Project Report

NEWS NEST

An Android Application for Keeping Up with the Latest Headlines

SMARTINTERNZ EXTERNSHIP

TEAM ID : 590941

TEAM MEMBERS:

PARIMI MONISH 21BCE8397

MOHITH TADI 21BCE7056

ALLU DEVA SAI VISHNU VARDHAN 21BCE7466

MANTHRI ASHISH 21BCE7870

Index

NEWS NEST

An Android Application For Keeping Up With The Latest Headlines

INTRODUCTION

PROJECT OVERVIEW

News Nest is a cutting-edge news application developed using a tech stack centered around Kotlin, Jetpack Compose, XML, SQLite, Retrofit, Room Database, and other advanced tools. This app empowers users with access to the latest headlines and top news stories, while offering the flexibility to read full articles through a pay-per-article model or by subscribing to a plan. With robust user authentication features including login, registration, and account retrieval, News Nest ensures a secure and seamless experience for its users. Leveraging technologies like Work Manager for efficient task scheduling, Coil for optimized image loading, and Navigation Component for seamless app navigation, News Nest provides a user-friendly interface and top-notch performance. The integration of NewsAPI enriches the content offering, while the utilization of Room Database and SQLite ensures robust data management. This amalgamation of cutting-edge technologies sets News Nest apart in the realm of news applications, promising a dynamic and engaging user experience.

PURPOSE

News Nest fulfills a critical need in today's fast-paced world by offering a streamlined and personalized news consumption experience. With a focus on top-notch technology such as Kotlin, Jetpack Compose, and Room Database, the app provides a seamless platform for users to access the latest headlines and in-depth articles. The dual monetization model, comprising pay-per-article and subscription plans, not only ensures sustainable revenue generation but also gives users the flexibility to choose how they consume news ad-free. By incorporating features like account management and password retrieval, News Nest prioritizes user convenience and engagement, solidifying its position as a go-to destination for staying informed.

Furthermore, News Nest's innovative approach sets it apart in the competitive landscape of news applications. Leveraging cutting-edge technologies like Retrofit and Work Manager, the app promises efficiency and responsiveness in delivering timely news updates. With a user-centric design and an eye towards future scalability, the platform has the potential to grow its user base and explore avenues for expansion. Whether it's through targeted marketing strategies or refining the user experience based on feedback, News Nest is poised to be a dynamic and indispensable tool for individuals seeking curated, reliable news content.

LITERATURE SURVEY

EXISTING PROBLEMS

Certainly, before the introduction of News Nest, several existing problems in the news consumption space may have prompted the need for a new approach:

1. Information Overload: With the abundance of news sources and articles available online, users often face the challenge of information overload. Sorting through the vast amount of content to find reliable and relevant news can be time-consuming and overwhelming.

2. Inconvenient Payment Models: Some existing platforms may have rigid payment models that do not align with user preferences. For example, a subscription-only model may deter users who prefer to pay per article.

3. Limited Accessibility: Not all users may have equal access to quality news content due to factors such as geographical restrictions, language barriers, or financial constraints associated with subscription-based models.

4. Security Concerns: Security and privacy issues may arise on certain news platforms, causing users to hesitate when providing personal information.

5. Outdated User Interfaces: Older news applications may have outdated or less user-friendly interfaces, potentially leading to a less engaging user experience.

6. Lack of Account Recovery Options: In case of forgotten passwords or account retrieval needs, some platforms may lack efficient and user-friendly account recovery mechanisms.

7. Technological Stagnation: Some existing platforms may not be leveraging the latest technologies and advancements, potentially resulting in slower load times or less responsive interfaces.

8. Fragmented User Experience: Users might need to switch between multiple apps or websites to get a comprehensive view of news across different categories or sources, leading to a fragmented and less efficient user experience.

REFERENCES

https://www.geeksforgeeks.org/kotlin-android-tutorial/

https://developer.android.com/jetpack?gclid=CjwKCAiA3aeqBhBzEiwAxFiOBqaCfU6t4N7JGoiVEXgKBcS3UFEN6MIQfS3iNYpyjCN9V79P4jt_DhoCQt0QAvD_BwE&gclsrc=aw.ds

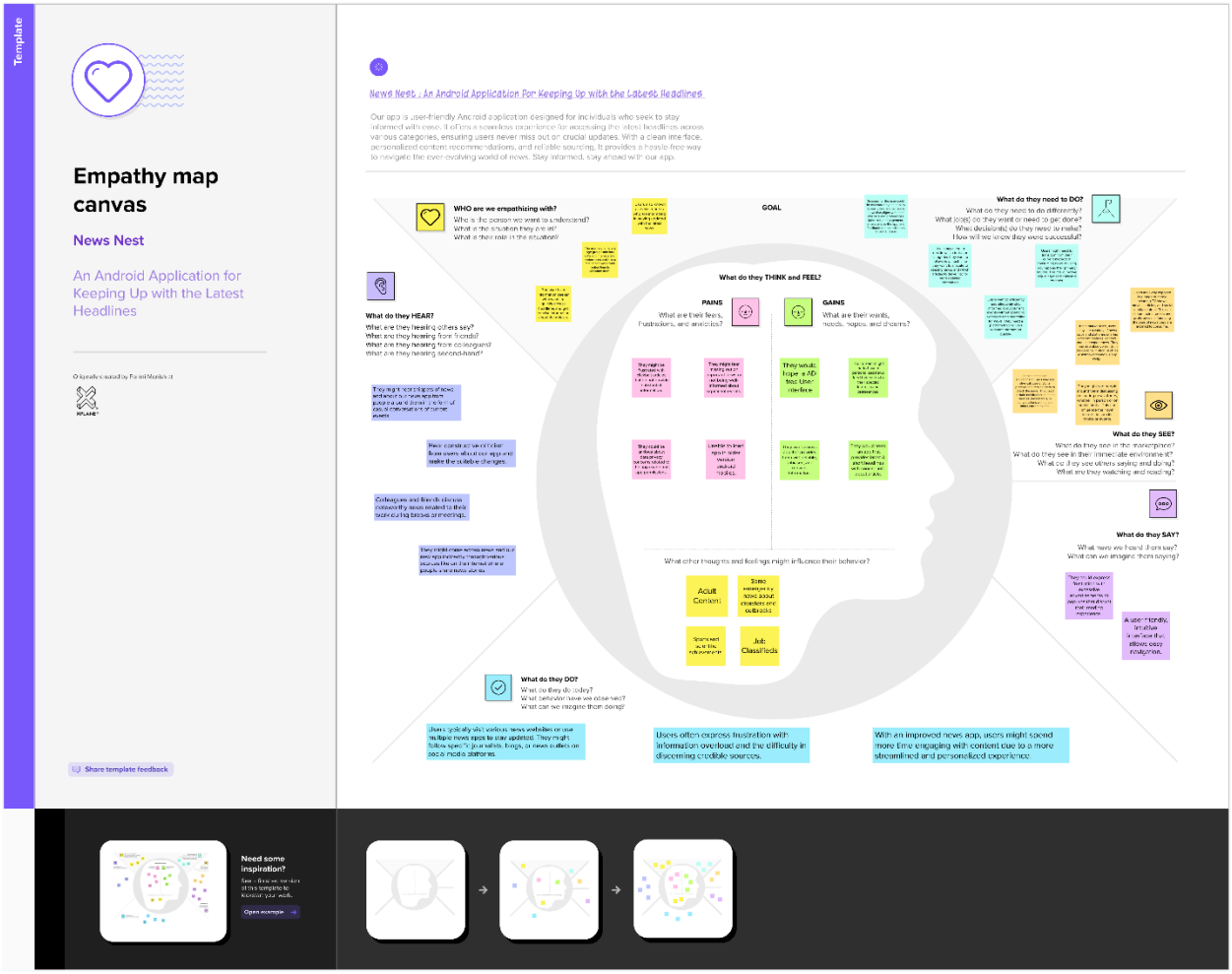
https://developer.android.com/kotlin?gclid=CjwKCAiA3aeqBhBzEiwAxFiOBqSr6Z0Phay1SV_SOp-_JVNtxz1Q_9szO0gY0ar0wB2n8XrBIGvZVxoC4-sQAvD_BwE&gclsrc=aw.ds

PROBLEM STATEMENT DEFINITION

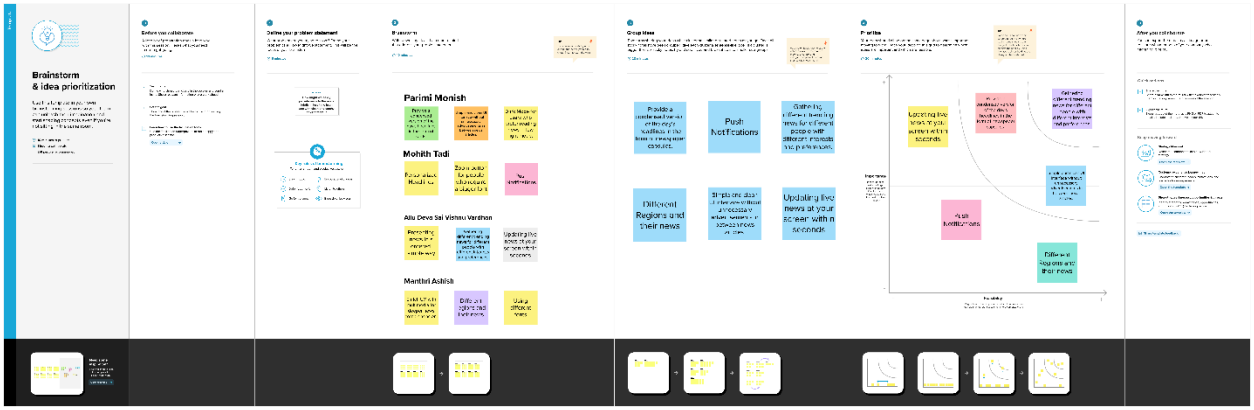
How might we easily provide news to the users ad-free briefly in less time in an app with clean and catchy environment?

IDEATION & PROPOSED SOLUTION

EMPATHY MAP CANVAS



IDEATION & BRAINSTORMING



REQUIREMENT ANALYSIS

FUNCTIONAL REQUIREMENTS

These functional requirements outline the core features and capabilities of the News Nest app, ensuring a comprehensive and user-friendly experience for its users.

Certainly! Here are some of the functional requirements for the News Nest app:

1. User Registration and Authentication:
   - Allow users to create accounts with a unique username and password.
   - Implement email verification or confirmation for new registrations.
   - Provide secure login functionality for registered users.

2. Account Management:
   - Enable users to update their profile information (e.g., name, email, profile picture).
   - Allow users to change their password through a secure process.

3. News Feed:
   - Display a curated list of top and latest news headlines.
   - Organize news articles by categories (e.g., politics, technology, sports).

4. Article Details:
   - Allow users to view the full content of an article.
   - Provide options for sharing articles on social media or via email.

5. Monetization Features:
   - Implement a pay-per-article functionality for users who prefer to make one-time payments.
   - Offer subscription plans for users who want unlimited access to articles within a specified period.

6. Payment Integration:
   - Integrate a secure payment gateway for processing payments.
   - Ensure a smooth and seamless payment experience for users.

7. Account Retrieval and Password Reset:
   - Implement mechanisms for users to retrieve their accounts using either their username or email.
   - Provide a secure process for users to reset their passwords.

8. Push Notifications:
   - Allow users to opt in or out of receiving push notifications for important news updates.
   - Ensure notifications are timely and relevant to user preferences.

NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements define the attributes of the system that are not directly related to specific behaviors or functionalities, but rather pertain to aspects like performance, security, usability, and more.

Here are some non-functional requirements for the News Nest app:

1. Performance:

- The app should load quickly and respond promptly to user interactions, ensuring a smooth and seamless user experience.
- Response times for actions such as loading articles, searching, and making payments should be within acceptable limits.

2. Scalability:

- The system should be able to handle an increasing number of users and a growing volume of content without significant degradation in performance.

3. Reliability:

- The app should be available and functional at all times, with minimal downtime or service disruptions.
- Implement robust error handling and recovery mechanisms to handle unexpected situations.

