Unit Topics Hours
1 Introduction to Flutter: Flutter Framework, Working of Flutter and Dart,
installation of Flutter SDK, Using Hot Reload and themes, External packages
import, project templates, organizing files and folders.
Introduction to Dart: Declaring and referencing variables, using operators, flow
control statements, using functions and classes, implementing asynchronous
Programming.
08
2 Widget : state full and state less widgets
Using common widgets: Scaffold, AppBar, SafeArea, Container, Text,
Buttons, column, row, using images and icons, decorators, form widgets,
Various Buttons,
User Inputs: Radio,Checkbox,Slider, Textediting Controller.
State management: Handling state and Set state, calculator app
10
3 App’s Navigation: Using Navigator, named navigator route, Hero widget,
Bottom Navigation Bar Bottom Map Bar, Tab Bar, Tab Bar View, Drawer
Saving data with local persistence: Reading and Writing files, working with
key- value pairs, Supporting multiple locales,
10
4 Building layouts: Scrolling Lists: Card widget, Using ListView and
ListTitle, GridView, Stack widget, Customizing CustomScrollView using
Slivers.
Http API Call: Working with Future object, working with streams, building,
widgets based on stream and futures, handling various types of data: JSON,
Sending HTTP request, JSON Encode Decode.

Firebase : Introduction to Firebase and cloud Firestore.

give code for all topics and
also give one whole code which contain all

Flutter All Topics Main

Done — I created a single-file Flutter example that covers every topic you listed (and small focused demos for each). It's saved as a code document named **Flutter_All_Topics_Main.dart** in the canvas.

What I included (brief):

- Dart basics and async usage.
- Stateless and Stateful widget examples.
- Common widgets: Scaffold, AppBar, SafeArea, Container, Text, Buttons, Column, Row, Image, Icons and decorated containers.
- Form widgets: TextField with TextEditingController, Radio, Checkbox, Slider and validation.
- Simple state management with `setState` (calculator and counter examples).
- Navigation: named routes, `Navigator.push` , `Hero` animation, `Drawer` , `BottomNavigationBar` , `TabBar` / `TabBarView` .
- Local persistence: `shared_preferences` example and file read/write using `path_provider` .
- Layouts: `ListView` , `Card` , `ListTile` , `GridView` , `Stack` , `CustomScrollView` with `SliverAppBar` and `SliverList` .
- HTTP API calls using `http` and `FutureBuilder` .
- Streams and `StreamBuilder` .
- Firebase Cloud Firestore: basic add + stream example (requires Firebase native setup).
- A combined single-file app that ties everything together with routes to individual demo pages.

How to run (short):

1. Create a new Flutter project: `flutter create my_app`
2. Replace `lib/main.dart` with the code I added in the canvas.
3. Add these dependencies to `pubspec.yaml` (under `dependencies`): `http` , `shared_preferences` , `path_provider` , `firebase_core` , `cloud_firestore` , and `flutter_localizations` (see exact versions in the file header comment).
4. Run `flutter pub get` .

5. If you want Firestore features, follow FlutterFire docs to configure Android/iOS (add `google-services.json / GoogleService-Info.plist`) and enable Firebase in your project.
6. Run on device/emulator with `flutter run` .

Files & communication explanation (how the pieces talk — useful for viva):

- In a real project you split the single file into many Dart files (pages, services, models). Widgets communicate by:
  - Passing data in constructors (e.g., `NavDetail(title: ..., index: ...)`).
  - Using named routes (`MaterialApp.routes`) or `Navigator.push` / `pushNamed` .
  - Global services (HTTP client, Firestore) are accessed as singletons or via dependency injection/provider.
  - Persistence layers (SharedPreferences or local files) expose async functions; UI calls them and updates state via `setState` or a state-management solution.
- The provided single-file app demonstrates those patterns; converting it to multiple files is straightforward (extract Widgets to separate dart files and import them).

If you want next steps, I can:

- Split the single file into a full project structure (`lib/screens/*` , `lib/services/*`) and create a ready `pubspec.yaml` .
- Remove Firebase parts and keep only offline + HTTP examples (if you prefer not to configure Firebase).
- Add `provider` or `riverpod` examples for more advanced state management.
- Create slides / a short viva script explaining each topic step-by-step.

Which follow-up would you like?

also give explaination of all code in detail

Below is a **complete, viva-ready, line-by-line style explanation** of the **entire code** I gave you.
I'll explain **what**, **why**, and **how it works**, exactly the way examiners expect.

You can **read this before demo + viva** and confidently explain everything.

◆ OVERALL IDEA OF THE PROJECT

This is a **single Flutter application** that demonstrates **all syllabus topics**:

- Dart basics & async programming
- Stateless & Stateful widgets
- UI widgets and layouts
- Forms and user inputs
- State management using `setState`
- Navigation (routes, Hero, tabs, drawer, bottom bar)
- Local storage (SharedPreferences + File)
- HTTP API calls (Future & Stream)
- Firebase Cloud Firestore

Instead of many small apps, **everything is integrated into one app**, which is **best for demonstration and viva**.

