Misr Higher Institute
for Engineering & Technology

Department of Communications
and Computer Engineering Department

Brain Pulse

A project submitted in partial fulfillment for the
requirements of the degree of B. Sc. In

Computer Engineering

By

Ahmed Elsayed ElSherbiny
Amira ElSayed Mohamed
Donia Osama Saad
Doha Mohamed Ezzat
Esraa Ali Abdelaziz
Ibrahim Ahmed Elbaz
Mina Eha Mohen
Mohamed Alaa ElBosaty
Mohamed Alaa ElSayyad
Mohamed Ayman Abdelfattah
Mohamed Ragab Ali
Yossef Sabry Farag

Supervisors

Assoc. Prof. Dr. Mohammed Moawad

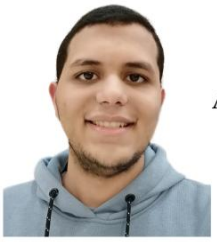
Eng. Aya Ayman Amin

Communications and Computer Engineering Department

Misr Higher Institute for Engineering & Technology

2025

Project Group Members



Ahmed Elsayed ElSherbiny

Mina Ehab Moheb

Amira Elsayed Mohamed

Mohamed Alaa Elbosaty

Donia Osama Saad

Mohamed Alaa Elsayad

Doha Mohamed Ezzat

Mohamed Ayman Abdelfattah

Esraa Ali Abdelaziz

Mohamed Rajab Ali

Ibrahim Ahmed Elbaz

Youssef Sabry Farag

Acknowledgement

We have taken effort in this project; it would not have been possible without the help and support of many contributors. We would like to extend our warmly thanks to all. First, we would like to express our gratitude to God, who gave us the faith and patience needed at all stages of our work. We would like to sincerely thank our supervisors:

- **Dr\ Mohamed Moawad**
- **Eng\ Aya Ayman Amin**

for their guidance, knowledge and patience. We are grateful for their supports and invaluable insight, which have significantly improved the final result of our work. We appreciate his kind supervision, and the helpful suggestions he provided during all stages of this project. We would not have accomplished anything without our parents, who from an early age inspired us to pursue our interests. Great thanks to them for their love, support and their continual care. Finally, we would like to express our love to all those who stayed beside us through the years. Our appreciation also goes to our colleagues, and all the people who have helped us with their invaluable skills.

Abstract

Brain Pulse is an integrated intelligent medical system developed to support neurologists in managing patient data and analyzing electroencephalogram (EEG) signals. The system consists of a mobile application built using Flutter, a pre-trained classification model for EEG signals, and a specially designed EEG helmet equipped with sensors, signal amplifiers, and noise-isolating filters to accurately record brain activity.

The mobile app enables doctors to add, edit, or delete patient records, view EEG readings, and analyze them through organized medical reports — all within a smooth and secure user interface. Data is stored locally using Shared Preferences, and state is managed using the BLoC pattern.

The EEG helmet plays a central role in the system by recording real-time brain activity and transmitting the signals to the application via Bluetooth, Wi-Fi, or USB, allowing for use beyond laboratory environments.

A Convolutional Neural Network (CNN) model is integrated into the system to classify brain activity and detect abnormal patterns such as seizures, GPD, GRDA, LPD, and LRDA. This model was trained on real clinical data from the HMS Harmful Brain Activity Classification Challenge on Kaggle and achieved an accuracy of 91.4%.

Brain Pulse is a smart and flexible system that merges hardware and software to improve the efficiency of neurological diagnosis and enhance the accuracy and ease of medical case tracking.

List of Contents

List of Contents

List of Figures

List of Figures

Chapter 1

Introduction

Chapter 1
Introduction

1.1 Overview:

Healthcare is one of the core pillars of development. With Egypt's Vision 2030 emphasizing full digital transformation, it has become essential to integrate smart technologies and mobile applications in the medical field.

This project presents Brain Pulse — a comprehensive intelligent system designed to support neurologists through a mobile application that enables the management of patient data and intelligent analysis of brainwave signals (EEG).

The system consists of three core components:

1. A Flutter-based mobile application.
2. A pre-trained classification model to analyze EEG data.
3. An EEG helmet, equipped with an electronic circuit to capture brain signals and transmit them to the mobile app.

The helmet plays a crucial role in the system. It contains EEG electrodes to detect brain activity and a built-in amplifier and filter circuit. The recorded signals are transmitted via Bluetooth or USB to the application for processing and analysis.

❖ Problem Definition:

Diagnosing brain disorders such as epilepsy or abnormal brainwave activity traditionally requires manual review of EEG signals — a process that is time-consuming, costly, and prone to human error or limited expertise.

In addition, managing patient records manually or through non-structured systems leads to inefficiencies in accessing or tracking previous medical data.

There is a clear need for a smart solution that enables doctors to:

- Capture and analyze EEG signals easily using a portable helmet.
- Send the readings directly to the mobile app for immediate processing.
- Manage patient data securely and systematically.
- Accelerate diagnosis using an intelligent, mobile-based tool.