4. Security:

- User data should be stored securely, with strong encryption protocols for sensitive information like passwords and payment details.
- Implement secure authentication and authorization mechanisms to protect user accounts.

5. Privacy:

- Ensure compliance with data privacy regulations (e.g., GDPR) and provide transparent information about how user data is collected, used, and stored.

6. Compatibility:

- The app should be compatible with a wide range of devices and screen sizes to accommodate different user preferences and needs.
- Ensure cross-platform compatibility, if applicable.

7. Localization:

- Provide support for multiple languages and regions to cater to a diverse user base.

8. Compliance and Legal Requirements:

- Ensure the app complies with relevant industry standards, regulations, and legal requirements.

9. Performance under Load:

- The app should be able to handle peak loads (e.g., during high traffic periods) without significant performance degradation.

10. Resource Utilization:

- Optimize resource utilization, such as memory and CPU usage, to ensure the app runs efficiently on a variety of devices.

11. Documentation and Training:

- Provide comprehensive documentation for developers and end-users, along with any necessary training materials.

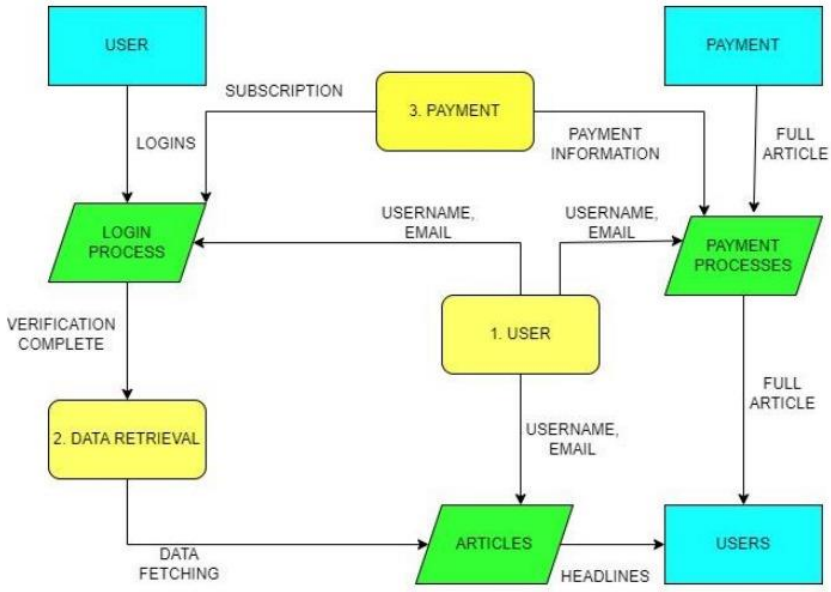
PROJECT DESIGN

DATA FLOW DIAGRAMS & USER STORIES

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored. They are a visual representation of how data moves and is processed within a system and are a powerful tool for system analysis, design, and documentation. DFDs use standardized symbols and notation to depict the flow of data and the processes that act on that data.

DFD Level 0

DFD Level 1

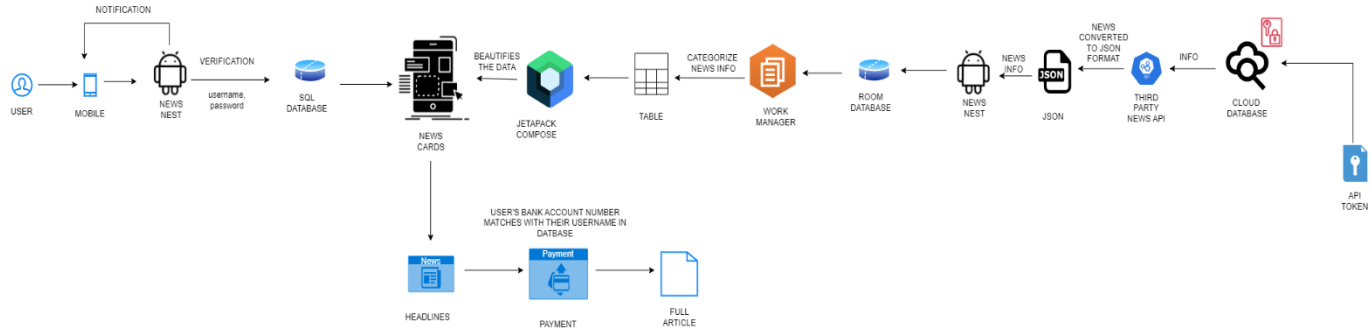


User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my username, email and password and my info will be checked when I am entering information whether I am entering correct type of information in its field.	My info will be saved in app's database once validated successfully.	High	Sprint-1
	Login	USN-2	As a user, I can login to the app by entering my username and password and will be verified with SQL database in backend.	I can start viewing news headlines.	High	Sprint-1
	Account retrieval	USN-3	As a user, I can retrieve my account by entering my username and email if I forgot password.	I can retrieve my account and start viewing headlines once again.	Medium	Sprint-2
	Sign Out	USN-4	As a user, I can sign out whenever I want.	I can sign out and sign in to the app only when I need	Low	Sprint-3
	Payment	USN-5	As a user, I want to read a specific article entirely so I can pay and get it.	I can read that article in a more detailed information.	Medium	Sprint-2
	Payment	USN-6	As a user, I can get a subscription and read all articles entirely for a specific period.	I can read whatever article I want entirely with detailed information.	Medium	Sprint-2
	UI / UX	USN-7	As a user, I can get user friendly interface.	I can get clean, well designed and ad free UI.	High	Sprint-1
Customer (Web user)	Anything	USN-8	As a web user, I can use the app just as the mobile user only if I have an android emulator.	I can do all functions as a mobile user if I have an emulator.	Low	Sprint-3
Customer Care Executive	User support and assistance	USN-9	As a CCE, I can help the users regarding their issues submitted through feedback.	The CCE provides clear and detailed response or help to address their issues / needs.	Medium	Sprint-3
Administrator	Data analytics and monetization	USN-10	As an administrator, I can analyse people interests and priorities and monetize my app accordingly.	The administrator has access to analytics that provide user activity and their interests. Administrator would monetize the app accordingly.	Medium	Sprint-3

SOLUTION ARCHITECTURE

The solution architecture for our NEWS NEST app, utilizing modern technologies such as Jetpack Compose and various APIs, encompasses user onboarding, data collection, personalization, database management, content curation, push notifications, user interface, monetization, security, analytics, scalability, testing, deployment, and maintenance. Users register and set preferences through an intuitive Jetpack Compose user interface, with data collected from various sources via NEWSAPI Token from cloud database. A powerful personalization engine, integrated with APIs, generates tailored recommendations, and the database stores user profiles, articles, and interactions. Content curation, supported by APIs, ensures relevance and ad removal. The user interface, built with Jetpack Compose, elegantly displays content and trending topics. Monetization strategies, including in app purchases and premium subscriptions, are optimized with API integration, Analytics, utilizing APIs, track user engagement, and scalability strategies ensure your app can grow dynamically with API enhancements.



Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

• Find the best tech solution to solve existing business problems.

• Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.

• Define features, development phases, and solution requirements.

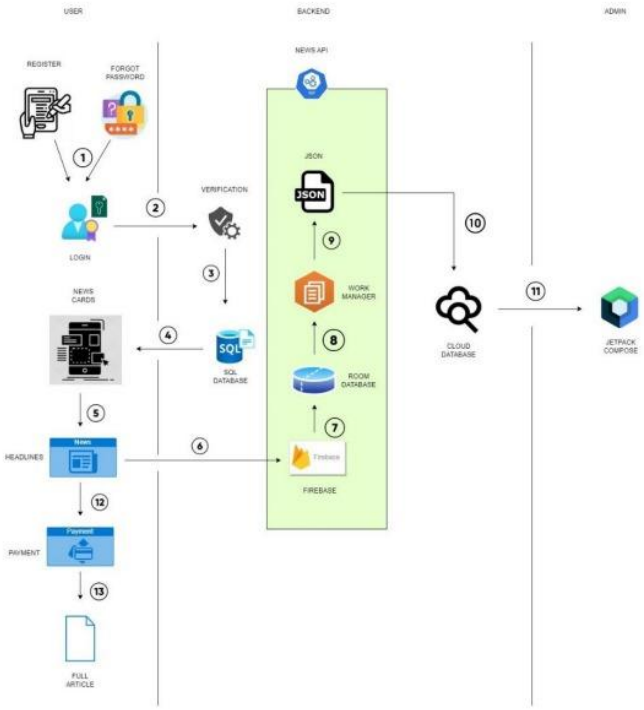
• Provide specifications according to which the solution is defined, managed, and delivered.

PROJECT PLANNING & SCHEDULING

TECHNICAL ARCHITECTURE

Technical architecture encompasses the fundamental structure and design of a software system, outlining its key components, their interactions, and the underlying framework that enables its functionality. This includes hardware infrastructure, software modules, databases, networking protocols, and interfaces. It also addresses critical considerations such as data storage, scalability, security measures, and compliance with industry standards. The architecture dictates how the system handles increasing loads and ensures optimal performance. It encompasses deployment strategies, whether on-premises or on cloud platforms, as well as procedures for error handling, recovery, and backups. Integration with external services and APIs is also specified, enabling seamless interaction with third-party applications. Furthermore, it delineates development, testing, staging, and production environments, ensuring consistency across different stages of the software development life cycle. Monitoring and logging mechanisms are put in place to track system behaviour, performance metrics, and errors. Maintenance procedures and strategies for upgrades are defined to keep the system up-to-date and efficient. Documentation is crucial, providing insights into architectural decisions and best practices, facilitating knowledge transfer among team members and aiding in troubleshooting. In essence, technical architecture serves as the foundational blueprint for the construction and maintenance of a robust and effective software system.

The Deliverable shall include the architectural diagram as below and the information as per the Table1 & Table 2.



SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration (Mobile User)	USN-1	As a user, I can register for the application by entering my username, email and password and my info will be checked when I am entering information whether I am entering correct type of information in its field.	2	High	Parimi Monish, Mohith
Sprint-1	Login (Mobile User)	USN-2	As a user, I can login to the app by entering my username and password and will be verified with SQL database in backend.	3	High	Mohith, Parimi Monish
Sprint-2	Account Retrieval (Mobile User)	USN-3	As a user, I can retrieve my account by entering my username and email if I forgot password.	2	Medium	Ashish
Sprint-3	Sign Out (Mobile User)	USN-4	As a user, I can sign out whenever I want.	1	Low	Vishnu
Sprint-2	Payment (Mobile User)	USN-5	As a user, I want to read a specific article entirely so I can pay and get it.	2	Medium	Parimi Monish
Sprint-2	Payment (Mobile User)	USN-6	As a user, I can get a subscription and read all articles entirely for a specific period.	2	Medium	Mohith
Sprint-1	UI / UX (Mobile User)	USN-7	As a user, I can get user friendly interface.	3	High	Parimi Monish
Sprint-3	Anything (Web User)	USN-8	As a web user, I can use the app just as the mobile user only if I have an android emulator.	1	Low	Vishnu
Sprint-3	User support and assistance (CCE)	USN-9	As a CCE, I can help the users regarding their issues submitted through feedback.	2	Medium	Vishnu, Parimi Monish, Mohith, Ashish
Sprint-3	Data analytics and monetization (Administrator)	USN-10	As an administrator, I can analyse people's interest and priorities and monetize my app accordingly.	2	Medium	Ashish, Vishnu, Mohith, Parimi Monish

SPRINT DELIVERY SCHEDULE

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	8	4 Days	28 Oct 2023	31 Oct 2023	8	31 Oct 2023
Sprint-2	6	2 Days	01 Nov 2023	02 Nov 2023	6	02 Nov 2023
Sprint-3	6	2 Days	03 Nov 2022	04 Nov 2022	6	04 Nov 2022

Velocity:

Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{sprint\ duration}{velocity}$$

