ABSTRACT

The system proposal document is the product of studying about analysis of seasonal trends agricultural yield, to understand how the seasonal tends impact yield production, food security and farming practices. This study seeks in optimizing agricultural practice, improving crop yields, improving technology in agricultural yields sustainable food production and improving soil health, this is by examining seasonal changes in weather conditions, crop growth cycle and market dynamics. The research will include the data collection through surveys, questioners, and field observation to capture diverse patterns of agricultural activities throughout the year. This proposal document has been arranged in order starting from the cover page and ending with the references.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

DEFINATIONATION OF KEY TERMS

Resilience – ability to withstand specific condition of events

Yield – to produce of agricultural products

Optimization- the action of fully utilizing resources

Data – fact and statistics collected together for reference

Intervention - act of process of interfering

Nutrition – process of providing nourishing material for better health

Disproportional – not being equal

ABREVIATIONS AND ACRONYMS

Adapt-	adaptation
SMTP-	Simple Mail Transfer Protocol
UI –	User Interface

CHAPTER ONE: RESEARCH OVERVIEW

1.1Statement of Problem

Agricultural yields experience significant variability due to seasonal changes, the changes cause food insecurity and economic instability for farmers. Many studies have focused just mainly in factors affecting yields, forgetting about the complex interplay of climate, ecological and management variables. This gives limited development of effective strategies for yield utilization specifically in climate changes, where the weather pattern is very unpredictable hence, makes agricultural planning difficult.

1.2Study Justification

The study of this platform will help people understand seasonal trends in agricultural yields even during the changing environmental conditions. The knowledge gained will be important for improving agricultural practices and having resilient and sustainable agriculture even in the future. This platform of study will help everyone in the society not just myself; it will also help in providing a stable economy.

1.3Research Objectives

1.2.1General objective

(i) To develop an app that gives can give suggestions to farmers of which crops to yield in his area depending in his or her preference of choice to choose.

1.1.1 Specific Objectives

(i) To analyse crop data of all regions in Kenya for better farming choice for the farmer.

(ii) To implement targeted crop management strategies based on seasonal trend analysis to optimize yields and reduce losses for smallholder farmers

(iii) To implement soil PH in certain regions of the country for proper farm utilisation, by giving types of crops that can be grown and have high yield production

1.4Research Questions

(i) What are key seasonal trends affecting agricultural yields across different regions?

(ii) What role do soil characteristics have in determining agricultural productivity during different seasons?

(iii) What are needed to be able to develop enhance resilience and stability of agricultural yields in response to climate variability?

(iv) What role do technology upgrade play in seasonal yield fluctuations?

(v) How do climatic factors such as temperature and precipitation influence yield variability for crops?

(vi) How do during extreme weather events such as floods and droughts affect seasonal crop yield?

(vii) How does timing of planting and harvesting affect yield outcomes in different climate condition?

(viii) What best practices can be identified from successful case studies of yield manage in variable climate?

(ix) How do socio economic factors influence farmers responses to seasonal changes in agricultural productivity?

(x) What region specific interventions can mitigate seasonal food insecurity caused by yield fluctuations?

1.5 Functional Requirements

USER	USER ACTIVITIES	FEATURES
Farmer	View yield forecast, analyse seasonal trends receive alerts	Yield forecasting seasonal trend analysis customizable notifications
Agricultural Researcher	Analyse historical yield data, identify correlations between seasonal factors and yields	Data visualization correlation analysis, regression analysis
Policy maker	View regional trends make informed decisions on agricultural policies	Regional yield trend analysis data-driven policy recommendation
Data analyst	Validate and clean data, perform advanced data analysis	Data validation data cleaning, advanced analytics
System administrator	Manage user access ensure data security and integrity	User management access Control, data encryption

1.6 Breakdown of tools and resources to be used

Tools	**Purpose**	**Budget (Ksh)**	**Alternative (for tools exceeding feasible budget)**
Laptop/PC	To install and run Android Studio for app development	**0 (already owned)**	**N/A**
Android Studio	The primary IDE for building, testing, and debugging the app	**Free (open source)**	**IntelliJ IDEA (Lightweight but lacks full Android SDK support)**
Jetpack Compose	UI toolkit for building modern, reactive user interfaces	**Free (open source)**	**XML-based UI (Legacy but functional)**
Kotlin Programming Language	App logic development, backend processing	**Free (open source)**	**Java (Alternative but less concise for modern Android apps)**
Room Database (Jetpack Library)	Local storage for managing seasonal yield data	**Free (open source)**	**SQLite (Also free but less integrated with Jetpack Compose)**
Firebase Fire store (Free Plan)	Cloud-based database for storing agricultural yield trends	**Free (limited tier)**	**Supa base (Free alternative to Firebase with SQL-like database features)**
Weather API (OpenWeatherMap - Free Tier)	Fetch real-time weather data for farmers	**Free (basic tier)**	**Weather bit API (Cheaper alternative)**
Charts Library (MPAndroidChart)	Data visualization for yield trends	**Free (open source)**	**Jetpack Compose Canvas (More code-heavy, but customizable)**
Mobile Survey Integration (Google Forms API / Firebase Fire store)	Collects real-time agricultural data from farmers	**Free (open source)**	**Manual surveys (Less efficient but viable)**

1.7 Project Schedule Breakdown

	PROJECT MILESTONES					
WEEKS	Project Planning & Analysis (System Documentation: Cover page & Chapter One)	Project Design & Modeling (System Documentation Chapter Two)	Project Development & Testing (System Documentation Chapter Three)	Project Deployment (System Documentation Chapter Three)	Final Touches of System Documentation (Preliminary Pages, Chapter Four & References)	Project Presentation
5-9 May	🟦 DONE					
12-16 May		🟦				
19-23 May			🟦			
26-30 May			🟦			
2-6 June			🟦			
9-13 June			🟦			
16-20 June			🟦			
23-27 June			🟦			
30-4 July			🟦			
7-11 July			🟦	🟦		
14-18 July				🟦		
21-25 July					🟦	
11 Aug						🟦

Table 1.7 Project Schedule Breakdown

CHAPTER TWO: DESIGN AND MODELLING

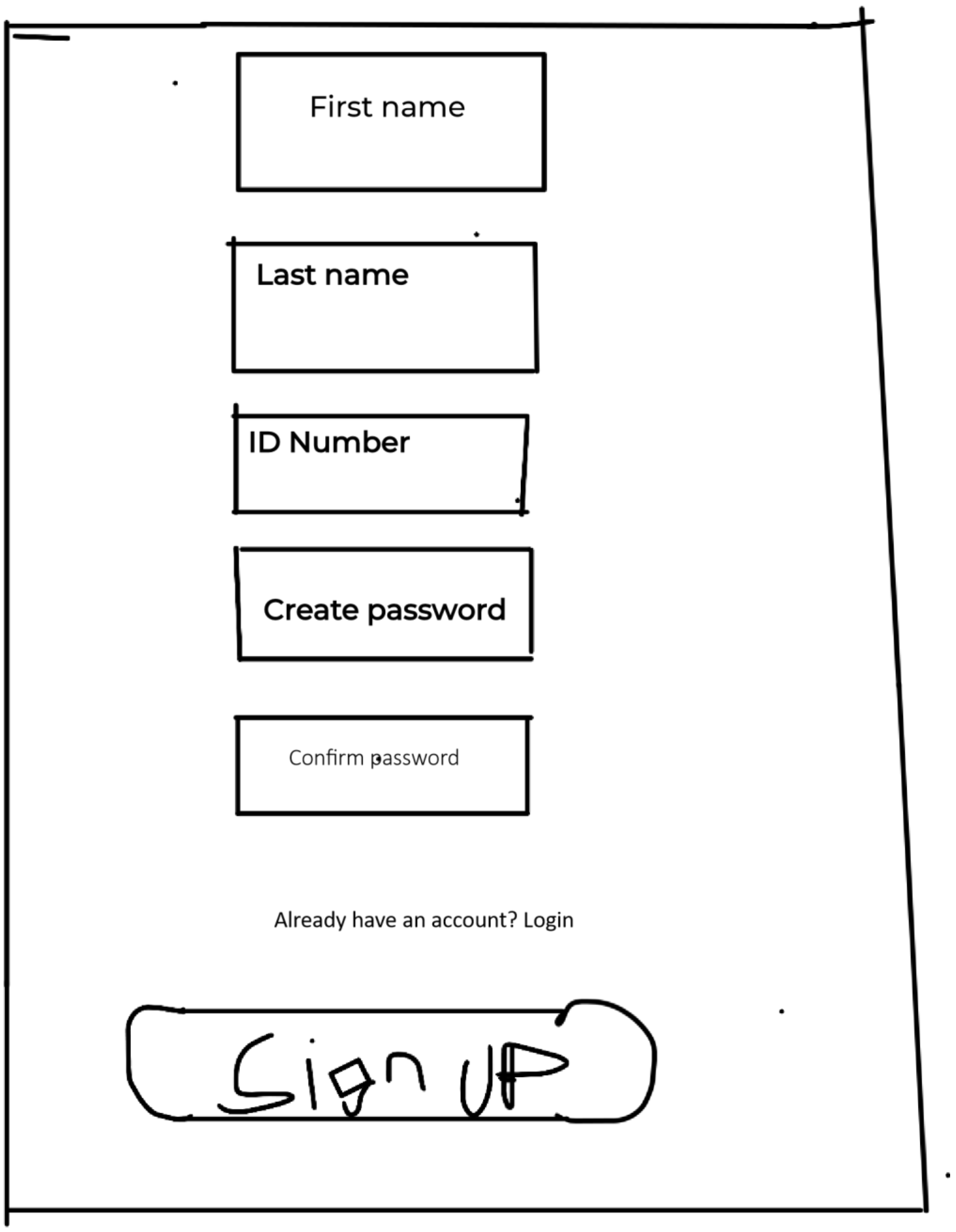
2.1 Introduction to modelling

. This section introduces the system's modelling approach, which uses user-selected regions to recommend suitable crops and ideal soil pH based on seasonal trends. It includes UI models for user interaction and logic models that handle data processing, making the app both user-friendly and agriculturally intelligent.

2.2User interface models

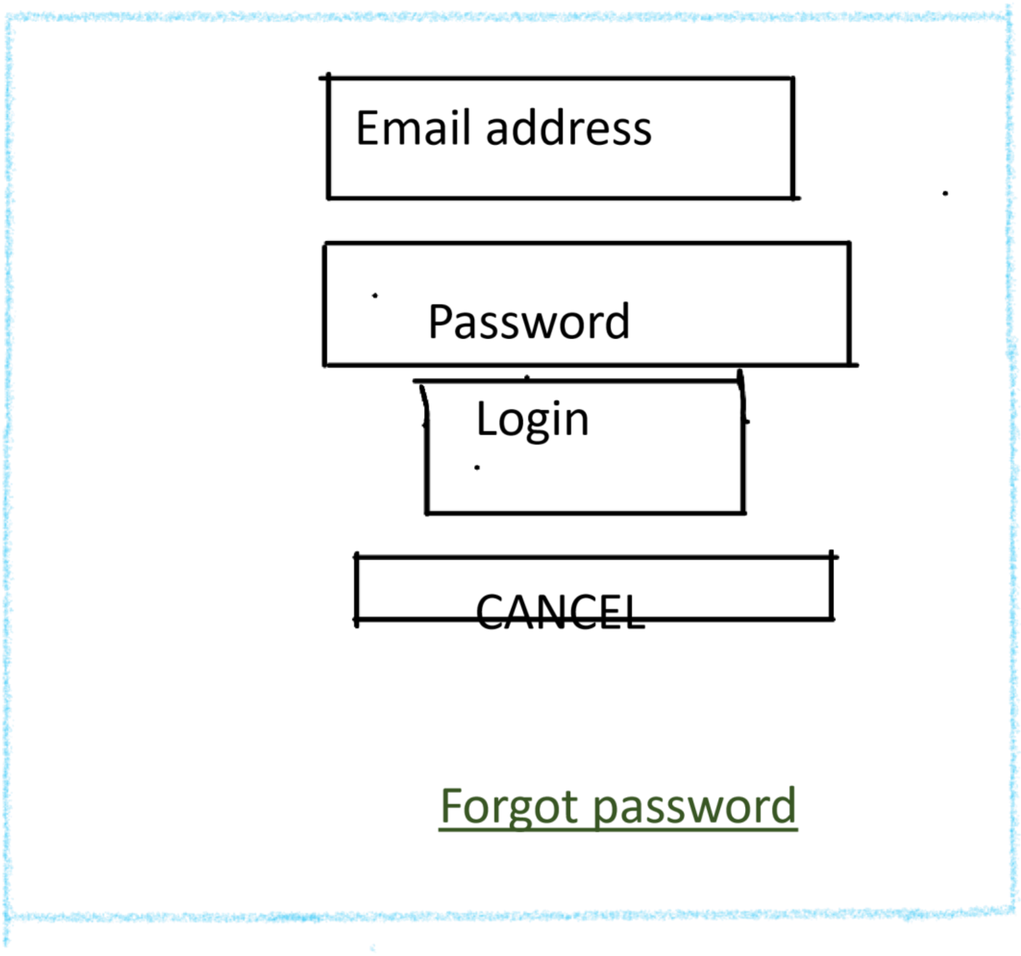
2.2.1Sign Up Form

This form will allow a new librarian to fill in their details so that they can start using the system and it will look as follows:



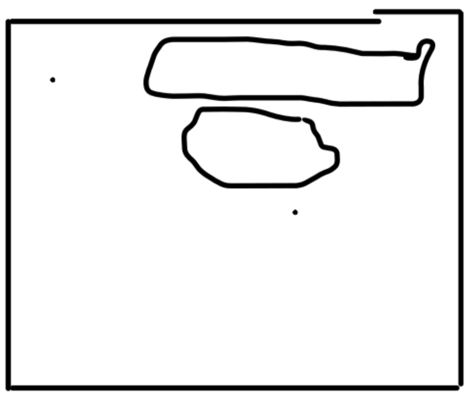
2.2.2 Login page

In this page, the librarian will enter his/her login credentials to access the system, and shall look as pictured below:



2.2.3 Main dashboard

This is the home page where the farmer or the investor will get to after a successful log in and will contain the primary controls for the system like weather condition of future, marketable crop to be planted in that area or region. The dashboard will look as follows.



