AI BASED CROP IDENTIFICATION MOBILE APP

A MINI PROJECT REPORT

submitted

in the partial fulfillment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

B. SAI HEMANTH REDDY (17B81A05H5)

T.P.N. NIKHIL (17B81A05F2)

D.V. KRISHNA KALYAN (17B81A05C2)

Under the guidance of

MS. M. SATHYA DEVI
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CVR COLLEGE OF ENGINEERING

(An Autonomous institution, NBA, NAAC Accredited and Affiliated to JNTUH, Hyderabad)

Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),

Rangareddy (D), Telangana- 501 510

June 2020

AI BASED CROP IDENTIFICATION MOBILE APP

A MINI PROJECT REPORT

submitted

in the partial fulfillment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

B. SAI HEMANTH REDDY (17B81A05H5)

T.P.N. NIKHIL (17B81A05F2)

D.V. KRISHNA KALYAN (17B81A05C2)

Under the guidance of

MS. M. SATHYA DEVI
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CVR COLLEGE OF ENGINEERING

(An Autonomous institution, NBA, NAAC Accredited and Affiliated to JNTUH, Hyderabad)

Vastunagar, Mangalpalli (V), Ibrahimpatnam (M),

Rangareddy (D), Telangana- 501 510

June 2020

Cherabuddi Education Society's

CVR COLLEGE OF ENGINEERING

(An Autonomous Institution)

ACCREDITED BY NBA, NAAC 'A' GRADE

(Approved by AICTE & Government of Telangana and Affiliated to JNTU Hyderabad) Vastunagar, Mangalpalli (V), Ibrahimpatnam (M), R.R.District.

Web: http://www.cvr.ac.in, email: info@cvr.ac.in

Ph : 08414 – 252222, 252369, Office Telefax : 252396, Principal : 252396 (O)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project entitled **"AI BASED CROP IDENTIFICATION MOBILE APP"** that is being submitted by **B. SAI HEMANTH REDDY (17B81A05H5)**, **T.P.N. NIKHIL (17B81A05F2)**, **D.V. KRISHNA KALYAN (17B81A05C2)**, in partial fulfillment for the award of Bachelor of Technology in Computer Science and Engineering to the CVR College of Engineering, is a record of bonafide work carried out by them under my guidance and supervision during the year 2019 - 2020.

The results embodied in this project work has not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of the project guide

MS. M. SATHYA DEVI

Assistant Professor

CSE Department

CVR College of Engineering

Signature of the HOD

DR. K. VENKATESWARA RAO

Head of the Department

CSE Department

CVR College of Engineering

External Examiner

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance has been a source of inspiration throughout the course of the project.

It is a great pleasure to convey our profound sense of gratitude to our Principal **Dr. Nayanathara K. S** and **Dr. K. Venkateswara Rao, Head of the CSE Department**, CVR College of Engineering for being kind enough for arranging the necessary facilities for executing the project in the college.

We would like to express our sincere gratitude to our guide, **Ms. M. Sathya Devi, Associate Prof., CSE Dept.,** CVR College of Engineering, whose guidance and valuable suggestions have been indispensable to bring about the successful completion of our project.

We wish to acknowledge special thanks to the Project Coordinator **Dr. M. Jaiganesh, CSE Dept.,** for assessing seminars, inspiration, moral support and giving us valuable suggestions in our project.

We would also like to express our gratitude to all the staff members and lab faculty, department of **Computer Science and Engineering, CVR College of Engineering** for the constant help and support.

We wish a deep sense of gratitude and heartfelt thanks to management for providing excellent lab facilities and tools. Finally, we thank all those whose guidance helped us in this regard.

ABSTRACT

India is an agricultural economy, with most of the population's livelihood is directly or indirectly associated with farming. Since late 1960s technology has advanced at an exponential rate but that advancements haven't improved farming. With the pressure to move to sustainable, green future, we need to advance farming techniques to improve energy requirements for farming, its carbon footprint, etc.

This project is an attempt at solving a small piece of that puzzle. Using AI to provide farmers with information, suggestions, precautions, and educate farmers, we can empower the farmers to help themselves.

The proposed model is currently targeted to support farmers, to educate them, give them information about their crops and diseases. This is created such that it can improve over time with new additional data and is developed using real-world data.

TABLE OF CONTENTS

LIST OF FIGURES

I INTRODUCTION

1.1 MOTIVATION

Intelligence has been considered as the major challenge in promoting economic potential and production efficiency of precision agriculture. Modern agriculture seeks to manage crops in controlled environments that are able to improve the production of plants or duplicate the environmental conditions of specific geographical areas to obtain imported products locally.

Now, it is possible to obtain highly accurate status of crops and form reasonable decisions to manage irrigation, enrich the soil nutrition in agricultural scenes. Automatic Plant Image Identification is the most promising solution towards bridging the botanical taxonomic gap, which receives considerable attention.

As the technology advances, sophisticated models have been proposed for automatic plant identification. With the popularity of technology, by the use of smartphones, many plant photos have been acquired. This data can be analysed and used to identify a plant. Improving the performance of the mobile-based plant identification helps users identify the crop and obtain the information and geolocation of the respective crop.

Government agencies and agricultural managers require information on the spatial distribution and area of cultivated crops for planning purposes. Agencies can more adequately plan the import and export of crop there by reducing price changes and stress on consumers.

1.2 PROBLEM STATEMENT

Develop a mobile application that can identify crop using the field photo of the crop. The application allows the user to take photos and automatically detects the crop. The photo of the crop along with its information and geolocation, are stored in

a database. To ensure farmers regarding latest methodologies and techniques in the process of cultivation and suggest the necessary remedies for the diseases in crops.

1.3 PROJECT REPORT ORGANIZATION

In this *"Project Report"s* a detailed description about the design challenges, proposed methodologies and the implementation of application to solve the real world problem is given. Different functionalities of the application are broken down into modules and explained with the help of use case diagrams and class diagrams. The working model of the application is shown using screenshots in this report.

This report is organized into six chapters. After this introductory chapter,

Chapter 2: Describes the characteristics of the problem, design challenge and proposed solution

Chapter 3: Provides software requirements which includes functional requirements, non-functional requirements, high level architecture of the proposed system and system specifications which includes software requirements and hardware requirements.

Chapter 4: Can be elaborated either in top down number or bottom up manner based on the development strategy adapted. It describes usecase diagrams, class diagrams, activity diagrams, architecture diagrams.

Chapter 5: In this chapter, implementation and testing is discussed in detail.

Chapter 6: Describes the conclusion and future enhancements.

II PROPOSED MODEL

2.1 INTRODUCTION TO THE CHARACTERISTICS OF THE PROBLEM

Though still in the beginning of its journey, ML-driven farms are already evolving into artificial intelligence systems. At present, machine learning solutions tackle individual problems, but with further integration of automated data recording, data analysis, machine learning, and decision-making into an interconnected system, farming practices would change into with the so-called knowledge-based agriculture that would be able to increase production levels and products quality.

The main characteristics include:-

Crop Detection: This feature allows the automated machines in future to differentiate the crops and suggest appropriate techniques in improving crop quality.

Disease Detection: The current existing method for plant disease detection is simply naked eye observation which sometimes may not give accurate results.With the help of ML the past disease remedies can be applied and get immediate solutions to prevent further catastrophe in farming.

2.2 DESIGN CHALLENGES

Creating an Accurate AI Model: The primary design challenge is to creating an AI model with acceptable accuracy. CNN[3] can be used for image detection. A simple NN can be created using transfer learning. This base model is essential to design an Minimum Viable Product (MVP). This MVP is essential for data collection for improving the AI model with Data Engine.

Managing Design Influence: Software is know for its high plasticity and the software requirements changes as stakeholders for the app change. This need to be primary consideration for design of any API/Data Models that are crucial for app's performance.

Ethical Practices: Deep learning model are inherently data hungry and these also tend to perform better with more data. So data collection is essential but this also presents an ethical question about privacy. So, any data collection should only be to improve the AI models and not breach user privacy. Steps need to be take to educate user's what kind of data is collected.

Performance: Any app need to be fast and scalable. All computationally intensive processes that need to be done of user's device need to be run in the background to prevent app from becoming unresponsive. This can be archived using threads. A thread of computation can be created for any heavy computational or network activity. This asynchronous nature can cause race conditions. This race conditions can be dealt with thread safe data models and semaphores.

Scalability: Backend also need to scale based on usage, traffic. So care need to be taken to design API/Date models that can easily scale in realtime based on requirements. Containerizing the backend makes it easy for scalability. Docker is an software environment that can be used to create, maintain and run the containers.

2.3 PROPOSED SOLUTION

Considering the limitations of existing systems, it seems obvious that we need to have a better system. We have developed a simple mobile application and helps farmers with learning about their crops, diagnose any crop diseases without having to waste their resources on futile solutions.

