Low power image sensor for accurate pest detection in large fields and for soil nutrient index level checking.

Software Design Document

PSG College of Technology
09-26-2021

TEAM MEMBERS:

19z353-Swathi.S
19z320-Harshitha P
19z332-Mridula M
19z354-Swetha GN
19z364-Chandhini B
20z466-YogeshKumar M

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Purpose

The purpose of this document is to outline the technical aspects of the Low power image sensor for accurate pest detection in large fields and for soil nutrient index level checking and the technologies used to develop and implement the application. The goal of this document is to give the reader a better understanding of how the application is being developed and implemented through examples of requirements, constraints, and system's architecture.

1.2 Scope

Integrated pest management relies on the accuracy of pest population monitoring techniques. At the farm level, human operators typically must perform periodical surveys of the traps disseminated through the field. This is a labor-, time- and cost-consuming activity, in particular for large plantations or large forestry areas, so it would be of great advantage to have an affordable system capable of doing this task automatically in an accurate and a more efficient way.

The proposed system will bring higher scalability, being able to deploy in small monitoring areas (greenhouses) as in large plantation extensions.

1.3 Overview

This document is written according to the standards for Software Design Documentation explained in "IEEE Recommended Practice for Software Design Documentation". Sections 3 – 5 contain discussions of the designs for the project with diagrams, section 6 shows samples of UI from the system, and section 7 contains the class diagrams. The appendices contain the setup and configuration needed for the UUIS, a list of functions that are implemented in this version, and that are to be implemented in future version, and a list of tools and environment used in the entire project, along with the time contribution of team members. The appendices also include the test report and test cases.

1.4 Reference Material

The user of this SDD may need the following documents for reference:

IEEE Standard 1016-1998, IEEE Recommended Practice for Software Requirements Specifications, IEEE Computer Society, 1998.

2.0 SYSTEM OVERVIEW

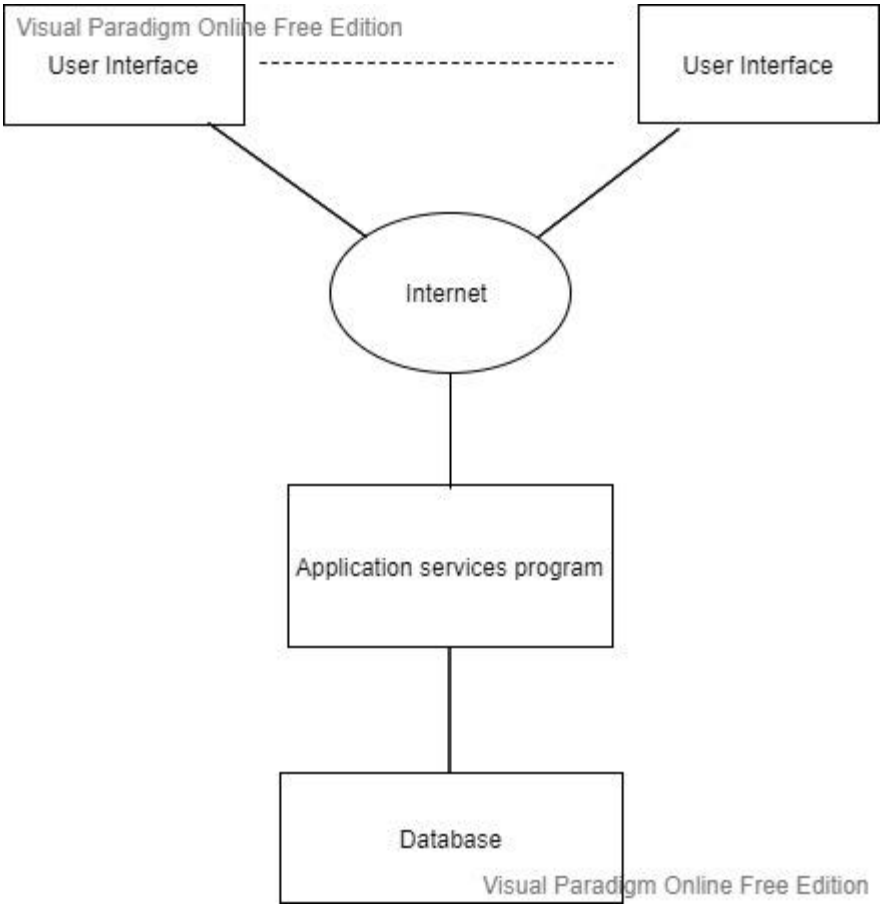
The proposed system is a new application to be developed to be of aid to farmers. This application is

intended to solve the following problems:

- Reduce the manual effort taken to gather trap data on individual pests
- Fix the issue of not being able to synchronize All monitoring traps to measure the target pest population.
- The traditional monitoring techniques are labor-intensive with a poor temporal resolution measurement
- Manual Minute detection takes more manpower and time, along with boundless usage of pesticides
- Current sensors lack the flexibility of power and persistent observation is not feasible.
- The estimation and the accuracy are limited using existing methods
- High power thermal sensors are more expensive and sensitive to environmental conditions

3.0 SYSTEM ARCHITECTURE

3.1 Architectural Design



A typical three-layer structure is used in the system: the database layer, the application service layer, the user interface layer. System architecture as shown in Figure 1.

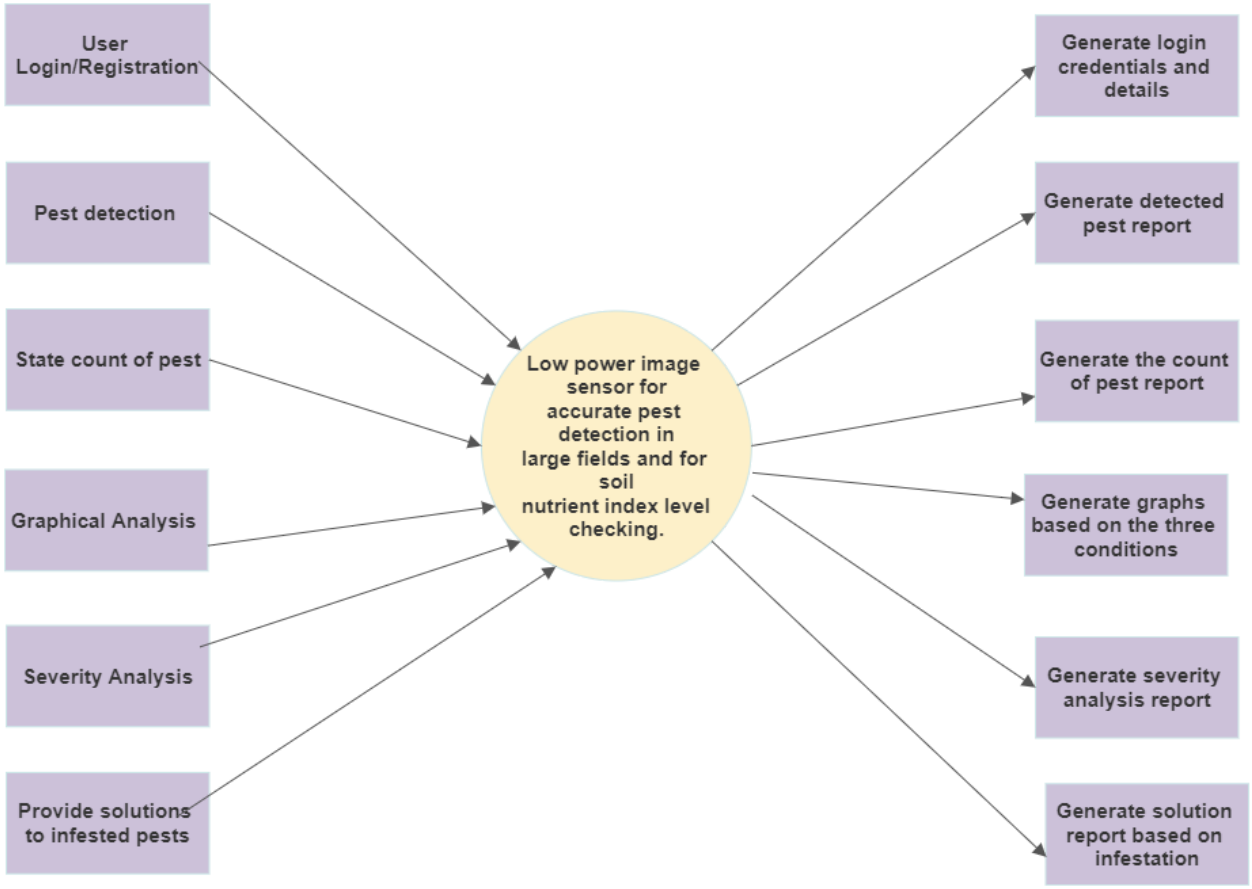
3.1.1 ***The database layer*** : The database is used to hold data, including user registration information, pest detection information, solution information and all of the other information.