❖ **Our Mission:**

Our mission is to provide an integrated digital environment for neurologists that enables them to:

- Add and manage patient profiles.
- Record and analyze EEG readings with a smart system.
- Display structured medical reports within a secure and intuitive interface.
- Use a wearable EEG helmet that sends real-time brain activity data to the app.
- Ensure privacy through in-app features like profile editing and account deletion.
- Bridge hardware (the helmet) and software (the app) to support rapid, efficient diagnosis.

1.2 Existing Solutions and New Ideas:

Current EEG analysis systems often:

- Target researchers, not clinical use.
- Require large, expensive laboratory equipment.
- Lack integrated patient management.
- Are not portable or user-friendly.

Brain Pulse introduces several innovations:

- A lightweight, wearable EEG helmet with a compact amplification and filtering circuit (based on AD620AN and signal filters).
- Real-time data transmission to the app — no lab required.
- Usable in hospitals, clinics, or even at home.
- EEG signal analysis via a pre-trained CNN model.
- Seamless patient record tracking through a clean mobile interface.

1.3 Methodology:

The team follows the Scrum methodology, which provides flexibility in handling multidisciplinary development — software, AI, and hardware.

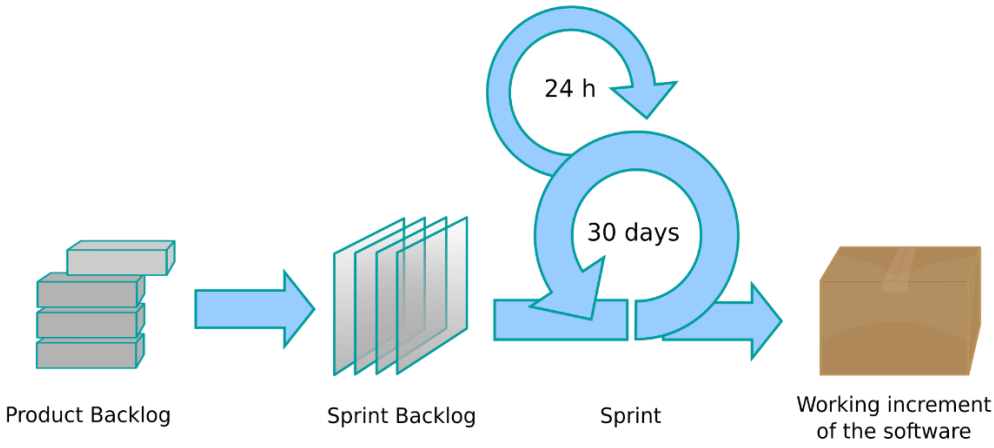


Fig1.1 Scrum Methodology

Scrum Practices:

- Sprint Planning: Identifying tasks related to app features, model integration, and EEG helmet development.
- Daily Stand-ups: Team members track progress and resolve blockers.
- Sprint Review: Assess deliverables like patient UI, EEG analysis modules, and hardware testing.
- Sprint Retrospective: Improve team performance for upcoming sprints.

Application of Methodology:

- In early sprints, the team built the mobile interface using Flutter and defined core models and state management (BLoC).
- Mid-sprints focused on integrating the pre-trained CNN model for EEG signal classification.
- A dedicated sprint was assigned to develop a real EEG acquisition helmet, equipped with sensors and analog filters to collect clean brain signals.
- Final sprints combined all components into one functional system that ensures an optimized and seamless doctor experience.

.

Chapter 2

Literature Review

Chapter 2

Literature Review

2.1 History and related work

In recent years, healthcare has witnessed a remarkable evolution in the integration of smart technologies and artificial intelligence with wearable devices to facilitate diagnosis and patient monitoring. One of the most notable trends is EEG signal analysis using specialized applications and intelligent models, supported by wearable EEG devices.

Examples of Existing Solutions:

1. Emotiv and NeuroSky (EEG Sensor Systems):

These are among the most well-known companies offering commercial EEG headsets used for research or entertainment purposes. These systems allow brain signal recording and display, or transmission to external applications. However, they typically do not offer a comprehensive medical platform or clinical-grade analysis.

2. OpenBCI:

An open-source system used in neuroscience research, offering customizable EEG devices and software. Despite its research capabilities, it requires technical expertise and lacks a user-friendly interface for doctors.

3. Muse EEG Headband:

A commercial device used primarily for tracking brain activity during meditation and relaxation. It is not intended for clinical use and does not include an integrated patient management system or medical record analysis.

2.2 What Makes Brain Pulse Different:

Brain Pulse offers a complete, integrated system that includes:

A dedicated medical mobile application designed specifically for neurologists, not general users or researchers.

An easy-to-use interface for managing patients, searching records, and analyzing EEG medical histories

A smart wearable EEG helmet, equipped with internal amplifiers and filters for noise reduction, which transmits data directly to the app via Bluetooth or USB.

A pre-trained CNN-based classification model capable of identifying abnormal brain activity patterns such as seizures, GPD, GRDA, LPD, and LRDA.

Direct integration between hardware and software, eliminating the need for lab equipment or complex systems.

❖ Summary:

While other systems offer basic EEG readings or partial analysis tools, Brain Pulse stands out as:

- A practical, daily-use clinical solution.
- A seamlessly integrated combination of software and hardware.
- An intelligent tool that empowers doctors to manage data and analyze brain signals accurately and efficiently.
- Suitable for hospitals, clinics, or even home use.