Company logo

Region

CHAPTER THREE; SYSTEM IMPLEMENTATION

(DEVELOPMENT, TESTING AND DEPLOYMENT)

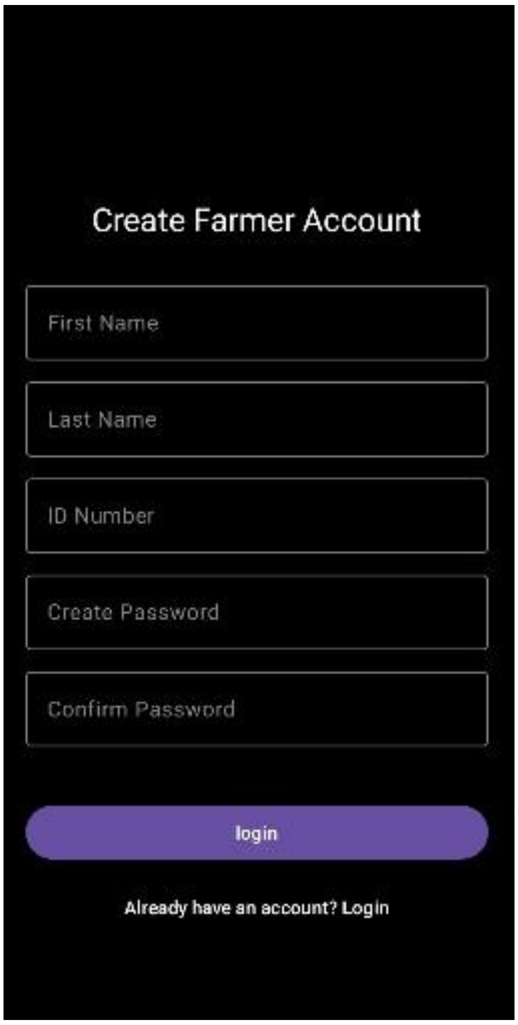
3.1 INTRODUCTION

This chapter presents the implementation experience of my seasonal trends yields analysis application and how the system went from idea to reality. The chapter starts and the development stage, where i developed the interface module, and also added logic functionality. This is followed by region selection, so as to choose the region of preference and user can also choose data analysis to be provided with a bar graph of analysis of different regions during different months of the year.

3.2 User Interface Development

3.2.1 Sign up page screenshot

A screen shot of how my sign-up page looks like where users will be required to insert their credentials in order to be registered



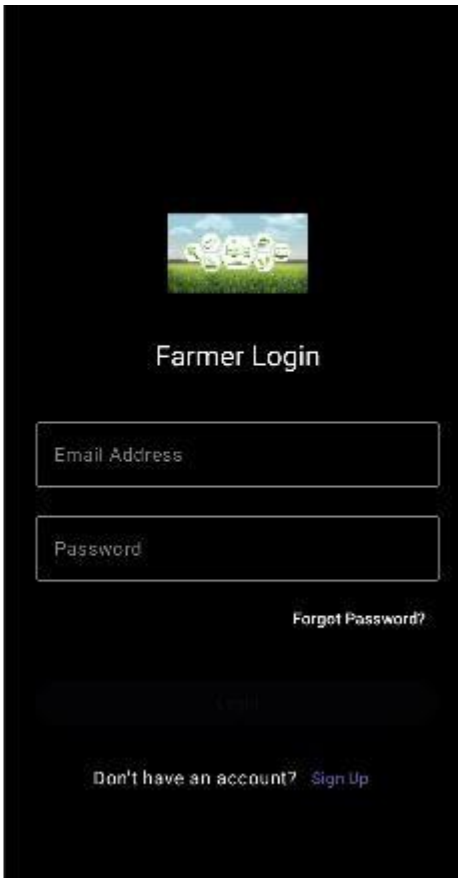
3.2.2 sign up page code

```
package
com.example.projtrends

import androidx.compose.foundation.layout.* import
androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.* import
androidx.compose.runtime.* import
androidx.compose.runtime.getValue import
androidx.compose.runtime.setValue import
androidx.compose.ui.Alignment import
androidx.compose.ui.Modifier import
androidx.compose.ui.graphics.Color import
androidx.compose.ui.text.input.KeyboardType import
androidx.compose.ui.text.input.PasswordVisualTransforma
tion import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp

@Composable fun SignUpScreen(     onSignUpClick: (firstName: String,
lastName: String, idNumber: String, password: String) -> Unit,
    onAlreadyHaveAccountClick: () -> Unit
) {
    var firstName by remember { mutableStateOf("") }     var
lastName by remember { mutableStateOf("") }     var
idNumber by remember { mutableStateOf("") }     var
password by remember { mutableStateOf("") }     var
confirmPassword by remember { mutableStateOf("") }
var isSigningUp by remember { mutableStateOf(false) }
    var passwordError by remember { mutableStateOf<String?>(null) }


    val textColor = Color.White     val textFieldColors =
OutlinedTextFieldDefaults.colors(        focusedTextColor =
textColor,        unfocusedTextColor = textColor,
focusedBorderColor = Color.LightGray,
unfocusedBorderColor = Color.Gray,
focusedLabelColor = Color.LightGray,
unfocusedLabelColor = Color.Gray,        cursorColor =
textColor,        errorBorderColor =
MaterialTheme.colorScheme.error,        errorLabelColor =
MaterialTheme.colorScheme.error
    )
    val buttonContainerColor = MaterialTheme.colorScheme.primary
val buttonContentColor = MaterialTheme.colorScheme.onPrimary

    Column(
        modifier = Modifier
            .fillMaxSize()
```

```
         .padding(24.dp),        verticalArrangement =
Arrangement.Center,        horizontalAlignment =
Alignment.CenterHorizontally
   ) {
      Text(
         text = "Create Farmer Account",            style =
MaterialTheme.typography.headlineSmall,
modifier = Modifier.padding(bottom = 24.dp),
color = textColor
      )

      OutlinedTextField(            value =
firstName,            onValueChange = {
firstName = it },            label = {
Text("First Name") },            singleLine =
true,            modifier =
Modifier.fillMaxWidth(),            colors =
textFieldColors
      )
      Spacer(modifier = Modifier.height(8.dp))

      OutlinedTextField(            value =
lastName,            onValueChange = {
lastName = it },            label = { Text("Last
Name") },            singleLine = true,
modifier = Modifier.fillMaxWidth(),
colors = textFieldColors
      )
      Spacer(modifier = Modifier.height(8.dp))

      OutlinedTextField(
         value = idNumber,
         onValueChange = {
         idNumber = it },
         label = { Text("ID Number") },
singleLine = true,
          keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
          modifier = Modifier.fillMaxWidth(),
colors = textFieldColors
      )
      Spacer(modifier = Modifier.height(8.dp))

      OutlinedTextField(
value = password,
onValueChange = {
password = it
passwordError = null
          },
          label = { Text("Create Password") },           singleLine =
true,           visualTransformation =
PasswordVisualTransformation(),           keyboardOptions =
KeyboardOptions(keyboardType = KeyboardType.Password),
          modifier =
Modifier.fillMaxWidth(),           colors =
textFieldColors,           isError =
passwordError != null
      )
      Spacer(modifier = Modifier.height(8.dp))

      OutlinedTextField(
value = confirmPassword,
onValueChange = {
confirmPassword = it
passwordError = null
          },
          label = { Text("Confirm Password") },
          singleLine = true,
          visualTransformation = PasswordVisualTransformation(),
          keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Password),
            modifier =
Modifier.fillMaxWidth(),            colors =
textFieldColors,            isError =
passwordError != null,
supportingText = {
                if (passwordError != null) {
                    Text(passwordError!!, color = MaterialTheme.colorScheme.error)
                }
            }
        )

        Spacer(modifier = Modifier.height(24.dp))

        Button(            onClick = {                if
(password != confirmPassword) {
                    passwordError = "Passwords do not match."
} else if (firstName.isBlank() || lastName.isBlank() ||
idNumber.isBlank() || password.isBlank()) {
passwordError = "All fields are required."
                }
                 else {
isSigningUp = true
passwordError = null

                    onSignUpClick(firstName, lastName, idNumber, password)

                    println("Sign Up Clicked: $firstName, $lastName, $idNumber")
                }
            },
            modifier = Modifier.fillMaxWidth(),
            enabled = !isSigningUp,
            colors = ButtonDefaults.buttonColors(
                containerColor = buttonContainerColor,
contentColor = buttonContentColor
            )
        ) {
            if (isSigningUp) {
                CircularProgressIndicator(
modifier = Modifier.size(24.dp),
color = buttonContentColor
                )
            } else {
                Text("login")
            }
        }

        Spacer(modifier = Modifier.height(8.dp))

        TextButton(onClick = {            if
(!isSigningUp) {
onAlreadyHaveAccountClick()
            }
}) {
            Text("Already have an account? Login", color = textColor)
        }
    }
}

@Preview(showBackground = true, backgroundColor = 0xFF000000)
@Composable
fun SignUpScreenPreview() {
    MaterialTheme {
        Surface(color = Color.Black) {            SignUpScreen(
onSignUpClick = { _, _, _, _ -> println("Preview login Click") },
onAlreadyHaveAccountClick = { println("Preview sign up Click") }            )
        }
    } }
```

3.2.3 Login page

A screenshot of how my login page looks like where users will be required to insert their credentials in order to login



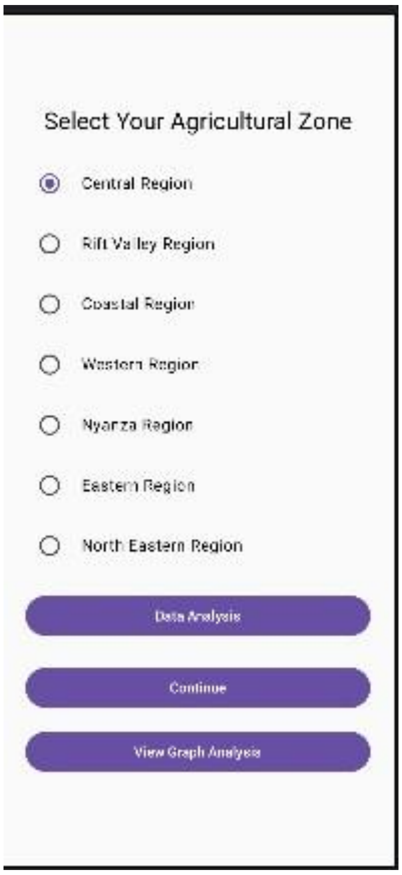
3.2.4 Login page code

```
package com.example.projtrends // Assuming this is your package, adjust
if necessary

import androidx.compose.foundation.Image import
androidx.compose.foundation.layout.Arrangement import
androidx.compose.foundation.layout.Column import
androidx.compose.foundation.layout.Row import
androidx.compose.foundation.layout.Spacer import
androidx.compose.foundation.layout.fillMaxSize import
androidx.compose.foundation.layout.fillMaxWidth import
androidx.compose.foundation.layout.height import
androidx.compose.foundation.layout.padding import
androidx.compose.foundation.layout.size import
androidx.compose.foundation.text.KeyboardOptions import
androidx.compose.material3.Button import
androidx.compose.material3.ButtonDefaults import
androidx.compose.material3.CircularProgressIndicator import
androidx.compose.material3.MaterialTheme import
androidx.compose.material3.OutlinedTextField import
androidx.compose.material3.OutlinedTextFieldDefaults import
androidx.compose.material3.Surface import
androidx.compose.material3.Text import
androidx.compose.material3.TextButton import
androidx.compose.runtime.Composable import
androidx.compose.runtime.getValue import
androidx.compose.runtime.mutableStateOf import
androidx.compose.runtime.remember import
androidx.compose.runtime.setValue import
androidx.compose.ui.Alignment import
androidx.compose.ui.Modifier import
androidx.compose.ui.graphics.Color import
androidx.compose.ui.res.painterResource import
androidx.compose.ui.text.input.KeyboardType import
androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.tooling.preview.Preview import
androidx.compose.ui.unit.dp


@Composable fun LoginScreenWithForgotPassword(
onLoginClick: (email: String, password: String) -> Unit,
onForgotPasswordClick: () -> Unit,     onSignUpClick: () ->
Unit // For "Don't have an account?"
   ) {
   var email by remember { mutableStateOf("") }     var
password by remember { mutableStateOf("") }     var
isLoggingIn by remember { mutableStateOf(false) }

   // Styling     val textColor = Color.White // Assuming black background as
per previous context     val textFieldColors =
OutlinedTextFieldDefaults.colors(        focusedTextColor = textColor,
unfocusedTextColor = textColor,        focusedBorderColor =
Color.LightGray,        unfocusedBorderColor = Color.Gray,
focusedLabelColor = Color.LightGray,        unfocusedLabelColor =
Color.Gray,        cursorColor = textColor
   )
   val buttonContainerColor = MaterialTheme.colorScheme.primary
val buttonContentColor = MaterialTheme.colorScheme.onPrimary

   Column(
modifier = Modifier
         .fillMaxSize()
         .padding(24.dp),        verticalArrangement =
Arrangement.Center,        horizontalAlignment =
Alignment.CenterHorizontally
   ) {
      Image(              painter = painterResource(id = R.drawable.agritrends), //
Image name is agritrends
         contentDescription = "AgriTrends Logo", // Updated content description
modifier = Modifier
            .size(120.dp) // Adjust size as needed
            .padding(bottom = 24.dp)
      )

      Text(           text = "Farmer Login",           style =
MaterialTheme.typography.headlineSmall,
modifier = Modifier.padding(bottom = 32.dp),
color = textColor
      )

      OutlinedTextField(           value =
email,           onValueChange = { email =
it },           label = { Text("Email Address")
},           singleLine = true,           modifier
= Modifier.fillMaxWidth(),           colors =
textFieldColors,
         keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Email)
        )
        Spacer(modifier = Modifier.height(16.dp))

        OutlinedTextField(            value = password,
onValueChange = { password = it },            label = {
Text("Password") },            singleLine = true,            modifier =
Modifier.fillMaxWidth(),            colors = textFieldColors,
visualTransformation = PasswordVisualTransformation(),
keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Password)
        )
        Spacer(modifier = Modifier.height(8.dp))

        TextButton(
onClick = {                if
(!isLoggingIn) {
                onForgotPasswordClick()
            }
          },
          modifier = Modifier.align(Alignment.End)
        ) {
          Text("Forgot Password?", color = textColor)
        }

        Spacer(modifier = Modifier.height(24.dp))

        Button(            onClick = {                if
(email.isNotBlank() && password.isNotBlank()) {
isLoggingIn = true                    onLoginClick(email,
password)
                // In a real app, isLoggingIn would be reset after the operation
completes
                println("Login Clicked: Email=$email")
            }
          },
          modifier = Modifier.fillMaxWidth(),           enabled = !isLoggingIn &&
email.isNotBlank() && password.isNotBlank(),           colors =
ButtonDefaults.buttonColors(              containerColor = buttonContainerColor,
contentColor = buttonContentColor
          )
      ) {
          if (isLoggingIn) {
              CircularProgressIndicator(
                  modifier = Modifier.size(24.dp),
color = buttonContentColor
              )
          } else {
              Text("Login")
          }
      }

      Spacer(modifier = Modifier.height(16.dp))

      Row(           modifier = Modifier.fillMaxWidth(),
horizontalArrangement = Arrangement.Center,
verticalAlignment = Alignment.CenterVertically
      ) {
          Text("Don't have an account?", color =
textColor)           TextButton(onClick = {              if
(!isLoggingIn) {                onSignUpClick()
              }
          }) {
              Text("Sign Up", color = MaterialTheme.colorScheme.primary)
          }
      }
  }
}

@Preview(showBackground = true, backgroundColor = 0xFF000000) //
Assuming black background for preview
@Composable
```

```
fun LoginScreenWithForgotPasswordPreview() {
   MaterialTheme { // Wrap with your app's theme if it defines specific colors
       Surface(color = Color.Black) { // Assuming your Surface is
black           LoginScreenWithForgotPassword(
onLoginClick = { _, _ -> println("Preview Login Click") },
onForgotPasswordClick = { println("Preview Forgot Password
Click") },              onSignUpClick = { println("Preview Sign Up
Click") }
           )
       }
   } }
```