3.1.2 ***The application service layer*** : The application service layer is the core of this three-layer structure, the system functions and business logic are handled in this layer. In this layer, the system's business logic is encapsulated, the application service interface is provided for the user interface layer and the system modules between the function calls. The application service layer also updates data in the database, according to the service request of the top layer.

3.1.3 ***The user interface layer*** : The user interface layer is a program that runs on a remote user computer or on a smartphone. It displays the provided services by the server to the user. When the user selects a service, this program sends a request to the server. When the server returns the processed result, this program shows it to the user**.**

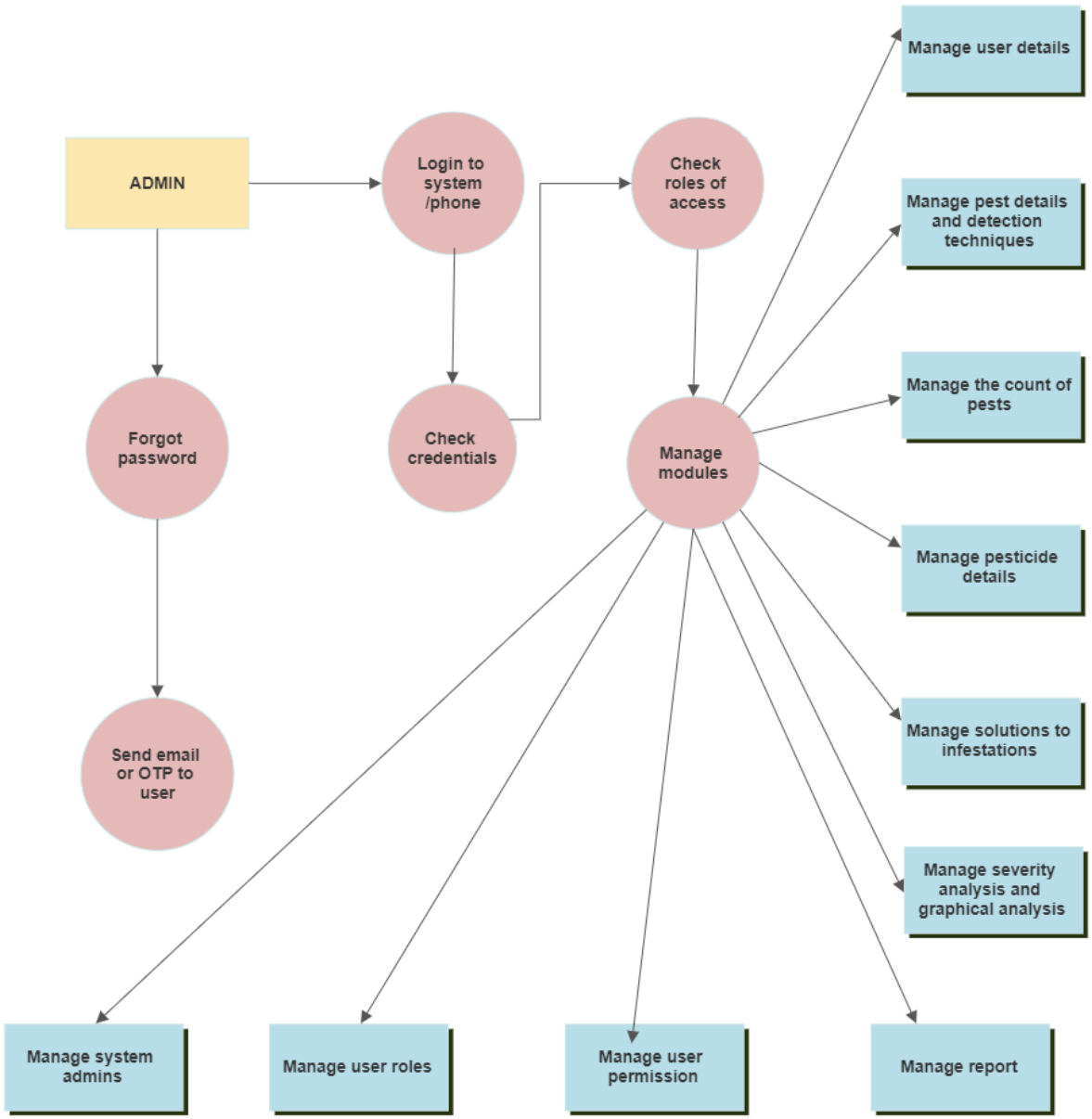
3.2 Decomposition Description

LEVEL 1 DATA FLOW DIAGRAM (DFD)