This makes Brain Pulse not just an app or a device, but a fully intelligent medical ecosystem, built upon cutting-edge medical and engineering technology to enhance neurological diagnostics and overall healthcare quality.

Chapter 3

Mobile Application

Chapter 3

Mobile Application

3.1 - Section 1: The Flutter Framework:

3.1.1 Overview of Flutter:

Flutter is an open-source UI toolkit developed by Google for building cross-platform applications (Android, iOS, Web, Desktop) from a single codebase. It uses the Dart programming language and offers high performance, modern design, and fast development.

Advantages of Flutter

- Widget-Based UI: Offers full control over the interface with customizable widgets.
- Hot Reload: Enables instant preview of code changes, speeding up development.
- Single Codebase: Reduces time and effort by allowing one codebase for multiple platforms.
- High Performance: Uses Skia rendering engine for smooth, native-like performance.
- Growing Community: Rich ecosystem with many packages available on pub.dev.

Advantages of Dart

- Easy to Learn: Similar to Java and JavaScript.
- Fast Execution: Compiled directly into native code.
- OOP Support: Strong object-oriented structure for scalable apps.

Disadvantages of Flutter

- Larger App Size: Due to embedded engine.
- Limited Native API Access: Sometimes requires native code for advanced features.
- Lower Performance for Complex Graphics: Not ideal for 3D or intensive graphics.
- Frequent Updates: Can cause compatibility issues.

Disadvantages of Dart

- Smaller Community: Compared to languages like Python or JavaScript.
- Mainly Used with Flutter: Limited adoption outside mobile development.
- Fewer Libraries: Less mature than older languages.

Conclusion

Despite its limitations, Flutter and Dart were excellent choices for the "Brain Pulse" project. They enabled a responsive, visually appealing UI with seamless integration of AI components across all target platforms.

3.2 - Section 2: Development Tool – Visual Studio Code:

3.2.1 Overview of VS Code:

Visual Studio Code (VS Code) is a free, open-source code editor by Microsoft. It's widely used in mobile and web development due to its flexibility, performance, and rich feature set.

Despite some drawbacks, VS Code is an ideal choice for Flutter development due to its speed, flexibility, and strong extension support. It played a vital role in the success of the "Brain Pulse" project and remains a top tool for cross-platform app development.

3.3 Section 3: System Requirements:

This Section aims to define the functional and non-functional requirements of the "Brain Pulse" system. These requirements serve as a foundation for designing and developing the system, ensuring that the final product meets the needs of its users (in this case, medical professionals) as well as the project's objectives.

3.3.1 Functional Requirements:

(Functional Requirements):

Functional requirements describe the specific behaviors and functionalities that the system must perform — in other words, what the system should do and how it should interact with the user.

1. User Authentication:
   - The system shall allow doctors to securely log in using a username and password.
   - The security system must be capable of verifying user identity and preventing unauthorized access.

2. Patient Management:
   - The system shall allow the addition of new patient records.
   - Doctors shall be able to edit or update existing patient information.
   - The system shall clearly display previous patient records.

3. Session Creation:
   - The system shall provide an interface for creating new evaluation sessions.
   - It shall allow the input of relevant medical and behavioral data related to the patient's condition.

4. AI Analysis Integration:
   - The system shall process entered data using an integrated artificial intelligence model.
   - The AI model shall return accurate and understandable analysis results.

5. View Session History:
   - The system shall allow doctors to view all previous sessions for each patient.
   - It shall show changes in the patient's condition over time.

6. Result Visualization:

- Results shall be displayed in a clear and easy-to-understand format.
- The output shall include analytical insights that support clinical decision-making.

7. Search and Filtering:

- The system shall support searching and filtering of patients and session records based on specific criteria (e.g., name, date, diagnosis).
- The filtering process must be simple and fast.

3.2.2: Non-Functional Requirements:

(Non-Functional Requirements):

Non-functional requirements focus on the quality attributes and performance characteristics of the system — not what the system does, but how well it performs its functions.

1. Performance:

- The system shall respond to user requests within a maximum of two seconds.

2. Scalability:

- The system architecture shall be scalable to accommodate more users and larger datasets in the future.

3. Usability:

- The user interface shall be intuitive and accessible, even for users without a strong technical background.

4. Security:

- Sensitive data such as patient records must be stored securely and encrypted during transmission.
- Strong security mechanisms such as encryption and role-based access control must be implemented.

5. Reliability:

- The system shall operate stably with minimal downtime.
- It shall be able to recover from failures without data loss.

6. Cross-Platform Compatibility:

- The system shall run smoothly on both Android and iOS devices.
- Web compatibility shall also be supported if added in future versions.

7. Maintainability:

- The system codebase shall be well-documented and modular, enabling ease of updates and bug fixes.

Conclusion:

This Section provided a comprehensive overview of the essential system requirements to be considered when designing and developing the "Brain Pulse" application. These requirements form the basis for the design and implementation phases and ensure alignment with the project's goals and the medical needs of the end users.

3.4 - Section 4: System Design:

3.4.1: User Flow Diagram:

The User Flow Diagram provides a step-by-step visualization of how the doctor interacts with the *Brain Pulse* application. It outlines the main navigation paths taken by the user to complete key tasks within the system.

