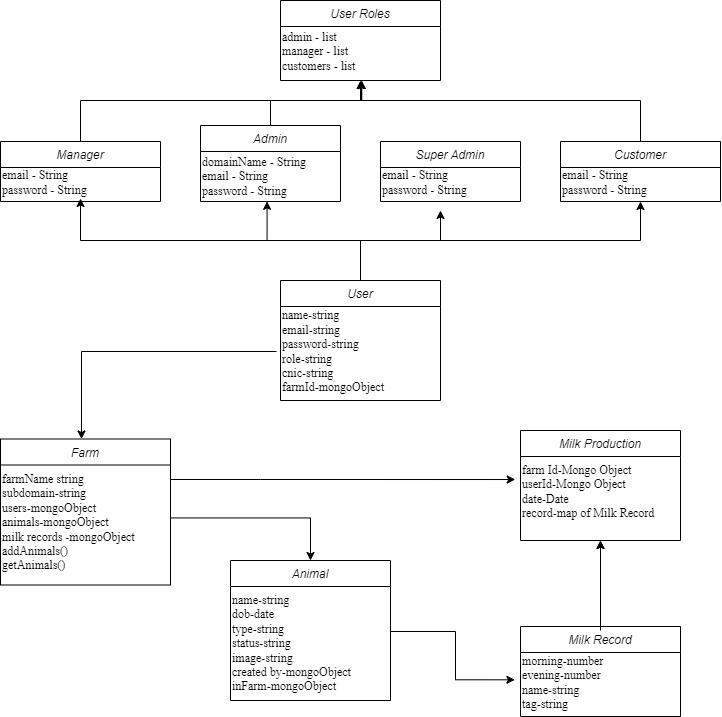
## Architecture Diagram—As it should-be



We started with the layered architecture when we were designing our software. But when the coding process started, we realized that MVC (Model View Controller) architecture would be best for our system as it would be more maintainable. The main reason for suggesting the layered architecture in the first place was to ensure security of data. MVC not only provides security of data but keeps the code in such a way that is more maintainable as compared to that in layered architecture. The technology stack that we are using is MERN stack and the full functionalities of MERN stacks can be unlocked using the MVC architecture.

The class descriptions are the same as before.

# Data Models



**Animal:**

This data model stores information for farm animals in a particular farm with a unique tag for each animal.

* **name-string**: This adds a name for the animal for ease of identification.
* **tag-string**: a unique id associated with each animal
* **dob-date**
* **type-string**: If the animal is a cow, heifer or etc.
* **status-string**: if the cow is milking, or not milking or active.
* **image-string**: This adds the picture of the animal to the database
* **created by-mongoObject:** stamped by the database
* **inFarm-mongoObject**: stamped by the database

An animal may have multiple milk production records for every different day. But for a single date, there will only be one record, having the evening and morning production.

**Farm:**

Each tenant can register their farm with a unique business domain name.

* **farmName - String**: Name of the farm
* **subdomain - String**: CNIC of the tenant that is creating the farm
* **users - Mongo Object:** Users on the farm
* **animals - Mongo Object:** Animals on the farm
* **milk records - Mongo Object:** Milk Production record of the animals

A farm can have only 1 Owner (1-1).

An Owner may have multiple farms present in the database. (1-N).

Similarly, a farm may have multiple animals (1-N).

A farm may have a cumulative milk production record linked to their respective farm id.

**User:**

* **name - String:** Name of the user
* **email - String:** Email address of the user used to register the account
* **password - String:** Password for account access
* **role - String:** Role of the user either manager/ tenant/farm owner
* **cnic - String:** The CNIC Number of the user
* **farmId – Mongo Object:** A schema for the database

The User Class is a parent class, it can have multiple subclasses, making it a (1 - N) mapping. Subclasses of Admin, Manager, Employee and Customer inherit from the User Class. Our Model assumes that there is only one super admin, in that case the relation would become (1-1).

**Milk Records**

* **morning - number:** The amount of milk produced in liters in the morning.
* **evening - number:** The amount of milk produced in liters in the evening.
* **name - string:** The name assigned to an animal, may or may not be unique.
* **tag - string:** The tag is a unique id assigned to an animal, must be unique.

Milk Record are for a single entry per day. This Object is used to cumulate the Milk Production object, provide detailed information regarding milk production over a longer period.