Automatic Plant/Disease Identification is the most promising solution that can be used to partially solve this problem. Solving the entire problem requires an update of basic infrastructure and education. Saying that, this is a small starting step to that future.

Improving the performance of the Mobile-based plant identification helps users identify the crop and obtain the information about the crop/crop disease.

III REQUIREMENTS AND SPECIFICATIONS

3.1 SOFTWARE REQUIREMENTS

3.1.1 Functional Requirements

1. User should be able to login using Email/Google account.
2. User should be able to take/use a picture to get predictions about the crop/crop disease.
3. Save the predictions in an database and access it from other devices.
4. Bookmark predictions.
5. Generate Report using the predictions.
6. Save report as PDF and share it easily using email or chat application (like iMessages, WhatsApp).

3.1.2 Non-Functional Requirements

Performance: Performance is essential for user experience. Performance is critical in both mobile app and backend. Slow performance is detrimental is app usage. With the help of optimization techniques, the entire processing pipeline is tested using strictly timed unit tests.

Scalabilty: This application is built to be have thousands of concurrent users. So, this project was developed with scalability in mind. The server is built on flask which is light-weight, requiring less resources. The web server is containerized, so, depending on the usage, docker containers can automatically be started or stopped.

Usability: In mobile app, localization support allow users to use the app in their preferred local language. Scalable, optimized backend means low latency for information processing, less downtime and better usability.

Availability: This app is available for both iOS and Android. Since android platform is omnipresent in mobile market, user penetration can easily be archived. The small share of iOS market is also not left alone. In, later versions, image classification can be performed on-device, making internet not a necessity.

Security: All requests to server are secured using TLS. User data is saved on device using on-device encryption. Data on firebase is automatically using user password.

Cost and Maintainability: Firebase is a pay-as-you-go model, lowering the costs considering that maintenance costs are essentially non-existential.

3.2 SYSTEM SPECIFICATIONS

3.2.1 Software Specifications

Pytorch: PyTorch[17] is an open-source machine learning library based on Torch library. PyTorch is a Tensor computing library and also a deep learning library with automatic differentiation system. In can be used to build state-of-the-art Neural Networks which are used in self-driving cars.

The crop detection and crop disease detection is done using a Convolutional Neural Network(CNN)[3] trained using transfer learning techniques. ResNet50[4] that is pre-trained on ImageNet[1] is used as a feature extractor. The current models have an accuracy of 92% and 94%. This is covered in detail in **5.5.1**.

NumPy: NumPy[15] is a numerical library that is used for accelerated tensor operations. This help is speeding up the computation by taking advantage of SIMD instructions on modern CPUs and GPUs. This library is useful for preprocessing data before classification.

Matplotlib: Matplotlib[14] is a plotting library for python. This is useful for EDA and performing error analysis while training and testing the AI models.

Pillow: Pilliow[16] is a Python Image Library is an free and open-source that supports opening, modifying, saving images. This is used to convert data sent over internet as bytes to images that can then be used in the AI models for inference. PIL is also used to load images from disk for training the models.

Colab: Google Colab[11] is a free cloud service which helps in developing deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch. In runs on Google servers with GPU or TPU acceleration which helps in reducing the

training period for the AI model. Giving team, the ability to iterate and improve quickly.

Flask and Docker: Flask[10] is a light-weight web framework for python. This can be used to run the inference server. Flask is ran inside a docker container. Docker is a set of platform as a service products that uses OS-level virtualization to deliver software in packages called containers. These container can easily be scaled up or down to automatically adjust for usage and network traffic.

Android Studio: Android Studio[7] is the official IDE for developing native Android Apps. Android Studio provides a unified environment where you can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. Structured code modules allow you to divide your project into units of functionality that you can independently build, test, and debug.

Android's Location library can be used to obtain user's location. Google Maps can we used to display user's location and provide additional information.

Xcode: Xcode[13] is the official IDE for developing native iOS apps. Xcode provides a unified environment for developing native apps for all apple devices and services. Swift programming language is used to write apps in Xcode. CoreLocation framework can be used to obtain user's location. MapKit (Apple Maps) can be used to display user's location and provide additional information.

Apps can be localized to make them be available in multiple language, so more people can use the app without english being a hindrance.

Firebase: Firebase[9] is a BaaS (Backend as a Service) provided by Google. Firebase is scalable, distributed and secure by design. It is especially geared towards business apps, with the intention of helping businesses grow their user bases and increase their profits through their mobile apps.

Firebase provides authentication(Auth), Database (Firestore), Cloud Storage (Storage), Analytics as a service.

3.2.2 Hardware Specifications

User Device Requirements:

Any Android phone running Android Oreo or above.

iPhone or iPad running iOS 13 or above.

An internet connection.

Minimum Server Requirements:

Processor: Any x86 CPU with clock speed of 2.5GHz and above.

RAM: 2GB.

Storage: At-least 10 G.B.

Above mentioned server requirements are minimum and need to be automatically scaled depending on usage and traffic.

