ระบบ แนะนำภาพยนต์ (DooRaiD)

เสนอ

อาจารย์ภัทร อัยรักษ์

นำเสนอโดย

6410210702 นางสาว อารีนา เกะรา

6510210279 นาย รัชชานนท์ ชูคง

6510210132 นาย ธีรนันท์ แนะนวน

6510210216 นาย พรหมพิริยะ พรหมสะอาด

6510210204 นาย ปุณณรัตน์ เฉียบแหลม

สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

มหาวิทยาลัยสงขลานครินท์ วิทยาเขตหาดใหญ่

แบบนำเสนอโครงการ (Project Proposal)

1. **ชื่อหัวข้อ**

   ระบบ แนะนำภาพยนต์ (DooraiD)

2. **รายชื่อนักศึกษา**

   6410210702 นางสาว อารีนา เกะรา

   6510210279 นาย รัชชานนท์ ชูคง

   6510210132 นาย ธีรนันท์ แนะนวน

   6510210216 นาย พรหมพิริยะ พรหมสะอาด

   6510210204 นาย ปุณณรัตน์ เฉียบแหลม

3. **ชื่ออาจารย์ที่ปรึกษา**

   อาจารย์ภัทร อัยรักษ์

4. **ที่มา และความสำคัญ**

   เมื่อผู้คนเริ่มหันมาบริโภคสื่อดิจิทัลผ่านแพลตฟอร์มออนไลน์มากขึ้น การค้นหาภาพยนตร์หรือรายการที่ตรงใจจากข้อมูลมหาศาลจึงกลายเป็นเรื่องท้าทาย ระบบแนะนำจึงเกิดขึ้นเพื่อช่วยผู้ใช้ค้นหาภาพยนตร์ที่น่าสนใจได้อย่างมีประสิทธิภาพ

   ระบบแนะนำภาพยนต์ (DooRaiD) นั้นจะถูกพัฒนาขึ้นเพื่อแก้ไขปัญหาในการค้นหาภาพยนต์ที่มีจำนวนมากและทำการจัดหมวดหมู่เพื่อการค้นหาภาพยนต์ที่ผู้ใช้ต้องการ

   ระบบแนะนำภาพยนต์ (DooRaiD) จึงเป็นเครื่องมือในการจัดการกับข้อมูลมหาศาล ที่ไม่เพียงช่วยเพิ่มความสะดวกสบายในการ แต่ยังสอดคล้องกับความต้องการและพฤติกรรมของผู้คนเริ่มหันมาบริโภคสื่อดิจิทัลผ่านแพลตฟอร์มออนไลน์

5. **วัตถุประสงค์ของโครงการ**

   1. **เพิ่มความสะดวกในการค้นหา**:
      ช่วยประหยัดเวลาในการเลือกภาพยนต์อย่างการใช้ตัวกรองเพื่อให้ได้ภาพยนต์ที่ตรงกับความต้องการของผู้ใช้หรือไลฟ์สไตล์ที่เข้ากับตัวเองได้อย่างสะดวกและรวดเร็ว
   2. **เพื่อการเข้าถึงข้อมูลและรีวิวภาพยนต์อย่างครบถ้วน**:
      ผู้ใช้สามารถเข้าถึงข้อมูลเกี่ยวกับภาพยนต์ เช่น เรื่องย่อ, รายละเอียดนักแสดง, และประเภทของหนังได้ในแอปเดียว ทำให้สามารถตัดสินใจได้ง่ายขึ้น

3. **เพื่อจัดทำรายการแนะนำที่ครอบคลุมและหลากหลาย**:
   ระบบจะสร้างรายการแนะนำที่ครอบคลุมทุกหมวดหมู่ภาพยนตร์
   ไม่ว่าจะเป็นภาพยนตร์ประเภทใดหรือจากภูมิภาคใด
   เพื่อให้ผู้ใช้มีตัวเลือกที่หลากหลายในการรับชม

6. **ขอบเขต**

1. ผู้ใช้สามารถค้นหาภาพยนต์จากฐานข้อมูลของแอป โดยใช้คำค้นหาต่างๆ เช่น ชื่อภาพยนต์, นักแสดง, หรือปีที่ฉาย
2. ผู้ใช้สามารถดูข้อมูลภาพยนต์อย่างละเอียด เช่น เรื่องย่อ, นักแสดง, ผู้กำกับ, ความยาว, ประเภทหรือตัวอย่างภาพยนต์, ภาพจากภาพยนต์ และเนื้อหาเพิ่มเติมอื่น ๆ ที่เกี่ยวข้อง

7. **ขั้นตอนและแผนการดำเนินโครงการ**

7.1 ขั้นตอนการดำเนินโครงการ

โครงงานนี้ดำเนินงานโดยใช้วงจรการพัฒนาระบบแบบ System Development Life Cycle (SDLC) โดยมีรายละเอียดดังแสดงในตารางที่ 1.1

ตารางที่ 1.1 ขั้นตอนการดำเนินงาน

ขั้นที่	กิจกรรม
1	กำหนดขอบเขตและวางแผนโครงการ (Project Planning) - ศึกษาปัญหาของคนที่ชอบดูภาพยนต์ - ศึกษาเครื่องมือและเทคโนโลยีที่จะนำมาใช้ในการแก้ไขปัญหา
2	การวิเคราะห์ระบบ (Analysis) - วิเคราะห์ความต้องการของคนที่ชอบดูภาพยนต์ - วิเคราะห์ความต้องการของระบบแนะนำภาพยนต์ (DooRaiD) - กำหนดขอบเขตความต้องการของระบบแนะนำภาพยนต์ (DooRaiD)
3	การออกแบบระบบ (Design) - ออกแบบโครงสร้างของแอป,การออกแบบโครงสร้างฐานข้อมูล - ออกแบบหน้าจอการแสดงผล - ออกแบบโครงสร้างฟังก์ชั้นต่างๆ
4	การพัฒนาระบบ (Development) - พัฒนาโปรแกรมโดยใช้ React Native จากข้อมูลที่ได้วิเคราะห์และออกแบบไว้ - การทดสอบหน่วยสำหรับแต่ละโมดูลเพื่อให้มั่นใจว่าโค้ดทำงานตามที่คาดหวัง

5	การทดสอบและปรับปรุงแก้ไข (Testing) - ทดสอบการใช้งาน เพื่อตรวจหาข้อบกพร่องที่อาจเกิดขึ้น - แก้ไขข้อผิดพลาดที่เกิดขึ้น

8. **เครื่องมือที่ใช้ในโครงการ**
   - การออกแบบ(Design) : Figma
   - ภาษาที่ใช้ในการเขียน(Programming Languages) : JavaScript (JS)
   - เฟรมเวิร์ก(Frameworks) : React Native
   - ฐานข้อมูล: Firebase

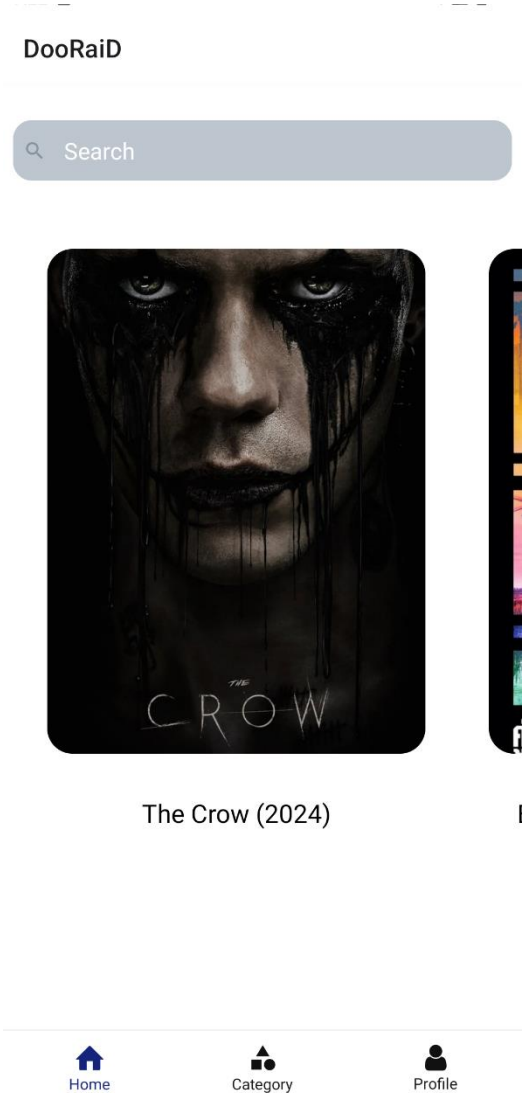
9. **ประโยชน์ที่คาดว่าจะได้รับ**
   1. ระบบแนะนำช่วยให้ผู้ใช้สามารถค้นหาภาพยนตร์ที่ตรงกับความสนใจได้ง่ายและรวดเร็ว ลดความยุ่งยากในการเลือกหนังจากตัวเลือกจำนวนมาก
   2. ระบบสามารถแนะนำภาพยนตร์ที่ผู้ใช้อาจจะไม่รู้จักมาก่อน ช่วยขยายขอบเขตของภาพยนตร์ที่ผู้ใช้ได้รับชม
   3. ระบบช่วยแยกแยะหมวดหมู่ ให้ตรงกับภาพยนต์ที่ผู้ใช้สนใจโดยการเฉพาะเจาะจงในหมวดหมู่ใดๆ ที่ต้องการค้นหา