3.2.5 Dashboard page/Homepage

A screenshot of how my dashboard page looks like where users will interact with different features



3.2.6 Dashboard page code

```
// File: RegionSelectionScreen.kt package com.example.projtrends // Ensure
this package matches your project structure

import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.material3.* import
androidx.compose.runtime.* import
androidx.compose.runtime.getValue import
androidx.compose.runtime.setValue import
androidx.compose.ui.Alignment import
androidx.compose.ui.Modifier import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp

@Composable
fun RegionSelectionScreen(     selectedRegion: String,
onRegionChange: (String) -> Unit,     onContinueClick: (selectedRegion:
String) -> Unit, // For the "Continue" button
   onNavigateToBarChart: () -> Unit,      // For "View Graph Analysis" AND "Data
Analysis" button
   // You could have a separate lambda for the new button if its navigation
target or logic is different
   // onNavigateToOverallAnalysis: () -> Unit
) {
   val regions = listOf(
"Central Region",
       "Rift Valley Region",
       "Coastal Region",
       "Western Region",
       "Nyanza Region",
       "Eastern Region",
       "North Eastern Region"

   )
   Column(
       modifier = Modifier
          .fillMaxSize()
          .padding(24.dp),
verticalArrangement = Arrangement.Center,
horizontalAlignment = Alignment.Start
   ) {
       Text(            text = "Select Your Agricultural Zone",
style = MaterialTheme.typography.headlineSmall,
modifier = Modifier.align(Alignment.CenterHorizontally)
       )

       Spacer(modifier = Modifier.height(16.dp))

       Column { // Ensure radio buttons and new button are listed
vertically           regions.forEach { region ->               Row(
verticalAlignment = Alignment.CenterVertically,                  modifier =
Modifier
                   .fillMaxWidth()
                   .padding(vertical = 6.dp)
                   .clickable { onRegionChange(region) }
              ) {
                  RadioButton(                    selected =
selectedRegion == region,                    onClick = {
onRegionChange(region) }
                  )                 Text(                    text =
region,                    modifier = Modifier.padding(start
= 8.dp),                    style =
MaterialTheme.typography.bodyLarge
                  )
              }
          }

          // New "Data Analysis" button after the regions list
```

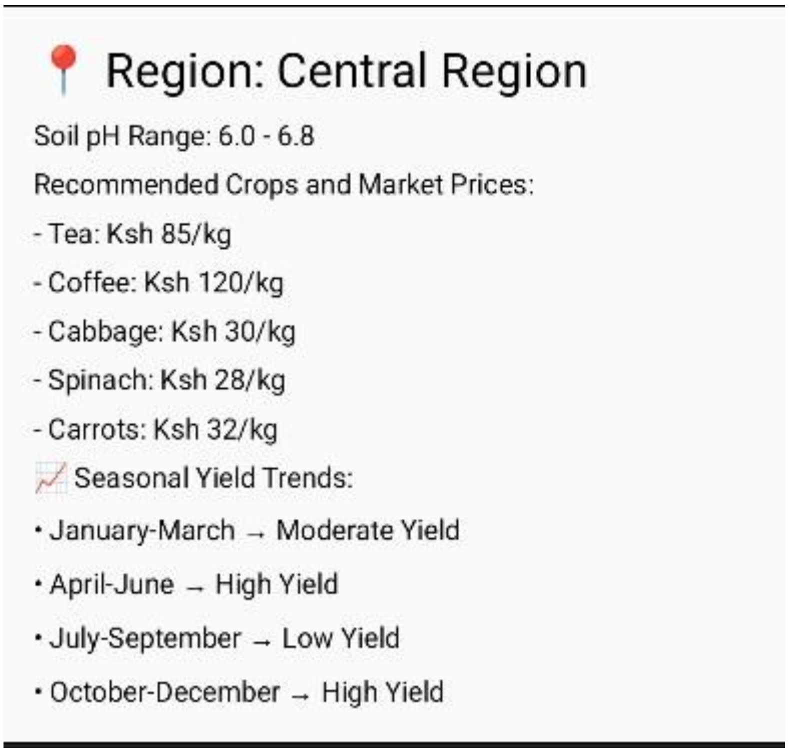
```
         Spacer(modifier = Modifier.height(16.dp)) // Spacing before the new
button
         Button(
            onClick = onNavigateToBarChart, // Reusing the same navigation
lambda
            modifier = Modifier.fillMaxWidth()
         ) {
            Text("Data Analysis")
         }
      }

      Spacer(modifier = Modifier.height(24.dp)) // Spacing before the original
bottom buttons

      // "Continue" button - uses its original logic
Button(           onClick = {
onContinueClick(selectedRegion) },           modifier =
Modifier.fillMaxWidth()
      ) {
         Text("Continue")
      }

      Spacer(modifier = Modifier.height(16.dp)) // Spacing between buttons

      // "View Graph Analysis" button - also navigates to the bar chart
Button(           onClick = onNavigateToBarChart, // Reusing the same
navigation lambda           modifier = Modifier.fillMaxWidth()
      ) {
         Text("View Graph Analysis")
      }
   }
}

@Preview(showBackground = true)
@Composable
```

fun RegionSelectionScreenPreview() {   // Local state for the preview   var previewSelectedRegion by remember **{** *mutableStateOf*("Central Region")
}

   MaterialTheme **{**
      RegionSelectionScreen(selectedRegion = previewSelectedRegion,   onRegionChange = **{** previewSelectedRegion = **it }**,   onContinueClick = **{** region **-
>**   *println*("Preview: Continue clicked with region: $region")
          },
          onNavigateToBarChart = **{**   *println*("Preview: Navigate to Bar Chart requested (from Data Analysis or View Graph Analysis).")
          }
)
   } }

3.2.7 Region repository central repository

A screenshot of how my region repository page looks like after selecting central region.in the region selection.



📍 Region: Central Region

Soil pH Range: 6.0 - 6.8

Recommended Crops and Market Prices:

- Tea: Ksh 85/kg
- Coffee: Ksh 120/kg
- Cabbage: Ksh 30/kg
- Spinach: Ksh 28/kg
- Carrots: Ksh 32/kg

📈 Seasonal Yield Trends:

• January-March → Moderate Yield
• April-June → High Yield
• July-September → Low Yield
• October-December → High Yield

3.2.8 region repository central region code

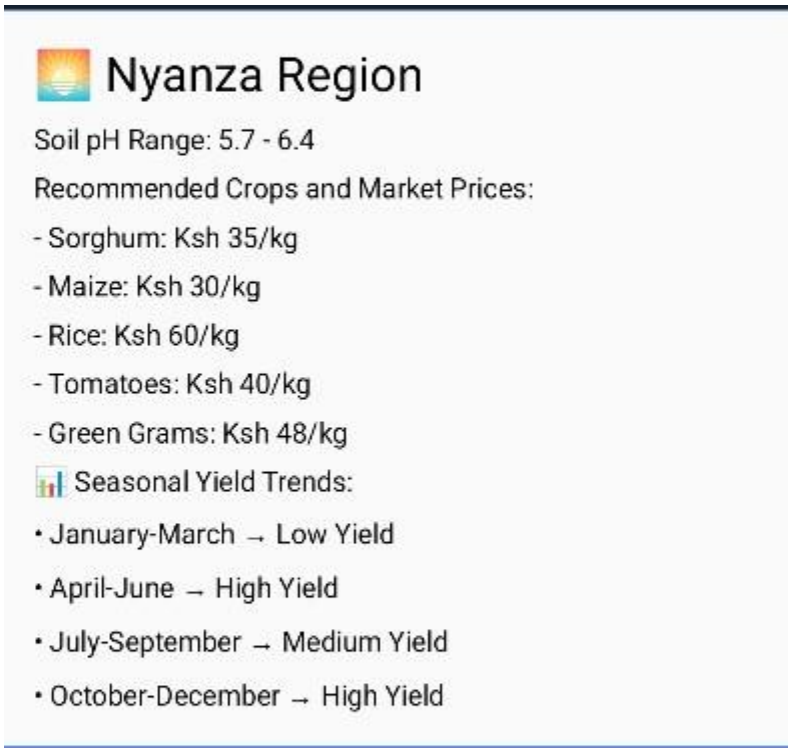
```
package com.example.projtrends.data

import android.content.Context import
androidx.compose.foundation.layout.* import
androidx.compose.material3.* import
androidx.compose.runtime.Composable import
androidx.compose.runtime.remember import
androidx.compose.ui.Modifier import
androidx.compose.ui.platform.LocalContext import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp import
org.json.JSONObject import java.io.IOException

// This object handles loading and accessing regional agricultural data object
RegionRepository {

  // Loads the entire JSON file from the assets folder
  fun loadRegionData(context: Context): JSONObject {
return try {          val jsonString =
context.assets.open("region_data.json")
            .bufferedReader()
            .use { it.readText() }
        JSONObject(jsonString)
} catch (e: IOException) {
e.printStackTrace()
        JSONObject()
     }
  }

  // Fetches a specific region's data from the full JSON     fun
getRegionInfo(context: Context, regionName: String): JSONObject? {
val regionData = loadRegionData(context)        return
regionData.optJSONObject(regionName)
  }
}
```

```
@Preview(showBackground = true)
@Composable fun
PreviewSampleRegionData() {     val
context = LocalContext.current
val region = "Central Region"

   // Safely load region data at preview time     val regionInfo = remember {
RegionRepository.getRegionInfo(context, region)
}

   Column(
modifier = Modifier
.fillMaxWidth()
         .padding(16.dp),        verticalArrangement =
Arrangement.spacedBy(8.dp)
   ) {
      Text("   Region: $region", style =
MaterialTheme.typography.headlineSmall)

      if (regionInfo != null) {
         // Display soil pH
         Text("Soil pH Range: ${regionInfo.optString("ph")}")

         // Display crops with prices           val crops =
regionInfo.optJSONArray("crops")           val market =
regionInfo.optJSONObject("marketValues")           val
cropCount = crops?.length() ?: 0

         if (cropCount > 0 && market != null) {
            Text("Recommended Crops and Market
Prices:")              for (i in 0 until cropCount) {
val crop = crops.getString(i)                  val price =
market.optInt(crop, 0)
               Text("- $crop: Ksh $price/kg")
            }
```

```
          }

          // Seasonal yield trends            val trends =
regionInfo.optJSONObject("seasonalTrends")            if (trends
!= null) {
             Text("   Seasonal Yield Trends:")
trends.keys().forEach { season ->                    val
trend = trends.optString(season, "")
                Text("• $season → $trend")
             }
          }
      } else {
          Text("   No region data found.")
      }
   }
}
```

3.2.9 region repository Nyanza region

A screenshot of how my region repository page looks like after selecting Nyanza region.in the region selection.



Nyanza Region

Soil pH Range: 5.7 - 6.4

Recommended Crops and Market Prices:

- Sorghum: Ksh 35/kg
- Maize: Ksh 30/kg
- Rice: Ksh 60/kg
- Tomatoes: Ksh 40/kg
- Green Grams: Ksh 48/kg

📊 Seasonal Yield Trends:

• January-March → Low Yield

• April-June → High Yield

• July-September → Medium Yield

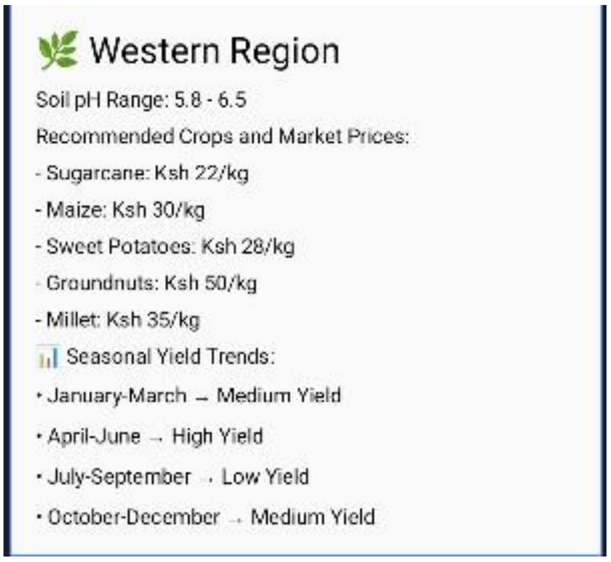
• October-December → High Yield

3.2.10 region repository Nyanza region code

```
package com.example.projtrends.data

import android.content.Context import
androidx.compose.foundation.layout.* import
androidx.compose.material3.* import
androidx.compose.runtime.Composable import
androidx.compose.runtime.remember import
androidx.compose.ui.Modifier import
androidx.compose.ui.platform.LocalContext import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp import
org.json.JSONObject

object RegionRepositoryNyanza {

   fun getNyanzaInfo(context: Context): JSONObject? {
val fullData = RegionRepository.loadRegionData(context)
return fullData.optJSONObject("Nyanza Region")
   }

   fun getPHRange(context: Context): String {
      return getNyanzaInfo(context)?.optString("ph") ?: "N/A"
   }

   fun getCrops(context: Context): List<String> {
return getNyanzaInfo(context)
         ?.optJSONArray("crops")
         ?.let { array ->
            List(array.length()) { i -> array.getString(i) }
         } ?: emptyList()
   }

   fun getMarketPrices(context: Context): Map<String, Int> {
return getNyanzaInfo(context)
         ?.optJSONObject("marketValues")
```

```
        ?.let { market ->
market.keys().asSequence().associateWith { crop ->
market.optInt(crop, 0)
            }
        } ?: emptyMap()
  }

  fun getSeasonalTrends(context: Context): Map<String, String> {
return getNyanzaInfo(context)
        ?.optJSONObject("seasonalTrends")
        ?.let { trends ->
trends.keys().asSequence().associateWith { season ->
trends.optString(season, "")
            }
        } ?: emptyMap()
  }
}

@Preview(showBackground = true)
@Composable fun
PreviewNyanzaRegionData() {     val
context = LocalContext.current

  val ph = remember { RegionRepositoryNyanza.getPHRange(context) }     val
crops = remember { RegionRepositoryNyanza.getCrops(context) }     val prices
= remember { RegionRepositoryNyanza.getMarketPrices(context) }     val
trends = remember {
RegionRepositoryNyanza.getSeasonalTrends(context) }

  Column(
modifier = Modifier
.fillMaxWidth()
        .padding(16.dp),       verticalArrangement =
Arrangement.spacedBy(8.dp)
  ) {
      Text("   Nyanza Region", style =
MaterialTheme.typography.headlineSmall)
        Text("Soil pH Range: $ph")

        if (crops.isNotEmpty()) {
            Text("Recommended Crops and Market
Prices:")           crops.forEach { crop ->                val
price = prices[crop] ?: 0
                Text("- $crop: Ksh $price/kg")
            }
        }

        if (trends.isNotEmpty()) {
            Text("   Seasonal Yield Trends:")
            trends.forEach { (season, trend) ->
                Text("• $season → $trend")
            }
        }
    } }
```