IV ANALYSIS AND DESIGN

4.1 USE CASE DIAGRAM

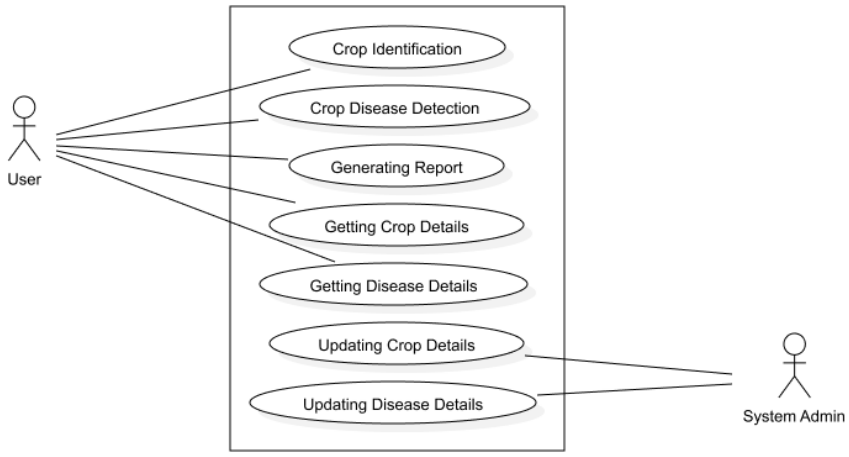

Figure 4.1: Use Case Diagram

4.2 CLASS DIAGRAMS

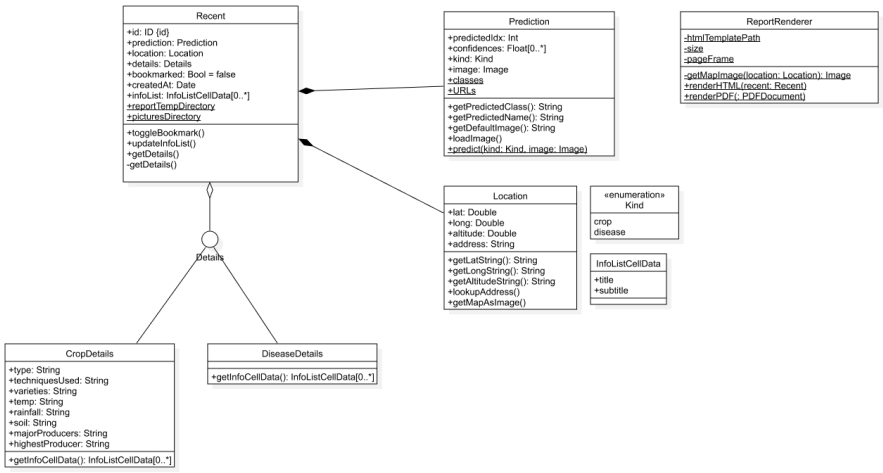

Figure 4.2: Class Diagram

4.3 ACTIVITY DIAGRAMS

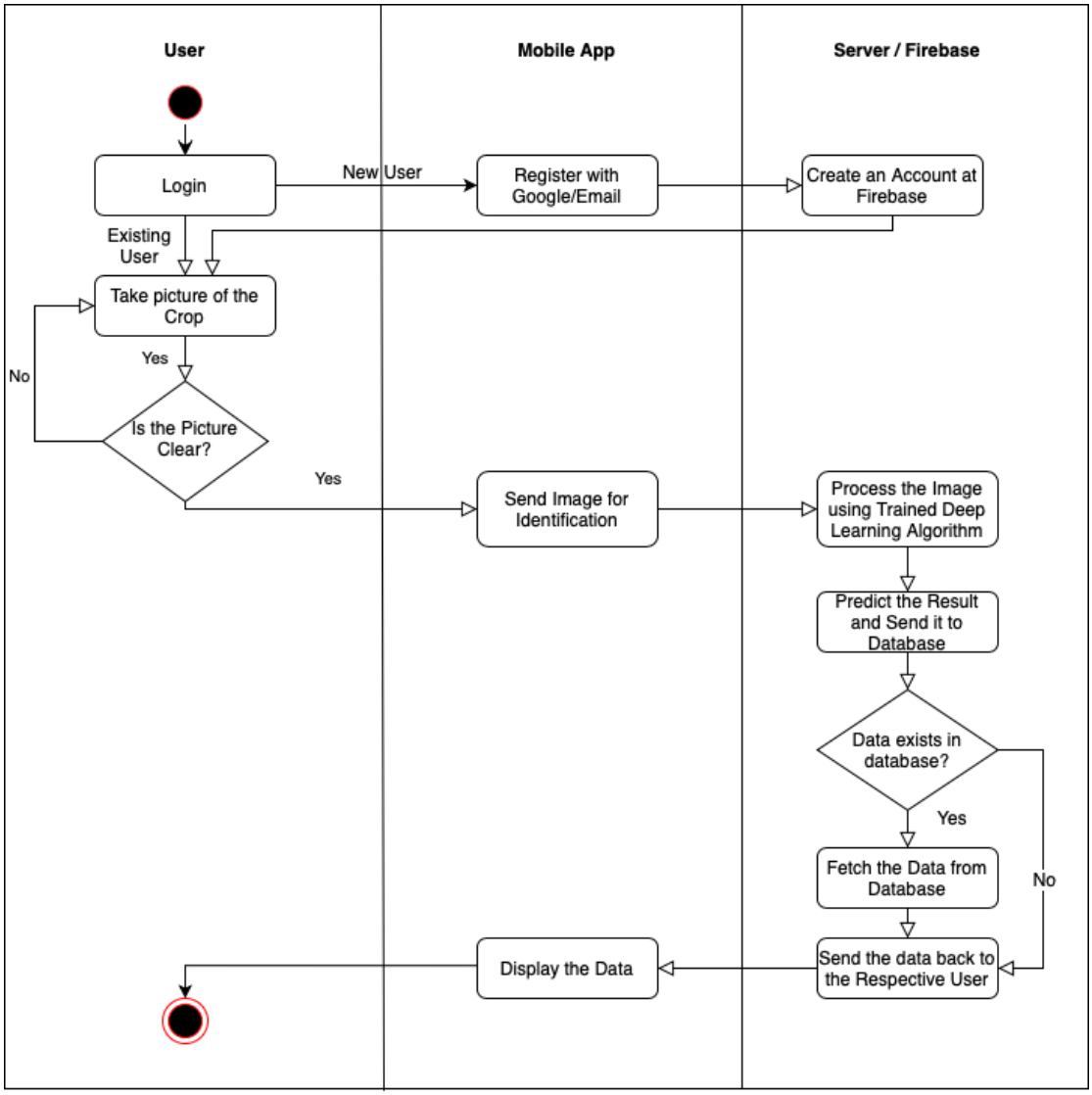

Figure 4.3: Activity Diagram for Crop Detection

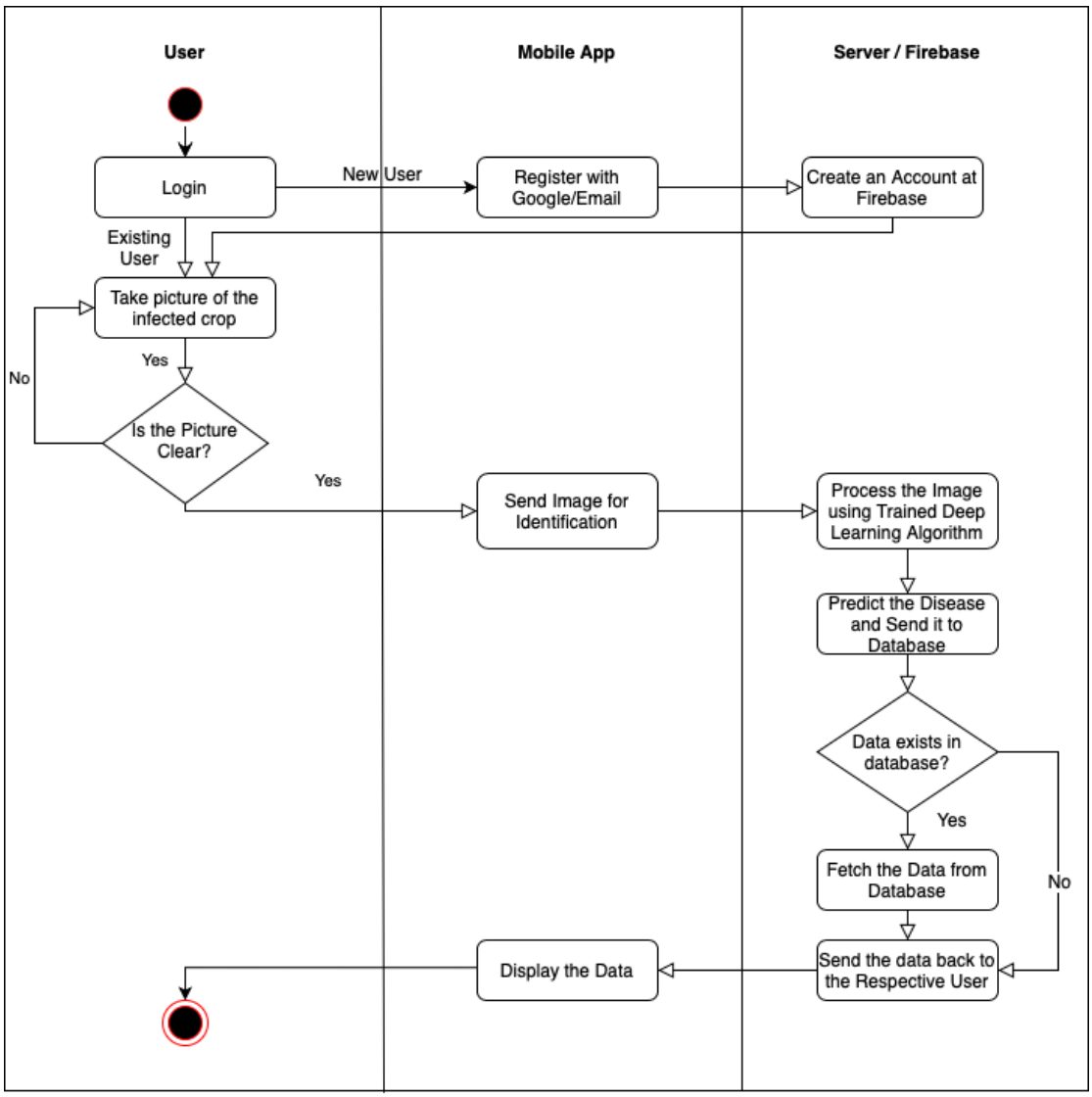

Figure 4.4: Activity Diagram for Disease Detection

4.4 SEQUENCE DIAGRAMS

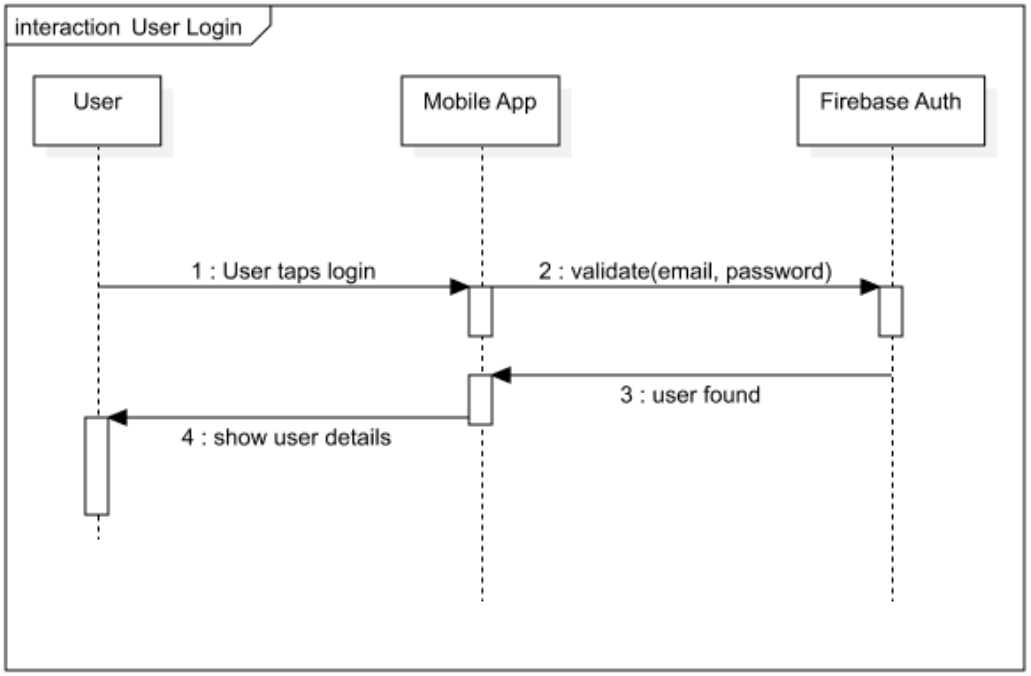

Figure 4.5: Sequence Diagram for User Login

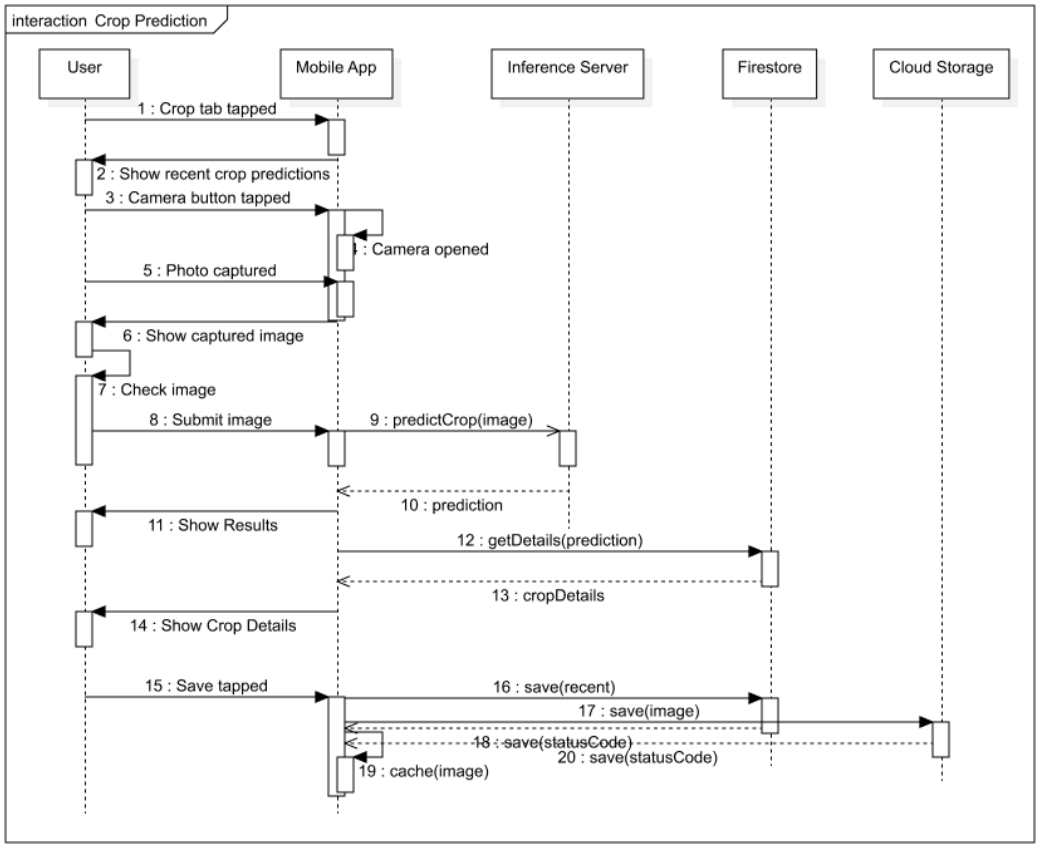