Overall, of Our App

แอพพลิเคชั่นจะมี การเข้าใช้ระบบต่างกันโดยแยกว่าเป็น User หรือ Admin

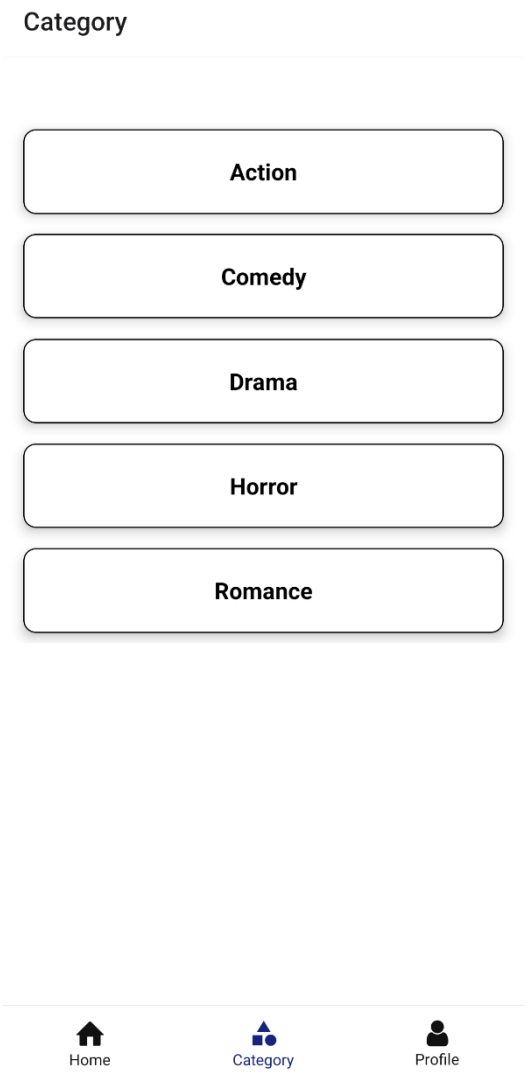
- **หน้า Home**

เป็นหน้าเริ่มต้นของแอพพลิเคชั่นวเมื่อผู้ใช้เข้ามาในแอพลิเคชั่น จะแสดงให้เห็นถึงหนังน่าดูประจำเดือน และมีพื้นที่สำหรับค้นหาหนังหากผู้ใช้อยากรู้เกี่ยวกับภาพโดยรวมของเรื่องนั้นๆ

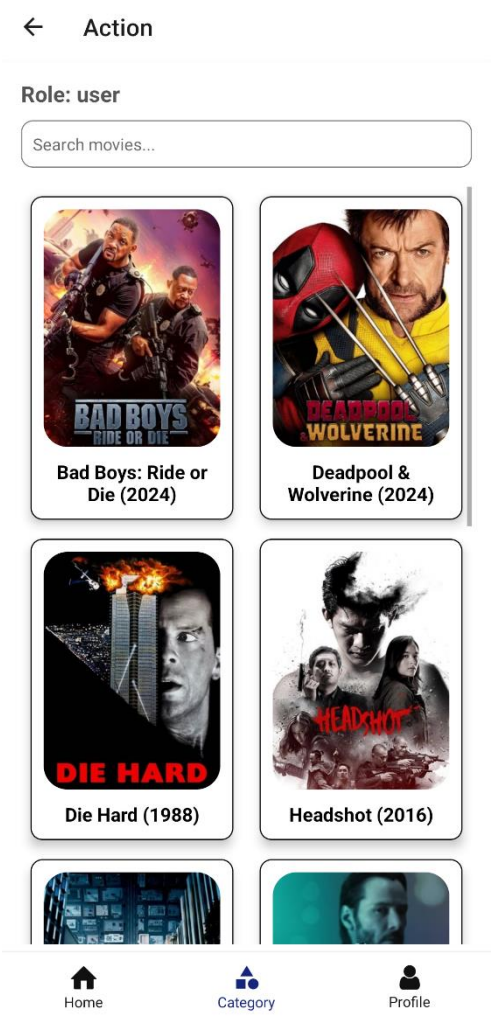


- **หน้า Category**

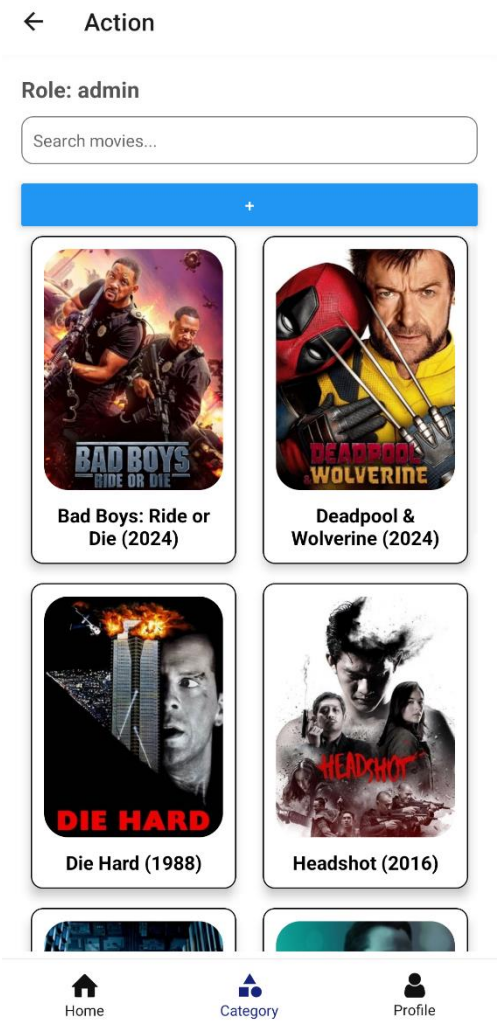
เป็นหน้าที่รวมหมวดหมู่ของหนัง เพื่ออำนวยความสะดวกให้ผู้ใช้สามารถเลือกตามหมวดหมู่ได้รวดเร็วยิ่งขึ้น



เมื่อผู้ใช้กดเข้าหมวดหมู่ในหมวดหมู่นึง จะแสดงให้เห็นถึงภาพยนต์ภายในหมวดหมู่นั้นๆ ตามฐานข้อมูลหลังบ้านที่ได้เรียบเรียงเอาไว้แล้ว