◆ 1. `main()` FUNCTION & APP ENTRY POINT

dart

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}
```

Copy code

Explanation:

- `main()` is the **starting point** of any Flutter app.
- `WidgetsFlutterBinding.ensureInitialized()`
  → Required before using **async services** like Firebase or file system.
- `Firebase.initializeApp()`
  → Initializes Firebase services.
- `runApp(MyApp())`
  → Starts the Flutter UI and loads the root widget.

📌 **Viva point**:

Flutter apps start from `main()` and UI rendering starts with `runApp()` .

🔹 2. `MyApp` – ROOT WIDGET

dart

```dart
class MyApp extends StatelessWidget
```

Copy code

Why StatelessWidget?

- `MyApp` does **not change** during runtime.
- It only configures:
  - Theme
  - Routes
  - Localization

MaterialApp Explanation

dart

```dart
MaterialApp(
  title: 'Flutter Topics Demo',
  theme: ThemeData(primarySwatch: Colors.blue),
  routes: {...},
)
```

Copy code

Property	Purpose
`title`	App name
`theme`	Global UI theme
`routes`	Named navigation routes
`supportedLocales`	Multi-language support

📌 **Viva**:

`MaterialApp` is the **root of widget tree** and controls routing, theming, and localization.

🔹 3. HOME PAGE – Stateful Widget

dart

```dart
class HomePage extends StatefulWidget
```

Copy code

Why Stateful?

- Uses **BottomNavigationBar**
- Changes page when user taps tabs
- Needs dynamic UI updates → `setState()`

🔷 Drawer Navigation

dart

```dart
Drawer(
  child: ListView(
    children: [
      ListTile(
        title: Text('Widgets'),
        onTap: () => Navigator.pushNamed(context, '/widgets'),
      ),
    ],
  ),
)
```

Copy code

Explanation:

- Drawer gives **side navigation**
- `Navigator.pushNamed()` loads screens using **named routes**

📌 **Viva**:

Named routes improve **code readability and maintainability**.

🔷 Bottom Navigation Bar

dart

```dart
BottomNavigationBar(
  currentIndex: _currentIndex,
  onTap: (i) => setState(() => _currentIndex = i),
)
```

Copy code

Explanation:

- `_currentIndex` controls which page is visible
- `setState()` rebuilds UI with new page

📌 **State management using** `setState`.

🔹 4. STATELESS vs STATEFUL WIDGET

Stateless Example

dart

```dart
class SimpleStatelessBox extends StatelessWidget
```

Copy code

- UI never changes
- No variables
- Used for static content

📌 **Viva:**

Stateless widgets are **lightweight and faster**.

Stateful Example

dart

```dart
class SimpleStatefulCounter extends StatefulWidget
```

Copy code

dart

```dart
int _count = 0;
setState(() => _count++);
```

Copy code

Explanation:

- `_count` is stored in memory
- `setState()` tells Flutter to **rebuild UI**

📌 **Key concept**:

> Flutter UI is **reactive**, not imperative.

🔹 5. COMMON WIDGETS

Widget	Purpose
`Scaffold`	Basic screen structure
`AppBar`	Top bar
`SafeArea`	Avoids notch/edges
`Container`	Styling + layout
`Row` / `Column`	Horizontal / Vertical layout
`Image.network()`	Load image
`Icon()`	Icons
`Decoration`	Styling

◆ 6. FORM & USER INPUTS

dart

```
TextEditingController _nameController
```

Copy code

Why Controller?

- Reads text from input field
- Required for validation and submission

Inputs Covered

Widget	Purpose
`TextFormField`	Text input
`Radio`	Single selection
`Checkbox`	Boolean input
`Slider`	Range input

dart

```
Form(
  key: _formKey,
)
```

📌 **Viva:**

Copy code

Forms use `GlobalKey<FormState>` for **validation and submission**.

🔹 7. CALCULATOR – STATE MANAGEMENT

dart

```
void _calculate(String op)
```

Copy code

Explanation:

- Reads input values
- Performs arithmetic
- Updates result using `setState`

📌 **Viva:**

This demonstrates **local state management** without external libraries.

🔹 8. NAVIGATION SYSTEM

Navigator Push

dart

```
Navigator.push(context, MaterialPageRoute(...))
```

Copy code

- Pushes a new screen on stack

Named Routes

dart

```
Navigator.pushNamed(context, '/calculator')
```

Copy code

- Decouples UI from navigation logic

Hero Animation

```dart
Hero(tag: 'logoHero', child: FlutterLogo())
```

Explanation:

- Smooth animation between screens
- Same `tag` connects widgets

📌 **Viva favorite question** ✔

Tabs

```dart
DefaultTabController
TabBar
TabBarView
```

Used for **swipeable navigation**.

◆ 9. LAYOUTS

ListView

```dart
ListView.builder()
```

- Efficient scrolling
- Lazy loading

Card + ListTile

- Material Design structure
- Clean UI

GridView

dart

```
GridView.count(crossAxisCount: 3)
```

Copy code

- Used for image grids, dashboards

Stack

dart

```
Stack(
  children: [...]
)
```

Copy code

- Overlapping widgets
- Used for badges, overlays

Slivers (Advanced)

dart

```
CustomScrollView
SliverAppBar
SliverList
```

Copy code

📌 **Viva:**

Slivers give **high-performance scrolling** for large UIs.

🔷 10. HTTP API CALL

dart

```
Future<List<Post>> fetchPosts()
```

Copy code

Explanation:

- Uses `http.get`
- Decodes JSON

- Returns Future data

FutureBuilder

dart

```
FutureBuilder(
  future: fetchPosts(),
)
```

Copy code

📌 **Why FutureBuilder?**

Because network calls are **asynchronous**.

◆ 11. STREAMS

dart

```
StreamController<DateTime>
```

Copy code

- Emits continuous data
- Used for real-time updates

dart

```
StreamBuilder
```

Copy code

📌 **Viva difference:**

Future	Stream
One-time	Continuous
API call	Live updates

◆ 12. LOCAL PERSISTENCE

SharedPreferences

dart