- ☐ User registration , credentials management
- ☐ Pest detection management system
- ☐ State count of pest and management
- ☐ Graphical representation , the analysis of pest attacks over various conditions
- ☐ Severity analysis
- ☐ Providing solutions to infested pest attack.

SECOND LEVEL DATA FLOW DIAGRAM(DFD)



SECOND LEVEL DATA FLOW DIAGRAM (DFD)

3.3 Design Rationale

The system is structured into the data access layer, business logic layer and business exterior layer. We implement user registration, user cancellation, pest inquiries and detection methods , online pesticide information, severity and graphical analysis including suggesting solutions to infestations . Business process design and database design is the focus of this system which are clearly and effectively designed by the business process diagrams . Pest detection is accurately done using machine learning algorithms and absolutely user friendly to farmers who can save time in manual labour.

3.3.1 The most significant limitation of the over project is its dependency over the database server because of this when it fails then the whole work is to be stopped.

3.3.2 Response time of the system will be stable over lower network connectivity areas also.

3.3.3 Another limitation of our project is that this software and hardware requirement is a high comparison of existing systems. There were many difficulties that came in our way in the process of development of this real time project some of them are illustrated below.

3.3.3.1 Real time transfer of the file with minimum time delay and efficiency through lower network range.

3.3.3.2 Searching for an efficient security package and studying its usage.

3.3.3.3 Enabling the multi user environment and simultaneous usage of files.

4.0 DATA DESIGN

4.1 Data Description

The data is stored in an Object Oriented database. Object-oriented databases use small, recyclable separates from software called objects. The entities are stored in classes and are processed using objects. The objects themselves are stored in the object-oriented database. The relations are expressed using Structural modelling.

4.2 Data Dictionary

Entity	Attributes/Methods	Parameters	Type	Description
	Login()	-	void	Proceeds to the login

Admin				page
	Logout()	-	void	Opens the exit page
	SignUp()	-	void	Proceeds to the Sign up page
_LoginState	AlreadyHaveAnAccountCheck()	press	void	Checks if accounts with the given credentials have been registered previously. If False, proceeds to SignUp page
	Button()	text,press	void	Login button that navigates to the Image Selection page.
	InputField()	hintText, OnChanged	String	Takes the email ID as input
	PasswordField()	OnChanged	String	Takes the password as input
Pest_Classification	detectedClass		array	Contains the detected classification label of pest
	ConfidenceInClass		array	Estimates the percentage likelihood covering the true model accuracy
	factorX		double	The x-coordinate of the detected pest
	factorY		double	The y-coordinate of the detected pest
	Positioned()	left,top,width, height,child	void	Provides the final interface of image detection,classification and confidence percentage.
	renderBoxes()	screen	void	Used for expressing each classification in a 2D coordinate system

TfliteHomeState	_busy		boolean	Used for selecting one Image at a time
	_image		File	Contains the selected pest image
	_imageHeight		double	Height of the Image
	_imageWidth		double	Width of the Image
	_model		String	Model used is ssd
	_recognition		List	Contains all the detected pests in a single image
	SelectFromImagePicker()	image	void	Used to select an Image from the database of pest image files
	loadmodel()	res, model, label	void	Loads the trained ssd model
ViewDetails	count		int	Provides count of pests found in each classification in an Image
	pest_identified		String	Provides the particular pest classification
	region		String	Provides the field of region where pest was detected
	inference		String	Provides the detailed analysis of pest manifestation
	solution		String	Provides possible solution to eradicate the identified pest

5.0 COMPONENT DESIGN

User Registration:

Local data:

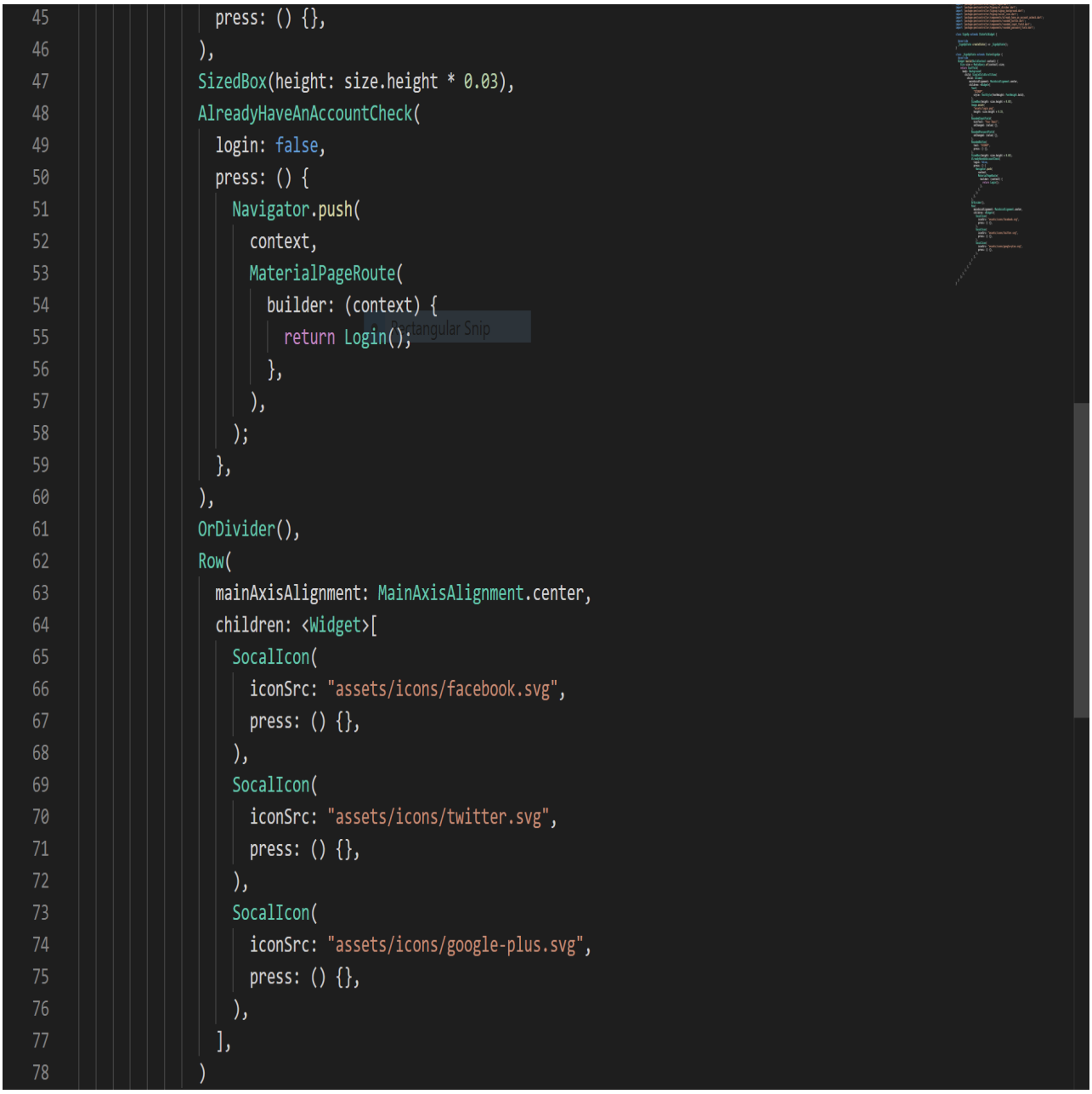
1)assets-png of the landing page

2)Password

```dart
class Background extends StatelessWidget {
  final Widget child;
  const Background({
    required this.child,
  });

  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;
    return Container(
      height: size.height,
      width: double.infinity,
      // Here i can use size.width but use double.infinity because both work as a same
      child: Stack(
        alignment: Alignment.center,
        children: <Widget>[
          Positioned(
            top: 0,
            left: 0,
            child: Image.asset(
              "assets/pest2.png",
              width: size.width * 0.35,
            ),
          ),
          Positioned(
            bottom: 0,
            left: 0,
            child: Image.asset(
              "assets/pest1.png",
              width: size.width * 0.25,
            ),
          ),
          child,
        ],
```