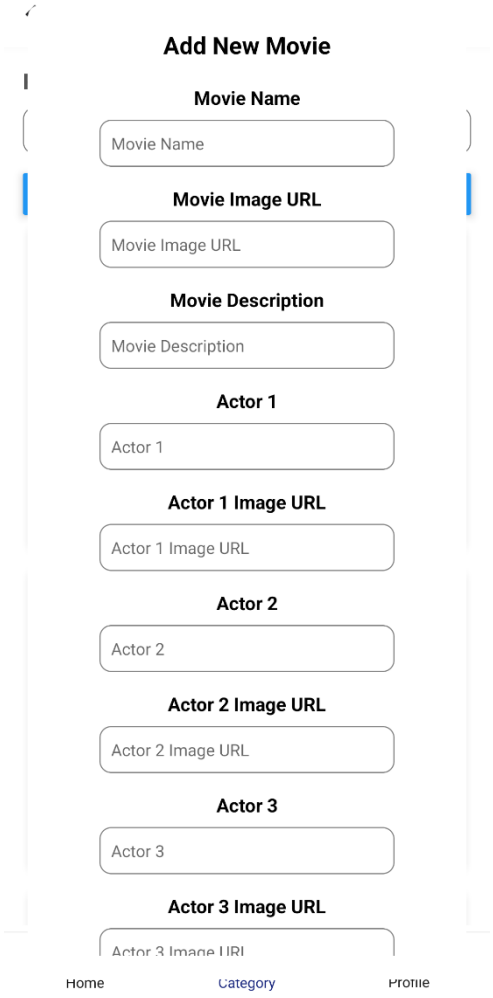
สำหรับ ผู้ใช้



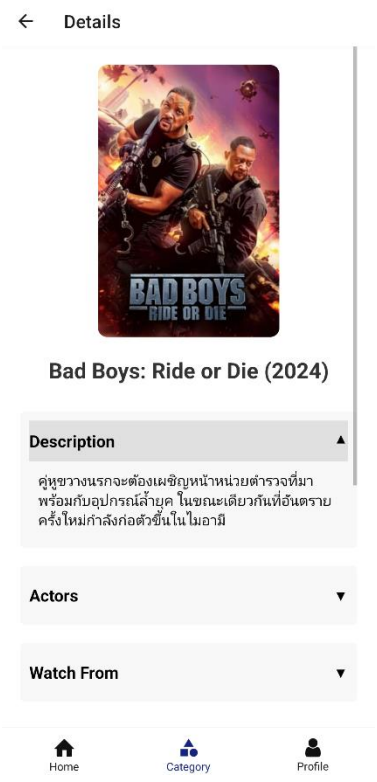
สำหรับ Admin

โดยในแต่ละหมวดหมู่ของ Category จะแยกให้สำหรับ ผู้ใช้ / Admin เพื่อจุดประสงค์การใช้งานที่ต่างกัน ในหน้านี้จะมีส่วนประกอบคือ Search Bar เอาไว้เพื่อค้นหาภาพยนต์ที่ผู้ใช้สนใจ มี Flat list โดยมีภาพยนต์ที่ฐานข้อมูลได้เก็บข้อมูลเอาไว้ในกรณีที่ผู้ใช้คิดไม่ออกว่าจะดูอะไรดี ในส่วนของ Admin จะมีปุ่มเพื่อเพิ่มภาพยนต์ลงไปใน Category นั้นได้เลยโดยตรง

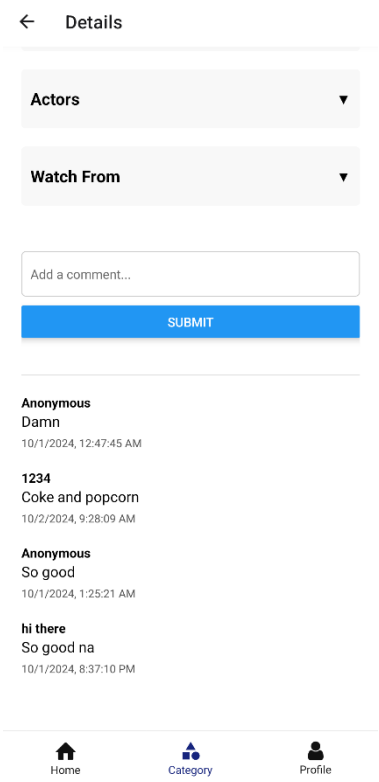
เมื่อ Admin กดปุ่ม (+) จะมี pop up ขึ้นมาเพื่อให้ Admin เพิ่มภาพยนต์ภายในหมวดหมู่นั้นๆ โดยจะรวบรวมข้อมูลที่อยู่ในภาพยนต์นั้นๆด้วย เช่น ชื่อภาพยนต์(Movie Name), รูปภาพของภาพยนต์(Movie Image), นักแสดง(Actor) เป็นต้น



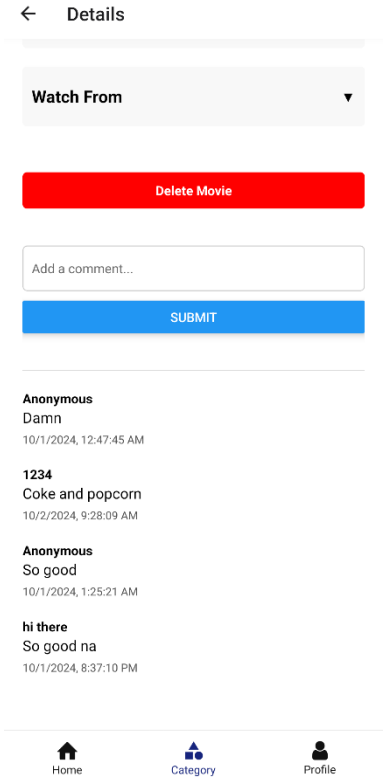
เมื่อผู้ใช้กดเข้ามาในภาพยนต์เรื่องใดเรื่องนึง จะแสดงให้เห็นเกี่ยวกับ ชื่อภาพยนต์(Movie Name), รูปภาพของภาพยนต์(Movie Image), เรื่องย่อ(Description), นักแสดง(Actor) และสามารถรับชมที่ใดได้บ้าง (Watch From) และยังมีส่วนของคอมเมนต์มีผู้ใช้สามารถแสดงความคิดเห็นของตัวเองต่อภาพยนต์เรื่องนี้ หรือสามารถอ่านความเห็นส่วนต่างของผู้รับชมคนอื่นได้