Figure 4.6: Sequence Diagram for Crop Detection

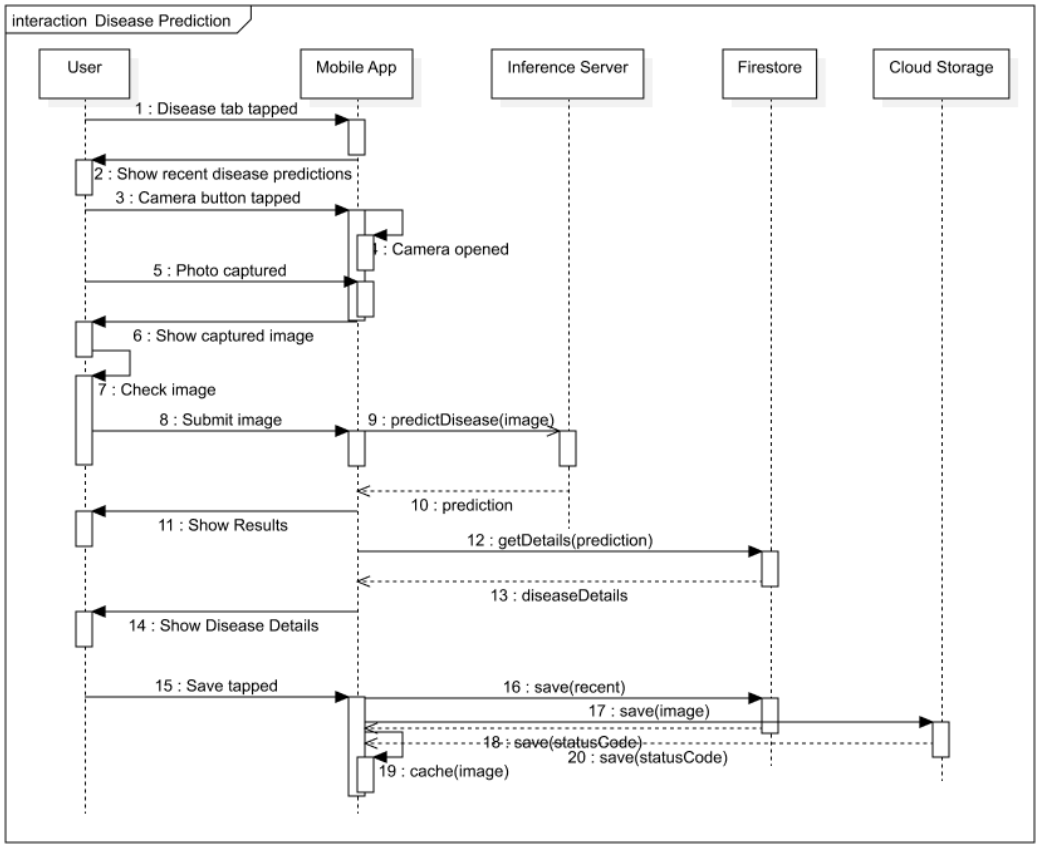

Figure 4.7: Sequence Diagram for Disease Detection

4.5 SYSTEM ARCHITECTURE

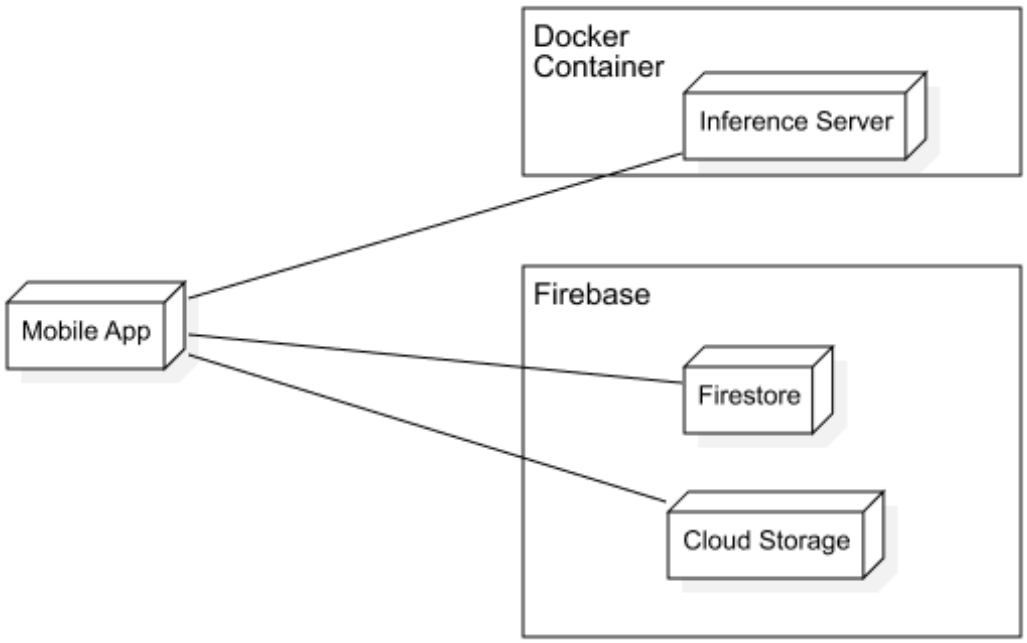

Figure 4.8: App Architecture Diagram

V IMPLEMENTATION AND TESTING

5.1 IMPLEMENTATION OF AI MODEL

5.1.1 Dataset

Any AI models need datasets, so the following are datasets:

Crop Prediction: Dataset for crop prediction is obtained by web scrapping google images. These images aren't perfect, so every image is manually checked if its correct and is not blurry, or has other elements.