3.2.11 region repository western region



Western Region

Soil pH Range: 5.8 - 6.5

Recommended Crops and Market Prices:

- Sugarcane: Ksh 22/kg
- Maize: Ksh 30/kg
- Sweet Potatoes: Ksh 28/kg
- Groundnuts: Ksh 50/kg
- Millet: Ksh 35/kg

Seasonal Yield Trends:

• January-March → Medium Yield

• April-June → High Yield

• July-September → Low Yield

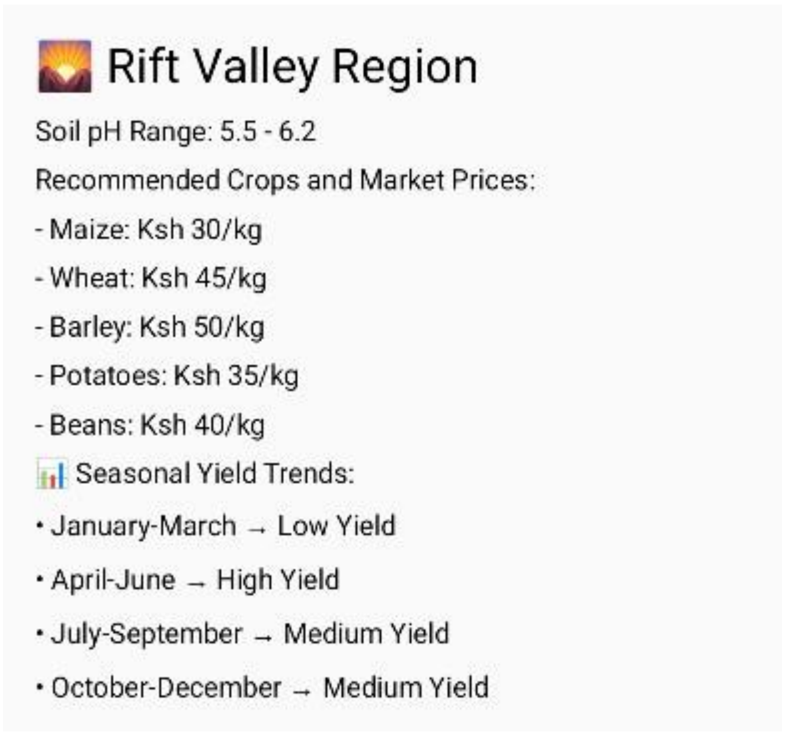
• October-December → Medium Yield

3.2.12 region repository western region code

```
package com.example.projtrends.data

import android.content.Context import
androidx.compose.foundation.layout.* import
androidx.compose.material3.* import
androidx.compose.runtime.Composable import
androidx.compose.runtime.remember import
androidx.compose.ui.Modifier import
androidx.compose.ui.platform.LocalContext import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp import
org.json.JSONObject

object RegionRepositoryWestern {

  fun getWesternInfo(context: Context): JSONObject? {
val fullData = RegionRepository.loadRegionData(context)
return fullData.optJSONObject("Western Region")
  }

  fun getPHRange(context: Context): String {        return
getWesternInfo(context)?.optString("ph") ?: "N/A"
  }

  fun getCrops(context: Context): List<String> {
return getWesternInfo(context)
         ?.optJSONArray("crops")
         ?.let { array ->
            List(array.length()) { i -> array.getString(i) }
         } ?: emptyList()
  }

  fun getMarketPrices(context: Context): Map<String, Int> {
return getWesternInfo(context)
         ?.optJSONObject("marketValues")
         ?.let { market ->
market.keys().asSequence().associateWith { crop ->
market.optInt(crop, 0)
            }
         } ?: emptyMap()
```

```
   }

   fun getSeasonalTrends(context: Context): Map<String, String> {
return getWesternInfo(context)
          ?.optJSONObject("seasonalTrends")
          ?.let { trends ->
trends.keys().asSequence().associateWith { season ->
trends.optString(season, "")
             }
          } ?: emptyMap()
   }
}

@Preview(showBackground = true)
@Composable fun
PreviewWesternRegionData() {
val context = LocalContext.current

   val ph = remember { RegionRepositoryWestern.getPHRange(context) }     val
crops = remember { RegionRepositoryWestern.getCrops(context) }     val
prices = remember { RegionRepositoryWestern.getMarketPrices(context)
}
   val trends = remember {
RegionRepositoryWestern.getSeasonalTrends(context) }

   Column(
modifier = Modifier
.fillMaxWidth()
          .padding(16.dp),        verticalArrangement =
Arrangement.spacedBy(8.dp)
   ) {
      Text("   Western Region", style =
MaterialTheme.typography.headlineSmall)
      Text("Soil pH Range: $ph")

      if (crops.isNotEmpty()) {
          Text("Recommended Crops and Market
Prices:")           crops.forEach { crop ->               val
price = prices[crop] ?: 0
             Text("- $crop: Ksh $price/kg")
          }
      }

      if (trends.isNotEmpty()) {
          Text("   Seasonal Yield Trends:")
trends.forEach { (season, trend) ->
             Text("• $season → $trend")
          }
      }
   } }
```

3.2.13 region repository Rift valley region code



Rift Valley Region

Soil pH Range: 5.5 - 6.2

Recommended Crops and Market Prices:

- Maize: Ksh 30/kg
- Wheat: Ksh 45/kg
- Barley: Ksh 50/kg
- Potatoes: Ksh 35/kg
- Beans: Ksh 40/kg

📊 Seasonal Yield Trends:

• January-March → Low Yield

• April-June → High Yield

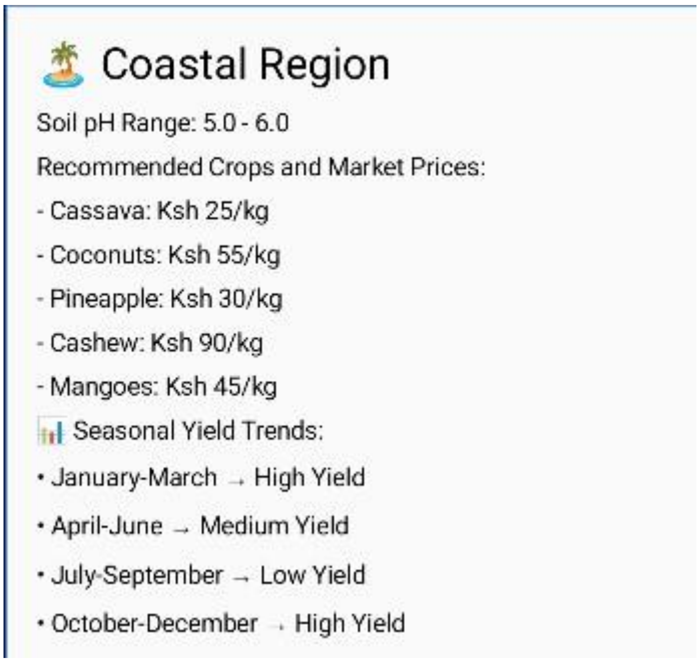
• July-September → Medium Yield

• October-December → Medium Yield

3.2.14 Region repository Rift valley region code

```
package com.example.projtrends.data

import android.content.Context import
androidx.compose.foundation.layout.* import
androidx.compose.material3.* import
androidx.compose.runtime.Composable import
androidx.compose.runtime.remember import
androidx.compose.ui.Modifier import
androidx.compose.ui.platform.LocalContext import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp import
org.json.JSONObject

object RegionRepositoryRiftValley {

   fun getRiftValleyInfo(context: Context): JSONObject? {
val fullData = RegionRepository.loadRegionData(context)
return fullData.optJSONObject("Rift Valley Region")
   }

   fun getCrops(context: Context): List<String> {
return getRiftValleyInfo(context)
          ?.optJSONArray("crops")
          ?.let { array ->
             List(array.length()) { i -> array.getString(i) }
          } ?: emptyList()
   }

   fun getPHRange(context: Context): String {        return
getRiftValleyInfo(context)?.optString("ph") ?: "N/A"
   }

   fun getMarketPrices(context: Context): Map<String, Int> {
return getRiftValleyInfo(context)
          ?.optJSONObject("marketValues")
          ?.let { market ->
market.keys().asSequence().associateWith { crop ->
market.optInt(crop, 0)
             }
          } ?: emptyMap()
   }

   fun getSeasonalTrends(context: Context): Map<String, String> {
return getRiftValleyInfo(context)
?.optJSONObject("seasonalTrends")
         ?.let { trends ->
trends.keys().asSequence().associateWith { season ->
trends.optString(season, "")
            }
         } ?: emptyMap()
   }
}

@Preview(showBackground = true)
@Composable
fun PreviewRiftValleyData() {     val
context = LocalContext.current

   val ph = remember { RegionRepositoryRiftValley.getPHRange(context) }     val
crops = remember { RegionRepositoryRiftValley.getCrops(context) }     val
prices = remember { RegionRepositoryRiftValley.getMarketPrices(context)
}
   val trends = remember {
RegionRepositoryRiftValley.getSeasonalTrends(context) }

   Column(
modifier = Modifier
.fillMaxWidth()
         .padding(16.dp),        verticalArrangement =
Arrangement.spacedBy(8.dp)
   ) {
      Text("   Rift Valley Region", style =
MaterialTheme.typography.headlineSmall)
      Text("Soil pH Range: $ph")

      if (crops.isNotEmpty()) {
         Text("Recommended Crops and Market
Prices:")           crops.forEach { crop ->              val
price = prices[crop] ?: 0               Text("- $crop: Ksh
$price/kg")
         }
      }

      if (trends.isNotEmpty()) {
         Text("   Seasonal Yield Trends:")
trends.forEach { (season, trend) ->
            Text("• $season → $trend")
         }
      }
   } }
```

3.2.15 Region repository coastal region code



Coastal Region

Soil pH Range: 5.0 - 6.0

Recommended Crops and Market Prices:

- Cassava: Ksh 25/kg
- Coconuts: Ksh 55/kg
- Pineapple: Ksh 30/kg
- Cashew: Ksh 90/kg
- Mangoes: Ksh 45/kg

📊 Seasonal Yield Trends:

• January-March → High Yield

• April-June → Medium Yield

• July-September → Low Yield

• October-December → High Yield

3.2.14 Region repository coastal region code

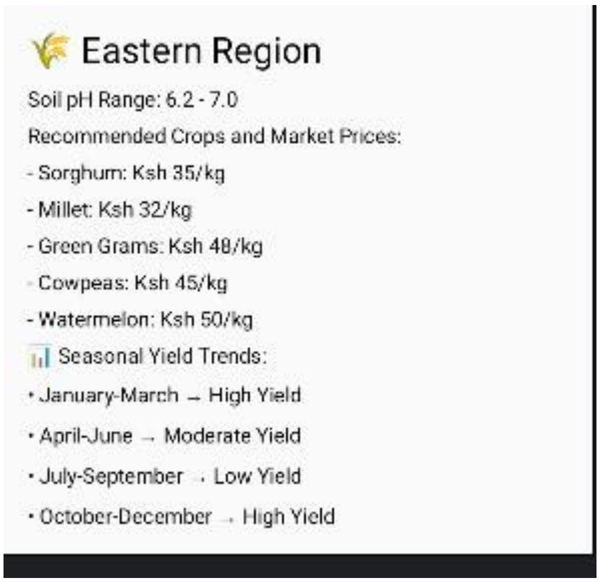
```
package com.example.projtrends.data

import android.content.Context import
androidx.compose.foundation.layout.* import
androidx.compose.material3.* import
androidx.compose.runtime.Composable
```

```
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier import
androidx.compose.ui.platform.LocalContext
import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp import
org.json.JSONObject

object RegionRepositoryCoastal {

   fun getCoastalInfo(context: Context): JSONObject? {
val fullData = RegionRepository.loadRegionData(context)
return fullData.optJSONObject("Coastal Region")
   }

   fun getCrops(context: Context): List<String> {
return getCoastalInfo(context)
         ?.optJSONArray("crops")
         ?.let { array ->
            List(array.length()) { i -> array.getString(i) }
         } ?: emptyList()
   }

   fun getPHRange(context: Context): String {
      return getCoastalInfo(context)?.optString("ph") ?: "N/A"
   }

   fun getMarketPrices(context: Context): Map<String, Int> {
return getCoastalInfo(context)
         ?.optJSONObject("marketValues")
         ?.let { market ->
market.keys().asSequence().associateWith { crop ->
market.optInt(crop, 0)
            }
         } ?: emptyMap()
   }
   fun getSeasonalTrends(context: Context): Map<String, String> {
return getCoastalInfo(context)
          ?.optJSONObject("seasonalTrends")
          ?.let { trends ->
trends.keys().asSequence().associateWith { season ->
trends.optString(season, "")
             }
          } ?: emptyMap()
   }
}

@Preview(showBackground = true)
@Composable
fun PreviewCoastalRegionData() {
val context = LocalContext.current

   val ph = remember { RegionRepositoryCoastal.getPHRange(context) }     val
crops = remember { RegionRepositoryCoastal.getCrops(context) }     val prices
= remember { RegionRepositoryCoastal.getMarketPrices(context) }     val
trends = remember {
RegionRepositoryCoastal.getSeasonalTrends(context) }

   Column(
modifier = Modifier
.fillMaxWidth()
          .padding(16.dp),        verticalArrangement =
Arrangement.spacedBy(8.dp)
   ) {
      Text("   Coastal Region", style =
MaterialTheme.typography.headlineSmall)
      Text("Soil pH Range: $ph")

      if (crops.isNotEmpty()) {
         Text("Recommended Crops and Market Prices:")
crops.forEach { crop ->
```

```
            val price = prices[crop] ?: 0
            Text("- $crop: Ksh $price/kg")
          }
        }

        if (trends.isNotEmpty()) {
          Text("   Seasonal Yield Trends:")
trends.forEach { (season, trend) ->
            Text("• $season → $trend")
          }
        }
    } }
```

3.2.15 Region repository eastern region

A screenshot of the eastern region repository.