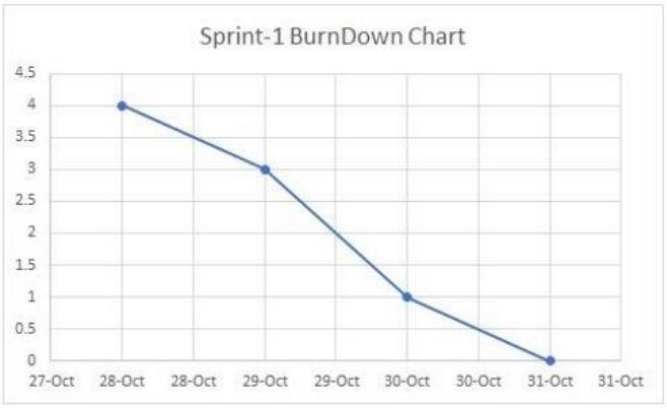
$$\text{Sprint 1 AV} = \frac{8}{4} = 2$$

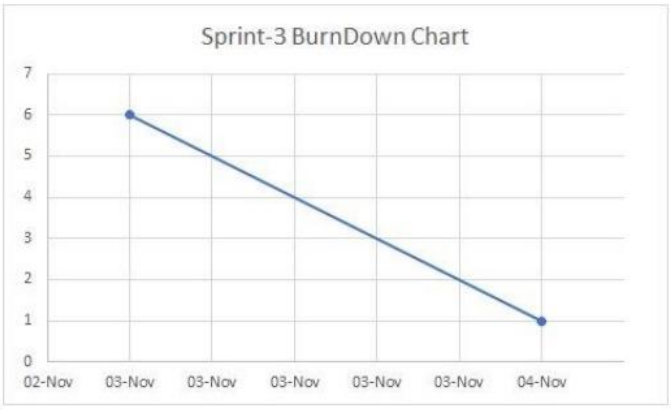
$$\text{Sprint 2 AV} = \frac{6}{2} = 3$$

$$\text{Sprint 3 AV} = \frac{6}{2} = 3$$

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.





CODING & SOLUTIONING

LOGIN ACTIVITY

LoginActivity Class:

This is an Android activity class that extends ComponentActivity. The LoginActivity is responsible for handling the login process. It initializes and manages shared preferences to check if the user is already logged in. If the user is already logged in, it immediately starts the MainActivity and finishes the login activity. If not logged in, it sets up the UI for user login.

LoginScreen Composable Function:

This is a Jetpack Compose composable function used to create the user interface for the login screen. It takes two parameters: a Context object and a UserDatabaseHelper object. It defines the UI elements and logic for the login screen, including the layout, images, text fields, buttons, and error handling.

UI Elements in LoginScreen:

The LoginScreen composable function sets up a user interface using Compose, including an image, login title, input fields for username and password, error message display, and login and registration buttons. It uses OutlinedTextField for username and password input, with icons for better user experience. A custom UserDatabaseHelper class is used to interact with the local database to validate user credentials. It handles user input, performs validation, and displays error messages using a Toast if needed.

Transition to MainActivity:

When the user successfully logs in, the code saves this information in shared preferences and starts the MainActivity while finishing the LoginActivity.

Registration and Account Retrieval Buttons:

The login screen also provides options for users to register a new account or retrieve a forgotten password through the "REGISTER NOW" and "RETRIEVE ACCOUNT" text buttons. Clicking these buttons launches other activities (RegistrationActivity and AccountRetrievalActivity).

Code:

```
@file:OptIn(ExperimentalMaterial3Api::class)

package com.example.newsnest

import android.app.Activity
import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Lock
import androidx.compose.material.icons.filled.Person
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Divider
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
```

```
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat


class LoginActivity : ComponentActivity() {
    private lateinit var sharedPreferences: SharedPreferences

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        sharedPreferences = getSharedPreferences("login_prefs",
Context.MODE_PRIVATE)
        val isLoggedIn = sharedPreferences.getBoolean("isLoggedIn", false)

        if (isLoggedIn) {
            startMainPage(this)
            finish()
            return
        }
        var databaseHelper = UserDatabaseHelper(this)
        setContent {
            LoginScreen(this, databaseHelper)
        }
    }

    private fun startMainPage(context: Context) {
        val intent = Intent(context, MainActivity::class.java)
        ContextCompat.startActivity(context, intent, null)
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun LoginScreen(context: Context, databaseHelper: UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    val sharedPreferences = context.getSharedPreferences("login_prefs",
Context.MODE_PRIVATE)

    Column(
        Modifier
            .fillMaxHeight()
            .fillMaxWidth()
            .background(Color.White),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center,
    )
    {
        Image(
```

```
            painter = painterResource(id = R.drawable.newsnest_login),
            contentScale = ContentScale.Crop,
            contentDescription = null,
            modifier = Modifier
                .weight(1f)
        )
        Spacer(modifier = Modifier.height(10.dp))
        Row {
            Divider(
                color = Color(0xFFFFA500),
                thickness = 2.dp,
                modifier = Modifier
                    .width(110.dp)
                    .padding(top = 20.dp, end = 20.dp)
            )
            Text(
                text = "LOGIN",
                color = Color(0xFF3064FC),
                fontWeight = FontWeight.Bold,
                fontSize = 24.sp,style =
MaterialTheme.typography.headlineMedium
            )
            Divider(
                color = Color(0xFFFFA500),
                thickness = 2.dp,
                modifier = Modifier
                    .width(110.dp)
                    .padding(top = 20.dp, start = 20.dp)
            )
        }
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = username,
            onValueChange ={username=it},
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Person,
                    contentDescription = "USERNAME",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER USERNAME", color =
Color.DarkGray) },
            label = { Text(text = "USERNAME", color = Color.Black)},
            textStyle = TextStyle(color = Color.Black)
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = password,
            onValueChange ={password=it},
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Lock,
                    contentDescription = "PASSWORD",
                    tint = Color(0xFFFFA500)
                )
            },
```

```
            placeholder = { Text(text = "ENTER PASSWORD", color =
Color.DarkGray) },
            label = { Text(text = "PASSWORD", color = Color.Black)},
            textStyle = TextStyle(color = Color.Black),
            visualTransformation = PasswordVisualTransformation(),
            //colors = TextFieldDefaults.textFieldColors(Color.Black)
        )
        Spacer(modifier = Modifier.height(8.dp))
        if (error.isNotEmpty())
        {
            Toast.makeText(context,error,Toast.LENGTH_SHORT).show()
        }
        Button(
            onClick = {
                if (username.isNotEmpty() && password.isNotEmpty())
                {
                    val user = databaseHelper.getUserByUsername(username)
                    if (user != null && user.password == password)
                    {
                        error = "Successfully login"
                        sharedPreferences.edit().putBoolean("isLoggedIn",
true).apply()

context.startActivity(Intent(context,MainActivity::class.java))
                        (context as? Activity)?.finish()
                    }
                    else
                    {
                        error = "Invalid username OR password"
                    }
                }
                else
                {
                    error = "Please fill all fields"
                }
            },
            shape = RoundedCornerShape(20.dp),
            colors = ButtonDefaults.buttonColors(Color(0xFFFFA500)),
            modifier = Modifier
                .width(200.dp)
        )
        {
            Text(text = "LOGIN", fontWeight = FontWeight.Bold, color =
Color.White)
        }
        Spacer(modifier = Modifier.height(4.dp))
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(0.dp),
            horizontalArrangement = Arrangement.Center,
            verticalAlignment = Alignment.CenterVertically
        )
        {
            Text(
                text = "JOIN THE NEWS COMMUNITY",
                color = Color.Black,
```

```
                fontSize = 12.sp
            )
            TextButton( onClick =
{context.startActivity(Intent(context,RegistrationActivity::class.java))} )
            {
                Text(
                text = "REGISTER NOW",
                color = Color(0xFF3064FC),
                fontSize = 12.sp
                )
            }
        }
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(0.dp),
            horizontalArrangement = Arrangement.Center,
            verticalAlignment = Alignment.CenterVertically
        )
        {
            Text(
                text = "FORGOT YOUR PASSWORD ?",
                color = Color.Black,
                fontSize = 12.sp
            )
            TextButton( onClick =
{context.startActivity(Intent(context,AccountRetrievalActivity::class.java))}
)
            {
                Text(
                    text = "RETRIEVE ACCOUNT",
                    color = Color(0xFF3064FC),
                    fontSize = 12.sp
                )
            }
        }
    }
}
```

REGISTRATION ACTIVITY

RegistrationActivity Class:
This is an Android activity class that extends ComponentActivity. The RegistrationActivity is responsible for handling the user registration process. It initializes and uses a UserDatabaseHelper to interact with a local database for user registration. RegistrationScreen Composable Function:
This is a Jetpack Compose composable function used to create the user interface for the registration screen.
It takes two parameters: a Context object and a UserDatabaseHelper object.
It defines the UI elements and logic for the registration screen, including the layout, image, input fields for username, password, and email, error handling, and registration button.

UI Elements in RegistrationScreen:
The RegistrationScreen composable function sets up a user-friendly user registration interface using Jetpack Compose. It uses OutlinedTextField for the user to input their username, password, and email, with icons for improved user experience. It provides validation for password complexity. The password should contain at least 5 alphabetic

characters and at least 1 digit. It displays an error message using a Toast if necessary. Upon successful registration, it inserts the user's information into the local database and navigates the user to the login screen using an Intent.

Transition to LoginActivity:
When the user successfully registers, the code navigates to the LoginActivity to allow the user to log in with the newly created credentials.
"HAVE AN ACCOUNT?":
The registration screen also provides an option for users who already have an account to navigate to the login screen. This is achieved through the "LOGIN" text button.
Additional Details:
The code uses Jetpack Compose for creating a visually appealing and responsive user interface.
It uses custom fonts and styling to make the registration screen attractive. The isPasswordValid function checks if the provided password meets the required complexity criteria.