```
prefs.setInt('counter', _counter);
```

Copy code

- Stores **key-value data**
- Data persists after app restart

File Read / Write

dart

```
File('${dir.path}/demo.txt')
```

Copy code

- Stores files in app directory
- Uses `path_provider`

📌 **Viva**:

SharedPreferences → small data

File → larger structured data

🔹 13. FIREBASE CLOUD FIRESTORE

dart

```
FirebaseFirestore.instance.collection('demo').add(...)
```

Copy code

Explanation:

- NoSQL cloud database
- Real-time sync
- Uses streams

dart

```
snapshots()
```

Copy code

📌 **Very important viva topic** ✔️

🔹 14. HOW FILES COMMUNICATE (IMPORTANT VIVA ANSWER)

In real projects:

Layer	**Communication**
UI Widgets	Constructor parameters
Screens	Navigator routes
Services	Singleton / Provider
Data	Async functions
Firebase	Streams

📌 In this demo → **all combined in one file** for simplicity.

✅ FINAL VIVA SUMMARY (YOU CAN SAY THIS)

> “This Flutter application demonstrates all syllabus concepts in a single integrated project. It covers Dart basics, widget lifecycle, UI components, form handling, state management using setState, navigation using routes and Hero animations, layout systems including slivers, asynchronous programming using Future and Stream, HTTP API integration, local data persistence, and Firebase Cloud Firestore for real-time data.”

If you want next:

- 🎤 **Viva questions & answers**
- 🧩 **Diagram explaining widget tree**
- 📝 **Short explanation notes for writing exam**
- 📁 **Multi-file project structure version**

Just tell me 👍

give code for login , sign up and after that it goes to some application like after login it goes to calculator , e-commerce app etc give one code for whole one scenario which conatin whole given syllabus

```
// Flutter_All_Topics_Full_Scenario.dart
// Single-file Flutter app demonstrating:
// - Signup & Login (using SharedPreferences)
// - Protected home flows (after login) leading to Calculator, E-commerce, and o
// - Stateless & Stateful widgets, forms, navigation (named routes + push), Hero
// - Drawer, BottomNavigationBar, TabBar, Slivers, ListView, GridView, Stack
// - Local persistence (SharedPreferences + File IO)
// - HTTP calls (FutureBuilder) and Streams (StreamBuilder)
// - Simple in-memory e-commerce flow (product list, product detail, cart)
// - Notes on Firebase: optional (commented) — you can re-enable if you configur

/*
Add to pubspec.yaml under dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
  shared_preferences: ^2.1.1
  path_provider: ^2.0.14
  http: ^0.13.6
  flutter_localizations:
    sdk: flutter
  # firebase_core: ^2.10.0
  # cloud_firestore: ^4.7.0

Place this file at lib/main.dart
Run: flutter pub get && flutter run
*/

import 'dart:async';
import 'dart:convert';
import 'dart:io';