Eastern Region

Soil pH Range: 6.2 - 7.0

Recommended Crops and Market Prices:

- Sorghum: Ksh 35/kg
- Millet: Ksh 32/kg
- Green Grams: Ksh 48/kg
- Cowpeas: Ksh 45/kg
- Watermelon: Ksh 50/kg

📊 Seasonal Yield Trends:

• January-March → High Yield

• April-June → Moderate Yield

• July-September → Low Yield

• October-December → High Yield

3.2.16 Region repository eastern region code

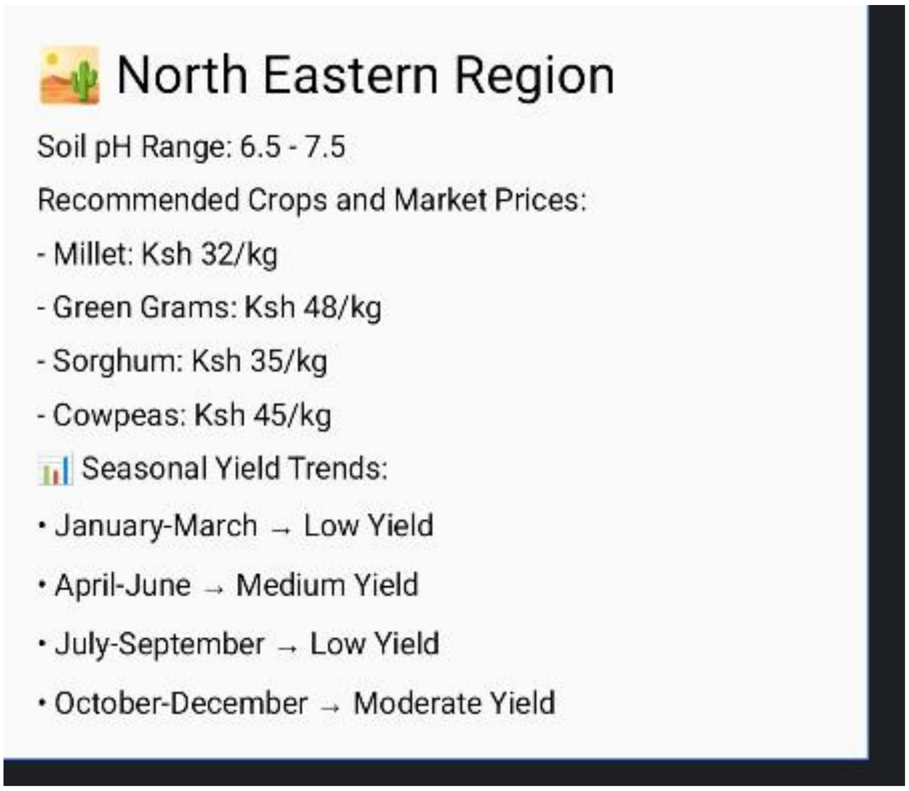
```
package com.example.projtrends.data

import android.content.Context import
androidx.compose.foundation.layout.* import
androidx.compose.material3.* import
androidx.compose.runtime.Composable
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier import
```

```
androidx.compose.ui.platform.LocalContext
import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp import
org.json.JSONObject

object RegionRepositoryEastern {

   fun getEasternInfo(context: Context): JSONObject? {
val fullData = RegionRepository.loadRegionData(context)
return fullData.optJSONObject("Eastern Region")
   }

   fun getPHRange(context: Context): String {
      return getEasternInfo(context)?.optString("ph") ?: "N/A"
   }

   fun getCrops(context: Context): List<String> {
return getEasternInfo(context)
         ?.optJSONArray("crops")
         ?.let { array ->
            List(array.length()) { i -> array.getString(i) }
         } ?: emptyList()
   }

   fun getMarketPrices(context: Context): Map<String, Int> {
return getEasternInfo(context)
         ?.optJSONObject("marketValues")
         ?.let { market ->
market.keys().asSequence().associateWith { crop ->
market.optInt(crop, 0)
            }
         } ?: emptyMap()
   }
   fun getSeasonalTrends(context: Context): Map<String, String> {
return getEasternInfo(context)
          ?.optJSONObject("seasonalTrends")
          ?.let { trends ->
trends.keys().asSequence().associateWith { season ->
trends.optString(season, "")
             }
          } ?: emptyMap()
   }
}

@Preview(showBackground = true)
@Composable
fun PreviewEasternRegionData() {
val context = LocalContext.current

   val ph = remember { RegionRepositoryEastern.getPHRange(context) }     val
crops = remember { RegionRepositoryEastern.getCrops(context) }     val prices
= remember { RegionRepositoryEastern.getMarketPrices(context) }     val
trends = remember {
RegionRepositoryEastern.getSeasonalTrends(context) }

   Column(
modifier = Modifier
.fillMaxWidth()
          .padding(16.dp),        verticalArrangement =
Arrangement.spacedBy(8.dp)
   ) {
      Text("   Eastern Region", style =
MaterialTheme.typography.headlineSmall)
      Text("Soil pH Range: $ph")

      if (crops.isNotEmpty()) {
         Text("Recommended Crops and Market Prices:")
crops.forEach { crop ->
            val price = prices[crop] ?: 0
```

```
                Text("- $crop: Ksh $price/kg")
            }
        }

        if (trends.isNotEmpty()) {
            Text("   Seasonal Yield Trends:")
trends.forEach { (season, trend) ->
                Text("• $season → $trend")
            }
        }
    } }
```

3.2.17 Region repository northeastern region code

A screenshot of how the northeastern region data analysis will look like


North Eastern Region

Soil pH Range: 6.5 - 7.5

Recommended Crops and Market Prices:

- Millet: Ksh 32/kg
- Green Grams: Ksh 48/kg
- Sorghum: Ksh 35/kg
- Cowpeas: Ksh 45/kg

📊 Seasonal Yield Trends:

• January-March → Low Yield

• April-June → Medium Yield

• July-September → Low Yield

• October-December → Moderate Yield

3.2.18 Region repository north eastern region code

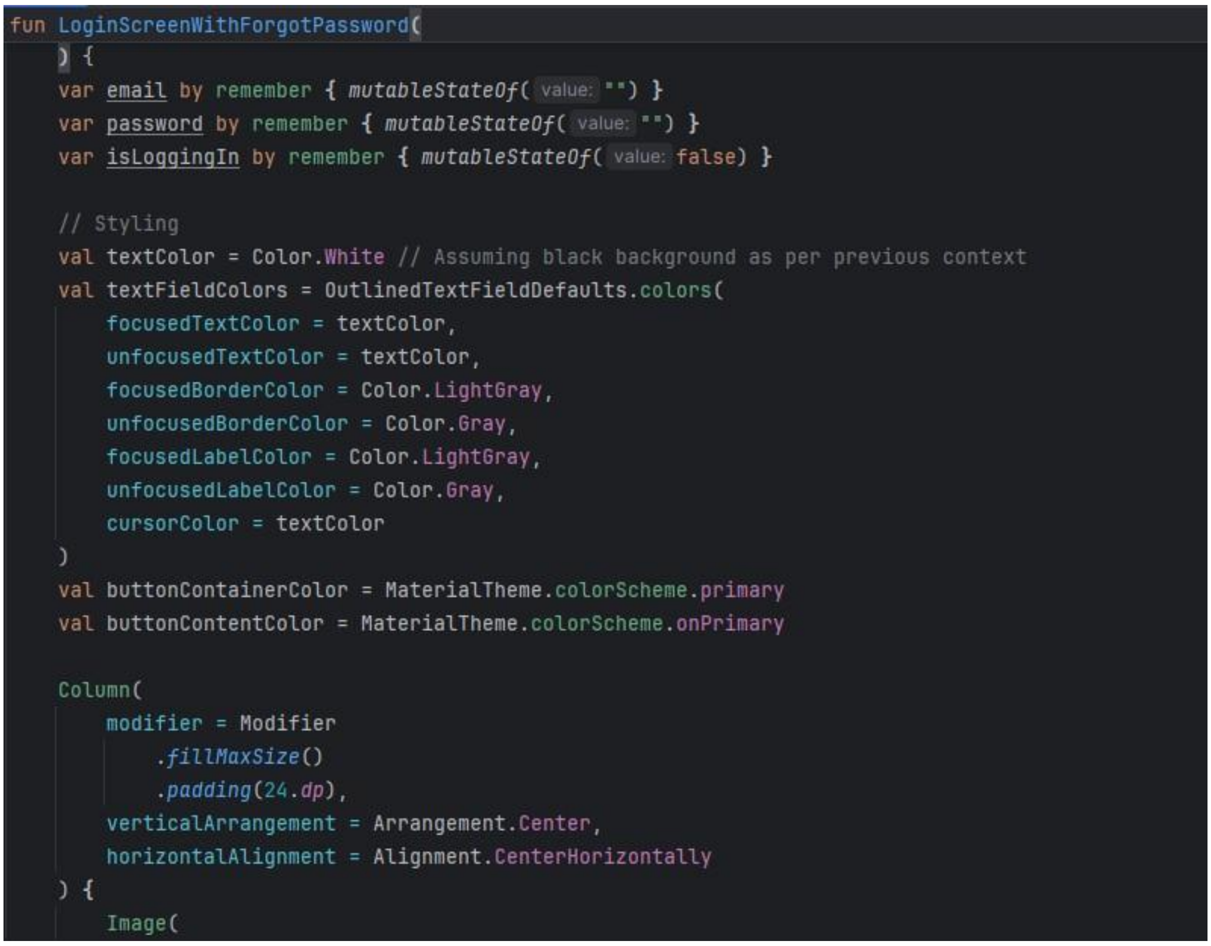
```
package com.example.projtrends.data

import android.content.Context import
androidx.compose.foundation.layout.* import
androidx.compose.material3.* import
androidx.compose.runtime.Composable import
androidx.compose.runtime.remember import
androidx.compose.ui.Modifier import
androidx.compose.ui.platform.LocalContext import
androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp import
org.json.JSONObject

object RegionRepositoryNorthEastern {

   fun getNorthEasternInfo(context: Context): JSONObject? {
val fullData = RegionRepository.loadRegionData(context)
return fullData.optJSONObject("North Eastern Region")
   }

   fun getPHRange(context: Context): String {        return
getNorthEasternInfo(context)?.optString("ph") ?: "N/A"
   }

   fun getCrops(context: Context): List<String> {
return getNorthEasternInfo(context)
         ?.optJSONArray("crops")
         ?.let { array ->
            List(array.length()) { i -> array.getString(i) }
         } ?: emptyList()
   }

   fun getMarketPrices(context: Context): Map<String, Int> {
return getNorthEasternInfo(context)
?.optJSONObject("marketValues")
         ?.let { market ->
market.keys().asSequence().associateWith { crop ->
market.optInt(crop, 0)
            }
         } ?: emptyMap()
```

```
    }

    fun getSeasonalTrends(context: Context): Map<String, String> {
return getNorthEasternInfo(context)
            ?.optJSONObject("seasonalTrends")
            ?.let { trends ->
trends.keys().asSequence().associateWith { season ->
trends.optString(season, "")
                }
            } ?: emptyMap()
    }
}

@Preview(showBackground = true)
@Composable fun
PreviewNorthEasternRegionData() {
val context = LocalContext.current

    val ph = remember { RegionRepositoryNorthEastern.getPHRange(context) }
val crops = remember { RegionRepositoryNorthEastern.getCrops(context) }
val prices = remember {
RegionRepositoryNorthEastern.getMarketPrices(context) }
val trends = remember {
RegionRepositoryNorthEastern.getSeasonalTrends(context) }

    Column(
modifier = Modifier
.fillMaxWidth()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Text("    North Eastern Region", style =
MaterialTheme.typography.headlineSmall)
        Text("Soil pH Range: $ph")

        if (crops.isNotEmpty()) {
            Text("Recommended Crops and Market
Prices:")            crops.forEach { crop ->                val
price = prices[crop] ?: 0
                Text("- $crop: Ksh $price/kg")
            }
        }

        if (trends.isNotEmpty()) {
            Text("   Seasonal Yield Trends:")
trends.forEach { (season, trend) ->
                Text("• $season → $trend")
            }
        }
    } }
```

3.3 Logic Development

3.3.1 Login Validation logic

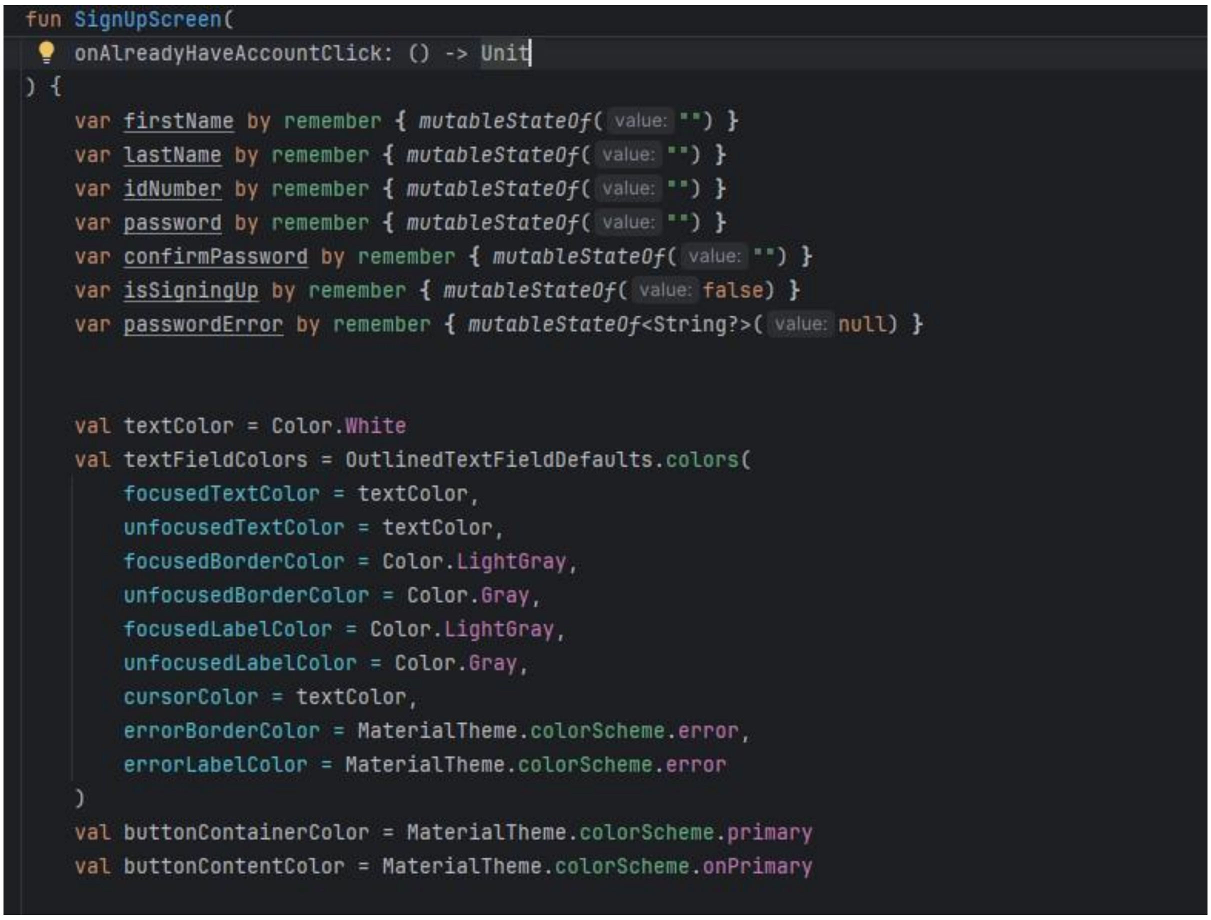
Login validation checks connectivity, ensures fields are empty, then verifies the username and password against stored credentials granting dashboard access if matched or showing an error toast if not.



```
fun LoginScreenWithForgotPassword(
    ) {
    var email by remember { mutableStateOf( value: "") }
    var password by remember { mutableStateOf( value: "") }
    var isLoggingIn by remember { mutableStateOf( value: false) }

    // Styling
    val textColor = Color.White // Assuming black background as per previous context
    val textFieldColors = OutlinedTextFieldDefaults.colors(
        focusedTextColor = textColor,
        unfocusedTextColor = textColor,
        focusedBorderColor = Color.LightGray,
        unfocusedBorderColor = Color.Gray,
        focusedLabelColor = Color.LightGray,
        unfocusedLabelColor = Color.Gray,
        cursorColor = textColor
    )
    val buttonContainerColor = MaterialTheme.colorScheme.primary
    val buttonContentColor = MaterialTheme.colorScheme.onPrimary

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(24.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Image(
```

3.3.2 Sign up validation logic

During sign up the system ensures internet connection before proceeding. It checks that all fields are filled, validates the email format and confirms that the password matches its confirmation. If everything is valid it stores the data and redirect to the login screen.