Code:

```
package com.example.newsnest

import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Email
import androidx.compose.material.icons.filled.Lock
import androidx.compose.material.icons.filled.Person
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Divider
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.SnackbarHostState
import androidx.compose.material3.Text
import androidx.compose.material3.TextButton
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.Font
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
```

```
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.content.ContextCompat

class RegistrationActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            RegistrationScreen(this,databaseHelper)
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun RegistrationScreen(context: Context, databaseHelper: UserDatabaseHelper)
{
    var username by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }
    val snackbarHostState = remember { SnackbarHostState() }

    Column(
        modifier = Modifier
            .fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    )
    {
        Spacer(modifier = Modifier.height(4.dp))
        Card(
            modifier = Modifier
                .weight(1f)
                .padding(20.dp),
            shape = RoundedCornerShape(18.dp),
            elevation = CardDefaults.cardElevation(4.dp),
            colors = CardDefaults.cardColors(Color(0xFF080404))
        )
        {
            Image(
                painter = painterResource(id = R.drawable.newsnest_register),
                contentDescription = null,
                contentScale = ContentScale.Fit,
                modifier = Modifier
                    .fillMaxSize()
                    .weight(1f)
                    .clip(RoundedCornerShape(18.dp))
            )
        }
        Text(
            text = "REGISTER NOW",
            color = Color(0xFF3064FC),
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,
```

```
            style = MaterialTheme.typography.headlineLarge,
            fontFamily = FontFamily(Font(R.font.arialbold)),
            modifier = Modifier.padding(bottom = 5.dp)
        )
        Divider(
            color = Color(0xFFFFA500),
            thickness = 2.dp,
            modifier = Modifier
                .width(250.dp)
        )
        Spacer(modifier = Modifier.padding(4.dp))
        OutlinedTextField(
            value = username,
            onValueChange ={ username = it },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Person,
                    contentDescription = "USERNAME",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER USERNAME", color =
Color.DarkGray) },
            label = { Text(text = "USERNAME", color = Color.Black)},
            textStyle = TextStyle(color = Color.Black)
        )
        Spacer(modifier = Modifier.height(4.dp))
        OutlinedTextField(
            value = password,
            onValueChange ={ password = it },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Lock,
                    contentDescription = "PASSWORD",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER PASSWORD", color =
Color.DarkGray) },
            label = { Text(text = "PASSWORD", color = Color.Black)},
            textStyle = TextStyle(color = Color.Black),
            visualTransformation = PasswordVisualTransformation()
        )
        Spacer(modifier = Modifier.height(4.dp))
        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Email,
                    contentDescription = "E-MAIL",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER E-MAIL", color =
Color.DarkGray) },
            label = { Text(text = "E-MAIL", color = Color.Black)},
```

```
            textStyle = TextStyle(color = Color.Black)
        )
        Spacer(modifier = Modifier.height(4.dp))
        if (error.isNotEmpty())
        {
            Toast.makeText(context, error, Toast.LENGTH_SHORT).show()
        }
        Button(
            onClick =
            {
                if (username.isNotEmpty() && password.isNotEmpty() &&
email.isNotEmpty())
                {
                    if (isPasswordValid(password))
                    {
                        val user = User(
                            id = null,
                            firstName = username,
                            lastName = null,
                            email = email,
                            password = password
                        )
                        databaseHelper.insertUser(user)
                        error = "User registered successfully"

context.startActivity(Intent(context,LoginActivity::class.java))
                    }
                    else
                    {
                        error = "Password should contain at least 5 alphabets
and 1 number"
                    }
                }
                else
                {
                    error = "Please fill all fields"
                }
            },
            shape = RoundedCornerShape(20.dp),
            colors = ButtonDefaults.buttonColors(Color(0xFFFFA500)),
            modifier = Modifier
                .width(200.dp)
                .padding(top = 16.dp)
        )
        {
            Text(text = "REGISTER", fontFamily =
FontFamily(Font(R.font.arialbold)), fontSize = 20.sp )
        }
        Row(
            modifier = Modifier,
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.Center
        )
        {
            Text(text = "HAVE AN ACCOUNT ?", fontSize = 16.sp, color =
Color.Black)
            TextButton( onClick =
```

```
{context.startActivity(Intent(context,LoginActivity::class.java))} )
            {
                Text(
                    text = "LOGIN",
                    fontSize = 16.sp,
                    color = Color(0xFF3064FC)
                )
            }
        }
    }
}

private fun startLoginActivity(context: Context) {
    val intent = Intent(context, LoginActivity::class.java)
    ContextCompat.startActivity(context, intent, null)
}

fun isPasswordValid(password: String): Boolean
{
    val alphabetCount = password.count { it.isLetter() }
    val digitCount = password.count { it.isDigit() }
    return alphabetCount >= 5 && digitCount >= 1
}
```

ACCOUNT RETREIVAL ACTIVITY

AccountRetrievalActivity Class:
This is an Android activity class that extends ComponentActivity. The AccountRetrievalActivity is responsible for handling the account retrieval process. It initializes and uses a UserDatabaseHelper to interact with a local database to retrieve user account information.

AccountRetrievalScreen Composable Function:
This is a Jetpack Compose composable function used to create the user interface for the account retrieval screen.
It takes two parameters: a Context object and a UserDatabaseHelper object.It defines the UI elements and logic for the account retrieval screen, including the layout, input fields for username and email, error handling, and the "Retrieve Account" button.

UI Elements in AccountRetrievalScreen:
The AccountRetrievalScreen composable function sets up a user-friendly interface for retrieving user account information using Jetpack Compose. It uses OutlinedTextField for the user to input their username and email, with icons for improved user experience. It provides error handling and displays an error message using a Toast if necessary. Upon successful retrieval, it checks if the provided username and email match an existing user in the local database and displays the user's password if found.

"STILL UNABLE TO RECOVER?":
The account retrieval screen provides options for users who are still unable to recover their accounts. It allows them to reset their password with their email or username. Clicking on these text buttons launches other activities, such as ResetPwdByEmailActivity and ResetPwdByUNActivity.

Code:

```
package com.example.newsnest

import android.content.Context
import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Email
import androidx.compose.material.icons.filled.Person
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
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.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class AccountRetrievalActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            AccountRetrievalScreen(this, databaseHelper)
        }
    }
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
```

```
fun AccountRetrievalScreen(context: Context, databaseHelper:
UserDatabaseHelper) {
    var username by remember { mutableStateOf("") }
    var email by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        Modifier
            .fillMaxHeight()
            .fillMaxWidth()
            .background(Color.White),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    )
    {
        Text(
            text = "ACCOUNT RETRIEVAL",
            color = Color(0xFF3064FC),
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,
            modifier = Modifier.padding(bottom = 24.dp),
            style = MaterialTheme.typography.headlineMedium
        )
        OutlinedTextField(
            value = username,
            onValueChange = { username = it },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Person,
                    contentDescription = "USERNAME",
                    tint = Color(0xFFFFA500)
                )
            },
            label = { Text(text = "USERNAME", color = Color.Black) },
            placeholder = { Text(text = "ENTER USERNAME", color =
Color.DarkGray) },
            textStyle = TextStyle(color = Color.Black)
        )
        Spacer(modifier = Modifier.height(12.dp))
        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Email,
                    contentDescription = "E-MAIL",
                    tint = Color(0xFFFFA500)
                )
            },
            label = { Text(text = "E-MAIL", color = Color.Black) },
            placeholder = { Text(text = "ENTER E-MAIL", color =
Color.DarkGray) },
            textStyle = TextStyle(color = Color.Black)
        )
        Spacer(modifier = Modifier.height(12.dp))

        if (error.isNotEmpty()) {
```

```
            Toast.makeText(context, error, Toast.LENGTH_SHORT).show()
        }

        Button(
            onClick = {
                if (username.isNotEmpty() && email.isNotEmpty()) {
                    val user = databaseHelper.getUserByUsername(username)
                    if (user != null && user.email == email) {
                        error = "Account found! Password: ${user.password}"
                    } else {
                        error = "Invalid username or email"
                    }
                } else {
                    error = "Please fill all fields"
                }
            },
            shape = RoundedCornerShape(20.dp),
            colors = ButtonDefaults.buttonColors(Color(0xFFFFA500)),
            modifier = Modifier
                .width(200.dp)
                .padding(top = 16.dp)
        )
        {
            Text(
                text = "RETRIEVE ACCOUNT",
                fontWeight = FontWeight.Bold,
                color = Color.White
            )
        }
        Column(
            modifier = Modifier,
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        )
        {
            Spacer(modifier = Modifier.height(4.dp))
            Text(text = "STILL UNABLE TO RECOVER ?", fontSize = 14.sp,
fontWeight = FontWeight.Bold, color = Color.Black)
            TextButton( onClick =
{context.startActivity(Intent(context,ResetPwdByEmailActivity::class.java))}
)
            {
                Text(
                    text = "RESET PASSWORD WITH E-MAIL",
                    fontSize = 14.sp,
                    color = Color(0xFF3064FC)
                )
            }
            TextButton( onClick =
{context.startActivity(Intent(context,ResetPwdByUNActivity::class.java))} )
            {
                Text(
                    text = "RESET PASSWORD WITH USERNAME",
                    fontSize = 14.sp,
                    color = Color(0xFF3064FC)
                )
            }
```

```
        }
    }
}
```

RESET PASSWORD BY EMAIL ACTIVITY

ResetPwdByEmailActivity Class:
This is an Android activity class that extends ComponentActivity. The ResetPwdByEmailActivity is responsible for handling the password reset process using an email. It initializes and uses a UserDatabaseHelper to interact with a local database to reset the user's password.

ResetPasswordByEmailScreen Composable Function:
This is a Jetpack Compose composable function used to create the user interface for the password reset screen using email.It takes two parameters: a Context object and a UserDatabaseHelper object. It defines the UI elements and logic for the password reset screen, including the layout, input fields for email, new password, and confirming the new password, error handling, and the "Reset Password" button.

UI Elements in ResetPasswordByEmailScreen:
The ResetPasswordByEmailScreen composable function sets up a user-friendly interface for resetting the user's password using Jetpack Compose. It uses OutlinedTextField for the user to input their email, a new password, and confirm the new password, with icons for improved user experience. It provides error handling and displays an error message if necessary. Upon successful password reset, it checks if the email exists in the local database and updates the password. It also ensures that the new password and confirmation match.

"RESET PASSWORD" Button:
When the user clicks the "RESET PASSWORD" button, the code validates the input fields and attempts to update the password in the database. It provides appropriate error messages if the update is successful or if there are issues with the input data.

Code:

```
package com.example.newsnest

import android.content.Context
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Email
import androidx.compose.material.icons.filled.Lock
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
```

```
import androidx.compose.material3.Text
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.platform.LocalContext
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class ResetPwdByEmailActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            ResetPasswordByEmailScreen(
                context = this,
                databaseHelper = databaseHelper
            )
        }
    }
}
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ResetPasswordByEmailScreen(
    context: Context,
    databaseHelper: UserDatabaseHelper
)
{
    var email by remember { mutableStateOf("") }
    var newPassword by remember { mutableStateOf("") }
    var confirmNewPassword by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        Modifier
            .fillMaxHeight()
            .fillMaxWidth(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    )
    {
        Text(
            text = "RESET PASSWORD",
            color = Color(0xFF3064FC),
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,style = MaterialTheme.typography.headlineMedium
        )
        Text(
```

```
            text = "WITH E-MAIL",
            color = Color(0xFF3064FC),
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,style = MaterialTheme.typography.headlineMedium
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Email,
                    contentDescription = "E-MAIL",
                    tint = Color(0xFFFFA500)
                )
            },
            label = { Text(text = "E-MAIL", color = Color.Black) },
            placeholder = { Text(text = "ENTER E-MAIL", color =
Color.DarkGray) },
            textStyle = TextStyle(color = Color.Black)
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = newPassword,
            onValueChange ={ newPassword = it},
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Lock,
                    contentDescription = "NEW PASSWORD",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER NEW PASSWORD", color =
Color.DarkGray) },
            label = { Text(text = "NEW PASSWORD", color = Color.Black)},
            textStyle = TextStyle(color = Color.Black),
            visualTransformation = PasswordVisualTransformation()
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = confirmNewPassword,
            onValueChange ={ confirmNewPassword = it},
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Lock,
                    contentDescription = "CONFIRM NEW PASSWORD",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER SAME PASSWORD", color =
Color.DarkGray) },
            label = { Text(text = "CONFIRM NEW PASSWORD", color =
Color.Black)},
            textStyle = TextStyle(color = Color.Black),
            visualTransformation = PasswordVisualTransformation()
        )
        Spacer(modifier = Modifier.height(16.dp))
```

```
        if (error.isNotEmpty()) {
            Text(
                text = error,
                color = MaterialTheme.colorScheme.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }

        Button(
            onClick = {
                if (email.isNotEmpty() && newPassword.isNotEmpty() &&
confirmNewPassword.isNotEmpty()) {
                    if (newPassword == confirmNewPassword) {
                        // Update password in the database
                        val updated =
databaseHelper.updatePasswordByEmail(email, newPassword)
                        if (updated != null) {
                            error = "Password updated successfully"
                        } else {
                            error = "Failed to update password"
                        }
                    } else {
                        error = "Passwords do not match"
                    }
                } else {
                    error = "Please fill all fields"
                }
            },
            shape = RoundedCornerShape(20.dp),
            colors = ButtonDefaults.buttonColors(Color(0xFF3064FC)),
            modifier = Modifier
                .width(200.dp)
                .padding(top = 16.dp)
        )
        {
            Text(text = "RESET PASSWORD", fontWeight = FontWeight.Bold)
        }
    }


}

@Preview(
    showSystemUi = true,
    showBackground = true
)
@Composable
fun ResetPasswordByEmailPreview(){
    ResetPasswordByEmailScreen(
        context = LocalContext.current, // Use LocalContext.current to get
the context
        databaseHelper = UserDatabaseHelper(LocalContext.current) // Use
LocalContext.current to get the context
    )
}
```

RESET PASSWORD BY USERNAME ACTIVITY

ResetPwdByUNActivity Class:
This is an Android activity class that extends ComponentActivity. The ResetPwdByUNActivity is responsible for handling the password reset process using a username. It initializes and uses a UserDatabaseHelper to interact with a local database to reset the user's password.