import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:path_provider/path_provider.dart';
import 'package:http/http.dart' as http;
import 'package:flutter_localizations/flutter_localizations.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const MyApp());
}
```

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  static const supportedLocales = [Locale('en'), Locale('hi')];

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Full Syllabus App',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primarySwatch: Colors.teal),
      localizationsDelegates: const [
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      supportedLocales: supportedLocales,
      initialRoute: '/',
      routes: {
        '/': (_) => const AuthGate(),
        '/login': (_) => const LoginPage(),
        '/signup': (_) => const SignupPage(),
        '/home': (_) => const HomeShell(),
        '/calculator': (_) => const CalculatorPage(),
        '/ecom': (_) => const ECommercePage(),
        '/cart': (_) => const CartPage(),
        '/local': (_) => const LocalPersistencePage(),
        '/httpstream': (_) => const HttpAndStreamPage(),
      },
    );
  }
}

// --------------------------- AUTH GATE ---------------------------
class AuthGate extends StatefulWidget {
  const AuthGate({Key? key}) : super(key: key);

  @override
  State<AuthGate> createState() => _AuthGateState();
}

class _AuthGateState extends State<AuthGate> {
  bool _loading = true;
  bool _loggedIn = false;
  String _user = '';

  @override
  void initState() {
    super.initState();
    _checkAuth();
  }

  Future<void> _checkAuth() async {
    final prefs = await SharedPreferences.getInstance();
    final logged = prefs.getBool('loggedIn') ?? false;
    final user = prefs.getString('user') ?? '';
    await Future.delayed(const Duration(milliseconds: 400));
    setState(() {
      _loading = false;
      _loggedIn = logged;
      _user = user;
    });
    if (logged) {
      Navigator.of(context).pushReplacementNamed('/home');
    }
  }

  @override
  Widget build(BuildContext context) {
    if (_loading) return const Scaffold(body: Center(child: CircularProgressIndi
    return const LoginPage();
  }
}

// --------------------------- SIMPLE AUTH STORAGE ---------------------------
class AuthService {
  // For demo only: password stored as plain text (DO NOT use in production)
  static Future<bool> signup(String username, String password) async {
    final prefs = await SharedPreferences.getInstance();
    // store user as JSON map of username -> password
    final usersJson = prefs.getString('users') ?? '{}';
    final Map<String, dynamic> users = jsonDecode(usersJson);
    if (users.containsKey(username)) return false; // exists
    users[username] = password;
    await prefs.setString('users', jsonEncode(users));
    return true;
  }

  static Future<bool> login(String username, String password) async {
    final prefs = await SharedPreferences.getInstance();
    final usersJson = prefs.getString('users') ?? '{}';
    final Map<String, dynamic> users = jsonDecode(usersJson);
    if (!users.containsKey(username)) return false;
    if (users[username] != password) return false;
    await prefs.setBool('loggedIn', true);
    await prefs.setString('user', username);
    return true;
  }

  static Future<void> logout() async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setBool('loggedIn', false);
    await prefs.remove('user');
  }
}
```

```
// --------------------------- LOGIN PAGE ---------------------------
class LoginPage extends StatefulWidget {
  const LoginPage({Key? key}) : super(key: key);

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final _formKey = GlobalKey<FormState>();
  final _userCtrl = TextEditingController();
  final _passCtrl = TextEditingController();
  bool _loading = false;
  String? _error;

  @override
  void dispose() {
    _userCtrl.dispose();
    _passCtrl.dispose();
    super.dispose();
  }

  Future<void> _tryLogin() async {
    if (!_formKey.currentState!.validate()) return;
    setState(() {
      _loading = true;
      _error = null;
    });
    final ok = await AuthService.login(_userCtrl.text.trim(), _passCtrl.text);
    await Future.delayed(const Duration(milliseconds: 400));
    if (!ok) {
      setState(() {
        _loading = false;
        _error = 'Invalid credentials';
      });
      return;
    }
    Navigator.of(context).pushReplacementNamed('/home');
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Login')),
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Form(
            key: _formKey,
            child: Column(mainAxisAlignment: MainAxisAlignment.center, children:
              const Hero(tag: 'appLogo', child: FlutterLogo(size: 72)),
              const SizedBox(height: 20),
              TextFormField(controller: _userCtrl, decoration: const InputDecora
              const SizedBox(height: 12),
              TextFormField(controller: _passCtrl, decoration: const InputDecora
              const SizedBox(height: 12),
              if (_error != null) Text(_error!, style: const TextStyle(color: Co
              const SizedBox(height: 12),
              _loading ? const CircularProgressIndicator() : ElevatedButton(onPr
              TextButton(onPressed: () => Navigator.of(context).pushNamed('/sign
            ]),
          ),
        ),
      ),
    );
  }
}

// --------------------------- SIGNUP PAGE ---------------------------
class SignupPage extends StatefulWidget {
  const SignupPage({Key? key}) : super(key: key);

  @override
  State<SignupPage> createState() => _SignupPageState();
}

class _SignupPageState extends State<SignupPage> {
  final _formKey = GlobalKey<FormState>();
  final _userCtrl = TextEditingController();
  final _passCtrl = TextEditingController();
  final _pass2Ctrl = TextEditingController();
  bool _loading = false;
  String? _message;

  @override
  void dispose() {
    _userCtrl.dispose();
    _passCtrl.dispose();
    _pass2Ctrl.dispose();
    super.dispose();
  }

  Future<void> _trySignup() async {
    if (!_formKey.currentState!.validate()) return;
    if (_passCtrl.text != _pass2Ctrl.text) {
      setState(() => _message = 'Passwords do not match');
      return;
    }
    setState(() {
      _loading = true;
      _message = null;
    });
    final ok = await AuthService.signup(_userCtrl.text.trim(), _passCtrl.text);
    await Future.delayed(const Duration(milliseconds: 400));
    if (!ok) {
      setState(() {
        _loading = false;
        _message = 'Username already exists';
      });
      return;
    }
    // Auto-login after signup
    await AuthService.login(_userCtrl.text.trim(), _passCtrl.text);
    Navigator.of(context).pushReplacementNamed('/home');
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Sign up')),
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Form(
            key: _formKey,