สำหรับ ผู้ใช้



สำหรับ ผู้ใช้

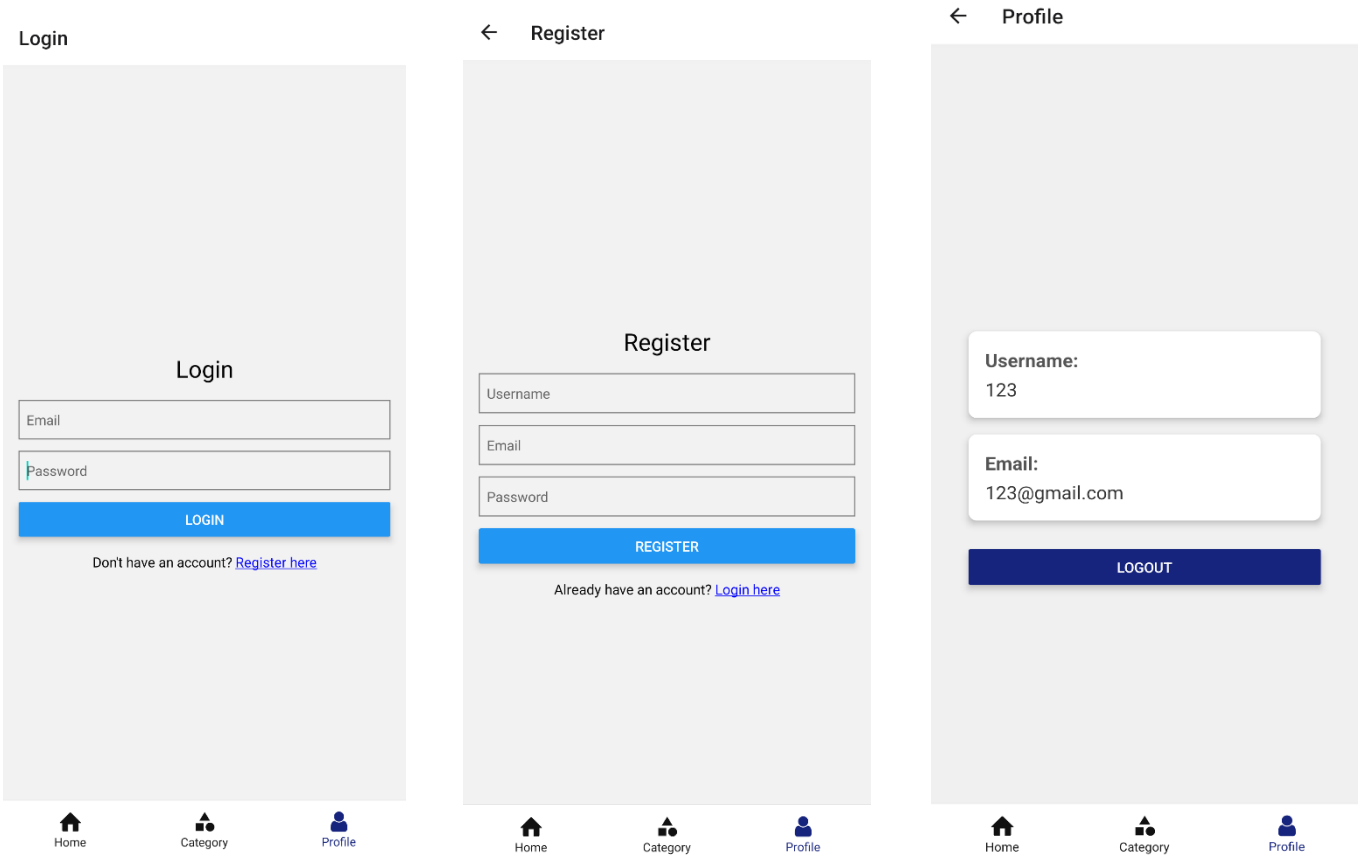


สำหรับ Admin

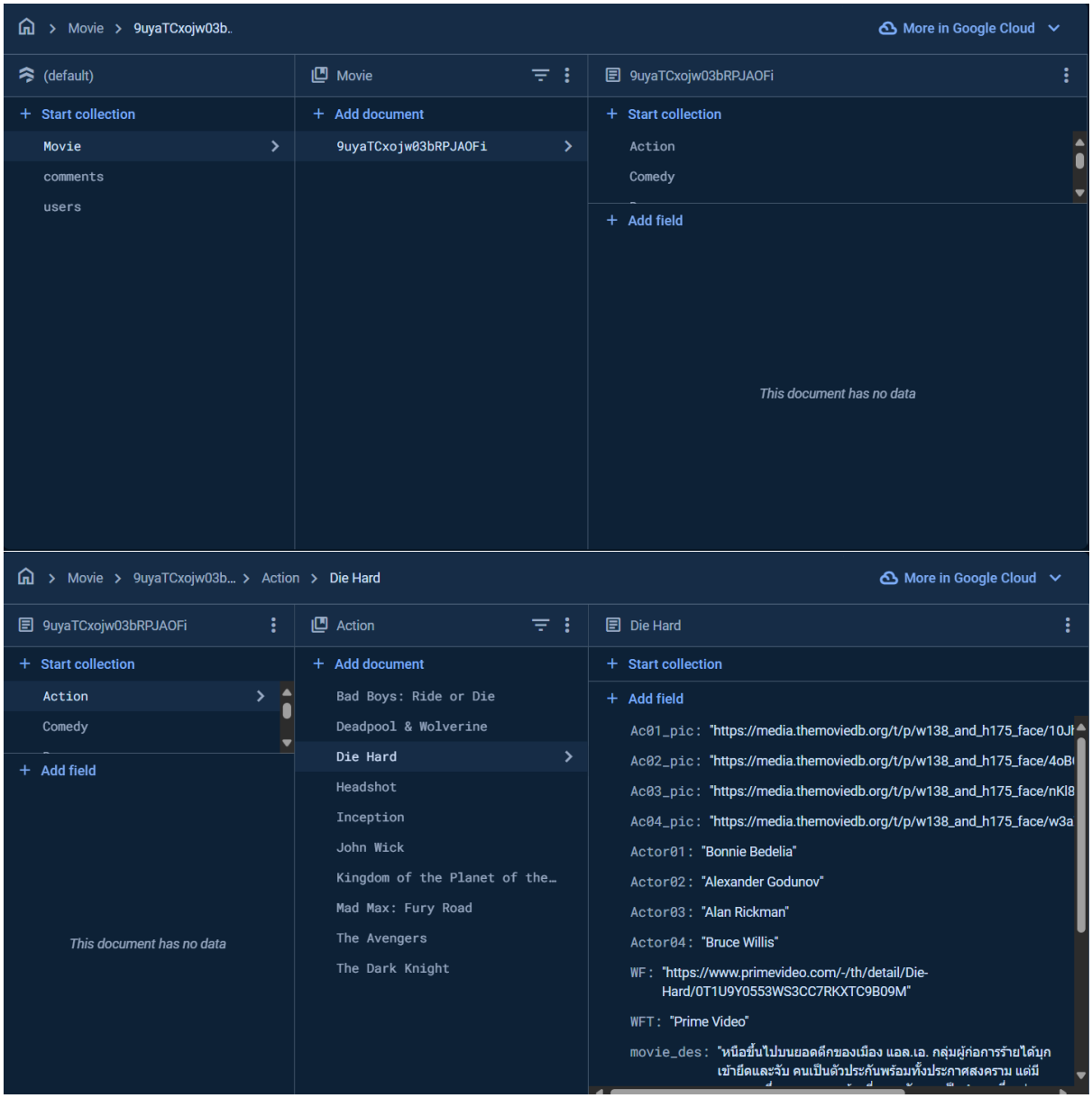
สำหรับส่วนของ Admin จะมีปุ่ม Delete เพื่อลบภาพยนต์เรื่องนั้นๆที่ไม่อยากให้แสดงอีกต่อไป ได้โดยตรงภายในภาพยนต์เรื่องนั้นได้เลย

- **หน้า Profile**

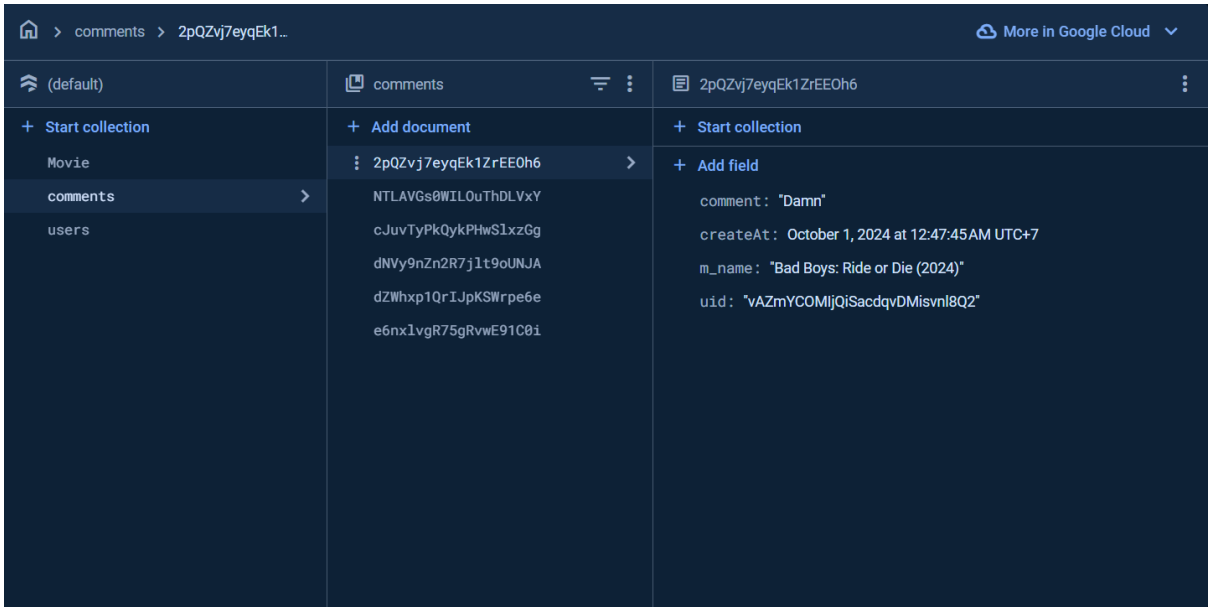
มีเพื่อลงชื่อเข้าใช้(Login) ในแอพพลิเคชั่นหากผู้ใช้มีบัญชีอยู่แล้ว หรือหากยังไม่มีก็สามารถลงทะเบียน (Register) ได้เลย เมื่อเข้าสู่ระบบได้ก็จะสามารถเข้าใช้ฟังชั่นที่แตกต่างกันออกไปภายในแอพพลิเคชั่น



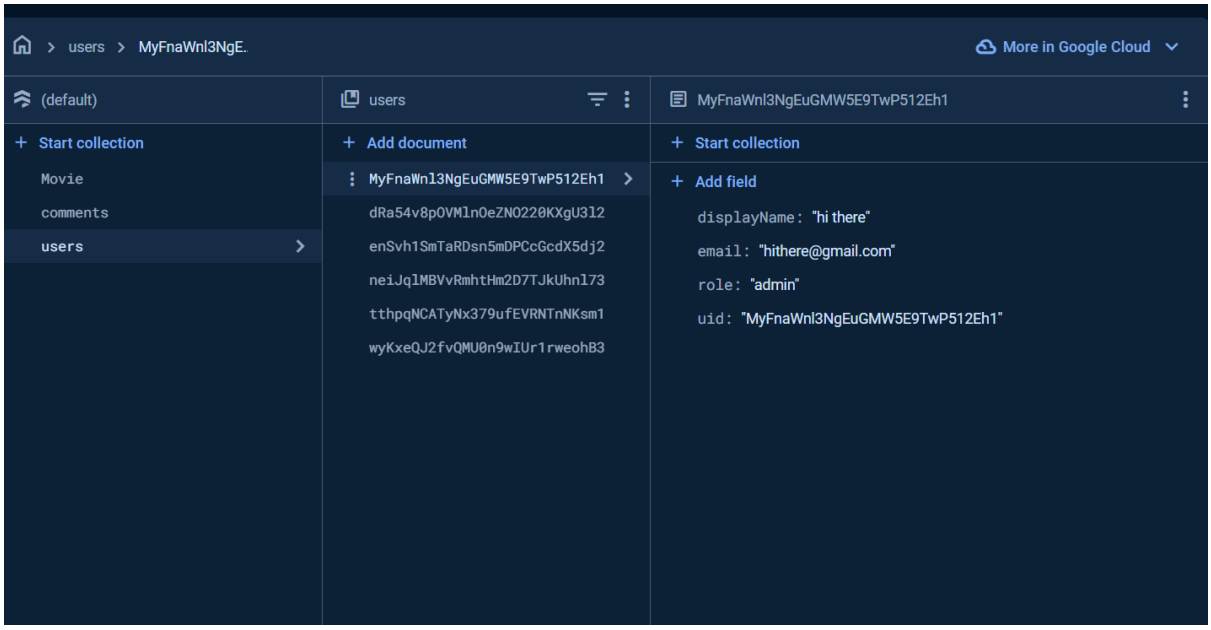
โครงสร้าง Database(FireBase)