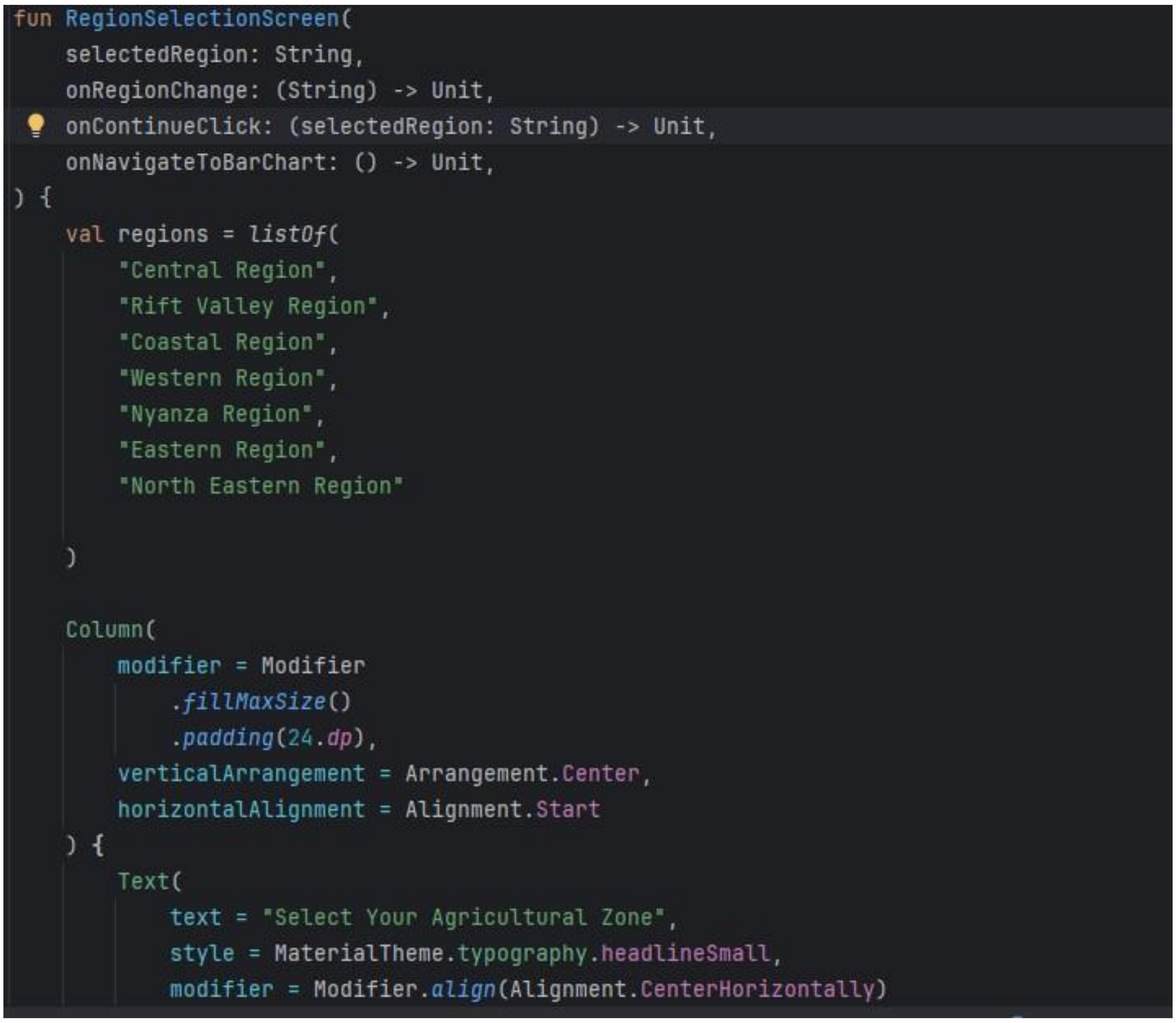


```kotlin
fun SignUpScreen(
    onAlreadyHaveAccountClick: () -> Unit
) {
    var firstName by remember { mutableStateOf( value: "") }
    var lastName by remember { mutableStateOf( value: "") }
    var idNumber by remember { mutableStateOf( value: "") }
    var password by remember { mutableStateOf( value: "") }
    var confirmPassword by remember { mutableStateOf( value: "") }
    var isSigningUp by remember { mutableStateOf( value: false) }
    var passwordError by remember { mutableStateOf<String?>( value: null) }


    val textColor = Color.White
    val textFieldColors = OutlinedTextFieldDefaults.colors(
        focusedTextColor = textColor,
        unfocusedTextColor = textColor,
        focusedBorderColor = Color.LightGray,
        unfocusedBorderColor = Color.Gray,
        focusedLabelColor = Color.LightGray,
        unfocusedLabelColor = Color.Gray,
        cursorColor = textColor,
        errorBorderColor = MaterialTheme.colorScheme.error,
        errorLabelColor = MaterialTheme.colorScheme.error
    )
    val buttonContainerColor = MaterialTheme.colorScheme.primary
    val buttonContentColor = MaterialTheme.colorScheme.onPrimary
```

3.3.3 Dashboard page validation logic

3.3.3.1 region_selection

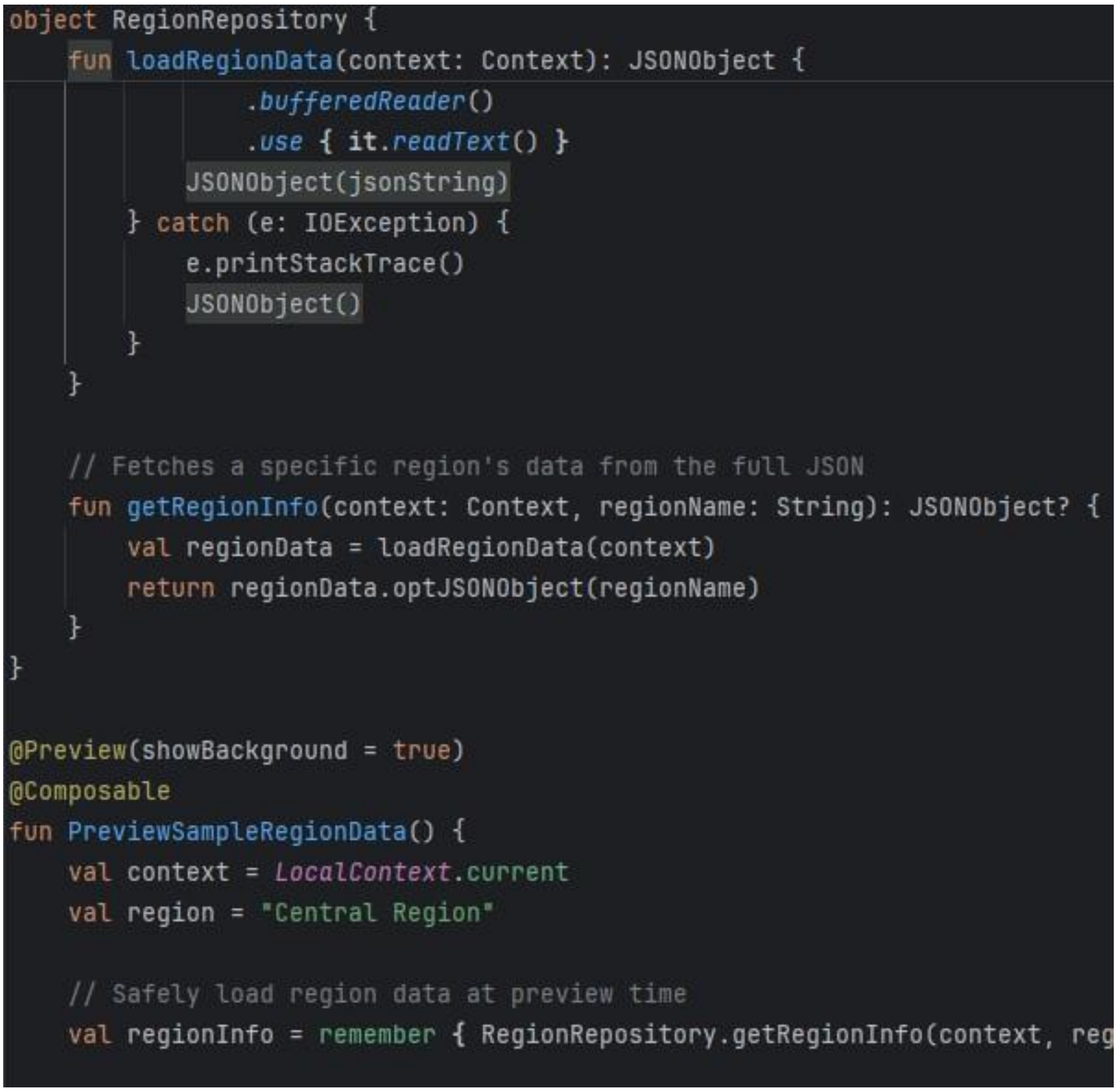
It show all the regions to be selected to be deployed to the user



```kotlin
fun RegionSelectionScreen(
    selectedRegion: String,
    onRegionChange: (String) -> Unit,
    onContinueClick: (selectedRegion: String) -> Unit,
    onNavigateToBarChart: () -> Unit,
) {
    val regions = listOf(
        "Central Region",
        "Rift Valley Region",
        "Coastal Region",
        "Western Region",
        "Nyanza Region",
        "Eastern Region",
        "North Eastern Region"

    )

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(24.dp),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.Start
    ) {
        Text(
            text = "Select Your Agricultural Zone",
            style = MaterialTheme.typography.headlineSmall,
            modifier = Modifier.align(Alignment.CenterHorizontally)
```

3.3.4 Region repository central region

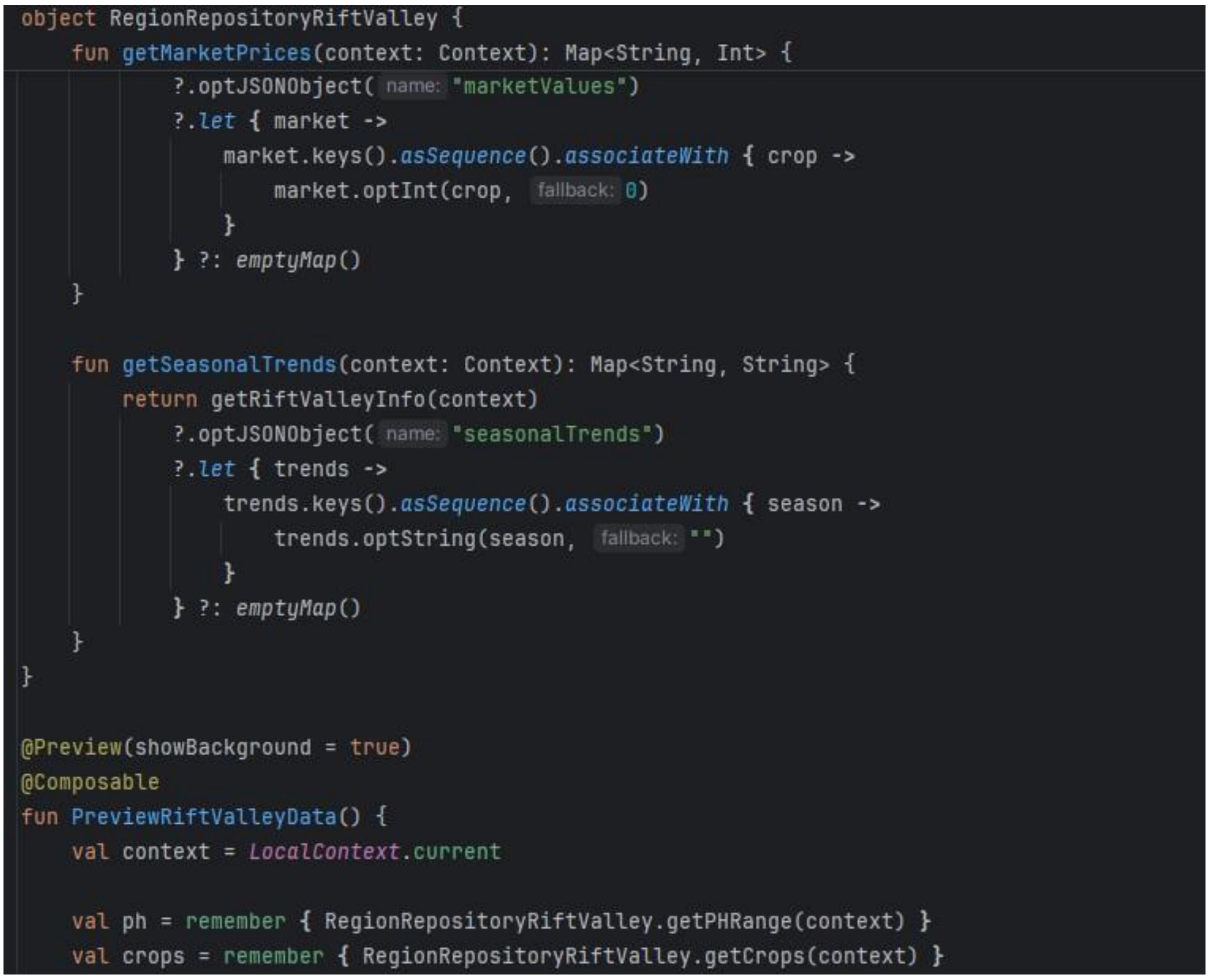
It launches the analysis of the best crops in central region, where the farmer has chosen.



```
object RegionRepository {
    fun loadRegionData(context: Context): JSONObject {
                .bufferedReader()
                .use { it.readText() }
            JSONObject(jsonString)
        } catch (e: IOException) {
            e.printStackTrace()
            JSONObject()
        }
    }

    // Fetches a specific region's data from the full JSON
    fun getRegionInfo(context: Context, regionName: String): JSONObject? {
        val regionData = loadRegionData(context)
        return regionData.optJSONObject(regionName)
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewSampleRegionData() {
    val context = LocalContext.current
    val region = "Central Region"

    // Safely load region data at preview time
    val regionInfo = remember { RegionRepository.getRegionInfo(context, reg
```