            child: Column(mainAxisAlignment: MainAxisAlignment.center, children:
              const Hero(tag: 'appLogo', child: FlutterLogo(size: 72)),
              const SizedBox(height: 20),
              TextFormField(controller: _userCtrl, decoration: const InputDecora
              const SizedBox(height: 12),
              TextFormField(controller: _passCtrl, decoration: const InputDecora
              const SizedBox(height: 12),
              TextFormField(controller: _pass2Ctrl, decoration: const InputDecor
              const SizedBox(height: 12),
              if (_message != null) Text(_message!, style: const TextStyle(color
              const SizedBox(height: 12),
              _loading ? const CircularProgressIndicator() : ElevatedButton(onPr
            ]),
          ),
        ),
      ),
    );
  }
}

// --------------------------- HOME SHELL (AFTER LOGIN) ------------------------
class HomeShell extends StatefulWidget {
  const HomeShell({Key? key}) : super(key: key);

  @override
  State<HomeShell> createState() => _HomeShellState();
}

class _HomeShellState extends State<HomeShell> {
  int _currentIndex = 0;
  String _username = '';

  final pages = const [DashboardPage(), ECommercePage(), CalculatorPage(), HttpA
  @override
  void initState() {
    super.initState();
    _loadUser();
  }

  Future<void> _loadUser() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() => _username = prefs.getString('user') ?? '');
  }

  Future<void> _logout() async {
    await AuthService.logout();
    Navigator.of(context).pushReplacementNamed('/login');
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Syllabus App')),
      drawer: Drawer(
        child: ListView(padding: EdgeInsets.zero, children: [
          UserAccountsDrawerHeader(accountName: Text(_username), accountEmail: T
          ListTile(leading: const Icon(Icons.dashboard), title: const Text('Dash
          ListTile(leading: const Icon(Icons.shopping_cart), title: const Text('
          ListTile(leading: const Icon(Icons.calculate), title: const Text('Calc
          ListTile(leading: const Icon(Icons.cloud), title: const Text('HTTP & S
          const Divider(),
          ListTile(leading: const Icon(Icons.save), title: const Text('Local Per
          ListTile(leading: const Icon(Icons.logout), title: const Text('Logout'
        ]),
      ),
      body: SafeArea(child: pages[_currentIndex]),
      bottomNavigationBar: BottomNavigationBar(currentIndex: _currentIndex, onTa
        BottomNavigationBarItem(icon: Icon(Icons.dashboard), label: 'Home'),
        BottomNavigationBarItem(icon: Icon(Icons.shop), label: 'Shop'),
        BottomNavigationBarItem(icon: Icon(Icons.calculate), label: 'Calc'),
        BottomNavigationBarItem(icon: Icon(Icons.cloud), label: 'Network'),
      ]),
      floatingActionButton: FloatingActionButton(child: const Icon(Icons.shoppin
    );
  }
}

// --------------------------- DASHBOARD PAGE (SHOWS VARIOUS WIDGETS) ----------
class DashboardPage extends StatelessWidget {
  const DashboardPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return CustomScrollView(slivers: [
      const SliverAppBar(expandedHeight: 140, floating: false, pinned: true, fle
      SliverList(delegate: SliverChildListDelegate([
        Padding(padding: const EdgeInsets.all(12.0), child: Column(crossAxisAlig
          const Text('Quick Actions', style: TextStyle(fontSize: 18, fontWeight:
          const SizedBox(height: 8),
          Row(children: [
            _ActionCard(icon: Icons.calculate, label: 'Calculator', onTap: () =>
            const SizedBox(width: 8),
            _ActionCard(icon: Icons.shop, label: 'Shop', onTap: () => Navigator.
            const SizedBox(width: 8),
            _ActionCard(icon: Icons.save, label: 'Persistence', onTap: () => Nav
          ]),
          const SizedBox(height: 12),
          const Text('Features Demo', style: TextStyle(fontSize: 18, fontWeight:
          const SizedBox(height: 8),
          Card(child: ListTile(leading: const Icon(Icons.input), title: const Te
          const SizedBox(height: 8),
          Card(child: ListTile(leading: const Icon(Icons.view_agenda), title: co
        ]))
      ]) )
    ]);
  }
}

class _ActionCard extends StatelessWidget {
  final IconData icon;
  final String label;
  final VoidCallback onTap;
  const _ActionCard({required this.icon, required this.label, required this.onTa

  @override
  Widget build(BuildContext context) {
    return Expanded(child: InkWell(onTap: onTap, child: Container(padding: const
  }
}

// --------------------------- FORMS PAGE (reused) ---------------------------
class FormsDemoPage extends StatefulWidget {
  const FormsDemoPage({Key? key}) : super(key: key);

  @override
  State<FormsDemoPage> createState() => _FormsDemoPageState();
}

class _FormsDemoPageState extends State<FormsDemoPage> {
  final _formKey = GlobalKey<FormState>();
  final _name = TextEditingController();
  bool _agree = false;
  double _slider = 0.3;
  int _radio = 0;

  @override
  void dispose() {
    _name.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: const Text('Forms Demo')), body: Paddi
      TextFormField(controller: _name, decoration: const InputDecoration(labelTe
      const SizedBox(height: 12),
      Row(children: [const Text('Gender:'), Radio<int>(value: 0, groupValue: _ra
      Row(children: [Checkbox(value: _agree, onChanged: (v) => setState(() => _a
      const SizedBox(height: 12),
      Slider(value: _slider, onChanged: (v) => setState(() => _slider = v)),
      const SizedBox(height: 12),
      ElevatedButton(onPressed: () { if (_formKey.currentState!.validate()) { sh
Agree: \$_agree
Slider: \${_slider.toStringAsFixed(2)}'))); } }, child: const Text('Submit'))
    ]))));
  }
}

// --------------------------- LAYOUTS PAGE ---------------------------
class LayoutsDemoPage extends StatelessWidget {
  const LayoutsDemoPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: const Text('Layouts Demo')), body: Cus
      const SliverToBoxAdapter(child: SizedBox(height: 12)),
      SliverPadding(padding: const EdgeInsets.symmetric(horizontal: 12), sliver:
        const Text('Cards List'), SizedBox(height: 200, child: ListView.builder(
        const SizedBox(height: 12),
        const Text('Grid'), SizedBox(height: 150, child: GridView.count(crossAxi
        const SizedBox(height: 12),
        const Text('Stack example'), SizedBox(height: 120, child: Stack(alignmen
      ])))
    ]));
  }
}

// --------------------------- CALCULATOR ---------------------------
class CalculatorPage extends StatefulWidget {
  const CalculatorPage({Key? key}) : super(key: key);

  @override
  State<CalculatorPage> createState() => _CalculatorPageState();
}

class _CalculatorPageState extends State<CalculatorPage> {
  final _a = TextEditingController();
  final _b = TextEditingController();
  String _res = '';

  void _op(String o) {