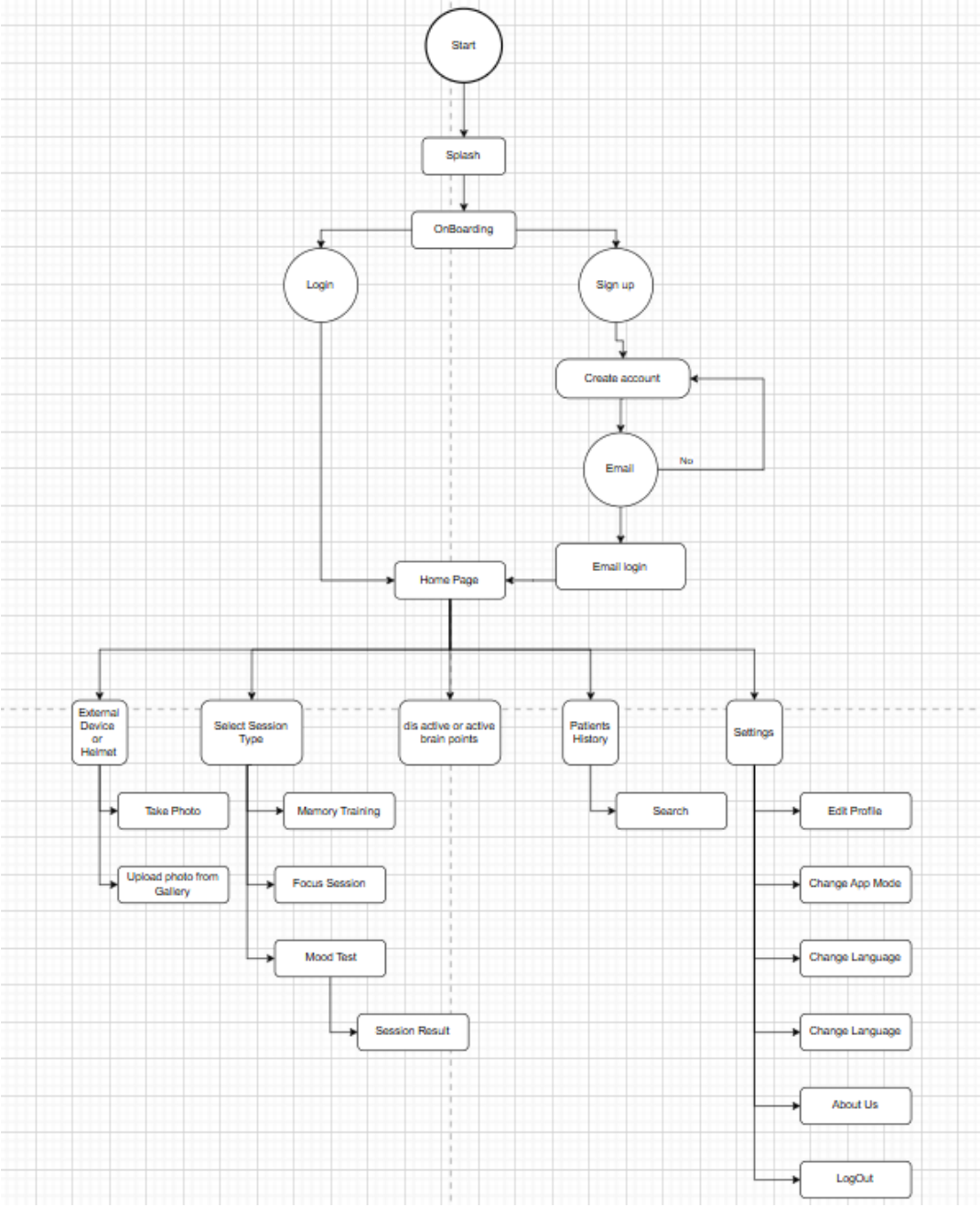


Fig.3.1 User Flow Diagram

This flow demonstrates the typical operations a doctor can perform and ensures a smooth, intuitive experience throughout the app.

3.5 - Section 5: System Architecture:

The "Brain Pulse" system is built upon a three-layer architecture, a widely adopted design pattern in modern application development. This architecture divides the system into three distinct layers, each responsible for specific functionalities. The detailed description of each layer is as follows:

3.5.1. Presentation Layer:

This layer serves as the interface through which users interact with the system. Developed using the Flutter framework, it is responsible for:

- Providing an intuitive and visually appealing user interface.
- Collecting input data from users, such as age, symptoms, and psychological status.
- Sending this input data to the underlying layers for processing.
- Displaying the processed results and medical recommendations to users in a clear and understandable manner.

3.5.2 Application Logic Layer:

Known as the core of the system, this layer encapsulates the main processing logic and decision-making components, including:

- The Artificial Intelligence models trained to analyze input data and generate diagnostic assessments.
- Business logic and clinical decision rules based on medical and scientific foundations.
- Serving as an intermediary between the Presentation Layer and the Data Layer, managing data flow and coordination.

3.5.3 Data Layer:

This layer manages all data storage and retrieval operations within the system. It includes:

- Databases storing user information, patient records, session histories, and analysis results.
- Storage of AI model training data, whether locally or through cloud services.