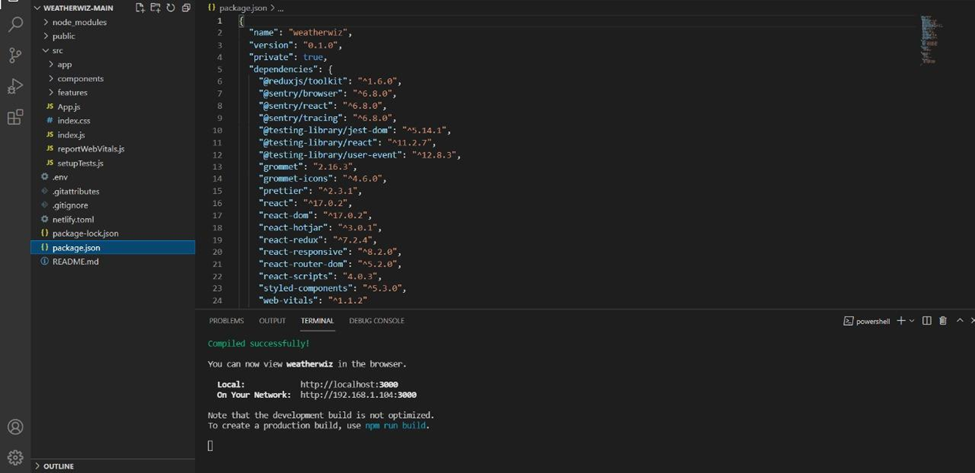
**Milk Production:**

* **farmId - Mongo Object:** To keep track of milk production of a certain farm
* **userId - Mongo Object:** An object to allow for a multi-tenant system, (yet to be deployed)
* **date - Date Object:** Date to keep track of milk production on a certain day.
* **record - Object of type Milk Record:** Schema mapping of object Milk Record

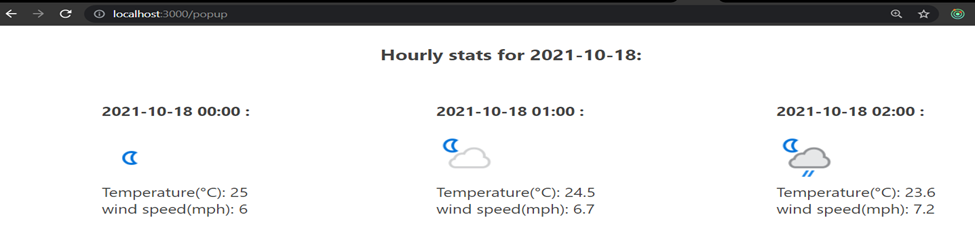
# Tools and Technologies

* The tools and technologies which will be used in our system are as follows:
* [ReactJS](mailto:ReactJS) version react@16.14.0 for front end development of the webapp.
* Front end tools like [Redux sagas](https://redux-saga.js.org/) redux@4.0.4 for state management and API call management.
* [React-router](mailto:React-router) react-router@5.2.0 a tool to navigate between components.
* [NodeJS](https://nodejs.org/en/) version v14.17.6 for backend development.
* [MongoDB](https://www.mongodb.com/) a NoSQL serverless database MongoDB4.4
* [Heroku](http://www.heroku.com)  has been used to deploy and host our website
* [Google Drive](https://www.google.com/drive/) for document management.
* [GitHub](https://github.com/) for code logistics.

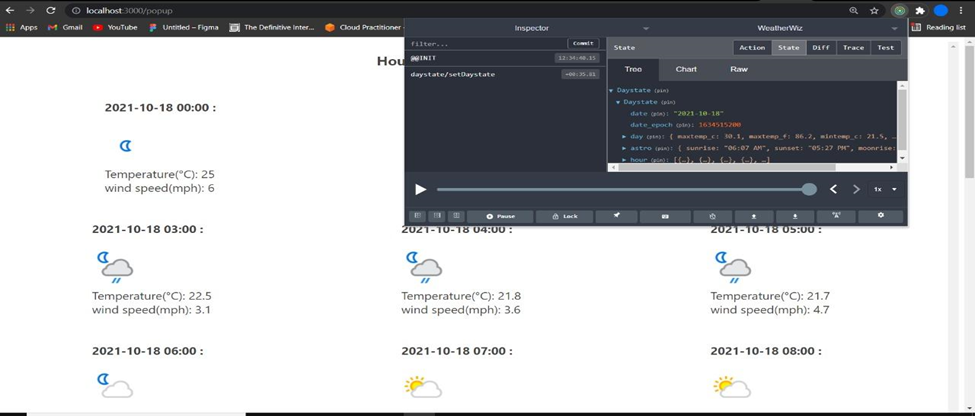
**ReactJS** app running below are the packages installed which show ReactJS, redux, and router dom from VScode.



The path shows the change in components have taken via “**React-router**”