3.3.5Region repository rift valley region

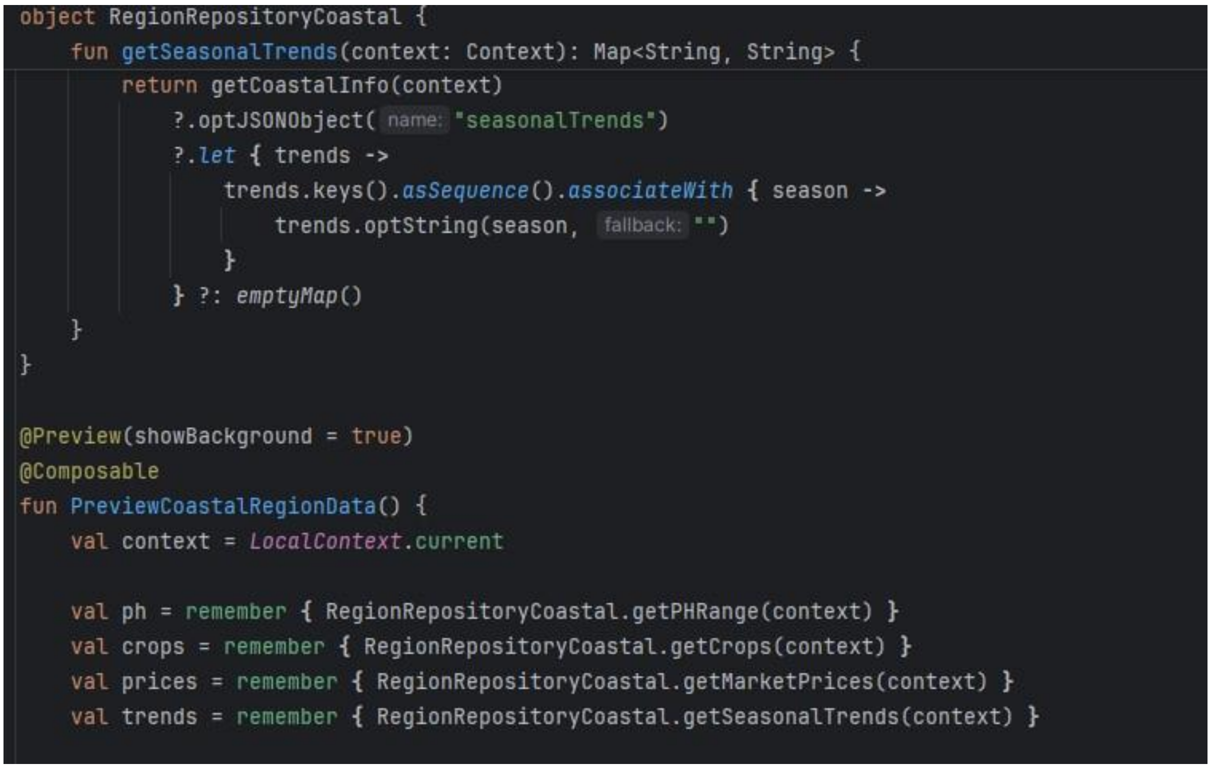
This is where the user after choosing the region the given data information of the area will be displayed.



```
object RegionRepositoryRiftValley {
    fun getMarketPrices(context: Context): Map<String, Int> {
            ?.optJSONObject( name: "marketValues")
            ?.let { market ->
                market.keys().asSequence().associateWith { crop ->
                    market.optInt(crop,  fallback: 0)
                }
            } ?: emptyMap()
    }

    fun getSeasonalTrends(context: Context): Map<String, String> {
        return getRiftValleyInfo(context)
            ?.optJSONObject( name: "seasonalTrends")
            ?.let { trends ->
                trends.keys().asSequence().associateWith { season ->
                    trends.optString(season,  fallback: "")
                }
            } ?: emptyMap()
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewRiftValleyData() {
    val context = LocalContext.current

    val ph = remember { RegionRepositoryRiftValley.getPHRange(context) }
    val crops = remember { RegionRepositoryRiftValley.getCrops(context) }
```

3.3.6 Region repository costal region

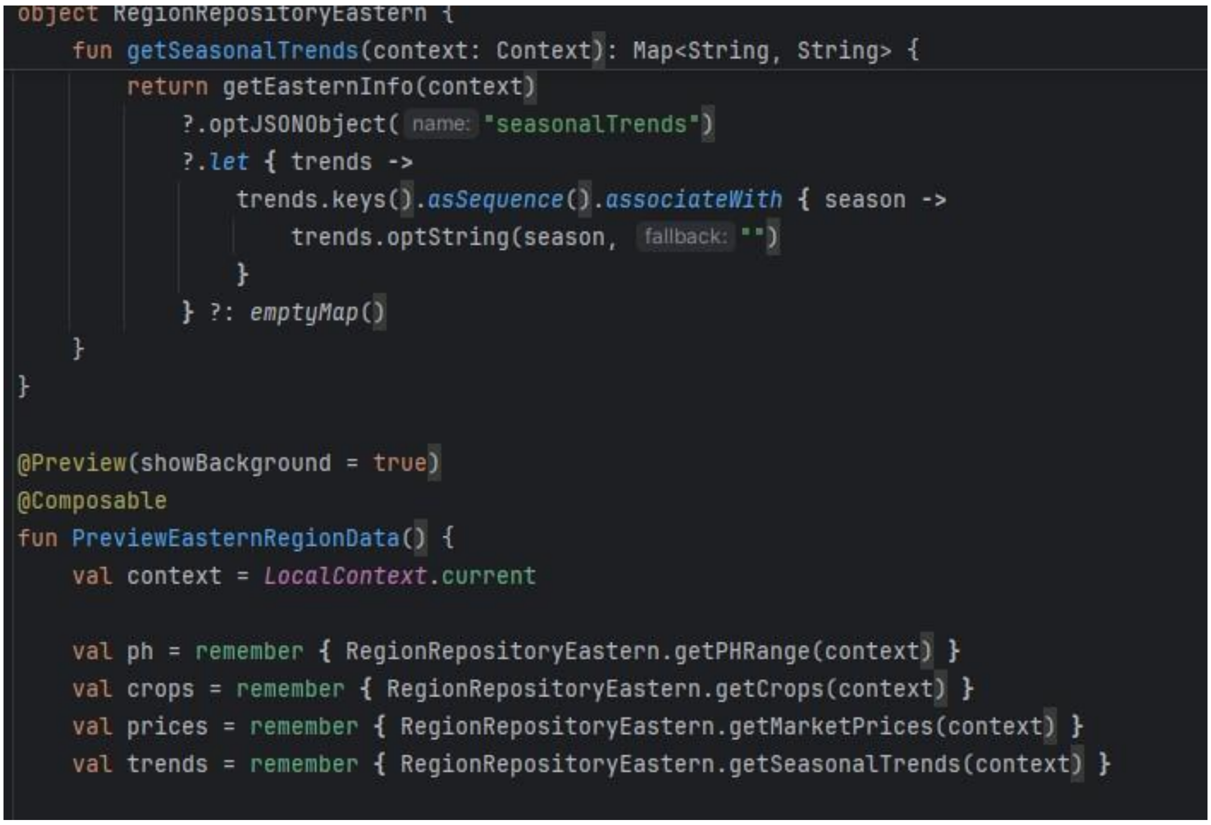
This is where the user after choosing the region the given data information of the area will be displayed.



```kotlin
object RegionRepositoryCoastal {
    fun getSeasonalTrends(context: Context): Map<String, String> {
        return getCoastalInfo(context)
            ?.optJSONObject( name: "seasonalTrends")
            ?.let { trends ->
                trends.keys().asSequence().associateWith { season ->
                    trends.optString(season,  fallback: "")
                }
            } ?: emptyMap()
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewCoastalRegionData() {
    val context = LocalContext.current

    val ph = remember { RegionRepositoryCoastal.getPHRange(context) }
    val crops = remember { RegionRepositoryCoastal.getCrops(context) }
    val prices = remember { RegionRepositoryCoastal.getMarketPrices(context) }
    val trends = remember { RegionRepositoryCoastal.getSeasonalTrends(context) }
```

3.3.7 Region repository Eastern region

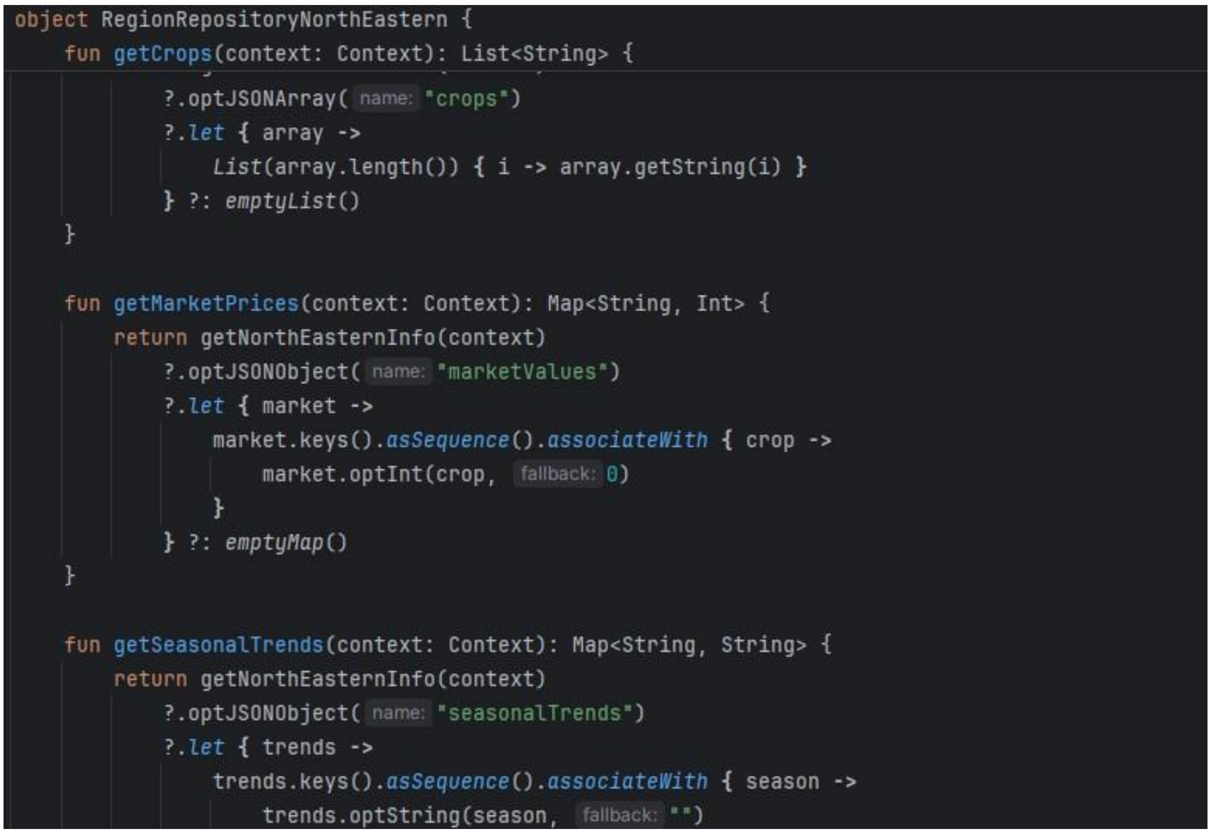
This is where the user after choosing the region the given data information of the area will be displayed.



```kotlin
object RegionRepositoryEastern {
    fun getSeasonalTrends(context: Context): Map<String, String> {
        return getEasternInfo(context)
            ?.optJSONObject( name: "seasonalTrends")
            ?.let { trends ->
                trends.keys().asSequence().associateWith { season ->
                    trends.optString(season,  fallback: "")
                }
            } ?: emptyMap()
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewEasternRegionData() {
    val context = LocalContext.current

    val ph = remember { RegionRepositoryEastern.getPHRange(context) }
    val crops = remember { RegionRepositoryEastern.getCrops(context) }
    val prices = remember { RegionRepositoryEastern.getMarketPrices(context) }
    val trends = remember { RegionRepositoryEastern.getSeasonalTrends(context) }
```