Signup:

```dart
class SignUp extends StatefulWidget {

  @override
  _SignUpState createState() => _SignUpState();
}

class _SignUpState extends State<SignUp> {
  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;
    return Scaffold(
      body: Background(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text(
                "SIGNUP",
                style: TextStyle(fontWeight: FontWeight.bold),
              ),
              SizedBox(height: size.height * 0.03),
              Image.asset(
                "assets/login.png",
                height: size.height * 0.15,
              ),
              RoundedInputField(
                hintText: "Your Email",
                onChanged: (value) {},
              ),
              RoundedPasswordField(
                onChanged: (value) {},
              ),
              RoundedButton(
                text: "SIGNUP",
```



```
        press: () {},
      ),
      SizedBox(height: size.height * 0.03),
      AlreadyHaveAnAccountCheck(
        login: false,
        press: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) {
                return Login();
              },
            ),
          );
        },
      ),
      OrDivider(),
      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          SocalIcon(
            iconSrc: "assets/icons/facebook.svg",
            press: () {},
          ),
          SocalIcon(
            iconSrc: "assets/icons/twitter.svg",
            press: () {},
          ),
          SocalIcon(
            iconSrc: "assets/icons/google-plus.svg",
            press: () {},
          ),
        ],
      )
```

Detecting pest and classification of pest:

```dart
List<Widget> renderBoxes(Size screen) {
  if (_recognitions == null) return [];
  if (_imageWidth == null || _imageHeight == null) return [];

  double factorX = screen.width;
  double factorY = _imageHeight / _imageHeight * screen.width;

  Color blue = Colors.red;

  return _recognitions.map((re) {
    return Positioned(
      left: re["rect"]["x"] * factorX,
      top: re["rect"]["y"] * factorY,
      width: re["rect"]["w"] * factorX,
      height: re["rect"]["h"] * factorY,
      child: Container(
        decoration: BoxDecoration(
            border: Border.all(
          color: blue,
          width: 2,
        )),
        child: Text(
          "${re["detectedClass"]} ${(re["confidenceInClass"] * 100).toStringAsFixed(0)}%",
          style: TextStyle(
            background: Paint()..color = blue,
            color: Colors.white,
            fontSize: 15,
          ),
        ),
      ),
    );
  }).toList();
}
```

Loading Tensorflow models:

```dart
const String ssd = "SSD MobileNet";
const String yolo = "Tiny YOLOv2";

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: TfliteHome(),
    );
  }
}

class TfliteHome extends StatefulWidget {
  @override
  _TfliteHomeState createState() => _TfliteHomeState();
}

class _TfliteHomeState extends State<TfliteHome> {
  String _model = ssd;
  File _image;

  double _imageWidth;
  double _imageHeight;
  bool _busy = false;

  List _recognitions;

  @override
  void initState() {
    super.initState();
    _busy = true;

    loadModel().then((val) {
```

```
    loadModel().then((val) {
      setState(() {
        _busy = false;
      });
    });
  }

  loadModel() async {
    Tflite.close();
    try {
      String res;
      if (_model == yolo) {
        res = await Tflite.loadModel(
          model: "assets/tflite/yolov2_tiny.tflite",
          labels: "assets/tflite/yolov2_tiny.txt",
        );
      } else {
        res = await Tflite.loadModel(
          model: "assets/tflite/damn.tflite",
          labels: "assets/tflite/label.txt",
        );
      }
      print(res);
    } on PlatformException {
      print("Failed to load the model");
    }
  }

  selectFromImagePicker() async {
    var image = await ImagePicker.pickImage(source: ImageSource.gallery);
    if (image == null) return;
    setState(() {
      _busy = true;
    });
```

6.0 HUMAN INTERFACE DESIGN

This section explains the functionality of the website from the user's perspective.

6.1 Overview of User Interface

6.1.1 Home Page :

As soon as the user enters the Application they are taken to the home screen.
On the Home screen, they can:

- Login or sign up options appears on the home screen
- Immediately Login or Sign up
- Here they can select the image to proceed with
- After selecting the image, they can able to view the name and count of the pest

- Get to know more about the particular peat in next screen
- Detailed statistics on the result
- Inference and the solution

6.1.2 The general flow :

This is a simple application which is very easy to use and will be helpful for the users who are new with using android applications

.

As soon as the user clicks on the application he/she is directed to the login/signup page. If the user is new, he/she can sign up by entering necessary details and can login to further proceed with. If the user has an account already he/she is directed to the login page to login.