Disease Detection: Dataset of Crop Disease Detection is a combination of two different datasets.

- First, is Plant Village Dataset[5] which contains 54,306 images of 14 crop species and 26 diseases. This is a dataset of plant leaves colleted in labs.
- Second, is a PlantDoc Dataset[6] which contains 2,598 images in total across 13 plant species and up to 17 classes of diseases. These images are collected in field and have a little more noise and are closer to real-world images.

5.1.2 Model Architecture

Convolutional Neural Network (CNN or ConvNet) [3] is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

CNNs makes it easy to perform Image Classification by sharing weights, this means CNNs use less memory compared to simple Artificial Neural Networks with same perform. But, CNNs still take a long time to train, so instead of trying to create a CNN from the bottom-up, we can use transfer learning to use a pre-trained ConvNet.

In this project, we use ResNet that has been pre-trained on ImageNet [1]. ImageNet is a large visual dataset containing more than 14 million images with over 20,000 categories. This gives us a huge advantage as the ConvNet has essentially learned the basics of vision, like detecting edges, lines, shapes, etc.

5.1.3 Data Pre-processing

Crop Prediction: Before passing image to ConvNet, they need to be slightly processed. This depends if we are training or testing/inference. During training, images are randomly cropped to size $320px \times 320px$, flipped and normalized. During testing/inference, images are resized to $480px \times 480px$, then center cropped to size $320px \times 320px$ and normalized.

```
data_transforms = {
  "train": transforms.Compose([
    transforms.RandomResizedCrop(320),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229,
        ↪ 0.224, 0.225])
  ]),

  "val": transforms.Compose([
    transforms.Resize(480),
    transforms.CenterCrop(320),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229,
        ↪ 0.224, 0.225])
  ])
}
```

Listing 5.1: Crop Prediction Image Pre-processing

Disease Detection : Pre-processing for disease detection is similar to crop prediction with slight differences.

```
data_transforms = {
  'train': transforms.Compose([
    transforms.RandomRotation(30),
    transforms.RandomResizedCrop(224),
```

```
      transforms.RandomHorizontalFlip(),
      transforms.ToTensor(),
      transforms.Normalize([0.485, 0.456, 0.406], [0.229,
          ↪ 0.224, 0.225])
    ]),
    'valid': transforms.Compose([
      transforms.Resize(256),
      transforms.CenterCrop(224),
      transforms.ToTensor(),
      transforms.Normalize([0.485, 0.456, 0.406], [0.229,
          ↪ 0.224, 0.225])
    ]),
  }
```

Listing 5.2: Disease Detection Image Pre-processing

5.1.4 Defining Model

As specified, we are using transfer learning technique, for crop detection, we use ResNet50 and for disease detection, we use ResNet152. We are not training the entire ResNet, so we simply remove the last layer of ResNet and replace with new layer(s) and freeze the other layers to prevent training of those layers.

The following is the code snippets for loading the model

```
model = models.resnet50(pretrained=True) # model = models.
    ↪ resnet152(pretrained=True) (for disease detection)
for param in model.parameters():
param.requires_grad = False

fc_in_ftrs = model.fc.in_features
fc_out_ftrs = len(class_names)
model.fc = nn.Sequential(nn.Linear(fc_in_ftrs, 1024),
                  nn.ReLU(),
                  nn.Dropout(0.2),
                  nn.Linear(1024, fc_out_ftrs))
```

Listing 5.3: Defining Model

5.1.5 Defining Cost, Optimization Function and Training Model

Crop Prediction: We are using Stochastic Gradient Descent (SGD) with momentum as optimization algorithm and CrossEntropyLoss as Loss function. The model is trained for 50 epochs with a batch size of 16 with learning rate decay rate γ of 0.1 for every 10 epochs.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.fc.parameters(), lr=0.001,
    ↪ momentum=0.9)
exp_mr_scheduler = lr_scheduler.StepLR(optimizer, step_size
    ↪ =10, gamma=0.1)
model, best_acc = train_model(model, criterion, optimizer,
    ↪ exp_mr_scheduler, num_epochs=50)
```

Listing 5.4: Disease Detection Image Pre-processing

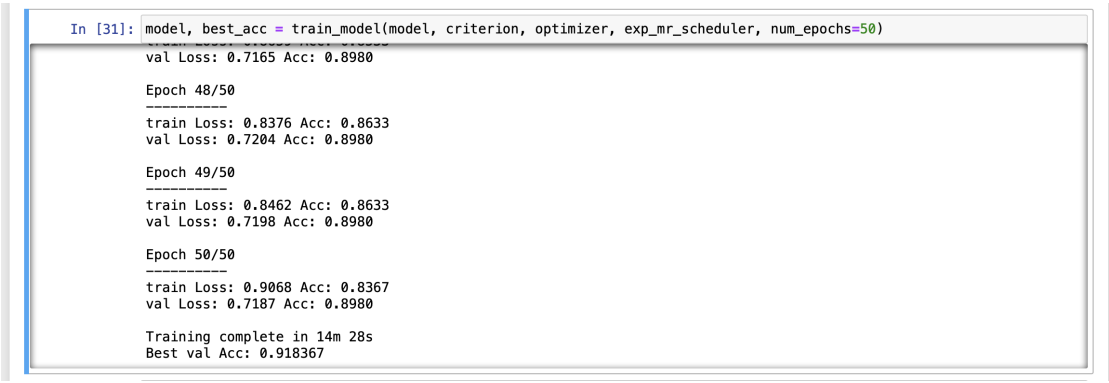

Figure 5.1: Crop Detection Accuracy

Disease Detection: We are using [2] Optimizer and Negative Log Likelihood (NLL-Loss) as Loss function. The model is trained for 10 epochs with a batch size of 16 with learning rate decay rate γ of 0.1 for every 5 epochs.

```
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size
    ↪ =5, gamma=0.1)

model, best_acc = train_model(model, criterion, optimizer,
    ↪ exp_lr_scheduler, num_epochs=10)
```

Listing 5.5: Disease Detection Image Pre-processing

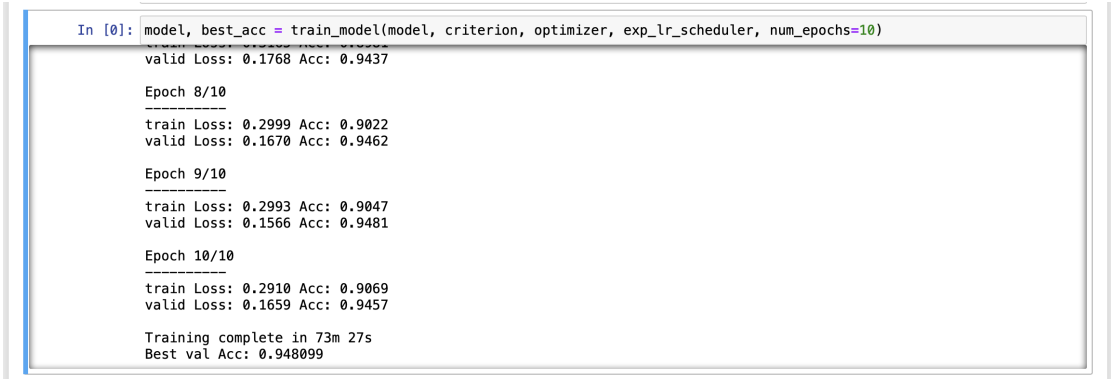

Figure 5.2: Crop Detection Accuracy

s

5.2 IMPLEMENTATION OF INFERENCE SERVER

Inference Server is implemented using flask to make is simple ad light-weight. This is a very simple server, it just has to endpoints.

- `http://<server_ip>/crop`
- `http://<server_ip>/disease`

When any one of these endpoints, receives a request, bytes of data are converted to image, pre-processing is applied appropriately and send to the respect AI model. Following is a snippet of the inference code.