    final x = double.tryParse(_a.text) ?? 0;
    final y = double.tryParse(_b.text) ?? 0;
    double r;
    switch (o) {
      case '+': r = x + y; break;
      case '-': r = x - y; break;
      case '*': r = x * y; break;
      case '/': r = y == 0 ? double.nan : x / y; break;
      default: r = 0;
    }
    setState(() => _res = r.isNaN ? 'Error' : r.toString());
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: const Text('Calculator')), body: Paddi
      TextField(controller: _a, decoration: const InputDecoration(labelText: 'A'
      TextField(controller: _b, decoration: const InputDecoration(labelText: 'B'
      const SizedBox(height: 12),
      Wrap(spacing: 8, children: [ElevatedButton(onPressed: () => _op('+'), chil
      const SizedBox(height: 12),
      Text('Result: \$_res')
    ])));
  }
}

// --------------------------- E-COMMERCE MINI APP ---------------------------
class Product { final int id; final String title; final String desc; final doubl

class ECommercePage extends StatefulWidget {
  const ECommercePage({Key? key}) : super(key: key);

  @override
  State<ECommercePage> createState() => _ECommercePageState();
}

class _ECommercePageState extends State<ECommercePage> {
  // sample in-memory products
  final products = List.generate(8, (i) => Product(id: i, title: 'Product \$i',

  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: const Text('Shop')), body: Padding(pad
  }
}

class ProductDetail extends StatelessWidget {
  final Product product;
  const ProductDetail({required this.product, Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: Text(product.title)), body: Column(chi
  }
```

```
}

// --------------------------- CART (local simple storage) ---------------------
class CartService {
  static const _key = 'cart_items';

  static Future<void> addToCart(Product p) async {
    final prefs = await SharedPreferences.getInstance();
    final s = prefs.getString(_key) ?? '[]';
    final List j = jsonDecode(s);
    j.add({'id': p.id, 'title': p.title, 'price': p.price, 'img': p.img});
    await prefs.setString(_key, jsonEncode(j));
  }

  static Future<List<Map<String, dynamic>>> getCart() async {
    final prefs = await SharedPreferences.getInstance();
    final s = prefs.getString(_key) ?? '[]';
    final List j = jsonDecode(s);
    return j.cast<Map<String, dynamic>>();
  }

  static Future<void> clearCart() async { final prefs = await SharedPreferences.
}

class CartPage extends StatefulWidget {
  const CartPage({Key? key}) : super(key: key);

  @override
  State<CartPage> createState() => _CartPageState();
}

class _CartPageState extends State<CartPage> {
  List<Map<String, dynamic>> items = [];
  bool _loading = true;

  @override
  void initState() {
    super.initState();
    _load();
  }

  Future<void> _load() async {
    final s = await CartService.getCart();
    setState(() { items = s; _loading = false; });
  }

  Future<void> _checkout() async {
    // Demo checkout: just clear
    await CartService.clearCart();
    await _load();
    ScaffoldMessenger.of(context).showSnackBar(const SnackBar(content: Text('Ord
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: const Text('Cart')), body: _loading ?
  }
}

// --------------------------- LOCAL PERSISTENCE (FILE + SharedPreferences) ----
class LocalPersistencePage extends StatefulWidget {
  const LocalPersistencePage({Key? key}) : super(key: key);

  @override
  State<LocalPersistencePage> createState() => _LocalPersistencePageState();
}

class _LocalPersistencePageState extends State<LocalPersistencePage> {
  int _count = 0;
  String _fileContent = '';

  @override
  void initState() {
    super.initState();
    _loadCounter();
    _readFile();
  }

  Future<void> _loadCounter() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() => _count = prefs.getInt('counter_demo') ?? 0);
  }

  Future<void> _increment() async {
    final prefs = await SharedPreferences.getInstance();
    setState(() { _count++; prefs.setInt('counter_demo', _count); });
  }

  Future<void> _writeFile() async {
    final dir = await getApplicationDocumentsDirectory();
    final file = File('${dir.path}/demo_app.txt');
    await file.writeAsString('Saved at \${DateTime.now()} - count: \$_count');
    await _readFile();
  }

  Future<void> _readFile() async {
    try {
      final dir = await getApplicationDocumentsDirectory();
      final file = File('${dir.path}/demo_app.txt');
      final exists = await file.exists();
      if (!exists) return setState(() => _fileContent = 'No file yet');
      final c = await file.readAsString();
      setState(() => _fileContent = c);
    } catch (e) {
      setState(() => _fileContent = 'Error: \$e');
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: const Text('Persistence')), body: Padd
  }
}

// --------------------------- HTTP & STREAMS DEMO ---------------------------
class HttpAndStreamPage extends StatefulWidget {
  const HttpAndStreamPage({Key? key}) : super(key: key);

  @override
  State<HttpAndStreamPage> createState() => _HttpAndStreamPageState();
}

class _HttpAndStreamPageState extends State<HttpAndStreamPage> {
  late Future<List<Post>> _futurePosts;
  final StreamController<DateTime> _clock = StreamController<DateTime>();

  @override
  void initState() {
    super.initState();
    _futurePosts = fetchPosts();
    Timer.periodic(const Duration(seconds: 1), (_) => _clock.add(DateTime.now())
  }

  @override
  void dispose() {
    _clock.close();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(appBar: AppBar(title: const Text('HTTP & Streams')), body: P
      const Text('Stream (live clock)'),
      StreamBuilder<DateTime>(stream: _clock.stream, builder: (_, snap) { if (!s
      const SizedBox(height: 12),
      const Text('FutureBuilder: Posts (sample HTTP)'),
      FutureBuilder<List<Post>>(future: _futurePosts, builder: (_, snap) { if (s
    ])));
  }
}

class Post { final int id; final String title; final String body; Post({required

Future<List<Post>> fetchPosts() async {
  try {
    final res = await http.get(Uri.parse('https://jsonplaceholder.typicode.com/p
    if (res.statusCode == 200) {
      final List j = jsonDecode(res.body);
      return j.map((e) => Post.fromJson(e)).toList();
    } else {
      throw Exception('Failed: \${res.statusCode}');
    }
  } catch (e) {
    if (kDebugMode) print('HTTP Error: \$e');
    rethrow;
  }
}

/*
Notes:
- This single-file app ties all syllabus topics into one flow.
- Signup/Login uses SharedPreferences (simple demo). In production use proper ba
- E-commerce is an in-memory demo with cart persisted in SharedPreferences.
- HTTP calls use jsonplaceholder.typicode.com as sample data.
- Streams are shown via a clock StreamController.
- Layouts and widgets demos are included (forms, slivers, grid, stack, hero anim

If you want, I can:
- Split this into multiple files (`lib/screens/*`, `lib/services/*`) and provide
```