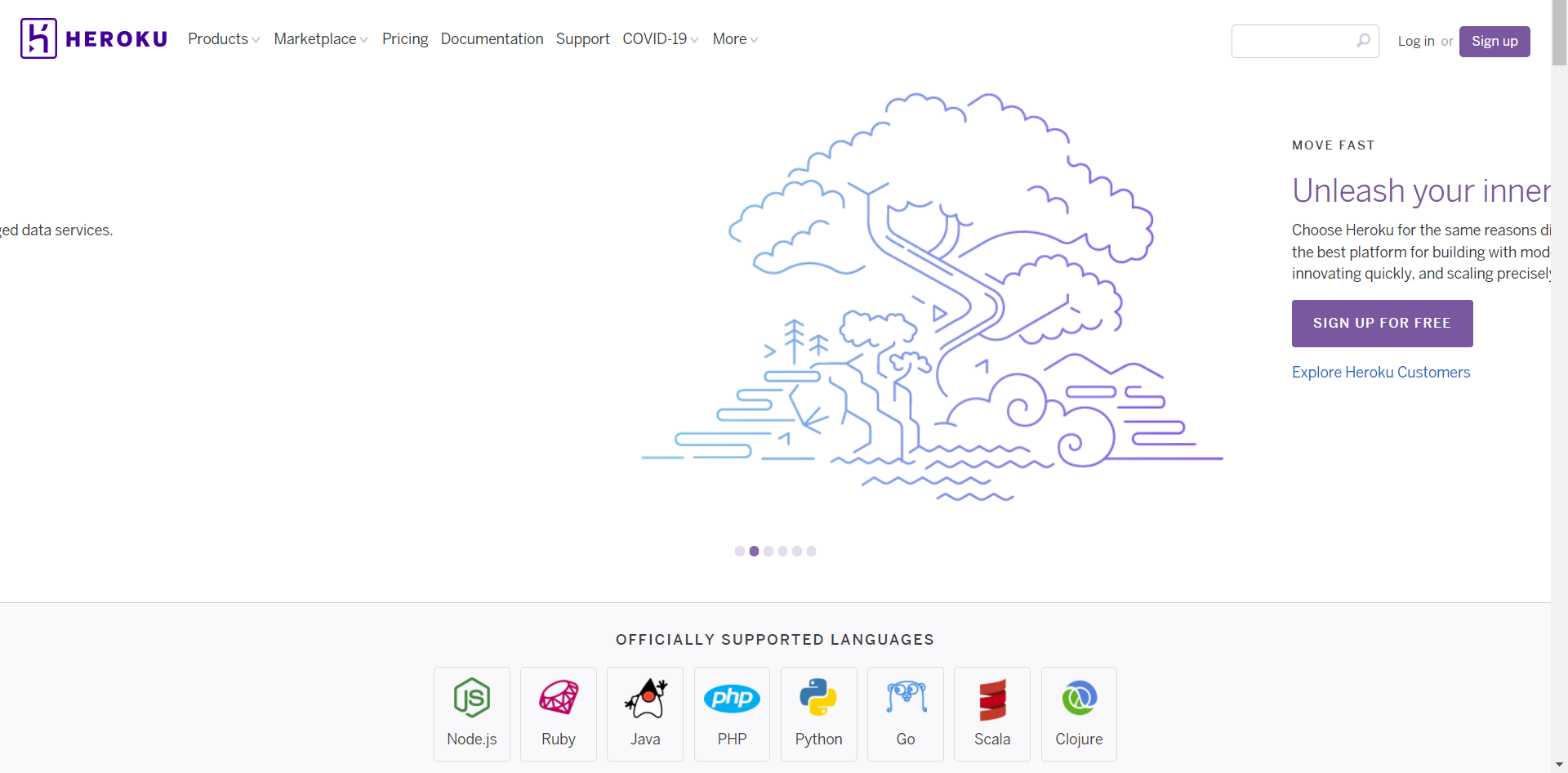


The **React redux** store responsible for state management is logging states fetched from the API call or updated within the component.



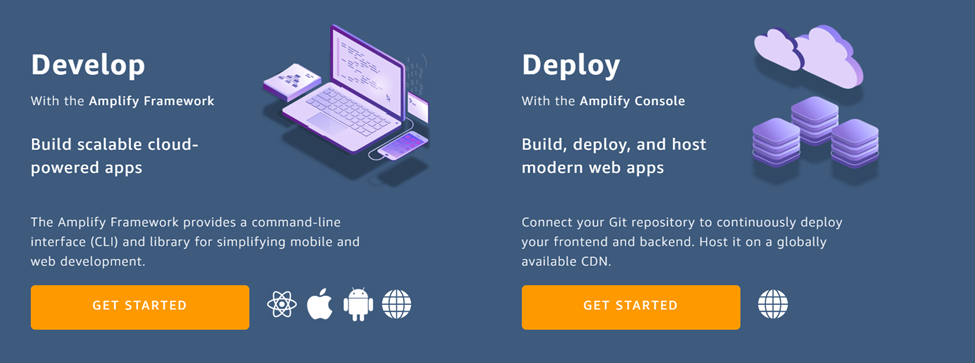
**Final Deployment Platform**

Initially we have deployed our web app on Heroku.



Final deployment may be on AWS.

**AWS amplify console** provides hosting facility for webapps. websites can be hosted from github repositories. It has a git-based workflow that means it supports continuous deployment. You will not be charged for services in the free tier.



# Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| Khawaja Junaid | ERmodel and tools and technologies |
| Saad Qadeer | ERmodel and tools and technologies, formatting |
| Furqan Athar | Architectural diagram as it is in prototype |
| Abdullah Saleem | Architectural diagram as it should be, intro |

# Review checklist

Before submission of this deliverable, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

|  |  |
| --- | --- |
| **Section** **Title** | **Reviewer Name(s)** |
| Khawaja Junaid | Tools and Technologies |
| Saad Qadeer | Data Models |
| Furqan Athar | Introduction, Architecture Diagram |
| Abdullah Saleem | Architecture Diagram |