```
def get_crop_prediction(image_bytes):
  tensor = crop_transform_image(image_bytes)
  outputs = crop_model.forward(tensor).squeeze(0)
  _, pred = outputs.max(0)
  outputs = F.softmax(outputs, dim=0)
  preds = [float(f"{outputs[i].item():.4f}") for i in range(len(
     ↪ crop_class_names))]
  return {"pred": pred.item(), "cnf": preds, "kind": "crop"}
```

Listing 5.6: Crop Inference Code

```
def get_disease_prediction(image_bytes):
  tensor = disease_transform_image(image_bytes)
  outputs = disease_model.forward(tensor).squeeze(0)
  _, pred = outputs.max(0)
```

```
    preds = [float(f"{outputs[i].item():.4f}") for i in range(len(
        ↪ diseases_class_names))]
    return {"pred": pred.item(), "cnf": preds, "kind": "disease"}
```

Listing 5.7: Crop Inference Code

The returned value is a dictionary, this is converted to JSON and send back to the mobile app.

5.3 CONNECTION BETWEEN MOBILE APP AND INFERENCE SERVER

REST APIs are used for communication between Mobile App and Inference Server. But this communication cannot be done using only JSON. Image are too large to be sent using JSON, so multipart/form-data request is used, to send images to server as bytes. At the server-side, the images are converted from bytes and passed to AI model as shown above in section. 5.2.

Both iOS and Android have different Implementations for preforming multipart/form-data. In android, a custom Volley[18] request is used and in iOS, default URLSession can be used for multipart/form-data request.

iOS Implementation:

```
let filename = "cropimage.png"
let boundary = UUID().uuidString

var request = URLRequest(url: URLs[kind]!)
request.httpMethod = "POST"
request.setValue("multipart/form-data; boundary=\(boundary)",
    ↪ forHTTPHeaderField: "Content-Type")

var data = Data()

data.append("\r\n--\(boundary)\r\n".data(using: .utf8)!)
data.append("Content-Disposition: form-data; name=\"img\";
    ↪ filename=\"\(filename)\"\r\n".data(using: .utf8)!)
data.append("Content-Type: image/png\r\n\r\n".data(using: .utf8)
    ↪ !)
data.append(image.pngData()!)
data.append("\r\n--\(boundary)--\r\n".data(using: .utf8)!)
```

```
URLSession.shared.uploadTask(with: request, from: data) { data,
    ↪ response, error in
  guard let data = data else {
    // An error occurred
    print(error?.localizedDescription ?? "Unknown Error")
    completionHandler(nil)
    return
  }

  // Response received, JSON is converted to an Swift Object
  let prediction = try? JSONDecoder().decode(Prediction.self,
      ↪ from: data)
  prediction?.image = image
  completionHandler(prediction)
}.resume()
```

Listing 5.8: iOS multipart/form-data request Implementation

Android Implementation:

```
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
bitmap.compress(Bitmap.CompressFormat.PNG, 100, outputStream);
final byte[] imageBytes = outputStream.toByteArray();

String url = getURL(context, kind);
VolleyMultipartRequest request = new VolleyMultipartRequest(
    ↪ Request.Method.POST, url,
    new Response.Listener<NetworkResponse>() {
      @Override
      public void onResponse(NetworkResponse response) {
        Gson gson = new GsonBuilder()
                .registerTypeAdapter(Prediction.class, new
                    ↪ PredictionDeserializer())
                .create();

        Prediction prediction = gson.fromJson(new String(response.
            ↪ data), Prediction.class);
        prediction.image = bitmap;
        callback.onCropPrediction(prediction);
```

```
      }
    },
    new Response.ErrorListener() {
      @Override
      public void onErrorResponse(VolleyError volleyError) {
        volleyError.printStackTrace();
        callback.onCropPrediction(null);
      }
    }) {
    @Override
    protected Map<String, String> getParams() {
      Map<String, String> params = new HashMap<>();
      return params;
    }

    @Override
    protected Map<String, DataPart> getByteData() {
      Map<String, DataPart> params = new HashMap<>();
      params.put("img", new DataPart("crop.jpg", imageBytes));

      return params;
    }
  };

  VolleySingleton.getInstance(context).addToRequestQueue(request)
    ↪ ;
```

Listing 5.9: Android multipart/form-data request Implementation

This is a partial implementation. The full implementation is very lengthy, please refer Volloy Docs for more details.

5.4 IMPLEMENTATION OF MOBILE APP

We developed native Android and iOS mobile apps with user the same Inference server and data models. This means the user can use any platform and all of the user's data is automatically synced between devices.

- Frequently accessed data from database is cached locally to speed up the process.
- All network requests are asynchronous, so that the app is always responsive
- State Observers are used to observer any changes in data and update the UI accordingly.
- All images are cached to prevent constant retrieval from the cloud.

GitHub Repos

iOS App: `https://github.com/SaiHemanthBR/CropPrediction-ios`

Android App: `https://github.com/SaiHemanthBR/CropPrediction-Android`

5.5 TESTING

Testing is done both manually and automatically. Manual testing involves running the app and checking if all the features are working as intended.

Automated testing involves writing test cases that evaluate the functionality of app as small individual parts and later after integration as a whole.

5.5.1 AI model accuracy validation

For both crop detection and disease detection, a subset of images from the dataset is made into test dataset. This test dataset is used to test the accuracy of AI model.

Crop Detection AI model has an accuracy of 91.8%.

```
In [40]: class_correct = [0. for i in range(len(class_names))]
         class_total = [0. for i in range(len(class_names))]

         with torch.no_grad():
             for data in data_loader["val"]:
                 images, labels = data
                 images = images.to(device)
                 labels = labels.to(device)
                 outputs = model(images)
                 _, predicted = torch.max(outputs, 1)
                 c = (predicted == labels)

                 for i in range(len(c)):
                         label = labels[i]
                         class_correct[label] += c[i].item()
                         class_total[label] += 1


         for i in range(len(class_names)):
             print(f"Accuracy of {class_names[i]:10s} : {class_correct[i] / class_total[i]:8.6f}")
```

```
Accuracy of coffee     : 1.000000
Accuracy of cotton     : 1.000000
Accuracy of jute       : 0.800000
Accuracy of maize      : 1.000000
Accuracy of millet     : 0.600000
Accuracy of rice       : 0.800000
Accuracy of sugarcane  : 1.000000
Accuracy of tea        : 1.000000
Accuracy of tomato     : 1.000000
Accuracy of wheat      : 1.000000
```

Figure 5.3: Crop Detection Class-wise Accuracy

Disease Detection AI model, is slightly better due to the larger dataset size, with an accuracy of 94.8%

```
In [0]: class_correct = [0. for i in range(len(class_names))]
        class_total = [0. for i in range(len(class_names))]

        with torch.no_grad():
            for data in data_loader["valid"]:
                images, labels = data
                images = images.to(device)
                labels = labels.to(device)
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)
                c = (predicted == labels)

                for i in range(len(c)):
                        label = labels[i]
                        class_correct[label] += c[i].item()
                        class_total[label] += 1


        for i in range(len(class_names)):
            print(f"Accuracy of {class_names[i]:10s} : {class_correct[i] / class_total[i]:8.6f} %")
```