Done — I updated the canvas with a **single-file Flutter app** (`Flutter_All_Topics_Full_Scenario.dart`) that includes signup/login and, after login, a complete app flow (calculator, e-commerce, persistence, HTTP & streams, layouts, forms, navigation, etc.).

How to use it:

1. Open the updated document in the canvas (it's the code file I added/updated).
2. Copy the code into `lib/main.dart` of a new Flutter project.
3. Add the dependencies listed at the top of the file to `pubspec.yaml` and run `flutter pub get` .
4. Run with `flutter run` .

Want me to also:

- Split the code into a proper multi-file project structure and produce a ready `pubspec.yaml` ?
- Replace SharedPreferences auth with Firebase Authentication?
- Add Provider/riverpod state management examples?

Tell me which follow-up you prefer and I'll update the canvas accordingly.

Unit Topics Hours
1 Introduction to Flutter: Flutter Framework, Working of Flutter and Dart,
installation of Flutter SDK, Using Hot Reload and themes, External packages
import, project templates, organizing files and folders.
Introduction to Dart: Declaring and referencing variables, using operators, flow
control statements, using functions and classes, implementing asynchronous
Programming.
08
2 Widget : state full and state less widgets
Using common widgets: Scaffold, AppBar, SafeArea, Container, Text,
Buttons, column, row, using images and icons, decorators, form widgets,
Various Buttons,
User Inputs: Radio,Checkbox,Slider, Textediting Controller.
State management: Handling state and Set state, calculator app
10
3 App’s Navigation: Using Navigator, named navigator route, Hero widget,
Bottom Navigation Bar Bottom Map Bar, Tab Bar, Tab Bar View, Drawer
Saving data with local persistence: Reading and Writing files, working with
key- value pairs, Supporting multiple locales,
10
4 Building layouts: Scrolling Lists: Card widget, Using ListView and
ListTitle, GridView, Stack widget, Customizing CustomScrollView using
Slivers.
Http API Call: Working with Future object, working with streams, building,
widgets based on stream and futures, handling various types of data: JSON,
Sending HTTP request, JSON Encode Decode.
Firebase : Introduction to Firebase and cloud Firestore.

explain whole syllabus i have practical exam tomorrow so explain according to that with code

Below is a **practical-exam–oriented, viva-ready explanation of your COMPLETE SYLLABUS**, explained **unit-wise**, **with short, clear code snippets**, exactly how examiners expect.

I'll explain:

- **Concept → Why used → Small code**
- Focus on **what to say in practical + viva**
- No extra theory, only **exam-useful points**

✅ UNIT–1 (08 Hours)

🔹 1. Introduction to Flutter

What is Flutter?

Flutter is a **UI framework by Google** used to build **cross-platform apps** (Android, iOS, Web, Desktop) from **a single codebase** using **Dart**.

How Flutter Works (VERY IMPORTANT FOR VIVA)

- Flutter uses **Widgets** for everything
- Uses **Skia rendering engine**
- No XML (unlike Android)
- UI updates using **Hot Reload**

📌 **Viva Line**

> Flutter renders UI directly using widgets and does not depend on platform UI components.

🔹 Flutter SDK Installation (Say in Viva)

1. Download Flutter SDK

2. Add to PATH
3. `flutter doctor`
4. Create project: `flutter create project_name`

🔹 Hot Reload

- Updates UI instantly without restarting app
- Preserves state

📌 **Viva**

> Hot Reload helps in faster UI development and debugging.

🔹 External Packages

Add in `pubspec.yaml`:

yaml

```yaml
dependencies:
  http: ^0.13.6
```

Copy code

Import:

dart

```dart
import 'package:http/http.dart' as http;
```

Copy code

🔹 Project Structure (IMPORTANT)

css

```css
lib/
 ├─ main.dart
 ├─ screens/
 ├─ widgets/
 ├─ services/
```

📌 **Viva**

Copy code

Code is organized for readability and maintainability.

🔹 Introduction to Dart

Variables

dart

```dart
int a = 10;
double b = 5.5;
String name = "Flutter";
bool isOk = true;
```

Operators

Copy code

dart

```dart
int sum = a + 5;
bool result = a > 5 && b < 10;
```

Copy code

Control Statements

dart

```dart
if (a > 5) {
  print("Greater");
} else {
  print("Smaller");
}

for (int i = 0; i < 5; i++) {
  print(i);
}
```

Functions

Copy code

dart

```
int add(int x, int y) {
  return x + y;
}
```

Copy code

Classes

dart

```
class Student {
  String name;
  Student(this.name);
}
```

Copy code

Asynchronous Programming

dart

```
Future<String> fetchData() async {
  await Future.delayed(Duration(seconds: 2));
  return "Data Loaded";
}
```

Copy code

📌 **Viva**

Async programming prevents UI blocking.