- Integration with external APIs or data sources that provide supplementary medical information.

Summary:

Adopting a three-layer architecture enhances system organization by clearly separating concerns. This separation results in:

- Improved system flexibility for updates and modifications.
- Easier maintenance and troubleshooting.
- Increased security and stability.

3.6 - Section6: System Implementation:

3.6.1 Main Entry point:

The main function serves as the entry point of the Brain Pulse mobile application. It initializes necessary configurations, sets up dependency injection, initializes shared preferences for session management, and launches the app wrapped in providers for state management and localization.

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  setupGetIt();
  Bloc.observer = MyBlocObserver();

  sharedPreferences = await SharedPreferences.getInstance();
  isLoggedIn = sharedPreferences.getString('token') != null;

  runApp(
    MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => ThemeProvider()),
        ChangeNotifierProvider(create: (_) => LocaleProvider()),
      ],
      child: const BrainPulse(),
    ),  // MultiProvider
  );
}
```

Fig.3.2 App entry point running

3.6.2 BrainPulse Class:

This section presents the main application widget BrainPulse, which initializes the required BLoC providers and sets up theming, localization, and routing using Flutter's MaterialApp. It uses ScreenUtilInit to support responsive UI design across different screen sizes.

```dart
class BrainPulse extends StatelessWidget {
  const BrainPulse({super.key});
  @override
  Widget build(BuildContext context) {
    return ScreenUtilInit(
      designSize: const Size(375, 812),
      minTextAdapt: true,
      child: MultiBlocProvider(
        providers: [
          BlocProvider(
            create: (context) => getIt<SendDataByDoctorCubit>(),),  // BlocProvider
          BlocProvider(
            create: (context) => getIt<GetAllPatientsCubit>(),),  // BlocProvider
          BlocProvider(
            create: (context) =>
                LoginCubit(loginRepoImple: getIt.get<LoginRepoImple>()),),  // BlocProvider
          BlocProvider(
            create: (context) => RegisterCubit(
                registerRepoImple: getIt.get<RegisterRepoImple>()),),  // RegisterCubit, BlocProvider
          BlocProvider(
            create: (context) => getIt<PredictionImageCubit>(),),],  // BlocProvider
        child: Consumer2<ThemeProvider, LocaleProvider>(
          builder: (context, themeProvider, localeProvider, child) {
            return MaterialApp(
              title: 'Brain Pulse',
              debugShowCheckedModeBanner: false,
              initialRoute: Routes.splashScreen,
              onGenerateRoute: AppRouter.generateRoute,
              theme: themeProvider.currentTheme,
              locale: localeProvider.currentLocale,);},),),);}}  // MaterialApp, Consumer2, MultiBlocProvider, ScreenUtilInit
```

Fig.3.3 BrainPulse app setup with theming, localization, and routing.

3.6.3 Splash Screen Implementation:

The splash screen is the initial screen that appears when the application starts. It provides a visual introduction while background checks are performed to determine the user's login status. After a short delay (2 seconds), the app navigates either to the main application screen or the onboarding screen depending on the stored login state using LoginCubit.

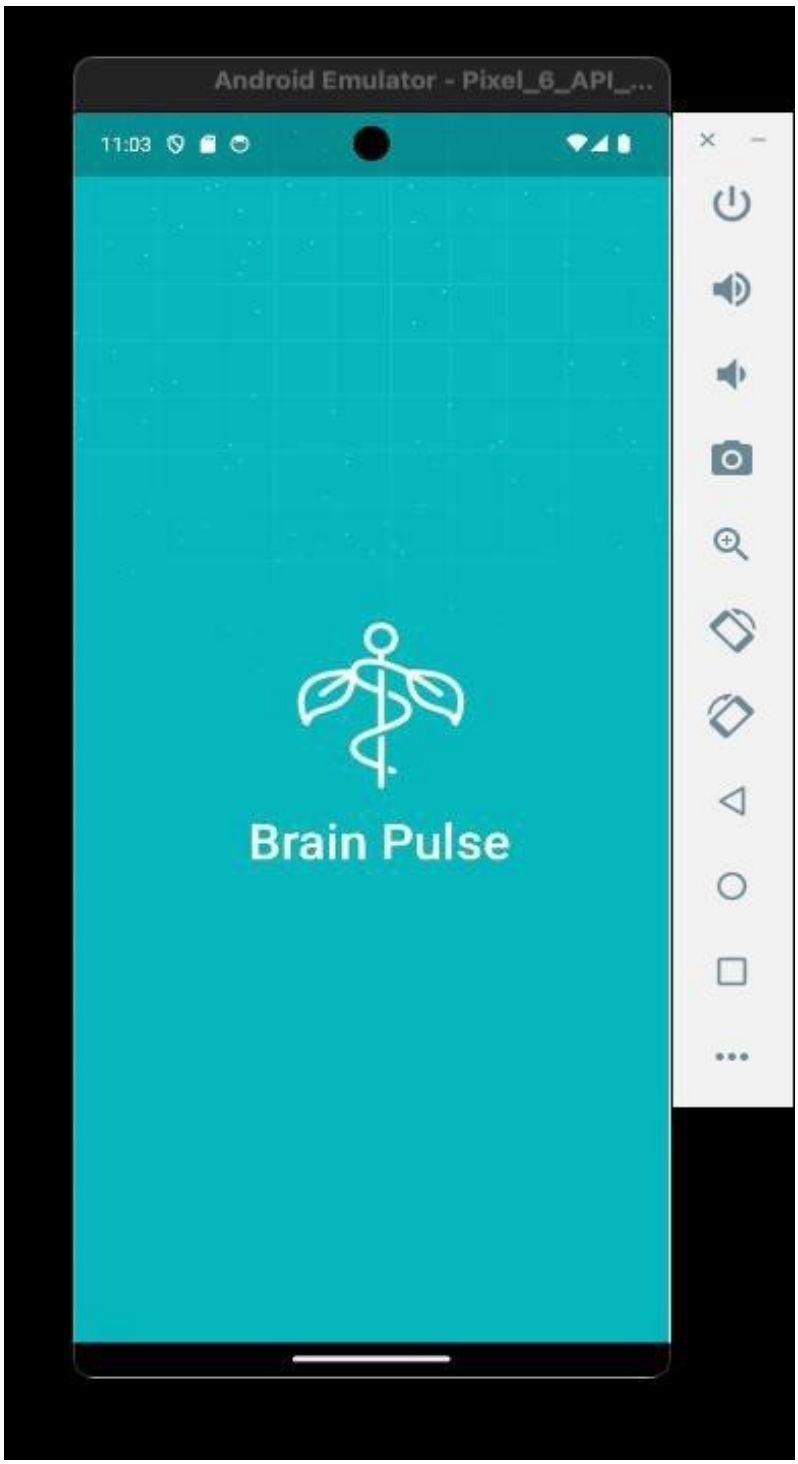


Fig.3.4 Splash screen of Brain Pulse app displaying the logo.

