**📂 Ollama UI - Project Structure, Explanation, and Design Choices**

**🛠 Hand-Off Guide for Developers**

This document provides:

* **Project file structure**
* **Explanation of each file**
* **How different parts of the project are interconnected**
* **Design choices for state management, UI, and installation logic**

**📂 Project File Structure**

📦 ollama\_ui

│

├── 📂 lib

│ ├── 📂 core

│ │ ├── 📂 services

│ │ │ ├── logging\_service.dart # Logs engine installation progress/errors

│ │ │

│ ├── 📂 features

│ │ ├── 📂 chat

│ │ │ ├── 📂 presentation

│ │ │ │ ├── chat\_screen.dart # Main chat UI, handles engine status and navigation

│ │ │

│ │ ├── 📂 engine\_management

│ │ │ ├── 📂 data

│ │ │ │ ├── engine\_repository.dart # Handles engine detection, installation, and startup

│ │ │ ├── 📂 domain

│ │ │ │ ├── engine\_provider.dart # State management using Riverpod

│ │ │ ├── 📂 presentation

│ │ │ │ ├── engine\_status\_screen.dart # Displays current engine status, includes sidebar

│ │ │

│ │ ├── 📂 model\_management

│ │ │ ├── 📂 presentation

│ │ │ │ ├── model\_list\_screen.dart # Placeholder screen for managing AI models

│ │ │

│ ├── 📂 navigation

│ │ ├── routes.dart # Defines named routes for the app

│ │

│ ├── 📂 shared

│ │ ├── 📂 components

│ │ │ ├── sidebar.dart # Sidebar for navigation

│ │

│ ├── app.dart # Root widget, initializes engine check

│ ├── main.dart # Entry point, sets up ProviderScope

│

├── 📂 assets

│

└── 📄 pubspec.yaml # Dependencies and Flutter setup

**📜 Explanation of Key Files & Interconnections**

**🔹 1. main.dart - Entry Point**

**Purpose**:

* Initializes the app within **ProviderScope** (Riverpod).
* Loads **OllamaUI**, which contains **routes and UI navigation**.

import 'package:flutter/material.dart';

import 'package:flutter\_riverpod/flutter\_riverpod.dart';

import 'package:ollama\_ui/app.dart';

void main() {

WidgetsFlutterBinding.ensureInitialized();

runApp(

const ProviderScope(

child: OllamaUI(),

),

);

}

**🔹 2. app.dart - Root Widget**

**Purpose**:

* Sets up **routes**.
* Runs **checkAndStartEngine()** on app startup.
* Uses **routes.dart** to define screens.

import 'package:flutter/material.dart';

import 'package:flutter\_riverpod/flutter\_riverpod.dart';

import 'package:ollama\_ui/features/engine\_management/domain/engine\_provider.dart';

import 'package:ollama\_ui/navigation/routes.dart';

class OllamaUI extends ConsumerStatefulWidget {

const OllamaUI({super.key});

@override

ConsumerState<OllamaUI> createState() => \_OllamaUIState();

}

class \_OllamaUIState extends ConsumerState<OllamaUI> {

@override

void initState() {

super.initState();

Future.microtask(() {

ref.read(engineProvider.notifier).checkAndStartEngine();

});

}

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Ollama UI',

theme: ThemeData(primarySwatch: Colors.blue),

routes: appRoutes,

initialRoute: '/',

debugShowCheckedModeBanner: false,

);

}

}

**🔹 3. routes.dart - Navigation**

**Purpose**:

* Defines **named routes** for different screens.

import 'package:flutter/material.dart';

import 'package:ollama\_ui/features/chat/presentation/chat\_screen.dart';

import 'package:ollama\_ui/features/engine\_management/presentation/engine\_status\_screen.dart';

import 'package:ollama\_ui/features/model\_management/presentation/model\_list\_screen.dart';

final Map<String, WidgetBuilder> appRoutes = {

'/': (context) => const ChatScreen(),

'/engine': (context) => const EngineStatusScreen(),

'/models': (context) => const ModelListScreen(),

};

**🔹 4. chat\_screen.dart - Main UI**

**Purpose**:

* **Checks engine state** and provides a **button to install/start Ollama**.
* **Displays chat UI** (future feature).
* **Includes a sidebar** for navigation.

import 'package:flutter/material.dart';

import 'package:flutter\_riverpod/flutter\_riverpod.dart';

import 'package:ollama\_ui/features/engine\_management/domain/engine\_provider.dart';

import 'package:ollama\_ui/shared/components/sidebar.dart';

class ChatScreen extends ConsumerWidget {

const ChatScreen({super.key});

@override

Widget build(BuildContext context, WidgetRef ref) {

final engineStatus = ref.watch(engineProvider);

final engineNotifier = ref.read(engineProvider.notifier);

return Scaffold(

appBar: AppBar(title: const Text('Chat')),

drawer: const Sidebar(),

body: Column(

children: [

// Engine Status Display

Container(

color: Colors.grey[900],

padding: const EdgeInsets.all(8),

child: Row(

mainAxisAlignment: MainAxisAlignment.spaceBetween,

children: [

Text(

engineStatus.state == EngineState.online

? 'Engine Online'

: engineStatus.state == EngineState.downloading

? 'Engine Downloading'

: engineStatus.state == EngineState.installing

? 'Engine Installing'

: 'Engine Offline',

style: TextStyle(

fontSize: 16,

color: engineStatus.state == EngineState.online

? Colors.green

: Colors.red,

),

),

ElevatedButton(

onPressed: () {

if (engineStatus.state == EngineState.offline) {

engineNotifier.installAndStartEngine();

}

},

child: Text(

engineStatus.state == EngineState.offline

? 'Install Ollama'

: 'Switch Model',

),

),

],

),

),

],

),

);

}

}

**🔹 5. engine\_repository.dart - Manages Ollama Installation**

**Purpose**:

* **Checks if Ollama is installed**.
* **Handles downloading & installing Ollama**.
* **Starts Ollama** after installation.

Future<void> installEngine({

required void Function(EngineState newState) onStateChange,

void Function(int progress)? onProgress,

}) async {

// Mark as downloading

onStateChange(EngineState.downloading);

// Download logic here...

// After downloading, mark as installing

onStateChange(EngineState.installing);

// Run the installer...

}

**🛠 Design Choices**

**1️⃣ State Management**

✅ **Riverpod (StateNotifierProvider)** is used for state management, allowing real-time updates for UI components like:

* Chat UI (ChatScreen)
* Engine status (EngineStatusScreen)

**2️⃣ UI/Navigation**

✅ **MaterialApp(routes: appRoutes)** keeps navigation **clean** and **declarative**.  
✅ **A Sidebar (Drawer)** allows users to quickly navigate between screens.

**3️⃣ Installation Process**

✅ **State updates trigger UI changes**:

* "Engine Downloading" → "Engine Installing" → "Engine Online".  
  ✅ **Ensures Ollama is correctly installed** by using ollama --version checks.

**🎯 Summary**

* **State management**: Riverpod ensures **real-time updates**.
* **Navigation**: routes.dart keeps things **organized**.
* **Installation logic**: engine\_repository.dart handles **engine detection & installation**.

This project **efficiently manages Ollama installation** and **provides a clean UI for navigation**. 🚀

Let me know if you need any refinements before handoff!