ResetPasswordByUsernameScreen Composable Function:
This is a Jetpack Compose composable function used to create the user interface for the password reset screen using a username. It takes two parameters: a Context object and a UserDatabaseHelper object. It defines the UI elements and logic for the password reset screen, including the layout, input fields for the username, new password, and confirming the new password, error handling, and the "Reset Password" button.

UI Elements in ResetPasswordByUsernameScreen:
The ResetPasswordByUsernameScreen composable function sets up a user-friendly interface for resetting the user's password using Jetpack Compose. It uses OutlinedTextField for the user to input their username, a new password, and confirm the new password, with icons for improved user experience. It provides error handling and displays an error message if necessary. Upon successful password reset, it checks if the username exists in the local database and updates the password. It also ensures that the new password and confirmation match.

"RESET PASSWORD" Button:
When the user clicks the "RESET PASSWORD" button, the code validates the input fields and attempts to update the password in the database. It provides appropriate error messages if the update is successful or if there are issues with the input data.
Code:

```
package com.example.newsnest

import android.content.Context
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxHeight
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Lock
import androidx.compose.material.icons.filled.Person
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
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.platform.LocalContext
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp

class ResetPwdByUNActivity : ComponentActivity() {
    private lateinit var databaseHelper: UserDatabaseHelper
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        databaseHelper = UserDatabaseHelper(this)
        setContent {
            ResetPasswordByUsernameScreen(
                context = this,
                databaseHelper = databaseHelper
            )
        }
    }
}
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ResetPasswordByUsernameScreen(
    context: Context,
    databaseHelper: UserDatabaseHelper
)
{
    var username by remember { mutableStateOf("") }
    var newPassword by remember { mutableStateOf("") }
    var confirmNewPassword by remember { mutableStateOf("") }
    var error by remember { mutableStateOf("") }

    Column(
        Modifier
            .fillMaxHeight()
            .fillMaxWidth(),
        horizontalAlignment = Alignment.CenterHorizontally,
        verticalArrangement = Arrangement.Center
    )
    {
        Text(
            text = "RESET PASSWORD",
            color = Color(0xFF3064FC),
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,
            style = MaterialTheme.typography.headlineMedium
        )
        Text(
            text = "WITH USERNAME",
            color = Color(0xFF3064FC),
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp,
```

```
            style = MaterialTheme.typography.headlineMedium
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = username,
            onValueChange = { username = it },
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Person,
                    contentDescription = "USERNAME",
                    tint = Color(0xFFFFA500)
                )
            },
            label = { Text(text = "USERNAME", color = Color.Black) },
            placeholder = { Text(text = "ENTER USERNAME", color =
Color.DarkGray) },
            textStyle = TextStyle(color = Color.Black)
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = newPassword,
            onValueChange ={ newPassword = it},
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Lock,
                    contentDescription = "NEW PASSWORD",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER NEW PASSWORD", color =
Color.DarkGray) },
            label = { Text(text = "NEW PASSWORD", color = Color.Black)},
            textStyle = TextStyle(color = Color.Black),
            visualTransformation = PasswordVisualTransformation()
        )
        Spacer(modifier = Modifier.height(8.dp))
        OutlinedTextField(
            value = confirmNewPassword,
            onValueChange ={ confirmNewPassword = it},
            leadingIcon = {
                Icon(
                    imageVector = Icons.Default.Lock,
                    contentDescription = "CONFIRM NEW PASSWORD",
                    tint = Color(0xFFFFA500)
                )
            },
            placeholder = { Text(text = "ENTER SAME PASSWORD", color =
Color.DarkGray) },
            label = { Text(text = "CONFIRM NEW PASSWORD", color =
Color.Black)},
            textStyle = TextStyle(color = Color.Black),
            visualTransformation = PasswordVisualTransformation()
        )
        Spacer(modifier = Modifier.height(16.dp))
        if (error.isNotEmpty()) {
            Text(
                text = error,
```

```
                color = MaterialTheme.colorScheme.error,
                modifier = Modifier.padding(vertical = 16.dp)
            )
        }
        Button(
            onClick = {
                if (username.isNotEmpty() && newPassword.isNotEmpty() &&
confirmNewPassword.isNotEmpty()) {
                    if (newPassword == confirmNewPassword) {
                        // Update password in the database
                        val updated =
databaseHelper.updatePasswordByUsername(username, newPassword)
                        if (updated != null) {
                            error = "Password updated successfully"
                        } else {
                            error = "Failed to update password"
                        }
                    } else {
                        error = "Passwords do not match"
                    }
                } else {
                    error = "Please fill all fields"
                }
            },
            shape = RoundedCornerShape(20.dp),
            colors = ButtonDefaults.buttonColors(Color(0xFF3064FC)),
            modifier = Modifier
                .width(200.dp)
                .padding(top = 16.dp)
        )
        {
            Text(text = "RESET PASSWORD", fontWeight = FontWeight.Bold)
        }
    }
}

@Preview(
    showSystemUi = true,
    showBackground = true
)
@Composable
fun ResetPasswordByUsernamePreview(){
    ResetPasswordByUsernameScreen(
        context = LocalContext.current, // Use LocalContext.current to get
the context
        databaseHelper = UserDatabaseHelper(LocalContext.current) // Use
LocalContext.current to get the context
    )
}
```

NEWS HEADLINES ACTIVITY

MainActivity Class:
This is the main activity of the application and extends ComponentActivity. It initializes a MainViewModel to manage data and user interactions. The onCreate method sets up the user interface and functionality of the main screen.

scheduleNotificationWorker() Method:
This method is used to schedule a one-time notification worker (a background task) using the Android WorkManager. The notification will be shown with a delay of 5 seconds after the activity starts.

signOut() Method:
This method is responsible for signing the user out of the application. It updates the user's login status in shared preferences and navigates the user to the LoginActivity.

NotificationWorker Class:
This is a worker class that extends Worker. It is responsible for showing a notification to the user.
The doWork method is executed when the worker runs. It shows a notification with information about trending headlines. The notification is created using NotificationCompat.Builder and is displayed using the NotificationManagerCompat.

ExpandableIcon Composable:
This composable function displays an expandable icon that, when clicked, reveals a menu with options such as "About," "Sign Out," and "Premium Plans."

User Interface (UI) Components:
The MainActivity sets up the user interface using Jetpack Compose. It displays the app's title, a hamburger icon for the expandable menu, and the top headlines.

ArticleList Composable:
This composable function displays a list of articles obtained from the mainViewModel.articleListResponse. It uses a LazyColumn to display the articles.

ArticleItem Composable:
This composable function represents an individual article item in the list. It displays the article's title and an image. When clicked, it opens a new activity (DisplayNews) to view the full article.

showDialog Function:
This function is used to display an "About" dialog with information about the app, such as the version and future features.

Code:

```
package com.example.newsnest

import android.app.AlertDialog
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
import android.content.Intent
import android.graphics.Color.parseColor
import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
```

```
import androidx.activity.compose.setContent
import androidx.activity.viewModels
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
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.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.itemsIndexed
import androidx.compose.foundation.selection.selectable
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.rounded.Menu
import androidx.compose.material3.Card
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
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.draw.clip
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.Font
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import androidx.work.OneTimeWorkRequest
import androidx.work.WorkManager
import androidx.work.WorkRequest
import androidx.work.Worker
import androidx.work.WorkerParameters
import coil.compose.rememberImagePainter
import coil.size.Scale
import com.example.news.Articles
import com.example.newsnest.ui.theme.NewsNestTheme
import java.util.concurrent.TimeUnit
```

```
class MainActivity : ComponentActivity() {
    private val mainViewModel by viewModels<MainViewModel>()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            NewsNestTheme {
                Column(
                    modifier = Modifier.background(Color.White)
                )
                {
                    var isExpanded by remember { mutableStateOf(false) }

                    Row(Modifier.fillMaxWidth())
                    {
                        Spacer(modifier = Modifier.padding(9.dp,0.dp))
                        ExpandableIcon(
                            modifier = Modifier
                                .align(Alignment.CenterVertically),
                            expanded = isExpanded,
                            onExpand = { isExpanded = !isExpanded },
                            onAboutClicked = { showDialog(this@MainActivity)
},
                            onSignOutClicked = { signOut() }
                        )
                        Text(
                            text = "News Nest 🗞",
                            fontSize = 30.sp,
                            fontWeight = FontWeight.Bold,
                            fontFamily = FontFamily(Font(R.font.arialbold)),
                            modifier = Modifier
                                .weight(1f)
                                .align(Alignment.CenterVertically),
                            textAlign = TextAlign.Center,
                            color = Color.Black
                        )
                    }
                    ArticleList(applicationContext, articleList =
mainViewModel.articleListResponse)
                    mainViewModel.getArticleList()
                }
            }
        }
        scheduleNotificationWorker()
    }

    private fun scheduleNotificationWorker() {
        val notificationWorkRequest: WorkRequest =
OneTimeWorkRequest.Builder(NotificationWorker::class.java)
            .setInitialDelay(5, TimeUnit.SECONDS)
            .build()

        WorkManager.getInstance(this).enqueue(notificationWorkRequest)
    }
```

```
    private fun signOut() {
        val sharedPreferences = getSharedPreferences("login_prefs",
Context.MODE_PRIVATE)
        sharedPreferences.edit().putBoolean("isLoggedIn", false).apply()
        startActivity(Intent(this, LoginActivity::class.java))
        finish()
    }
}

class NotificationWorker(private val context: Context, params:
WorkerParameters) : Worker(context, params) {
    override fun doWork(): Result {
        showNotification()

        return Result.success()
    }

    private fun showNotification() {
        val notificationChannelId = "default"
        val notificationId = 1

        val notificationManager = NotificationManagerCompat.from(context)

        if (android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.O) {
            val channelName = "Default"
            val importance = NotificationManager.IMPORTANCE_DEFAULT
            val channel = NotificationChannel(notificationChannelId,
channelName, importance)
            notificationManager.createNotificationChannel(channel)
        }

        val notificationBuilder = NotificationCompat.Builder(context,
notificationChannelId)
            .setSmallIcon(R.drawable.notification_logo)
            .setContentTitle("Latest & Trending headlines!")
            .setContentText("Explore and discover the most talked-about news
stories.")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .setAutoCancel(true)

        notificationManager.notify(notificationId,
notificationBuilder.build())
    }
}

@Composable
fun ExpandableIcon(
    modifier: Modifier = Modifier,
    expanded: Boolean,
    onExpand: () -> Unit,
    onAboutClicked: () -> Unit,
    onSignOutClicked: () -> Unit
) {
    val context = LocalContext.current
    Box(modifier = modifier)
    {
```

```
        Box(
            contentAlignment = Alignment.Center,
            modifier = Modifier
                .size(30.dp)
                .background(
                    Color(android.graphics.Color.parseColor("#3064FC")),
                    RoundedCornerShape(5.dp)
                )
        )
        {
            Icon(
                imageVector = Icons.Rounded.Menu,
                contentDescription = "Expandable Icon",
                tint = Color(0xFFFFA500),
                modifier = Modifier
                    .size(30.dp)
                    .clickable(onClick = onExpand)
            )
        }
        if (expanded) {
            Column(
                modifier = Modifier
                    .padding(top = 48.dp, start = 8.dp)
//                    .border(1.dp, Color(0xFF3064FC), shape =
RoundedCornerShape(8.dp))
                    .background(Color.White)
                    .clip(RoundedCornerShape(8.dp))
                    .width(120.dp)
            )
            {
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .border(2.dp, Color(0xFF3064FC), shape =
RoundedCornerShape(8.dp))
                        .background(Color.White)
                        .padding(4.dp)
                )
                {
                    Text(
                        text = "About",
                        color = Color(0xFFFFA500),
                        modifier = Modifier
                            .clickable(onClick = onAboutClicked)
                            .padding(8.dp)
                            .fillMaxWidth(),
                        fontFamily = FontFamily(Font(R.font.arialbold)),
                        fontSize = 12.sp
                    )
                }
                Spacer(modifier = Modifier.height(4.dp))
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .border(2.dp, Color(0xFF3064FC), shape =
RoundedCornerShape(8.dp))
                        .background(Color.White)
```

```
                        .padding(4.dp)
                )
                {
                    Text(
                        text = "Sign Out",
                        color = Color(0xFFFFA500),
                        modifier = Modifier
                            .clickable(onClick = onSignOutClicked)
                            .padding(8.dp)
                            .fillMaxWidth(),
                        fontFamily = FontFamily(Font(R.font.arialbold)),
                        fontSize = 12.sp
                    )
                }
                Spacer(modifier = Modifier.height(4.dp))
                Box(
                    modifier = Modifier
                        .fillMaxWidth()
                        .border(2.dp, Color(0xFF3064FC), shape =
RoundedCornerShape(8.dp))
                        .background(Color.White)
                        .padding(4.dp)
                )
                {
                    Text(
                        text = "Premium Plans",
                        color = Color(0xFFFFA500),
                        modifier = Modifier
                            .clickable(
                                onClick = {
                                    val intent = Intent(context,
PremiumPlans::class.java)
                                    context.startActivity(intent)
                                }
                            )
                            .padding(8.dp)
                            .fillMaxWidth(),
                        fontFamily = FontFamily(Font(R.font.arialbold)),
                        fontSize = 12.sp
                    )
                }
            }
        }
    }
}

@Composable
fun ArticleList(context: Context, articleList: List<Articles>) {
    var selectedIndex by remember { mutableStateOf(-1) }

    Column(
        modifier = Modifier
            .padding(8.dp),
        horizontalAlignment = Alignment.CenterHorizontally
    )
    {
        Text(
```

```
            text = "TOP HEADLINES",
            fontFamily = FontFamily(Font(R.font.arialbold)),
            color = Color.Red,
            textAlign = TextAlign.Center,
            fontSize = 28.sp,
            lineHeight = 32.sp
        )
        LazyColumn {
            itemsIndexed(items = articleList) { index, item ->
                ArticleItem(
                    context = context,
                    article = item,
                    index = index,
                    selectedIndex = selectedIndex
                ) { i ->
                    selectedIndex = i
                }
            }
        }
    }
}

@Composable
fun ArticleItem(
    context: Context,
    article: Articles,
    index: Int,
    selectedIndex: Int,
    onClick: (Int) -> Unit
) {
    Card(
        modifier = Modifier
            .padding(20.dp, 8.7.dp)
            .fillMaxWidth()
            .height(250.dp)
            .selectable(true, true, null,
                onClick = {
                    Log.i("test123abc", "ArticleItem:
$index/n$selectedIndex")
                })
            .clickable { onClick(index) },
        shape = RoundedCornerShape(8.dp)
    )
    {
        Surface(color = Color(parseColor("#3064FC")))
        {
            Row(
                Modifier
                    .padding(4.dp)
                    .fillMaxSize()
                    .background(Color.White), //brush = gradientBrush
                horizontalArrangement = Arrangement.Center
            )
            {
                Column(
                    verticalArrangement = Arrangement.Center,
                    modifier = Modifier
```

```
                        .padding(8.dp)
                        .selectable(true, true, null,
                            onClick = {
                                Log.i("test123abc", "ArticleItem:
$index/n${article.description}")
                                context.startActivity(
                                    Intent(context, DisplayNews::class.java)

.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
                                        .putExtra("desk",
article.description.toString())
                                        .putExtra("urlToImage",
article.urlToImage)
                                        .putExtra("title", article.title)
                                )
                            }
                        )
                )
                {
                    Box (
                        modifier = Modifier
                            .weight(1f)
                            .padding(8.dp)
                    )
                    {
                        Image(
                            painter = rememberImagePainter(
                                data = article.urlToImage,
                                builder = {
                                    scale(Scale.FILL)

placeholder(R.drawable.placeholder_loading)
                                    fallback(R.drawable.placeholder_news)
                                }
                            ),
                            contentScale = ContentScale.Crop,
                            contentDescription = article.description,
                            modifier = Modifier
                                .fillMaxWidth()
                                .clip(RoundedCornerShape(8.dp))
                        )
                    }
                    Spacer(modifier = Modifier.height(8.dp))
                    Box(
                        modifier = Modifier
                            .padding(8.dp),
                        contentAlignment = Alignment.Center
                    )
                    {
                        Text(
                            text = article.title.toString(),
                            fontSize = 16.sp,
                            style = MaterialTheme.typography.titleSmall,
                            textAlign = TextAlign.Justify,
                            fontFamily = FontFamily(Font(R.font.arialbold)),
                            color = Color.DarkGray
                        )
```

```
                    }
                }
            }
        }
    }
}


fun showDialog(context: Context) {
    val dialog = AlertDialog.Builder(context)
        .setTitle("ⓘAbout")
        .setMessage("Version 1.0\n" +
                "We will be releasing more features.\n" +
                "Stay Tuned")
        .setPositiveButton("OK")

        { _, _ ->
            // Handle OK button click if needed
        }
        .create()


    dialog.show()
}
```