มี main collection เป็น Movie และ Subcollection เป็น Category ต่างๆภายใน Category จะเก็บข้อมูลหนังเอาไว้ ตามที่ต้องการเช่น โปสเตอร์หนัง เรื่องย่อ นักแสดง เป็นต้น



มีการเก็บข้อมูล Comment ของแต่ละ User โดยมีการตรวจสอบความสอดคล้องด้วย uid



เมื่อ User มีการ Register ระบบจะบันทึกข้อมูล Gmail , Displayname , role และ uid

อธิบายการติดตั้งระบบ

ทรัพยากรที่ต้องใช้

- Node.js

สามารถคัดลอกโปรเจคได้จาก GitHub

- git clone https://github.com/HEALINGX/DooRaiD.git
- cd DooRaiD (ไปยังไดเรกทอรีของโปรเจ็กต์)
- npm install (ติดตั้งไลบรารีหรือโมดูลที่จำเป็นสำหรับโปรเจ็กต์)
- npm add expo (เพิ่มแพ็กเกจ expo ลงในโปรเจ็กต์)
- npx expo start (เริ่มต้นเซิร์ฟเวอร์ Expo สำหรับการพัฒนาแอปพลิเคชัน)

คำสั่งต่างภายในระบบ

- App.js

```
const LoginStack = () => (
  <Stack.Navigator screenOptions={{ headerShown: true }}>
    <Stack.Screen name="Login" component={Logins} />
    <Stack.Screen name="Register" component={Register} />
    <Stack.Screen name="Profile" component={Profile} />
  </Stack.Navigator>
);
```

```
const HomeStack = () => (
  <Stack.Navigator screenOptions={{ headerShown: true }}>
    <Stack.Screen
      name="DooRaiD"
      component={Home}
      options={{ tabBarStyle: { display: 'flex' } }} // แสดง tab bar ในหน้า Home
    />
    <Stack.Screen
      name="Details"
      component={Details}
      options={{ tabBarStyle: { display: 'none' } }} // ซ่อน tab bar ในหน้า Details
    />
  </Stack.Navigator>
);
```

```
const CategoryStack = () => (
  <Stack.Navigator screenOptions={{ headerShown: true }}>
    <Stack.Screen name="Category" component={Category} />
    <Stack.Screen
      name="CategoryDetails"
      component={CategoryDetails}
      options={({ route }) => ({
        title: route.params.category || 'Category Details',
        tabBarStyle: { display: 'none' },
      })}
    />
    <Stack.Screen
      name="Details"
      component={Details}
      options={{ tabBarStyle: { display: 'none' } }} // ซ่อน tab bar ในหน้า Details
    />
  </Stack.Navigator>
);
```

LoginStack: เป็น Stack Navigator สำหรับหน้าจอการเข้าสู่ระบบ
ซึ่งรวมถึงหน้าจอการลงทะเบียนและโปรไฟล์

HomeStack: เป็น Stack Navigator สำหรับหน้าจอ Home และ Details โดยจะแสดง tab bar ในหน้า Home และซ่อนในหน้า Details

CategoryStack: เป็น Stack Navigator สำหรับหน้าจอ Category, CategoryDetails และ Details โดยซ่อน tab bar ในหน้า CategoryDetails และ Details

```
export default function App() {
  return (
    <AuthProvider>
      <NavigationContainer>
        <Tab.Navigator screenOptions={screenOptions}>
          <Tab.Screen
            name="HomeTab"
            component={HomeStack}
            options={{
              tabBarIcon: ({ focused }) => (
                <View style={{ alignItems: "center", justifyContent: "center"}}>
                  <Entypo name="home" size={24} color={focused ? "#16247d" : "#111"} />
                  <Text style={{ fontSize: 12, color={focused ? "#16247d" : "#111" }}>
                    Home
                  </Text>
                </View>
              ),
            }}
          />
<Tab.Screen
  name="CategoryTab"
  component={CategoryStack}
  options={{
    tabBarIcon: ({ focused }) => (
      <View style={{ alignItems: "center", justifyContent: "center"}}>
        <MaterialIcons name="category" size={24} color={focused ? "#16247d" : "#111"} />
        <Text style={{ fontSize: 12, color={focused ? "#16247d" : "#111" }}>
          Category
        </Text>
      </View>
```

```
          <Tab.Screen
            name="ProfileTab"
            component={LoginStack}
            options={{
              tabBarIcon: ({ focused }) => (
                <View style={{ alignItems: "center", justifyContent: "center"}}>
                  <FontAwesome name="user" size={24} color={focused ? "#16247d" : "#111"} />
                  <Text style={{ fontSize: 12, color={focused ? "#16247d" : "#111" }}>
                    Profile
                  </Text>
                </View>
              ),
            }}
          />
        </Tab.Navigator>
      </NavigationContainer>
    </AuthProvider>
```

App Component: เป็น component หลักของแอปพลิเคชัน โดยใช้ AuthProvider เพื่อจัดการการรับรองตัวตน และ NavigationContainer เพื่อจัดการการนำทางหลัก โดยมี Tab Navigator ซึ่งประกอบด้วย HomeStack, CategoryStack, และ LoginStack

Tab Navigator: มีหน้าจอหลักสามหน้าคือ Home, Category, และ Profile แต่ละหน้าจะมีไอคอนและข้อความแสดงอยู่

- AuthContext.js

```
import { createContext, useContext, useEffect, useState } from "react";
import { auth } from "../Configs/Firebase";
import { createUserWithEmailAndPassword, signInWithEmailAndPassword, signOut } from "firebase/auth";
import { doc, setDoc } from "firebase/firestore";
import { db } from "../Configs/Firebase";
```

นำเข้าโมดูลต่างๆ:

createContext, useContext, useEffect, useState จาก react

auth และ db จาก ../Configs/Firebase

ฟังก์ชันจาก firebase/auth สำหรับการลงทะเบียน, เข้าสู่ระบบ, และออกจากระบบ

ฟังก์ชันจาก firebase/firestore สำหรับการจัดการเอกสารใน Firestore

```
const AuthContext = createContext(null);

export const useAuth = () => {
  return useContext(AuthContext);
}
```

สร้าง Context:

AuthContext สำหรับการรับรองตัวตน

useAuth เป็น hook ที่ช่วยให้สามารถใช้ context ได้ง่ายขึ้น

```
export const AuthProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const unsubscribe = auth.onAuthStateChanged(user => {
      setCurrentUser(user);
      setLoading(false);
    });

    return () => unsubscribe();
  }, []);
```

AuthProvider Component:

currentUser: สถานะของผู้ใช้ปัจจุบัน

loading: สถานะการโหลดข้อมูล

useEffect: ฟังก์ชันที่ใช้ในการตรวจสอบการเปลี่ยนแปลงสถานะการรับรองตัวตนของผู้ใช้ โดยเมื่อสถานะเปลี่ยนแปลงจะอัพเดตสถานะ currentUser และ loading

```
const signUpWithEmail = async (email, password, displayName) => {
  try {
    const userCredential = await createUserWithEmailAndPassword(auth, email, password);
    const user = userCredential.user;
    const userData = {
      uid: user.uid,
      displayName: displayName,
      email: user.email,
      role: "user"
    };
    await setDoc(doc(db, 'users', user.uid), userData);
    console.log("ผู้ใช้ถูกลงทะเบียนด้วยชื่อแสดง:", displayName);
  } catch (error) {
    console.error("เกิดข้อผิดพลาดระหว่างการลงทะเบียนด้วยอีเมล:", error.message);
    throw new Error("เกิดข้อผิดพลาดระหว่างการลงทะเบียนด้วยอีเมล: " + error.message);
  }
}
```

signUpWithEmail Function:

ฟังก์ชันสำหรับการลงทะเบียนผู้ใช้ใหม่ด้วยอีเมลและรหัสผ่าน

สร้างเอกสารผู้ใช้ใน Firestore หลังจากลงทะเบียนสำเร็จ

```
const logout = async () => {
  try {
    await signOut(auth);
  } catch (error) {
    console.error("เกิดข้อผิดพลาดระหว่างการออกจากระบบ:", error.message);
  }
};
```

logout Function:

ฟังก์ชันสำหรับการออกจากระบบ

```
  const value = {
    currentUser,
    loading,
    signInWithEmail,
    signUpWithEmail,
    logout,
  }

  return (
    <AuthContext.Provider value={value}>
      {!loading && children}
    </AuthContext.Provider>
  );
}
```