✅ UNIT–2 (10 Hours)

🔹 Stateless vs Stateful Widgets

Stateless Widget

dart

```
class MyText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text("Hello");
    }
  }
```

- UI never changes

Copy code

Stateful Widget

dart

```
class Counter extends StatefulWidget {
  @override
  _CounterState createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int count = 0;

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text("$count"),
        ElevatedButton(
          onPressed: () {
            setState(() {
              count++;
            });
          },
          child: Text("Increment"),
        )
      ],
    );
  }
}
```

📌 Viva

`setState()` rebuilds UI when data changes.

🔷 Common Widgets

Scaffold

Copy code

dart

```dart
Scaffold(
  appBar: AppBar(title: Text("App")),
  body: Text("Hello"),
);
```

Row & Column

Copy code

dart

```dart
Row(children: [Text("A"), Text("B")]);
Column(children: [Text("X"), Text("Y")]);
```

Copy code

Buttons

dart

```dart
ElevatedButton(onPressed: () {}, child: Text("Click"));
TextButton(onPressed: () {}, child: Text("Text"));
OutlinedButton(onPressed: () {}, child: Text("Outline"));
```

Copy code

Image & Icon

dart

```dart
Image.network("https://picsum.photos/200");
Icon(Icons.home);
```

Copy code

◆ User Inputs

TextEditingController

dart

```dart
TextEditingController nameCtrl = TextEditingController();
```

```
TextField(controller: nameCtrl);
```

Copy code

Radio

dart

```
Radio(value: 1, groupValue: gender, onChanged: (v) {});
```

Copy code

Checkbox

dart

```
Checkbox(value: agree, onChanged: (v) {});
```

Copy code

Slider

dart

```
Slider(value: volume, onChanged: (v) {});
```

Copy code

🔹 Calculator App (State Management)

dart

```
setState(() {
  result = a + b;
});
```

Copy code

📌 Viva

Local state management using setState.

✅ UNIT–3 (10 Hours)

🔹 Navigation

Navigator Push

dart

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => SecondPage()),
);
```

Copy code

Named Routes

dart

```dart
Navigator.pushNamed(context, "/home");
```

Copy code

Hero Widget

dart

```dart
Hero(
  tag: "logo",
  child: FlutterLogo(),
);
```

📌 **Viva Favorite Question** ✔️

Copy code

Bottom Navigation Bar

dart

```dart
BottomNavigationBar(
  items: [
    BottomNavigationBarItem(icon: Icon(Icons.home), label: "Home"),
  ],
);
```

Copy code

Drawer

dart

```
Drawer(
  child: ListView(children: [ListTile(title: Text("Menu"))]),
);
```

Copy code

Tab Bar

dart

```
DefaultTabController(
  length: 2,
  child: Scaffold(
    appBar: AppBar(bottom: TabBar(tabs: [Tab(text:"A"), Tab(text:"B")])),
    body: TabBarView(children: [Page1(), Page2()]),
  ),
);
```

Copy code

◆ Local Persistence

SharedPreferences

dart

Copy code

```
SharedPreferences prefs = await SharedPreferences.getInstance();
prefs.setInt("count", 10);
```

File Read / Write

dart

Copy code

```
File file = File(path);
await file.writeAsString("Hello");
```

Localization

dart

Copy code

```
  supportedLocales: [Locale('en'), Locale('hi')]
```

✅ UNIT–4 (10 Hours)

🔹 Layouts

ListView

dart Copy code

```dart
ListView.builder(
  itemCount: 10,
  itemBuilder: (context, i) => ListTile(title: Text("Item $i")),
);
```

Card

dart Copy code

```dart
Card(child: ListTile(title: Text("Card")));
```

GridView

dart Copy code

```dart
GridView.count(
  crossAxisCount: 2,
  children: [Text("A"), Text("B")],
);
```

Stack

dart Copy code

```dart
Stack(children: [
  Container(color: Colors.red),
  Text("Top"),
]);
```

Slivers

dart Copy code

```
CustomScrollView(
  slivers: [
    SliverAppBar(title: Text("App")),
    SliverList(delegate: SliverChildListDelegate([...]))
  ],
);
```

📌 Viva

Slivers are used for efficient scrolling layouts.

🔹 HTTP API Call

dart Copy code

```
Future<List> fetchData() async {
  final res = await http.get(Uri.parse(url));
  return jsonDecode(res.body);
}
```

JSON Encode Decode

dart Copy code

```
jsonEncode(map);
jsonDecode(string);
```

FutureBuilder

dart Copy code

```
FutureBuilder(
  future: fetchData(),
  builder: (context, snapshot) {
    if (snapshot.hasData) return Text("Loaded");
    return CircularProgressIndicator();
  },
);
```

StreamBuilder

dart Copy code

```
StreamBuilder(
  stream: stream,
  builder: (context, snapshot) => Text(snapshot.data.toString()),
);
```

🔹 Firebase & Cloud Firestore

dart Copy code

```
FirebaseFirestore.instance
  .collection("users")
  .add({"name": "Shivam"});
```

📌 Viva Line

Firestore is a NoSQL cloud database with real-time synchronization.

🎯 FINAL PRACTICAL EXAM ANSWER (MEMORIZE)

> "This Flutter project covers Dart fundamentals, widget architecture, state management using setState, form handling, navigation techniques including routes, drawer, bottom navigation and hero animations, local data persistence using SharedPreferences and files, advanced layouts using slivers, asynchronous programming with Future and Stream, HTTP API integration with JSON parsing, and an introduction to Firebase Cloud Firestore."

✅ If you want LAST-MINUTE HELP, I can:

- Give **10 most expected viva questions**
- Create **one-page revision PDF style notes**
- Explain **how to demo app step-by-step in lab**

Just tell me 🔥

Best of luck for your practical 💯