DISPLAY NEWS ACTIVITY

DisplayNews Class:
This class represents the DisplayNews activity and extends ComponentActivity. In the onCreate method, it retrieves data passed via intent, such as the news article's description (desk), title, and URL to the article's image. It then sets the content of the activity to the DisplayNewsContent composable, passing the retrieved data.

DisplayNewsContent Composable:
This composable function represents the content of the DisplayNews activity. It uses Jetpack Compose to create the user interface for displaying the news article.

DisplayNewsCard Composable:
This composable function displays the news article in a card format. It includes the news article's title, image, and a button to read the full article. The article's title, image URL, and description (desk) are passed as parameters.

HtmlText Composable:
This composable function is used to render HTML-formatted text within the article description. It uses an AndroidView to display the HTML content as a TextView.

User Interface (UI) Components:
The DisplayNews activity uses Jetpack Compose to create the UI.
It includes a card that displays the news article's title, an image, and the article description.

Image Display:
The news article's image is displayed using the Coil library for image loading and is scaled to fit the card.

Button:
A "READ FULL ARTICLE" button is provided, which allows users to access the full article by navigating to another activity, PaymentReadFullArticle, when clicked.

Code:

```
package com.example.newsnest

import android.content.Intent
import android.os.Bundle
import android.widget.TextView
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.Image
import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
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.shape.RoundedCornerShape
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.Card
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.text.font.Font
import androidx.compose.ui.text.font.FontFamily
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.text.style.TextOverflow
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import androidx.compose.ui.viewinterop.AndroidView
import androidx.core.text.HtmlCompat
import coil.compose.rememberImagePainter
import coil.size.Scale
import com.example.newsnest.ui.theme.NewsNestTheme

class DisplayNews : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val desk = intent.getStringExtra("desk") ?: ""
        val title = intent.getStringExtra("title") ?: ""
        val uriImage = intent.getStringExtra("urlToImage") ?: ""
        setContent {
            DisplayNewsContent(
                desk = desk,
```

```
                title = title,
                uriImage = uriImage
            )
        }
    }
}

@Composable
fun DisplayNewsContent(
    desk: String,
    title: String,
    uriImage: String
) {
    NewsNestTheme {
        Surface(
            modifier = Modifier
                .fillMaxSize(),
            color = Color.White
        ) {
            Column(
                modifier = Modifier
                    .fillMaxSize()
                    .padding(20.dp),
                horizontalAlignment = Alignment.CenterHorizontally,
                verticalArrangement = Arrangement.Center
            )
            {
                DisplayNewsCard(
                    desk = desk,
                    title = title,
                    uriImage = uriImage
                )
            }
        }
    }
}

@Composable
fun DisplayNewsCard(
    desk: String,
    title: String,
    uriImage: String
)
{
    val context = LocalContext.current
    Card(
        modifier = Modifier
            .fillMaxSize()
    )
    {
        Column(
            modifier = Modifier
                .border(8.dp, Color(0xFF3064FC), shape =
RoundedCornerShape(8.dp))
                .background(Color.White)
        )
        {
```

```
            Card (
                modifier = Modifier
                    .padding(20.dp)
            )
            {
                Image(
                    modifier = Modifier
                        .height(200.dp)
                        .border(8.dp, Color.Transparent, shape =
RoundedCornerShape(8.dp)),
                    painter = rememberImagePainter(
                        data = uriImage,
                        builder = {
                            scale(Scale.FILL)
                            placeholder(R.drawable.placeholder_loading)
                            error(R.drawable.placeholder_news)
                        }
                    ),
                    contentDescription = "News Image",
                    contentScale = ContentScale.Crop
                )
            }
            Spacer(modifier = Modifier.height(8.dp))
            Column(
                modifier = Modifier
                    .padding(
                        start = 20.dp,
                        end = 20.dp
                    )
                    .fillMaxSize()
            )
            {
                Text(
                    text = title,
                    fontFamily = FontFamily(Font(R.font.timesnewroman)),
                    color = Color(0xFFB01E3A),
                    fontSize = 24.sp,
                    lineHeight = 28.sp,
                    overflow = TextOverflow.Ellipsis
                )
                Spacer(modifier = Modifier.height(12.dp))
                Text(
                    text = desk,
                    fontFamily = FontFamily(Font(R.font.arialnormal)),
                    color = Color.Black,
                    textAlign = TextAlign.Justify,
                    fontSize = 16.sp,
                    lineHeight = 20.sp,
                    overflow = TextOverflow.Ellipsis
                )
                Spacer(modifier = Modifier.height(30.dp))
                Button(
                    onClick =
                    {
                        val intent = Intent(context, PaymentReadFullArticle
:: class.java)
                        context.startActivity(intent)
```

```
                    },
                    shape = RoundedCornerShape(20.dp),
                    colors = ButtonDefaults.buttonColors(Color(0xFFFFA500)),
                    modifier = Modifier
                        .fillMaxWidth()
                )
                {
                    Text(
                        text = "READ FULL ARTICLE",
                        fontWeight = FontWeight.Bold,
                        color = Color.White
                    )
                }
            }
        }
    }
}
@Composable
fun HtmlText(html: String, modifier: Modifier = Modifier) {
    AndroidView(
        modifier = modifier,
        factory = { context -> TextView(context) },
        update = { it.text = HtmlCompat.fromHtml(html,
HtmlCompat.FROM_HTML_MODE_COMPACT) }
    )
}
```

DATABASE SCHEMA (DATABASEHELPER ACTIVITY)

COLUMN_ID (INTEGER PRIMARY KEY AUTOINCREMENT):
This column represents the unique identifier for each user and is set as the primary key with auto-increment. It ensures that each user has a unique ID.

COLUMN_FIRST_NAME (TEXT):
This column stores the user's first name as text.

COLUMN_LAST_NAME (TEXT):
This column stores the user's last name as text.

COLUMN_EMAIL (TEXT):
This column stores the user's email address as text. Email addresses should be unique to each user to ensure data integrity.

COLUMN_PASSWORD (TEXT):
This column stores the user's password as text. It is important to note that storing passwords in plain text is not a secure practice in a real-world application. In a production application, passwords should be securely hashed and salted before storage for security reasons.

The database is created and managed using the SQLiteOpenHelper class, which includes methods to create and upgrade the database schema. The onCreate method creates the user_table with the specified columns, while the onUpgrade method drops the table if it exists and recreates it when the database version changes. Additionally, the UserDatabaseHelper class provides methods for performing various database operations, such as inserting a new user (insertUser), updating a user's password by email (updatePasswordByEmail), updating a user's password by username

(updatePasswordByUsername), retrieving a user by username (getUserByUsername), retrieving a user by ID (getUserById), and retrieving all users (getAllUsers).