ค่า Context:

value เป็นวัตถุที่รวมฟังก์ชันและสถานะที่เกี่ยวข้องกับการรับรองตัวตน

AuthContext.Provider ส่งผ่านค่า value ไปยัง component ลูก

เมื่อ loading เป็น false จะแสดง component ลูก

- Category.js

```
import React from 'react';
import { View, Text, FlatList, StyleSheet, TouchableOpacity } from 'react-native';

const categories = [
  { id: '1', name: 'Action', subcollection: 'Action', documentId: '9uyaTCxojw03bRPJAOFi' },
  { id: '2', name: 'Comedy', subcollection: 'Comedy', documentId: '9uyaTCxojw03bRPJAOFi' },
  { id: '3', name: 'Drama', subcollection: 'Drama', documentId: '9uyaTCxojw03bRPJAOFi' },
  { id: '4', name: 'Horror', subcollection: 'Horror', documentId: '9uyaTCxojw03bRPJAOFi' },
  { id: '5', name: 'Romance', subcollection: 'Romance', documentId: '9uyaTCxojw03bRPJAOFi'
  // เพิ่มหมวดหมู่อื่น ๆ ตามที่ต้องการ
];
```

ข้อมูลหมวดหมู่:

categories เป็นอาเรย์ที่เก็บข้อมูลหมวดหมู่ต่างๆ ที่จะถูกแสดงในหน้าจอ

```
export default function Category({ navigation }) {
  const renderItem = ({ item }) => (
    <TouchableOpacity
      style={styles.item}
      onPress={() => navigation.navigate('CategoryDetails', {
        subcollection: item.subcollection,
        documentId: item.documentId,
        category: item.name
      })}
    >
      <Text style={styles.title}>{item.name}</Text>
    </TouchableOpacity>
  );

  return (
    <View style={styles.container}>
      <FlatList
        data={categories}
        renderItem={renderItem}
        keyExtractor={(item) => item.id}
      />
    </View>
  );
}
```

Category Component:

- ฟังก์ชัน Category เป็นฟังก์ชันคอมโพเนนต์ที่รับ navigation เป็นพร็อพสำหรับการนำทาง
- renderItem เป็นฟังก์ชันที่ใช้ในการเรนเดอร์แต่ละรายการหมวดหมู่
  - ใช้ TouchableOpacity เพื่อให้ผู้ใช้สามารถแตะได้
  - เมื่อแตะที่รายการ จะนำทางไปยังหน้าจอ CategoryDetails พร้อมส่งพร็อพ subcollection, documentId, และ category
- FlatList ใช้สำหรับการแสดงรายการหมวดหมู่
  - data เป็นข้อมูลหมวดหมู่ที่จะแสดง
  - renderItem เป็นฟังก์ชันที่ใช้ในการเรนเดอร์แต่ละรายการ
  - keyExtractor ใช้ในการกำหนดคีย์สำหรับแต่ละรายการ

- **CategoryDetail.js**

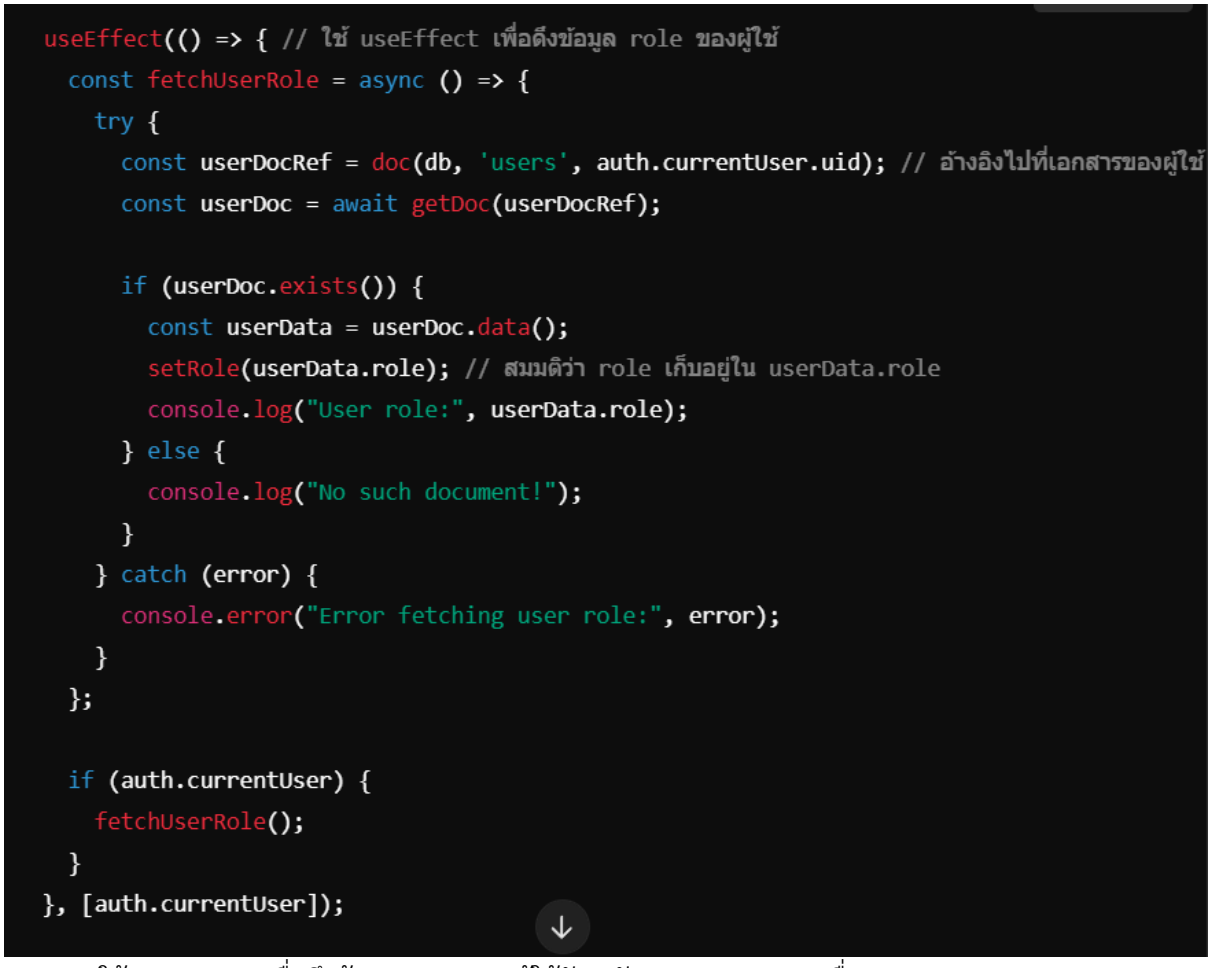
```
import React, { useEffect, useState } from 'react';
import { useFocusEffect } from '@react-navigation/native';

import { View, Text, FlatList, StyleSheet, Image, TouchableOpacity, TextInput, Button, Modal, ScrollView } from 'react-native';
import { db } from '../Configs/Firebase';
import { collection, getDocs, doc, getDoc, setDoc } from 'firebase/firestore';
import { useAuth } from '../context/AuthContext'; // Assuming you have a context for authentication
```

ส่วนนี้เป็นการนำเข้าฟังก์ชันและ component ต่าง ๆ ที่จะใช้ใน CategoryDetails component โดยใช้ React, React Native, Firebase Firestore และ context สำหรับการตรวจสอบตัวตน

```
export default function CategoryDetails({ route, navigation }) { // สร้าง component Category
  const { subcollection, documentId, category } = route.params; // ดึง params จาก route
  const [movies, setMovies] = useState([]); // สร้าง state สำหรับเก็บข้อมูลภาพยนตร์
  const [search, setSearch] = useState(''); // สร้าง state สำหรับเก็บข้อความค้นหา
  const [filteredMovies, setFilteredMovies] = useState([]); // สร้าง state สำหรับเก็บข้อมูลภาพย
  const [role, setRole] = useState(''); // สร้าง state สำหรับเก็บ role ของผู้ใช้
  const [modalVisible, setModalVisible] = useState(false); // สร้าง state สำหรับการแสดง modal
  const [newMovie, setNewMovie] = useState({ // สร้าง state สำหรับเก็บข้อมูลภาพยนตร์ใหม่
    movie_name: '',
    movie_image: '',
    movie_des: '',
    Actor01: '',
    Actor02: '',
    Actor03: '',
    Actor04: '',
    Ac01_pic: '',
    Ac02_pic: '',
    Ac03_pic: '',
    Ac04_pic: ''
  });
  const auth = useAuth(); // ใช้ context สำหรับการตรวจสอบตัวตนเพื่อดึงข้อมูลผู้ใช้ปัจจุบัน
```