```
Accuracy of Apple___Apple_scab : 0.938492 %
Accuracy of Apple___Black_rot : 0.989940 %
Accuracy of Apple___Cedar_apple_rust : 0.931818 %
Accuracy of Apple___healthy : 0.972112 %
Accuracy of Blueberry___healthy : 0.969163 %
Accuracy of Cherry_(including_sour)___Powdery_mildew : 0.985748 %
Accuracy of Cherry_(including_sour)___healthy : 0.995614 %
Accuracy of Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot : 0.821951 %
Accuracy of Corn_(maize)___Common_rust_ : 0.997904 %
Accuracy of Corn_(maize)___Northern_Leaf_Blight : 0.958071 %
Accuracy of Corn_(maize)___healthy : 1.000000 %
Accuracy of Grape___Black_rot : 0.963983 %
Accuracy of Grape___Esca_(Black_Measles) : 0.987500 %
Accuracy of Grape___Leaf_blight_(Isariopsis_Leaf_Spot) : 0.967442 %
Accuracy of Grape___healthy : 1.000000 %
Accuracy of Orange___Haunglongbing_(Citrus_greening) : 0.998012 %
Accuracy of Peach___Bacterial_spot : 0.984749 %
Accuracy of Peach___healthy : 0.995370 %
Accuracy of Pepper,_bell___Bacterial_spot : 0.981172 %
Accuracy of Pepper,_bell___healthy : 0.953722 %
Accuracy of Potato___Early_blight : 0.987629 %
Accuracy of Potato___Late_blight : 0.944330 %
Accuracy of Potato___healthy : 0.868421 %
Accuracy of Raspberry___healthy : 0.986517 %
Accuracy of Soybean___healthy : 0.976238 %
Accuracy of Squash___Powdery_mildew : 0.995392 %
Accuracy of Strawberry___Leaf_scorch : 1.000000 %
Accuracy of Strawberry___healthy : 1.000000 %
Accuracy of Tomato___Bacterial_spot : 0.917647 %
Accuracy of Tomato___Early_blight : 0.854167 %
Accuracy of Tomato___Late_blight : 0.896328 %
Accuracy of Tomato___Leaf_Mold : 0.838298 %
Accuracy of Tomato___Septoria_leaf_spot : 0.935780 %
Accuracy of Tomato___Spider_mites Two-spotted_spider_mite : 0.836782 %
Accuracy of Tomato___Target_Spot : 0.638950 %
Accuracy of Tomato___Tomato_Yellow_Leaf_Curl_Virus : 0.983673 %
Accuracy of Tomato___Tomato_mosaic_virus : 0.979911 %
Accuracy of Tomato___healthy : 0.970894 %
```

Figure 5.4: Crop Disease Detection Class-wise Accuracy

5.5.2 Testing Inference Server

Testing of Inference Server is done using a tool called *"Postman"*. Postman is an application that can be used for making HTTP requests like HTTP GET, POST, CREATE, PUT, DELETE. This tool also has an option to send multiple requests to test APIs.

For testing, the inference server, is run on local machine and Postman is used to send a POST request with image and compare the JSON responses with expected responses.

Following are some screenshots of automated API tests.

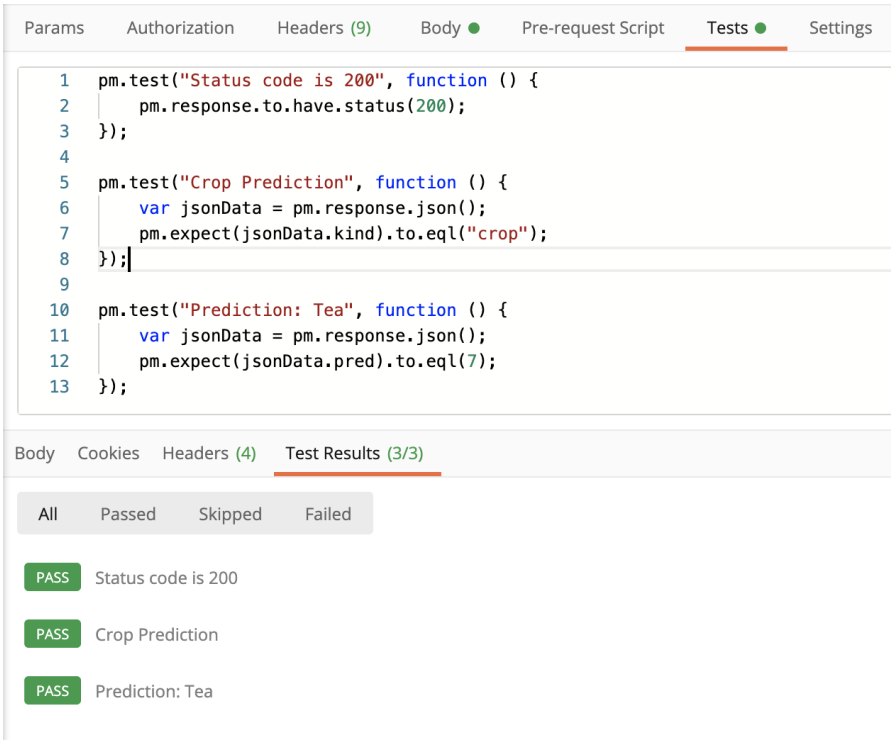

Figure 5.5: Test for Crop Detection API

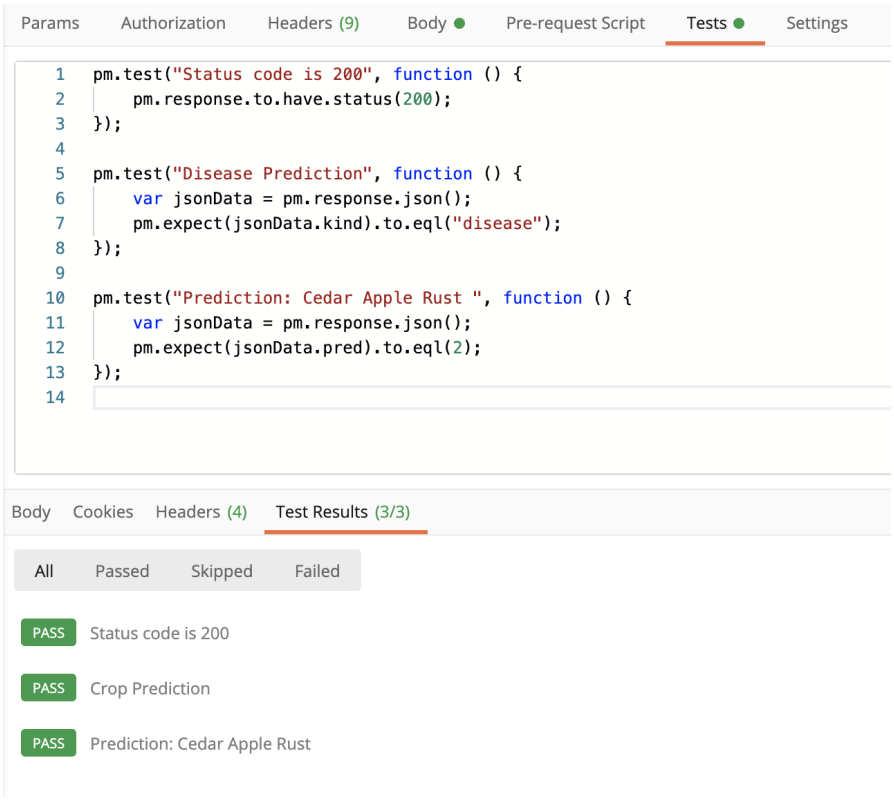

Figure 5.6: Test for Crop Disease Detection API

5.6 MOBILE APP SCREENSHOTS

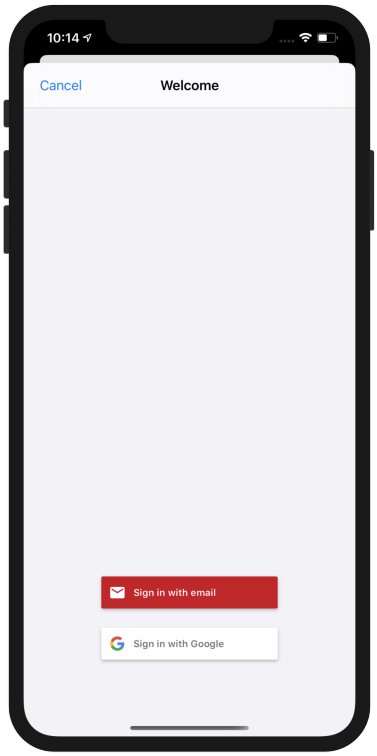

Figure 5.7: User Login

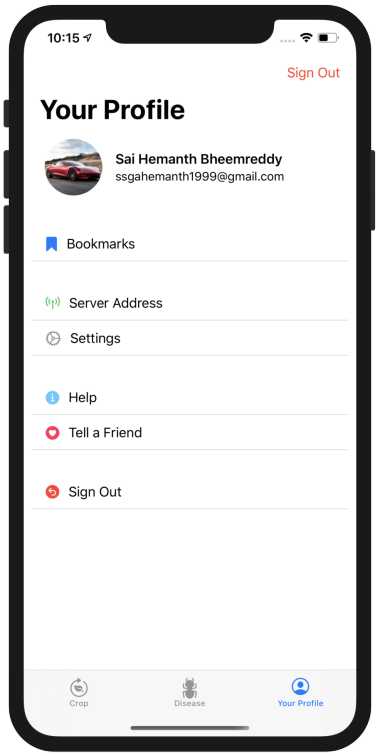

Figure 5.8: User Profile Tab

When the user first launches the app, the user is asked to login.

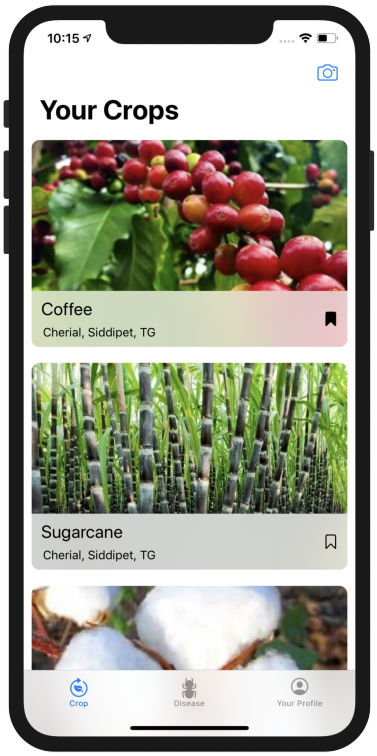

Figure 5.9: Crops Tab

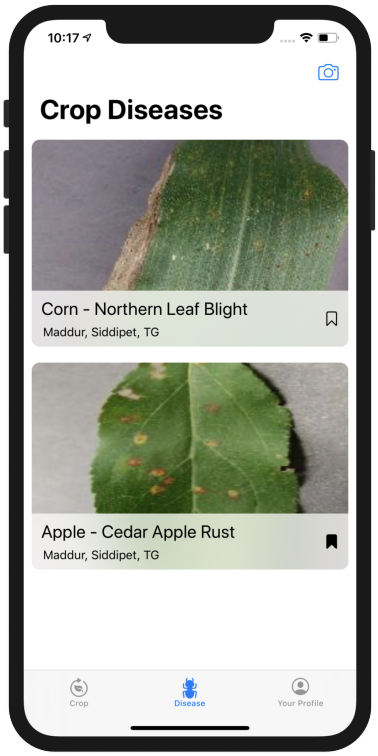

Figure 5.10: Crop Disease Tab

These tabs show user's crops and all the information is available within single tap.

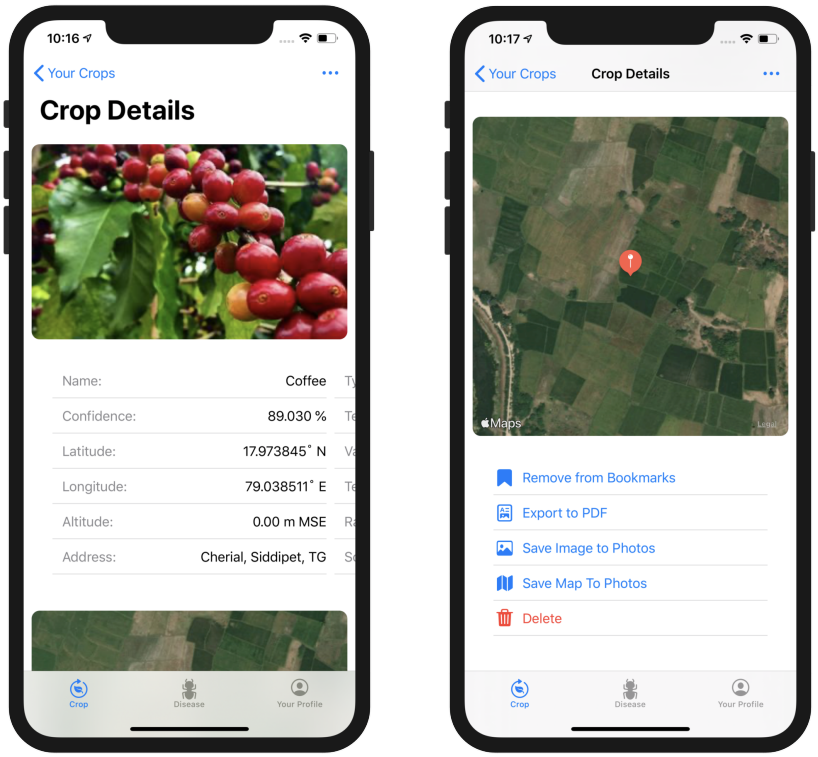

Figure 5.11: Crop Details Screen

User can tap on any of the card in crop tab to get more details about his crops.

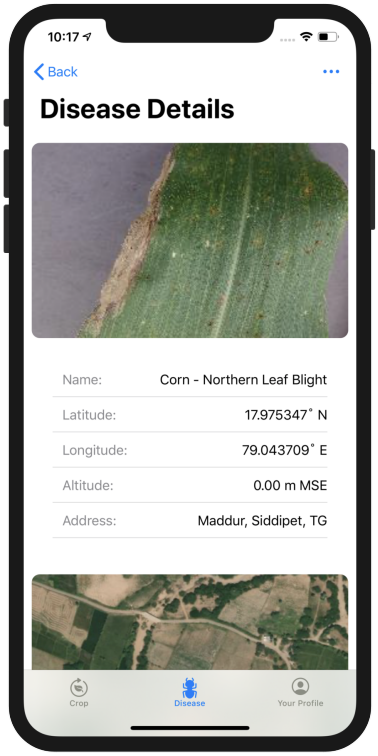

Figure 5.12: Crop Disease Screen