3.6.4: OnBoarding Screen:

The above code demonstrates the implementation of the Onboarding screen, which contains two informative pages navigated using a PageView. It includes action buttons like “Skip” and “Let’s go” that allow the user to either bypass the onboarding or proceed to the login screen. Custom widgets such as CustomButton and AnimatedContainerOnBoarding are used to provide a smooth and interactive user experience.

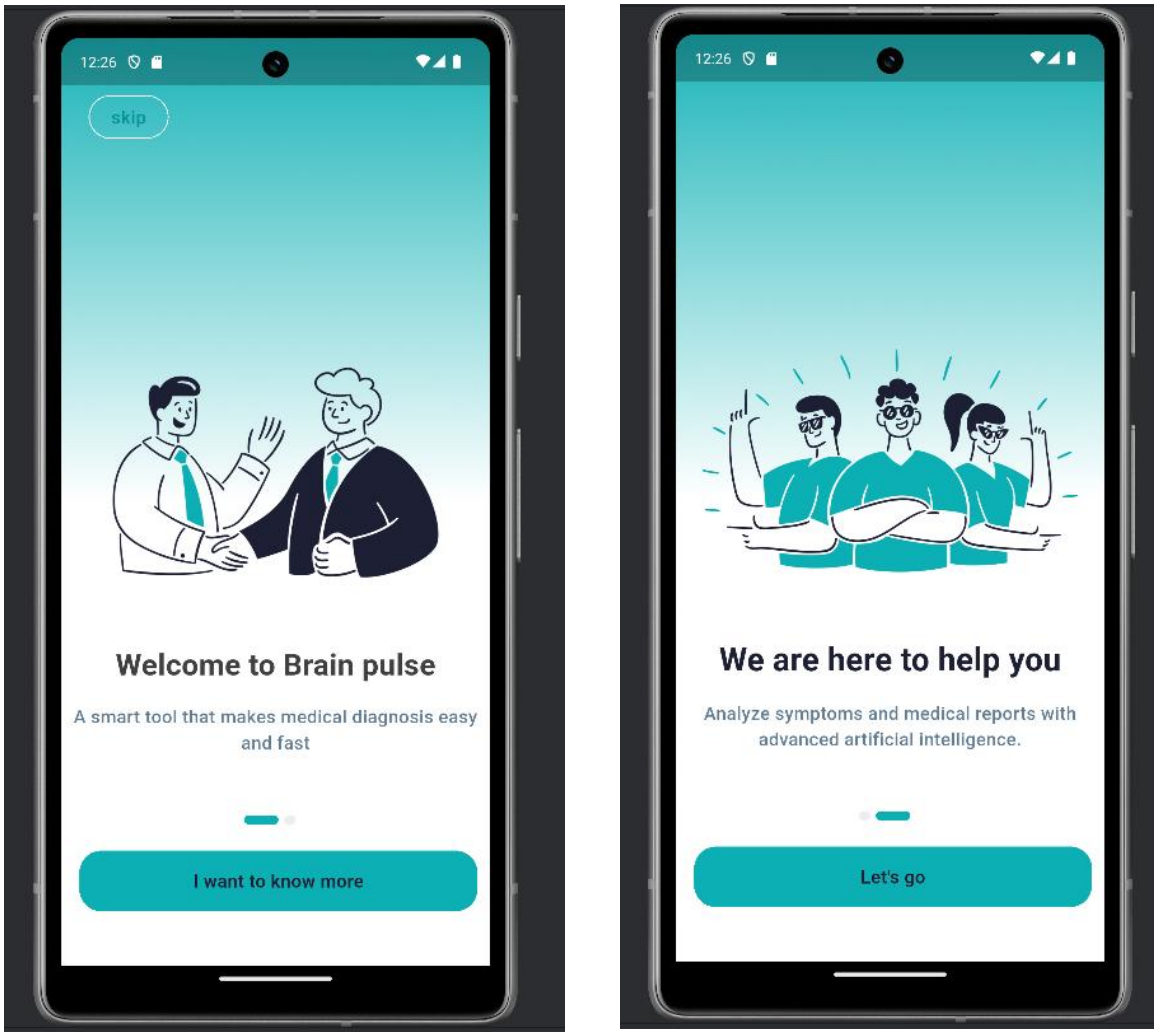


Fig.3.5 The following figures demonstrate different parts of the Onboarding screen Output

3.6.5: Login Screen:

The LogInScreen class displays a login interface using Flutter and utilizes BlocConsumer to listen for the login state from LoginCubit. Upon successful registration, the user is taken to the main screen, and if an error occurs, a warning message is displayed. The screen includes fields for email and password, a login button with a loading state, and options such as 'Remember Me' and 'Forgot Password', with a responsive design that fits various screen sizes.

Fig.3.6 Login Screen.

3.6.6: Register Screen:

The RegisterScreen class provides a user registration interface with fields for name, email, phone number, and password. It uses RegisterCubit and BlocConsumer to manage state, handle form validation, and interact with the backend. On successful registration, users are redirected to the login screen, while errors trigger appropriate messages.

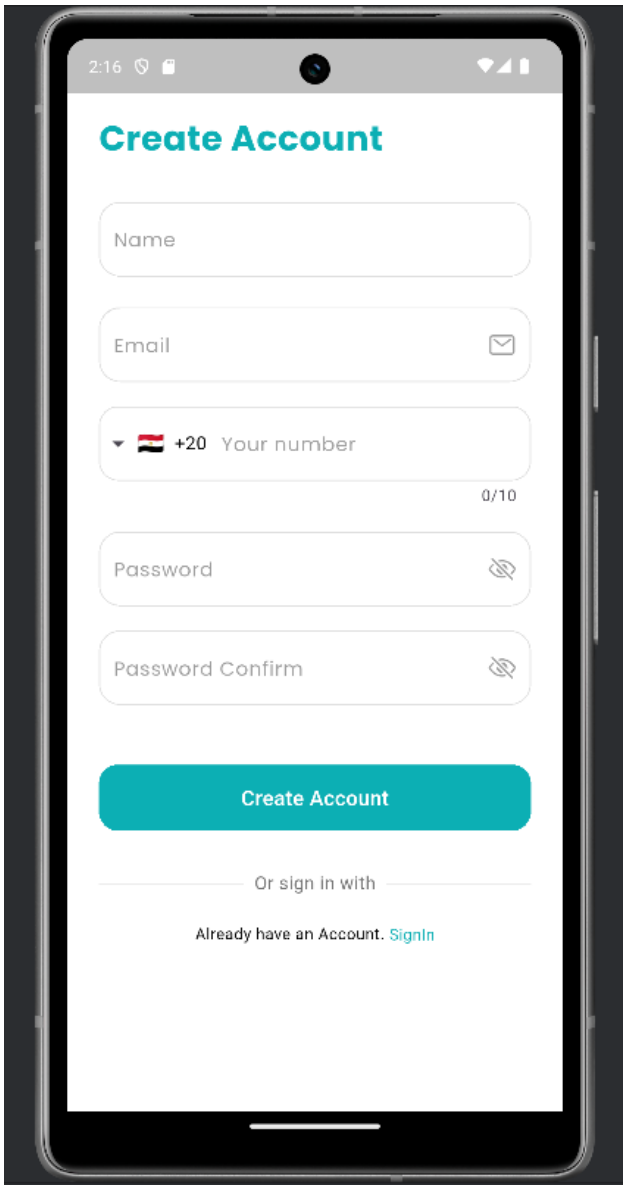


Fig.3.7 Register Screen

3.6.7: Home Screen:

The Home Screen is the main interface that appears to the doctor after logging in. It acts as the starting point for navigating and using the application.

1. Welcome & App Introduction

At the top of the screen, a welcome message is displayed along with a brief description stating that Brain Pulse is a medical platform that helps diagnose and monitor neurological disorders using EEG (Electroencephalography) technology.

2. Interactive Display of EEG Signals

The screen displays various brainwave signal types with their medical meaning and possible conditions:

- Seizure
  - Sudden abnormal brain electrical activity
  - Indicates: Epilepsy, tumors, infections, trauma
  - Treatment: Anti-seizure medication, surgery, lifestyle changes
- LPD (Localized Periodic Discharges)
  - Repetitive discharges from one hemisphere
  - Indicates: Stroke, infections, tumors
  - Treatment: Treat underlying cause + medication
- GPD (Generalized Periodic Discharges)
  - Discharges from both hemispheres
  - Indicates: Brain injury, coma, metabolic disturbances
  - Treatment: Supportive care + manage root cause
- LRDA (Lateralized Rhythmic Delta Activity)
  - Slow rhythmic waves from one side of the brain
  - Indicates: Tumors, trauma, stroke
  - Treatment: Based on cause + optional medication
- GRDA (Generalized Rhythmic Delta Activity)
  - Slow rhythmic waves from both sides of the brain
  - Indicates: Encephalopathy, coma, metabolic issues
  - Treatment: Supportive care + root cause management

When selecting any of these signals, the app displays:

- A simple description
- Related medical conditions
- Recommended treatments

3. Floating Action Button (FAB)

A circular button appears at the bottom of the screen. When pressed, it opens a shortcut menu offering:

- By Hospital

  Navigate to EEG image diagnosis from a hospital

- By Doctor

  Navigate to manual diagnosis by the doctor

- Note:

  This screen is designed to help the doctor access diagnostic tools quickly and interactively using a clean, intuitive UI.