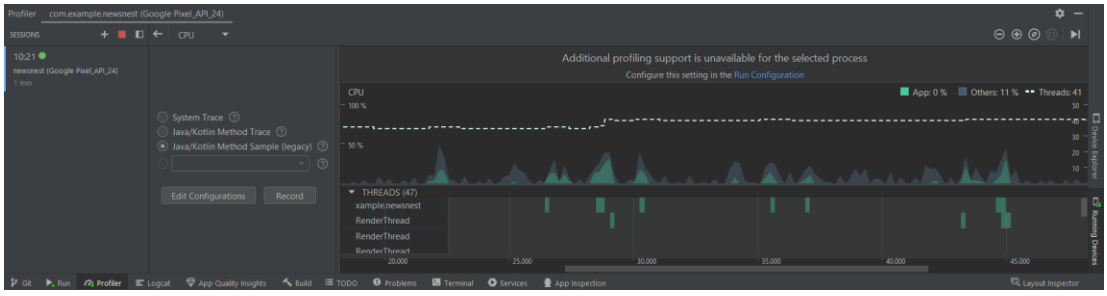
```
package com.example.newsnest

import android.annotation.SuppressLint
import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class UserDatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    companion object {
        private const val DATABASE_VERSION = 1
        private const val DATABASE_NAME = "UserDatabase.db"
        private const val TABLE_NAME = "user_table"
        private const val COLUMN_ID = "id"
        private const val COLUMN_FIRST_NAME = "first_name"
        private const val COLUMN_LAST_NAME = "last_name"
        private const val COLUMN_EMAIL = "email"
        private const val COLUMN_PASSWORD = "password"
    }

    override fun onCreate(db: SQLiteDatabase?) {
        val createTable = "CREATE TABLE $TABLE_NAME (" +
                "$COLUMN_ID INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "$COLUMN_FIRST_NAME TEXT, " +
                "$COLUMN_LAST_NAME TEXT, " +
                "$COLUMN_EMAIL TEXT, " +
                "$COLUMN_PASSWORD TEXT" +
                ")"

        db?.execSQL(createTable)
    }

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion:
Int) {
        db?.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }

    fun insertUser(user: User) {
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_FIRST_NAME, user.firstName)
        values.put(COLUMN_LAST_NAME, user.lastName)
        values.put(COLUMN_EMAIL, user.email)
        values.put(COLUMN_PASSWORD, user.password)
        db.insert(TABLE_NAME, null, values)
        db.close()
    }

    fun updatePasswordByEmail(email: String, newPassword: String): Int {
        val db = writableDatabase
        val values = ContentValues()
```

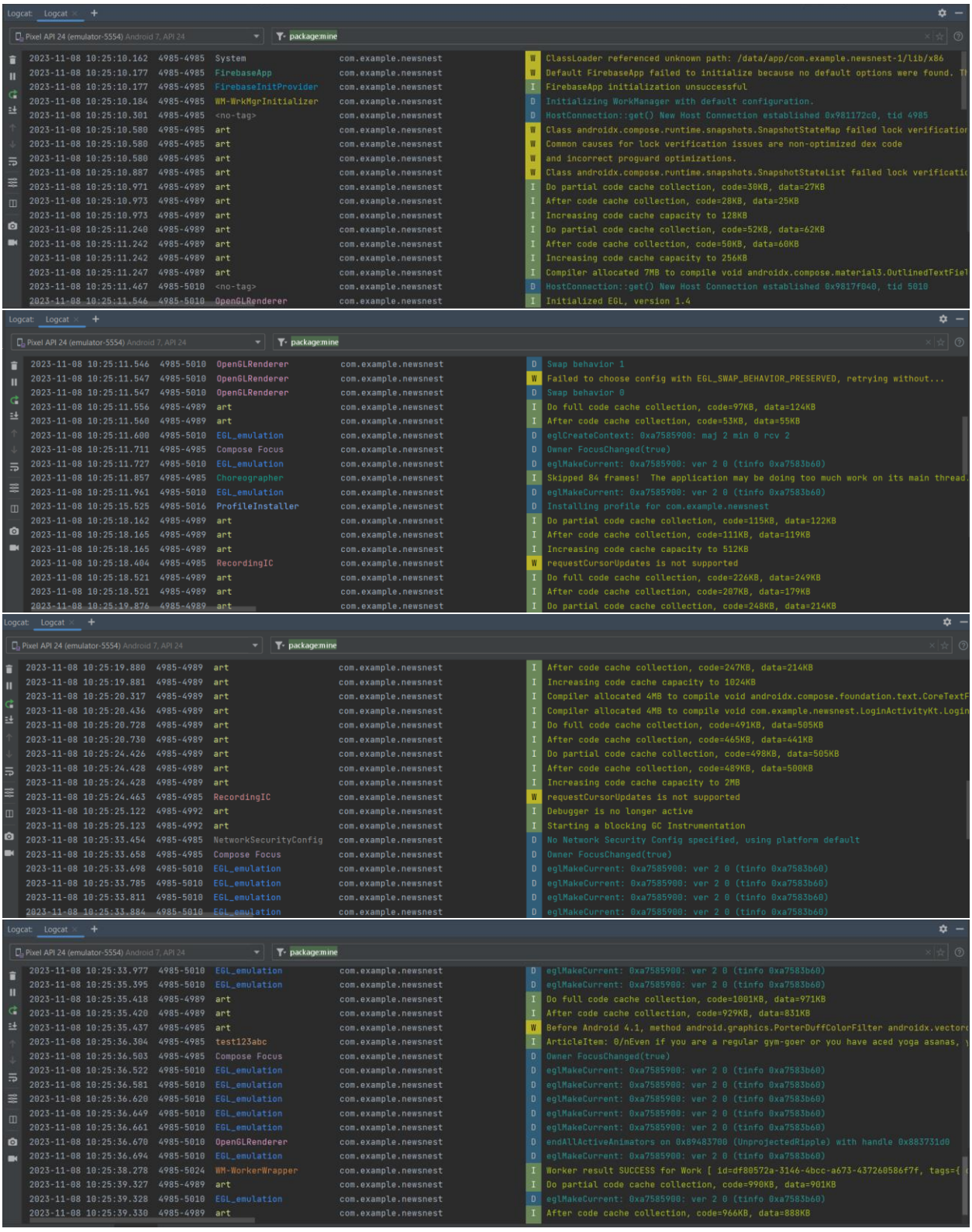
```
        values.put(COLUMN_PASSWORD, newPassword)
        return db.update(TABLE_NAME, values, "$COLUMN_EMAIL = ?",
arrayOf(email))
    }
    fun updatePasswordByUsername(username: String, newPassword: String): Int
{
        val db = writableDatabase
        val values = ContentValues()
        values.put(COLUMN_PASSWORD, newPassword)
        return db.update(TABLE_NAME, values, "$COLUMN_FIRST_NAME = ?",
arrayOf(username))
    }

    @SuppressLint("Range")
    fun getUserByUsername(username: String): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_FIRST_NAME = ?", arrayOf(username))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
        return user
    }
    @SuppressLint("Range")
    fun getUserById(id: Int): User? {
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME WHERE
$COLUMN_ID = ?", arrayOf(id.toString()))
        var user: User? = null
        if (cursor.moveToFirst()) {
            user = User(
                id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
            )
        }
        cursor.close()
        db.close()
```

```
        return user
    }

    @SuppressLint("Range")
    fun getAllUsers(): List<User> {
        val users = mutableListOf<User>()
        val db = readableDatabase
        val cursor: Cursor = db.rawQuery("SELECT * FROM $TABLE_NAME", null)
        if (cursor.moveToFirst()) {
            do {
                val user = User(
                    id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID)),
                    firstName =
cursor.getString(cursor.getColumnIndex(COLUMN_FIRST_NAME)),
                    lastName =
cursor.getString(cursor.getColumnIndex(COLUMN_LAST_NAME)),
                    email =
cursor.getString(cursor.getColumnIndex(COLUMN_EMAIL)),
                    password =
cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD)),
                )
                users.add(user)
            } while (cursor.moveToNext())
        }
        cursor.close()
        db.close()
        return users
    }

}
```

PERFORMANCE TESTING

PERFORMACE METRICS





Logcat: Logcat

Pixel API 24 (emulator-5554) Android 7, API 24 — package:mine

```
2023-11-08 10:25:10.162  4985-4985  System                  com.example.newsnest  W  ClassLoader referenced unknown path: /data/app/com.example.newsnest-1/lib/x86
2023-11-08 10:25:10.177  4985-4985  FirebaseApp             com.example.newsnest  W  Default FirebaseApp failed to initialize because no default options were found. Th
2023-11-08 10:25:10.177  4985-4985  FirebaseInitProvider    com.example.newsnest  I  FirebaseApp initialization unsuccessful
2023-11-08 10:25:10.184  4985-4985  WM-WrkMgrInitializer    com.example.newsnest  D  Initializing WorkManager with default configuration.
2023-11-08 10:25:10.301  4985-4985  <no-tag>                com.example.newsnest  D  HostConnection::get() New Host Connection established 0x981172c0, tid 4985
2023-11-08 10:25:10.580  4985-4985  art                     com.example.newsnest  W  Class androidx.compose.runtime.snapshots.SnapshotStateMap failed lock verificatior
2023-11-08 10:25:10.580  4985-4985  art                     com.example.newsnest  W  Common causes for lock verification issues are non-optimized dex code
2023-11-08 10:25:10.580  4985-4985  art                     com.example.newsnest  W  and incorrect proguard optimizations.
2023-11-08 10:25:10.887  4985-4985  art                     com.example.newsnest  W  Class androidx.compose.runtime.snapshots.SnapshotStateList failed lock verificatic
2023-11-08 10:25:10.971  4985-4989  art                     com.example.newsnest  I  Do partial code cache collection, code=30KB, data=27KB
2023-11-08 10:25:10.973  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=28KB, data=25KB
2023-11-08 10:25:10.973  4985-4989  art                     com.example.newsnest  I  Increasing code cache capacity to 128KB
2023-11-08 10:25:11.240  4985-4989  art                     com.example.newsnest  I  Do partial code cache collection, code=52KB, data=62KB
2023-11-08 10:25:11.242  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=50KB, data=60KB
2023-11-08 10:25:11.242  4985-4989  art                     com.example.newsnest  I  Increasing code cache capacity to 256KB
2023-11-08 10:25:11.247  4985-4989  art                     com.example.newsnest  I  Compiler allocated 7MB to compile void androidx.compose.material3.OutlinedTextFiel
2023-11-08 10:25:11.467  4985-5010  <no-tag>                com.example.newsnest  D  HostConnection::get() New Host Connection established 0x9817f040, tid 5010
2023-11-08 10:25:11.546  4985-5010  OpenGLRenderer          com.example.newsnest  I  Initialized EGL, version 1.4
```

Logcat: Logcat

Pixel API 24 (emulator-5554) Android 7, API 24 — package:mine

```
2023-11-08 10:25:11.546  4985-5010  OpenGLRenderer          com.example.newsnest  D  Swap behavior 1
2023-11-08 10:25:11.547  4985-5010  OpenGLRenderer          com.example.newsnest  W  Failed to choose config with EGL_SWAP_BEHAVIOR_PRESERVED, retrying without...
2023-11-08 10:25:11.547  4985-5010  OpenGLRenderer          com.example.newsnest  D  Swap behavior 0
2023-11-08 10:25:11.556  4985-4989  art                     com.example.newsnest  I  Do full code cache collection, code=97KB, data=124KB
2023-11-08 10:25:11.560  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=53KB, data=55KB
2023-11-08 10:25:11.600  4985-5010  EGL_emulation           com.example.newsnest  D  eglCreateContext: 0xa7585900: maj 2 min 0 rcv 2
2023-11-08 10:25:11.711  4985-4985  Compose Focus           com.example.newsnest  D  Owner FocusChanged(true)
2023-11-08 10:25:11.727  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:11.857  4985-4985  Choreographer           com.example.newsnest  I  Skipped 84 frames!  The application may be doing too much work on its main thread.
2023-11-08 10:25:11.961  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:15.525  4985-5016  ProfileInstaller        com.example.newsnest  D  Installing profile for com.example.newsnest
2023-11-08 10:25:18.162  4985-4989  art                     com.example.newsnest  I  Do partial code cache collection, code=115KB, data=122KB
2023-11-08 10:25:18.165  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=111KB, data=119KB
2023-11-08 10:25:18.165  4985-4989  art                     com.example.newsnest  I  Increasing code cache capacity to 512KB
2023-11-08 10:25:18.404  4985-4985  RecordingIC             com.example.newsnest  W  requestCursorUpdates is not supported
2023-11-08 10:25:18.521  4985-4989  art                     com.example.newsnest  I  Do full code cache collection, code=226KB, data=249KB
2023-11-08 10:25:18.521  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=207KB, data=179KB
2023-11-08 10:25:19.876  4985-4989  art                     com.example.newsnest  I  Do partial code cache collection, code=248KB, data=214KB
```