ส่วนนี้ประกาศ component CategoryDetails และสร้าง state ต่าง ๆ ที่ใช้ภายใน component เช่น ข้อมูลภาพยนตร์ ข้อความค้นหา role ของผู้ใช้ และสถานะของ modal



```
useEffect(() => { // ใช้ useEffect เพื่อดึงข้อมูล role ของผู้ใช้
  const fetchUserRole = async () => {
    try {
      const userDocRef = doc(db, 'users', auth.currentUser.uid); // อ้างอิงไปที่เอกสารของผู้ใช้
      const userDoc = await getDoc(userDocRef);

      if (userDoc.exists()) {
        const userData = userDoc.data();
        setRole(userData.role); // สมมติว่า role เก็บอยู่ใน userData.role
        console.log("User role:", userData.role);
      } else {
        console.log("No such document!");
      }
    } catch (error) {
      console.error("Error fetching user role:", error);
    }
  };

  if (auth.currentUser) {
    fetchUserRole();
  }
}, [auth.currentUser]);
```

ใช้ useEffect เพื่อดึงข้อมูล role ของผู้ใช้ปัจจุบันจาก Firestore เมื่อ component ถูก mount หรือเมื่อ auth.currentUser เปลี่ยนแปลง

```
useEffect(() => {
  const fetchData = async () => {
    try {
      console.log("Fetching data for subcollection:", subcollection);
      const subcollectionRef = collection(db, 'Movie', documentId, subcollection);
      const querySnapshot = await getDocs(subcollectionRef);
      const docsData = querySnapshot.docs.map(doc => doc.data());
      setMovies(docsData);
      setFilteredMovies(docsData); // Initialize filteredMovies with all movies
    } catch (error) {
      console.error("Error fetching data: ", error);
    }
  };

  fetchData();
}, [subcollection, documentId]);

useEffect(() => {
  if (search) {
    const newData = movies.filter(item =>
      item.movie_name.toLowerCase().includes(search.toLowerCase())
    );
    setFilteredMovies(newData);
  } else {
    setFilteredMovies(movies);
  }
}, [search, movies]);
```

ใช้ useEffect เพื่อดึงข้อมูลภาพยนตร์จาก Firestore และกรองข้อมูลภาพยนตร์ตามข้อความค้นหา

```
const addMovie = async () => {
  try {
    const newMovieDocRef = doc(db, `Movie/${documentId}/${category}/${newMovie.movie_name}`);
    await setDoc(newMovieDocRef, newMovie);
    setMovies([...movies, newMovie]);
    setFilteredMovies([...movies, newMovie]);
    setModalVisible(false); // Close modal after adding

    // Reset form after adding the movie
    setNewMovie({
      movie_name: '',
      movie_image: '',
      movie_des: '',
      Actor01: '',
      Actor02: '',
      Actor03: '',
      Actor04: '',
      Ac01_pic: '',
      Ac02_pic: '',
      Ac03_pic: '',
      Ac04_pic: ''
    });
  } catch (error) {
    console.error("Error adding movie:", error);
  }
};
```

ฟังก์ชัน addMovie ใช้สำหรับเพิ่มภาพยนตร์ใหม่เข้าไปใน Firestore และอัปเดต state ของ movies และ filteredMovies

```
const renderItem = ({ item }) => ( // ฟังก์ชันสำหรับ render รายการภาพยนตร์
  <TouchableOpacity
    style={styles.item}
    onPress={() =>
      navigation.navigate('Details', {
        item,
        userRole: role,
        documentId,
        category,
        movie_name: item.movie_name // ส่งชื่อภาพยนตร์ที่นี่
      })
    }
  >
    <Image style={styles.image} source={{ uri: item.movie_image }} />
    <Text style={styles.title}>{item.movie_name}</Text>
  </TouchableOpacity>
);
```

ฟังก์ชัน renderItem ใช้สำหรับ render รายการภาพยนตร์แต่ละรายการใน FlatList โดยแต่ละรายการจะเป็น TouchableOpacity ที่นำไปสู่หน้า Details เมื่อถูกกด

- Detail.js

```
const WatchBar = ({ item, header, active, toggleContent }) => {
  const [contentVisible, setContentVisible] = useState(false);
  const opacity = useRef(new Animated.Value(0)).current;

  const handleToggle = () => {
    toggleContent();
    setContentVisible((prev) => !prev);

    Animated.timing(opacity, {
      toValue: contentVisible ? 0 : 1,
      duration: 150,
      useNativeDriver: true,
    }).start();
  };
```

WatchBar เป็นคอมโพเนนต์ที่แสดงข้อมูลเกี่ยวกับแหล่งที่สามารถรับชมภาพยนตร์ได้ มีการใช้ Animation เพื่อทำให้การแสดง/ซ่อนเนื้อหาดูนุ่มนวลขึ้น โดยใช้ Animated.View และ Animated.timing

- Home.js

```
const DetailBar = ({ text, header, active, toggleContent }) => {
  const [contentVisible, setContentVisible] = useState(true);
  const opacity = useRef(new Animated.Value(1)).current;

  const handleToggle = () => {
    toggleContent();
    setContentVisible((prev) => !prev);

    Animated.timing(opacity, {
      toValue: contentVisible ? 0 : 1,
      duration: 150,
      useNativeDriver: true,
    }).start();
  };
```

DetailBar ใช้สำหรับแสดงรายละเอียดของภาพยนตร์ มีการใช้ Animation เช่นเดียวกับ WatchBar

```
const ActorBar = ({ item, header, active, toggleContent }) => {
  const [contentVisible, setContentVisible] = useState(false);
  const opacity = useRef(new Animated.Value(0)).current;

  const handleToggle = () => {
    toggleContent();
    setContentVisible((prev) => !prev);

    Animated.timing(opacity, {
      toValue: contentVisible ? 0 : 1,
      duration: 150,
      useNativeDriver: true,
    }).start();
  };
```

ActorBar แสดงข้อมูลนักแสดงในภาพยนตร์ ใช้ Animation ในการแสดง/ซ่อนเนื้อหา และแสดงรูปภาพและชื่อของนักแสดงแต่ละคน

```
export default function Details({ route }) {
  const { item, userRole, documentId, category, movie_name } = route.params;
  const { currentUser } = useAuth();
  const navigation = useNavigation();
  const [comments, setComments] = useState([]);
  const [newComment, setNewComment] = useState('');
  const [activeDetail, setActiveDetail] = useState(true);
  const [activeActors, setActiveActors] = useState(false);
  const [activeWatch, setActiveWatch] = useState(false);

  useFocusEffect(
    React.useCallback(() => {
      const fetchData = async () => {
        // Fetch comments logic...
      };

      fetchData();
    }, [item.movie_name])
  );

 const handleAddComment = async () => {
   // Add comment logic...
 };

 const handleDeleteMovie = async () => {
   // Delete movie logic...
 };
```

Details เป็นคอมโพเนนต์หลักที่รวมทุกส่วนเข้าด้วยกัน มีการใช้ useFocusEffect เพื่อโหลดความคิดเห็นเมื่อหน้าจอถูกโฟกัส มีฟังก์ชันสำหรับเพิ่มความคิดเห็นใหม่และลบภาพยนตร์ (สำหรับ admin) แสดงรูปภาพ ชื่อ และรายละเอียดของภาพยนตร์โดยใช้ WatchBar, DetailBar, และ ActorBar

- Logins.js

```
import React, { useState } from 'react';
import { View, TextInput, Button, Text, StyleSheet, Alert } from 'react-native';
import { useAuth } from '../context/AuthContext';
```

- **React**: นำเข้า React และ useState สำหรับจัดการสถานะในคอมโพเนนต์
- **React Native**: นำเข้าคอมโพเนนต์ต่างๆ เช่น View, TextInput, Button, Text, StyleSheet, และ Alert สำหรับการสร้าง UI
- **useAuth**: นำเข้าคอนเท็กซ์สำหรับการจัดการการยืนยันตัวตนของผู้ใช้

```
const Logins = ({ navigation }) => {
  const auth = useAuth();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
```

- **Logins**: เป็นฟังก์ชันคอมโพเนนต์ที่ใช้สำหรับหน้าล็อกอิน
- **navigation**: รับ props สำหรับการนำทางไปยังหน้าต่างๆ
- **auth**: ใช้ useAuth เพื่อเข้าถึงฟังก์ชันการล็อกอิน
- **useState**: ใช้เพื่อสร้างสถานะสำหรับอีเมลและรหัสผ่าน

```
const handleLogin = async () => {
  try {
    await auth.signInWithEmail(email, password);

    Alert.alert('Login Success', 'Welcome back!');
    navigation.navigate('Profile');
  } catch (error) {
    Alert.alert('Login Failed', error.message);
  }
};
```