```
child: ListView.builder(
  scrollDirection: Axis.horizontal,
  itemCount: brainSignalData.length,
  itemBuilder: (context, index) {
    return GestureDetector(
      onTap: () {
        setState(() {
          selectedIndex = index;
        });},
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 8.0),
        child: Column(
          children: [
            Container( width: 60.w, height: 60.h,
              decoration: BoxDecoration(
                color: selectedIndex == index
                    ? ColorsApp.primary.withAlpha(800) : Colors.grey.withAlpha(900),
                shape: BoxShape.circle,
                border: Border.all(
                  color: selectedIndex == index
                      ? ColorsApp.primary.withAlpha(900) : Colors.grey.withAlpha(900), width: 2.0,
                ),),
              child: Center(
                child: Text(
                  brainSignalData[index]['title'],
                  style: TextStyle(
                    color: selectedIndex == index
                        ? ColorsApp.primary.withAlpha(900) : Colors.white, fontWeight: FontWeight.bold, fontSize: 14, // Adjust as needed
                  ),),),),],),),);},),
```

Fig.3.8 Binary Signal Data – Displaying EEG Brain Signals.

This code displays a horizontal scrollable list containing EEG signal names (like Seizure, GPD, etc.) interactively. When the user taps on a tab, the content on the screen updates based on the selected brain signal.