Logcat: Logcat

Pixel API 24 (emulator-5554) Android 7, API 24 — package:mine

```
2023-11-08 10:25:19.880  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=247KB, data=214KB
2023-11-08 10:25:19.881  4985-4989  art                     com.example.newsnest  I  Increasing code cache capacity to 1024KB
2023-11-08 10:25:20.317  4985-4989  art                     com.example.newsnest  I  Compiler allocated 4MB to compile void androidx.compose.foundation.text.CoreTextF
2023-11-08 10:25:20.436  4985-4989  art                     com.example.newsnest  I  Compiler allocated 4MB to compile void com.example.newsnest.LoginActivityKt.Login
2023-11-08 10:25:20.728  4985-4989  art                     com.example.newsnest  I  Do full code cache collection, code=491KB, data=505KB
2023-11-08 10:25:20.730  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=465KB, data=441KB
2023-11-08 10:25:24.426  4985-4989  art                     com.example.newsnest  I  Do partial code cache collection, code=498KB, data=505KB
2023-11-08 10:25:24.428  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=489KB, data=500KB
2023-11-08 10:25:24.428  4985-4989  art                     com.example.newsnest  I  Increasing code cache capacity to 2MB
2023-11-08 10:25:24.463  4985-4985  RecordingIC             com.example.newsnest  W  requestCursorUpdates is not supported
2023-11-08 10:25:25.122  4985-4992  art                     com.example.newsnest  I  Debugger is no longer active
2023-11-08 10:25:25.123  4985-4992  art                     com.example.newsnest  I  Starting a blocking GC Instrumentation
2023-11-08 10:25:33.454  4985-4985  NetworkSecurityConfig   com.example.newsnest  D  No Network Security Config specified, using platform default
2023-11-08 10:25:33.658  4985-4985  Compose Focus           com.example.newsnest  D  Owner FocusChanged(true)
2023-11-08 10:25:33.698  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:33.785  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:33.811  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:33.884  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
```

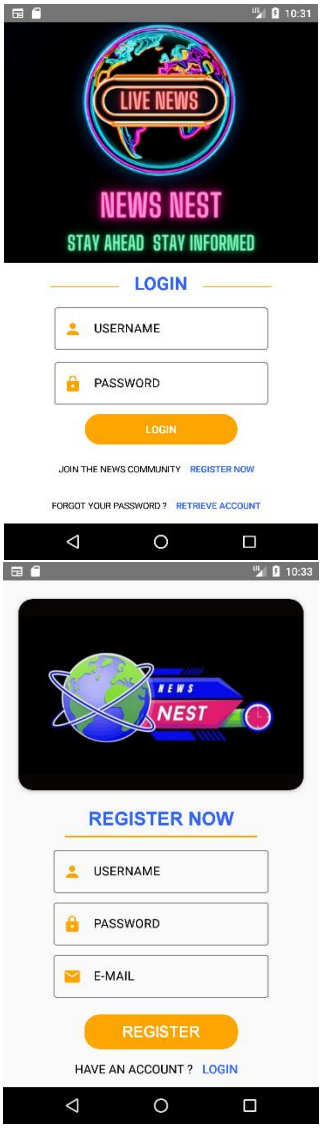
Logcat: Logcat

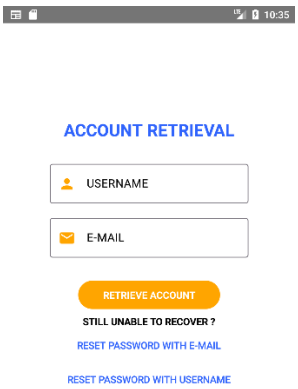
Pixel API 24 (emulator-5554) Android 7, API 24 — package:mine

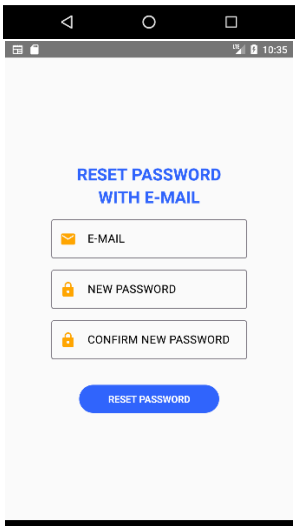
```
2023-11-08 10:25:33.977  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:35.395  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:35.418  4985-4989  art                     com.example.newsnest  I  Do full code cache collection, code=1001KB, data=971KB
2023-11-08 10:25:35.420  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=929KB, data=831KB
2023-11-08 10:25:35.437  4985-4985  art                     com.example.newsnest  W  Before Android 4.1, method android.graphics.PorterDuffColorFilter androidx.vectorc
2023-11-08 10:25:36.304  4985-4985  test123abc              com.example.newsnest  I  ArticleItem: 0/nEven if you are a regular gym-goer or you have aced yoga asanas, y
2023-11-08 10:25:36.503  4985-4985  Compose Focus           com.example.newsnest  D  Owner FocusChanged(true)
2023-11-08 10:25:36.522  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:36.581  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:36.620  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:36.649  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:36.661  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:36.670  4985-5010  OpenGLRenderer          com.example.newsnest  D  endAllActiveAnimators on 0x89483700 (UnprojectedRipple) with handle 0x883731d0
2023-11-08 10:25:36.694  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:38.278  4985-5024  WM-WorkerWrapper        com.example.newsnest  I  Worker result SUCCESS for Work [ id=df80572a-3146-4bcc-a673-437260586f7f, tags={
2023-11-08 10:25:39.327  4985-4989  art                     com.example.newsnest  I  Do partial code cache collection, code=990KB, data=901KB
2023-11-08 10:25:39.328  4985-5010  EGL_emulation           com.example.newsnest  D  eglMakeCurrent: 0xa7585900: ver 2 0 (tinfo 0xa7583b60)
2023-11-08 10:25:39.330  4985-4989  art                     com.example.newsnest  I  After code cache collection, code=966KB, data=888KB
```

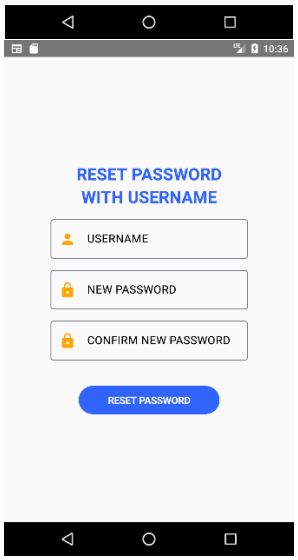
RESULTS

OUTPUT SCREENSHOTS




10:35
ACCOUNT RETRIEVAL
USERNAME
E-MAIL
RETRIEVE ACCOUNT
STILL UNABLE TO RECOVER ?
RESET PASSWORD WITH E-MAIL
RESET PASSWORD WITH USERNAME


10:35
RESET PASSWORD
WITH E-MAIL
E-MAIL
NEW PASSWORD
CONFIRM NEW PASSWORD
RESET PASSWORD


10:36
RESET PASSWORD
WITH USERNAME
USERNAME
NEW PASSWORD
CONFIRM NEW PASSWORD
RESET PASSWORD

ADVANTAGES & DISADVANTAGES

Certainly! Here are some potential advantages and disadvantages of the News Nest app:

Advantages:

1. Curated News Content: News Nest provides users with curated and reliable news content, saving them time and effort in sifting through an overwhelming amount of information.

2. Flexibility in Reading Options: Users have the choice between paying per article or subscribing for unlimited access, catering to a wide range of preferences and budgets.

3. Monetization Opportunities: The app offers a dual revenue model through pay-per-article and subscription plans, providing a sustainable business model for the creators.

4. Personalized Experience: By implementing features like bookmarking, favourites, and search functionality, News Nest allows users to tailor their news consumption to their specific interests.

5. Secure and User-Friendly: With robust user authentication and account management functionalities, News Nest ensures a secure and seamless experience for its users.

6. Technological Innovation: Leveraging advanced technologies like Kotlin, Jetpack Compose, and Room Database, the app showcases cutting-edge development practices.

7. Efficient Payment Integration: The integration of a secure payment gateway ensures smooth and hassle-free transactions for users.

8. Convenient Account Recovery: Users can retrieve their accounts or reset passwords easily, enhancing overall user satisfaction and convenience.

Disadvantages:

1. Competitive Landscape: The news app market is highly competitive, with established players and a plethora of alternatives. Gaining a substantial user base and standing out in this crowded space can be difficult.
2. Dependency on External Data Sources: Reliance on external APIs like NewsAPI may lead to potential issues if there are disruptions or changes in the data source.
3. Platform Dependency: Developing and maintaining the app for multiple platforms (e.g., Android, iOS) can be resource-intensive and may require separate development efforts.

CONCLUSION

In conclusion, News Nest emerges as a dynamic and innovative solution in the realm of news applications, offering a curated and personalized news consumption experience. By harnessing cutting-edge technologies like Kotlin, Jetpack Compose, and Room Database, the app delivers a seamless and efficient platform for users to access the latest headlines and in-depth articles. The dual monetization model, encompassing pay-per-article and subscription plans, provides users with the flexibility to choose how they engage with news content.

In essence, News Nest is poised to revolutionize how users interact with news content, offering a refined, reliable, and convenient platform for staying informed. With a focus on continual improvement and responsive user engagement, News Nest is well-positioned to make a significant impact in the ever-evolving landscape of digital news consumption.

FUTURE SCOPE

The future scope of News Nest holds exciting potential for expansion and enhancement. Here are some avenues for future development and improvement:

1. Multi-Platform Availability: Consider developing versions of News Nest for iOS and other platforms, ensuring a broader user base and greater accessibility.
2. Enhanced Personalization: Implement advanced algorithms and machine learning techniques to provide highly personalized news recommendations based on user preferences and behaviour.
3. Audio and Video Content Integration: Incorporate features for audio and video news content, catering to users who prefer multimedia formats.
4. Collaborations and Partnerships: Forge partnerships with reputable news sources or media outlets to offer exclusive content, further enriching the app's content library.
5. Geographical Expansion: Explore opportunities to provide region-specific news coverage, offering content tailored to different markets and languages.
6. Community and User-Generated Content: Introduce features such as user-submitted articles, comments, and community discussions where we verify user submitted articles and for other requirements to foster an engaged user community.
7. Real-time Updates and Alerts: Implement real-time notifications for breaking news and urgent updates, ensuring users receive timely information.
8. Voice Assistants Integration: Enable interaction with the app through voice commands, enhancing accessibility for users with disabilities or those seeking a hands-free experience.
9. Social Media Integration: Allow users to easily share articles and news updates on their preferred social media platforms, increasing the app's reach and user engagement.

10. AI-driven Content Verification: Employ artificial intelligence to verify the authenticity and credibility of news sources and articles, combating the spread of misinformation.
11. Content Monetization for Publishers: Create a platform for content creators and publishers to monetize their articles through partnerships or revenue-sharing models.
12. Data Analytics and User Insights: Utilize data analytics to gain insights into user behaviour, preferences, and engagement patterns, informing future content curation and app enhancements.
13. Offline Reading Enhancements: Develop features that allow users to save articles for offline reading, ensuring content remains accessible even without an internet connection.
14. Gamification and Rewards: Introduce gamified elements to incentivize user engagement, such as badges, rewards, and challenges tied to reading habits and interactions within the app.

APPENDIX

SOURCE CODE

https://github.com/smartinternz02/SI-GuidedProject-587987-1696931398/tree/project_code

GITHUB

https://github.com/smartinternz02/SI-GuidedProject-587987-1696931398

PROJECT DEMO LINK

https://drive.google.com/file/d/13SXIyw7jRpAqFWAonH8tZtqjHtDZmCUG/view