3.3.8 Region repository North eastern region

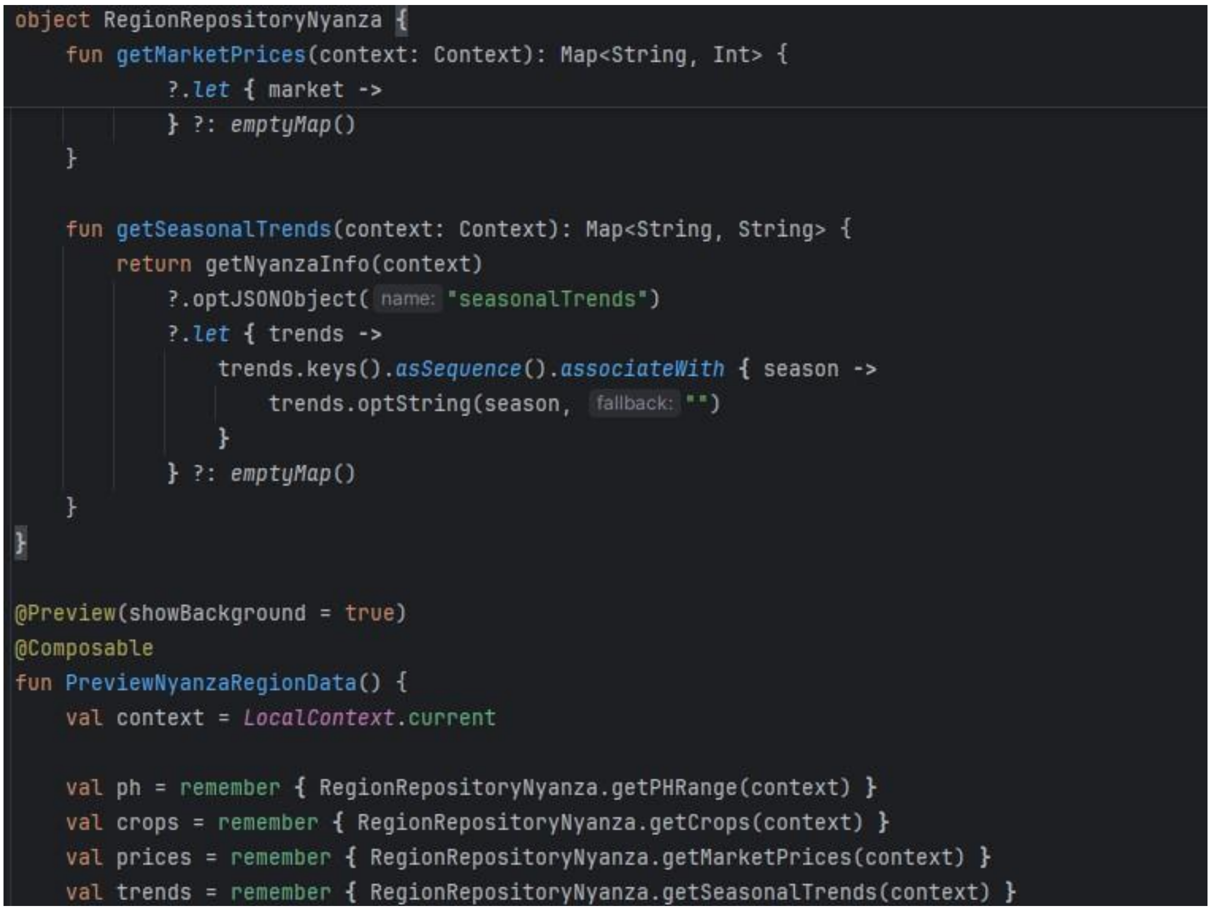
This is where the user after choosing the region the given data information of the area will be displayed.



```
object RegionRepositoryNorthEastern {
    fun getCrops(context: Context): List<String> {
            ?.optJSONArray( name: "crops")
            ?.let { array ->
                List(array.length()) { i -> array.getString(i) }
            } ?: emptyList()
    }

    fun getMarketPrices(context: Context): Map<String, Int> {
        return getNorthEasternInfo(context)
            ?.optJSONObject( name: "marketValues")
            ?.let { market ->
                market.keys().asSequence().associateWith { crop ->
                    market.optInt(crop,  fallback: 0)
                }
            } ?: emptyMap()
    }

    fun getSeasonalTrends(context: Context): Map<String, String> {
        return getNorthEasternInfo(context)
            ?.optJSONObject( name: "seasonalTrends")
            ?.let { trends ->
                trends.keys().asSequence().associateWith { season ->
                    trends.optString(season,  fallback: "")
```

3.3.9 Region repository Nyanza region

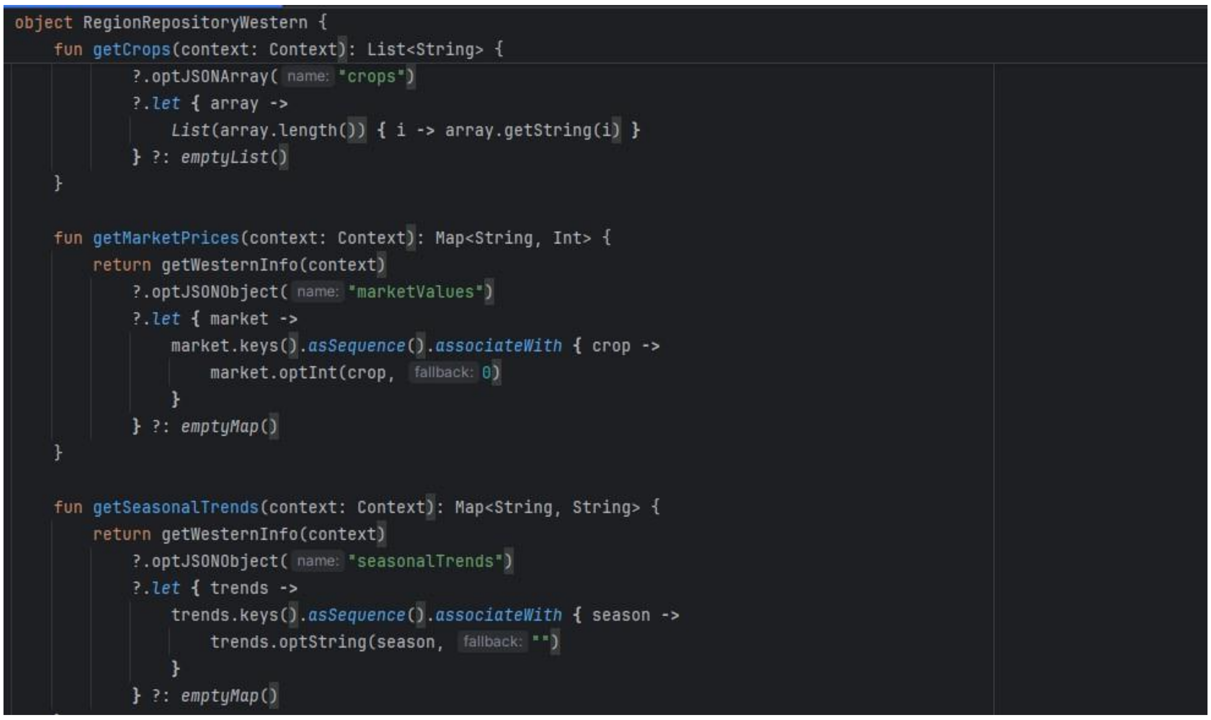
This is where the user after choosing the region the given data information of the area will be displayed.



```kotlin
object RegionRepositoryNyanza {
    fun getMarketPrices(context: Context): Map<String, Int> {
            ?.let { market ->
            } ?: emptyMap()
    }

    fun getSeasonalTrends(context: Context): Map<String, String> {
        return getNyanzaInfo(context)
            ?.optJSONObject( name: "seasonalTrends")
            ?.let { trends ->
                trends.keys().asSequence().associateWith { season ->
                    trends.optString(season,  fallback: "")
                }
            } ?: emptyMap()
    }
}

@Preview(showBackground = true)
@Composable
fun PreviewNyanzaRegionData() {
    val context = LocalContext.current

    val ph = remember { RegionRepositoryNyanza.getPHRange(context) }
    val crops = remember { RegionRepositoryNyanza.getCrops(context) }
    val prices = remember { RegionRepositoryNyanza.getMarketPrices(context) }
    val trends = remember { RegionRepositoryNyanza.getSeasonalTrends(context) }
```

3.3.10 Region repository Western region

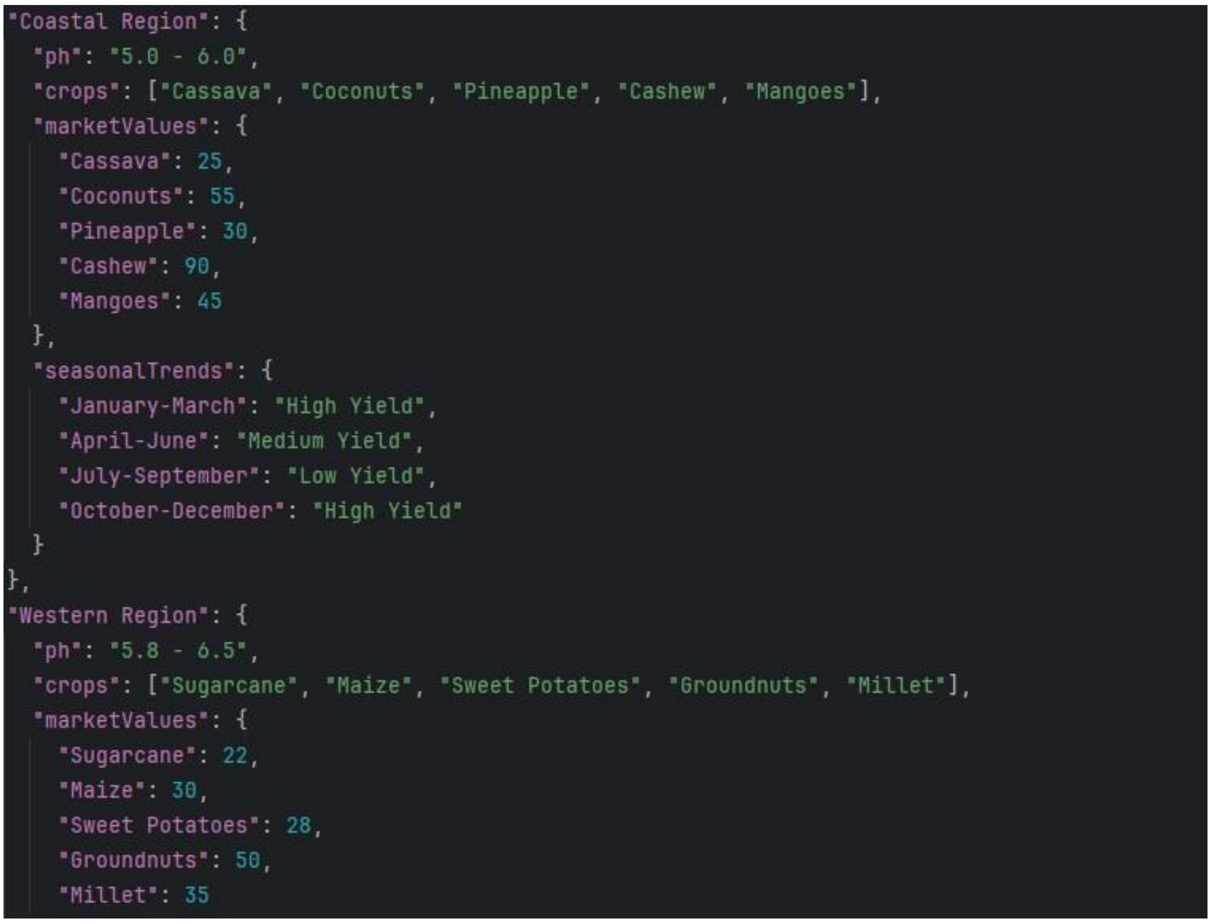
This is where the user after choosing the region the given data information of the area will be displayed.



```kotlin
object RegionRepositoryWestern {
    fun getCrops(context: Context): List<String> {
            ?.optJSONArray( name: "crops")
            ?.let { array ->
                List(array.length()) { i -> array.getString(i) }
            } ?: emptyList()
    }

    fun getMarketPrices(context: Context): Map<String, Int> {
        return getWesternInfo(context)
            ?.optJSONObject( name: "marketValues")
            ?.let { market ->
                market.keys().asSequence().associateWith { crop ->
                    market.optInt(crop,  fallback: 0)
                }
            } ?: emptyMap()
    }

    fun getSeasonalTrends(context: Context): Map<String, String> {
        return getWesternInfo(context)
            ?.optJSONObject( name: "seasonalTrends")
            ?.let { trends ->
                trends.keys().asSequence().associateWith { season ->
                    trends.optString(season,  fallback: "")
                }
            } ?: emptyMap()
```

3.4.data storage

This is where the analysis is stored in to the system.



```
"Coastal Region": {
  "ph": "5.0 - 6.0",
  "crops": ["Cassava", "Coconuts", "Pineapple", "Cashew", "Mangoes"],
  "marketValues": {
    "Cassava": 25,
    "Coconuts": 55,
    "Pineapple": 30,
    "Cashew": 90,
    "Mangoes": 45
  },
  "seasonalTrends": {
    "January-March": "High Yield",
    "April-June": "Medium Yield",
    "July-September": "Low Yield",
    "October-December": "High Yield"
  }
},
"Western Region": {
  "ph": "5.8 - 6.5",
  "crops": ["Sugarcane", "Maize", "Sweet Potatoes", "Groundnuts", "Millet"],
  "marketValues": {
    "Sugarcane": 22,
    "Maize": 30,
    "Sweet Potatoes": 28,
    "Groundnuts": 50,
    "Millet": 35
```

3.5 Testing

To guarantee that each critical feature of the app was correctly in place for the user experience. I followed with a robust source of testing through manual input to scenario, debug logging and validating hashing whether on the reset password screen, I keenly tested edge cases for incomplete fields, minimum password, length, matching fields. I used some samples to through out the project to ensure thing run smoothly. Improvements made were to develop a centralized toast to return for whole implementation(to easily set up for reuse), pull and secure up some of the input logical validation and all better modularized outby making parts and functions.

3.6 Deployment

CHAPTER FOUR: CONCLUSION AND RECOMMENDATION

4.1

Conclusion

In developing this application called analysis of seasonal trends in agricultural yields, were able to go from defining the problem of an analysed seasonal trends all in chapter one to three through laying out the system model and implementing some of the basic logic in application. The app encompasses secure authentication, simple by looking in our UI layouts, geocoding, and real time data analysis. I integrated Jakarta Mails SMTP for emailing to ensure i have a proper backend control of operation through SQLite database using asynchronous function. There were challenges for instance getting the proper data analysis and setting up the functionality of the up. These challenges have helped me build a proper functioning app.

4.2 Recommendation

For futurism for the analysis of seasonal trends i agricultural yields, i would recommend includes all the counties not just the regions, where the famers just input their locations. Which would help in a better functionality of the famers.

REFERENCES

Lobell, D.B., W., & Costa-Roberts, J. (2011)

Climate Trends and global crop production Since 1980

https://www.nature.com/articles/nclimate2108

Grassini, P.; van Oort, P. A.J.; Fischer, T.

Yield Gaps and Stagnation Risks "Global spatially explicit yield gap time trends reveal regions at risk of future crop yield stagnation"

https://www.nature.com/articles/s43016-023-00913-8

Anderson, W. B.; Seager, R.; Baethgen, W.; Cane, M.; You, L

“Global Within-season Yield Anomaly Prediction for Major Crops Derived Using Seasonal Forecast of large-scale climate indices and Regional Temperature and Precipitation.”

https://www.nature.com/articles/s43016-023-00913-8

Intergovernmental Panel on Climate change (IPCC) (2007)

Adaptation and vulnerability.

https://www.ipcc.ch/report/ar4/wg2/

Hatfielf, J L, & Prueger, J. H. (2004)

Impact of Extreme Weather on Plant performance and crop productivity
https://www.researchgate.net/publication/282542578_Temperature_extremes_Effect_on_plant_growth_and_development