- **handleLogin**: ฟังก์ชันที่เรียกใช้เมื่อผู้ใช้กดปุ่มล็อกอิน
- **auth.signInWithEmail**: ฟังก์ชันสำหรับล็อกอินโดยใช้อีเมลและรหัสผ่าน
- **Alert.alert**: แสดงกล่องข้อความสำหรับแสดงผลลัพธ์ล็อกอินสำเร็จหรือไม่สำเร็จ
- **navigation.navigate**: นำทางไปยังหน้าประวัติส่วนตัว (Profile) หากล็อกอินสำเร็จ

- Profile.js

```
import React, { useEffect, useState } from 'react';
import { View, Text, StyleSheet, ActivityIndicator, Button, Alert } from 'react-native';
import { useAuth } from '../context/AuthContext';
import { doc, getDoc } from "firebase/firestore";
import { db } from "../Configs/Firebase";
import { Image } from 'react-native-elements';
```

- **React**: นำเข้า React และ hooks useEffect และ useState สำหรับจัดการสถานะและทำงานกับ side effects
- **React Native**: นำเข้าคอมโพเนนต์ต่างๆ สำหรับการสร้าง UI เช่น View, Text, ActivityIndicator, Button, และ Alert
- **useAuth**: นำเข้าคอนเท็กซ์สำหรับการจัดการการยืนยันตัวตนของผู้ใช้
- **Firestore**: นำเข้าฟังก์ชัน doc และ getDoc สำหรับดึงข้อมูลจาก Firestore
- **db**: นำเข้าการตั้งค่า Firebase
- **Image**: นำเข้าคอมโพเนนต์สำหรับแสดงภาพจาก react-native-elements

```
const Profile = ({ navigation }) => {
    const { currentUser, logout } = useAuth();
    const [userData, setUserData] = useState(null);
    const [loading, setLoading] = useState(true);
```

- **Profile**: เป็นฟังก์ชันคอมโพเนนต์ที่ใช้สำหรับแสดงข้อมูลประวัติส่วนตัวของผู้ใช้
- **navigation**: รับ props สำหรับการนำทางไปยังหน้าต่างๆ
- **currentUser**: ดึงข้อมูลผู้ใช้ที่เข้าสู่ระบบจาก useAuth
- **logout**: ฟังก์ชันสำหรับออกจากระบบ
- **userData**: สถานะที่ใช้เก็บข้อมูลผู้ใช้
- **loading**: สถานะที่ใช้ระบุว่ากำลังโหลดข้อมูลอยู่หรือไม่

```
useEffect(() => {
    const fetchUserData = async () => {
        if (currentUser) {
            try {
                const docRef = doc(db, 'users', currentUser.uid);
                const docSnap = await getDoc(docRef);

                if (docSnap.exists()) {
                    setUserData(docSnap.data());
                } else {
                    console.log("No such document!");
                }
            } catch (error) {
                console.error("Error fetching user data: ", error);
            } finally {
                setLoading(false);
            }
        }
    };

    fetchUserData();
}, [currentUser]);
```

- **useEffect**: ใช้สำหรับดึงข้อมูลของผู้ใช้เมื่อคอมโพเนนต์ถูกสร้างขึ้นหรือเมื่อ currentUser เปลี่ยนแปลง
- **fetchUserData**: ฟังก์ชันภายในที่ใช้ดึงข้อมูลผู้ใช้จาก Firestore
- **docRef**: อ้างอิงถึงเอกสารของผู้ใช้ใน Firestore
- **getDoc**: ดึงเอกสารที่อ้างอิง
- **setUserData**: หากเอกสารมีอยู่ จะจัดเก็บข้อมูลใน userData
- **console.log**: แสดงข้อความหากไม่มีเอกสาร
- **setLoading(false)**: ปิดสถานะโหลดเมื่อเสร็จสิ้นการดึงข้อมูล

```
const handleLogout = async () => {
    try {
        await logout();
        navigation.navigate('Login'); // Navigate to login screen after logout
    } catch (error) {
        Alert.alert('Logout Failed', error.message);
    }
};
```

- **handleLogout**: ฟังก์ชันที่เรียกใช้เมื่อผู้ใช้กดปุ่มออกจากระบบ
- **logout**: ฟังก์ชันสำหรับออกจากระบบ
- **navigation.navigate**: นำทางไปยังหน้าล็อกอินเมื่อออกจากระบบสำเร็จ
- **Alert.alert**: แสดงกล่องข้อความหากการออกจากระบบไม่สำเร็จ

- Register.js

```
import React, { useState } from 'react';
import { View, TextInput, Button, Text, StyleSheet, Alert } from 'react-native';
import { useAuth } from '../context/AuthContext';
```

- **React**: นำเข้า React และ hook useState สำหรับจัดการสถานะภายในคอมโพเนนต์
- **React Native**: นำเข้าคอมโพเนนต์ต่างๆ สำหรับการสร้าง UI เช่น View, TextInput, Button, Text, StyleSheet, และ Alert
- **useAuth**: นำเข้าคอนเท็กซ์สำหรับการจัดการการยืนยันตัวตนของผู้ใช้

```
const Register = ({ navigation }) => {
  const auth = useAuth();
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [username, setUsername] = useState('');
```

- **Register**: ฟังก์ชันคอมโพเนนต์ที่ใช้สำหรับสร้างหน้าลงทะเบียน
- **navigation**: รับ props สำหรับการนำทางไปยังหน้าต่างๆ
- **auth**: ใช้เพื่อเข้าถึงฟังก์ชันการลงทะเบียน
- **email, password, username**: สถานะที่ใช้เก็บข้อมูลที่ผู้ใช้กรอก

```
const handleRegister = async () => {
  try {
    await auth.signUpWithEmail(email, password, username);
    Alert.alert('Registration Success', 'You can now log in!');
    navigation.navigate('Login');
  } catch (error) {
    Alert.alert('Registration Failed', error.message);
  }
};
```

handleRegister: ฟังก์ชันที่เรียกใช้เมื่อผู้ใช้กดปุ่มลงทะเบียน

- auth.signUpWithEmail: เรียกใช้ฟังก์ชันสำหรับการลงทะเบียนผู้ใช้ด้วยอีเมลและรหัสผ่าน พร้อมกับชื่อผู้ใช้
- Alert.alert: แสดงกล่องข้อความเมื่อการลงทะเบียนสำเร็จหรือไม่สำเร็จ
- navigation.navigate: นำทางไปยังหน้าล็อกอินเมื่อการลงทะเบียนสำเร็จ

- Firebase.js

```
import { initializeApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";
import { initializeAuth, getReactNativePersistence } from 'firebase/auth';
import ReactNativeAsyncStorage from '@react-native-async-storage/async-storage';
```

- initializeApp: ใช้ในการเริ่มต้นแอป Firebase
- getFirestore: ใช้ในการเข้าถึงบริการ Firestore ของ Firebase
- initializeAuth: ใช้ในการตั้งค่าการยืนยันตัวตนของ Firebase
- getReactNativePersistence: ใช้เพื่อกำหนดวิธีการจัดเก็บข้อมูลการยืนยันตัวตน
- ReactNativeAsyncStorage: ใช้สำหรับการจัดเก็บข้อมูลแบบไม่สัมพันธ์ (asynchronous storage) ใน React Native

```
const firebaseConfig = {
  apiKey: "AIzaSyCrX34GHOFGQsn4eJpd0CaZ8wkvKEtBZjo",
  authDomain: "dooraid.firebaseapp.com",
  projectId: "dooraid",
  storageBucket: "dooraid.appspot.com",
  messagingSenderId: "40111671558",
  appId: "1:40111671558:web:3a5adda6efa3e52869b687",
  measurementId: "G-BHHTX8LQY8"
};
```

firebaseConfig: วัตถุที่ใช้ในการกำหนดค่าการเชื่อมต่อกับ Firebase โดยประกอบไปด้วยข้อมูลสำคัญ เช่น apiKey, authDomain, projectId, storageBucket, messagingSenderId, appId, และ measurementId ซึ่งเป็นข้อมูลที่จำเป็นในการเชื่อมต่อกับโปรเจกต์ Firebase ของคุณ

```
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
const auth = initializeAuth(app, {
  persistence: getReactNativePersistence(ReactNativeAsyncStorage)
});
```

- initializeApp: เรียกใช้ฟังก์ชันเพื่อเริ่มต้น Firebase ด้วย firebaseConfig
- getFirestore: เรียกใช้ฟังก์ชันเพื่อเข้าถึงฐานข้อมูล Firestore
- initializeAuth: เรียกใช้ฟังก์ชันเพื่อเริ่มต้นบริการการยืนยันตัวตน โดยใช้ getReactNativePersistence เพื่อกำหนดให้ใช้ ReactNativeAsyncStorage เป็นวิธีการจัดเก็บข้อมูลการเข้าสู่ระบบ