After this process, next appears the screen which has a button that helps the user to select the picture to detect the pest. Then next it allows the user to connect with G drive. There the user can select any picture he got from the esp32 cam. This page tells the user what kind of pest it is, along with the detection boxes in the selected picture. The same pest details screen has a button called view details which give the detailed view of the pest that it detected.

The view details page tells the pest identified, number of pests, in which region the pest resides, inference and solution.

SCREEN 1: User login
The user is required to enter a valid email id or phone number used at the time of registration. This page is the basic authentication page with user-friendly minimalistic UI to meet the demands of the user.

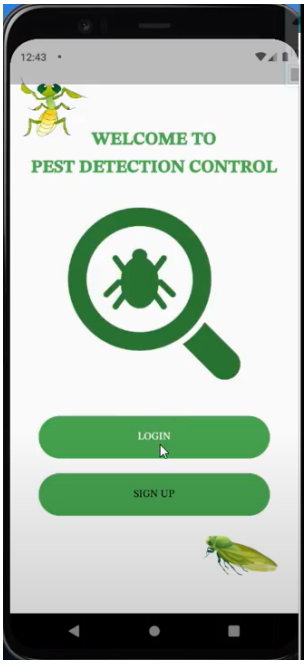
SCREEN 2:Classified result and count
This screen displays the count and classified pest population that is loaded each time the user refreshes the page.The classified image is shown with detection boxes and the classification labels is displayed down for user's readability.

SCREEN 3:Detailed statistics on the result
A statistical analysis of how the classified result and count varies with different acres of the same field is shown which enables the user to plan an effective integrated pest management on the specific acre of land.

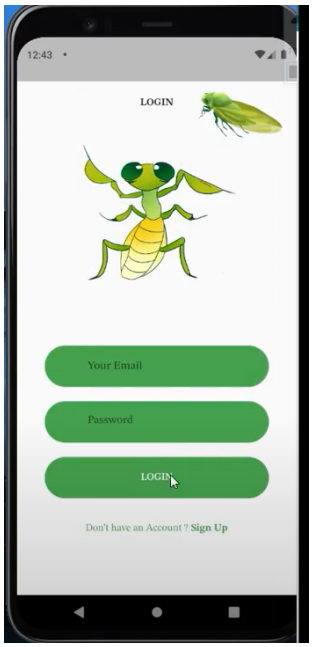
SCREEN 4:Inference and solution

Detailed solution to the specific pest detected is displayed along with the pesticide dealer's contacts so that the user can avail all information through the application itself.