User can tap on any of the card in disease tab to get more details about his crop disease.

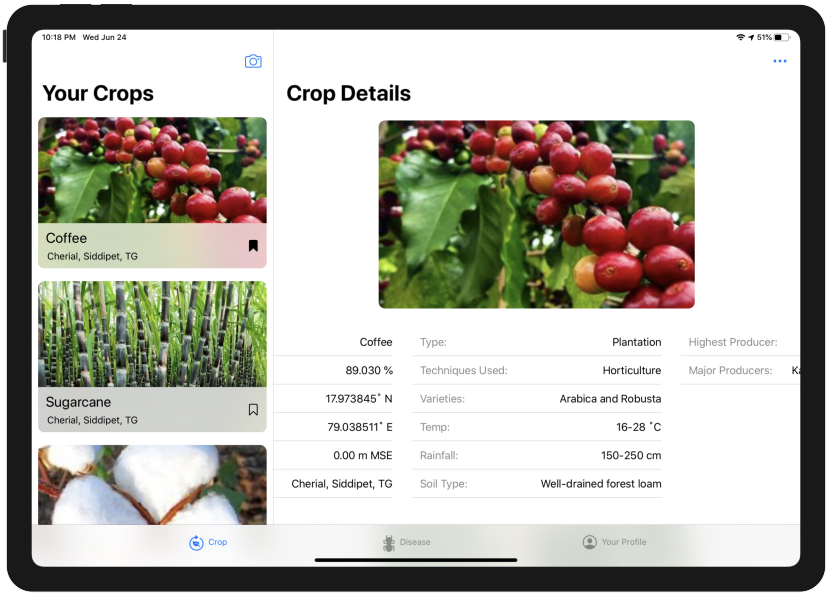

Figure 5.13: Dynamic UI

The app is made to run on any device and automatically adapts the UI to optimize the screen real estate.

VI CONCLUSION & FUTURE SCOPE

"Automation" is the frequent word being used in this decade. This project has been designed with a vision of the automation of farming, which promotes the usage of advanced machinery in agriculture. Farmer's economic growth depends on the quality of the products they produce, which relies on the plant's growth and the yield they get. With the need to move to environmental and sustainable future more than ever before, we need to find ways to produce good healthy food to feed the growing population.

We used CNN model for crop/disease detection, and have achieved an accuracy of 92% for crop detection and 90% for disease detection. But with more research in computer vision, more sophisticated model will be developed and that inevitably leads to better automation and better food production. So the short term goal is to improve the AI model, to work with more crops and increased accuracy.

The long term goal is to integrate this with an automated crop management system. This will create an end-to-end automated food production system, that can make better, healthy food with less resources, less carbon footprint and essentially zero pesticides and fertilizers. People will be able to order food on their mobile and receive freshly harvested food at their door step in less than an hour.

REFERENCES

[1] **Deng, Jia and Dong, Wei and Socher, Richard and Li, Li-Jia and Li, Kai and Fei-Fei, Li** *Imagenet: A large-scale hierarchical image database*. [*In 2009 IEEE conference on computer vision and pattern recognition, pp 248–255*] 2009.

[2] **Diederik P. Kingma and Jimmy Ba** *Adam: A Method for Stochastic Optimization*. `https://arxiv.org/abs/1412.6980`

[3] **Fukushima, Kunihiko and Miyake, Sei** *Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition*. Berlin, Heidelberg, 1982.

[4] **Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun** *Deep Residual Learning for Image Recognition*. 2015. `https://arxiv.org/abs/1512.03385v1`

[5] **Sharada Prasanna Mohanty and David P. Hughes and Marcel Salathé** *Using Deep Learning for Image-Based Plant Disease Detection*. 2016. `https://arxiv.org/pdf/1604.03169`

[6] **Singh, Davinder and Jain, Naman and Jain, Pranjali and Kayal, Pratik and Kumawat, Sudhakar and Batra, Nipun** *PlantDoc: A Dataset for Visual Plant Disease Detection*. New York, NY, USA. 2020. `https://doi.org/10.1145/3371158.3371196`

[7] **Android** `https://developer.android.com/docs`

[8] **Docker** `https://docs.docker.com`

[9] **Firebase** `https://firebase.google.com/docs`

[10] **Flask** `https://flask.palletsprojects.com/en/1.1.x/`

[11] **Google Colab**
`https://colab.research.google.com`

[12] **Google Images** *Crop Images Dataset* Collected by web scrapping images of crop from google images

[13] **iOS**
`https://developer.apple.com/documentation/`

[14] **Matplotlib**
`https://matplotlib.org/contents.html`

[15] **NumPy**
`https://numpy.org/doc/stable/`

[16] **Pillow**
`https://pillow.readthedocs.io/en/stable/`

[17] **PyTorch**
`https://pytorch.org/docs/stable/index.html`

[18] **Volloy**
`https://developer.android.com/training/volley`