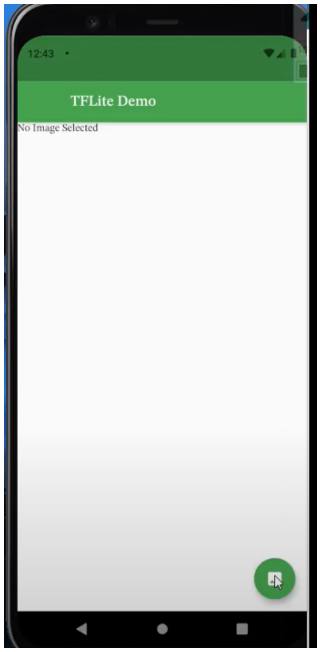
6.2 Screen Images



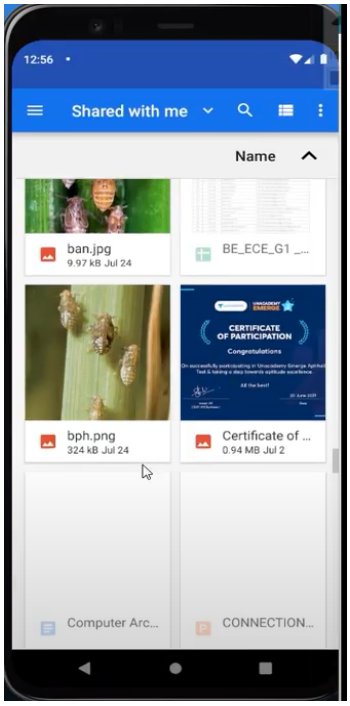
6.2.1 Login/Signup page



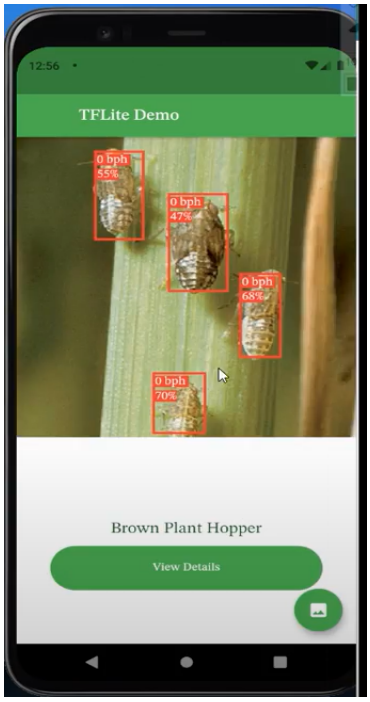
6.2.2. Login page



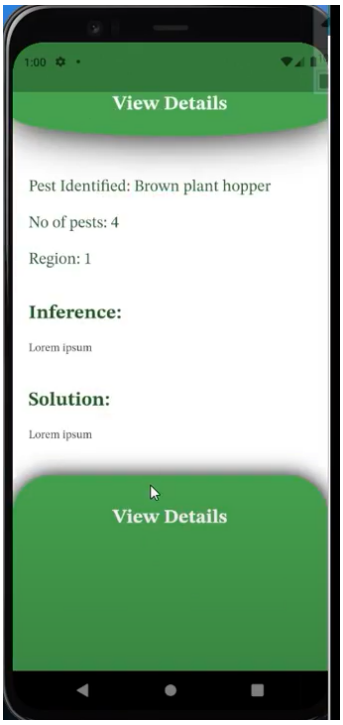
6.2 3. Page which makes us to select an image



6.2.4. Link to Drive Page



6.2.5. Pest details Page



6.2.6. View details page

6.3 Screen Objects and Actions

In image 6.2.1 (Object: action)

6.3.1 Logo : Pressing on the logo will take the user to the home page as well

6.3.2 Sign up: The user is directed to a page where they have to input their personal details. The login object will change to “View Profile” upon completion of signing up.

6.3.3 Login : Upon Login the navigation bar has “View Profile” now instead of the Sign Up| Login object.

In image 6.2.2 (Object: action)

6.3.4 Your mail (text box) : The space where we enter our mail id

6.3.5 Password (text box):The space where we enter our mail id

6.3.6 Login : Upon Login, The user directed to next screen

6.3.7 Don't have an account SIGN UP : Takes the user to signup page

In image 6.2.3 (Object: action)

6.3.8 Photo icon button: On clicking this button in this screen takes us to select the image

In image 6.2.4 (Object: action)
6.3.9 Selecting images : The space where we select our image for further processing

In image 6.2.5 (Object: action)
6.3.10 Selected Image : By clicking on the selected image, the bounding boxes may appear
6.3.11 view details : Takes the user to next screen where they can view the detailed information
6.3.12 Photo icon button: On clicking this button in this screen takes us to select the image again if we want to check for another image

7.0 REQUIREMENTS MATRIX

SYSTEM COMPONENTS	FUNCTION REQUIREMENT ID (Referring from SRS doc)	FUNCTIONAL REQUIREMENT
Detect the pest	REQ-1	A minimalistic Smartphone
	REQ-2	Mandatory to fill all the fields displayed on the screen
	REQ-3	Users must have an account
Give count of pest upon detection	REQ-1	A smartphone
	REQ-2	Pest control measures and preventive measures must be listed out in the application
	REQ-3	Application must show the highest accuracy results in detection and also detect the right pest.
Graphical Analysis (A visual representation)	REQ-1	A minimalist smartphone
	REQ-2	Application must ensure that mandatory fields are filled before moving onto viewing graphical analysis.

	REQ-3	When invalid information is entered, the application must prompt the user to re-enter the values.
Severity Analysis	REQ-1	A minimalist smartphone
	REQ-2	Application should have classified the pest images and taken count
	REQ-3	Application should have obtained the location of traps
Provide solutions to the infested pests	REQ-1	A smartphone
	REQ-2	Application should enable the users to view the measures and apply it practically in their fields i.e easy to understand.
	REQ-3	Application should provide multiple techniques to tackle infestation
	REQ-4	Users should be able to view detailed descriptions about how different techniques should be implemented
Provide contact details of the pesticide provider	REQ-1	A smartphone
	REQ-2	Users should be able to view the correct details about the pesticide providers , only reputed and verified provided must be displayed in the application.
	REQ-3	Users will be able to download a report of their analysis and pesticide providing information also as an add on feature.