```
Padding(padding: EdgeInsets.symmetric(vertical: 10.h),
  child: Column( crossAxisAlignment: CrossAxisAlignment.start,
    children: [ Text('Description :',
        style: TextStyle(
          fontSize: 18.sp, fontWeight: FontWeight.bold, color: ColorsApp.primary,),),
      const GapH(height: 8),
      Text(brainSignalData[selectedIndex]['description'],
        style: TextStyle( fontSize: 16.sp, color: ColorsApp.grey,),),],),),
Padding(padding: EdgeInsets.symmetric(vertical: 10.h),
  child: Column( crossAxisAlignment: CrossAxisAlignment.start,
    children: [ Text('Possible diseases :',
        style: TextStyle(
          fontSize: 18.sp, fontWeight: FontWeight.bold, color: ColorsApp.primary,
        ),),
      const GapH(height: 8),
      Text(brainSignalData[selectedIndex]['Possible diseases'],
        style: TextStyle( fontSize: 16.sp, color: ColorsApp.grey,),),],),),
Padding(padding: EdgeInsets.symmetric(vertical: 10.h),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [Text('Treatment :',
        style: TextStyle(fontSize: 18.sp, fontWeight: FontWeight.bold, color: ColorsApp.primary,),),
      const GapH(height: 8),
      Text(brainSignalData[selectedIndex]['Treatment'],
        style: TextStyle(fontSize: 16.sp, color: ColorsApp.grey,),),],),),
```

Fig.3.9 Displaying EEG Signal Details (Description, Diseases, and Treatment).

After selecting a specific signal from the list above, this section displays its description, the possible diseases related to it, and available treatment options. This helps the doctor quickly understand the patient's condition based on the EEG analysis.

```
body: HawkFabMenu(
  icon: AnimatedIcons.menu_arrow,
  fabColor: ColorsApp.primary,
  iconColor: Colors.white,
  hawkFabMenuController: hawkFabMenuController,
  items: [
    HawkFabMenuItem(
      label: 'By Hospital',
      ontap: () {
        Navigator.pushNamed(context, Routes.getImage);
      },
      icon: Icon(Icons.local_hospital, color: ColorsApp.primary, size: 25.h,),
      color: Colors.white, labelColor: ColorsApp.primary,
    ),
    HawkFabMenuItem(
      heroTag: 'doctor',
      label: 'By Doctor',
      ontap: () {
        Navigator.pushNamed(context, Routes.dataByDoctorScreen);
      },
      icon: Icon(Icons.local_hospital, color: ColorsApp.primary, size: 25.h,),
      color: Colors.white, labelColor: ColorsApp.primary,
    ),
```

Fig.3.10 Home Body – Floating Action Button and Screen Navigation.

This code creates a Floating Action Button Menu that allows the user to choose the diagnosis method—either by hospital or by doctor. When an option is selected, the user is navigated to the corresponding screen using Navigator.pushNamed.

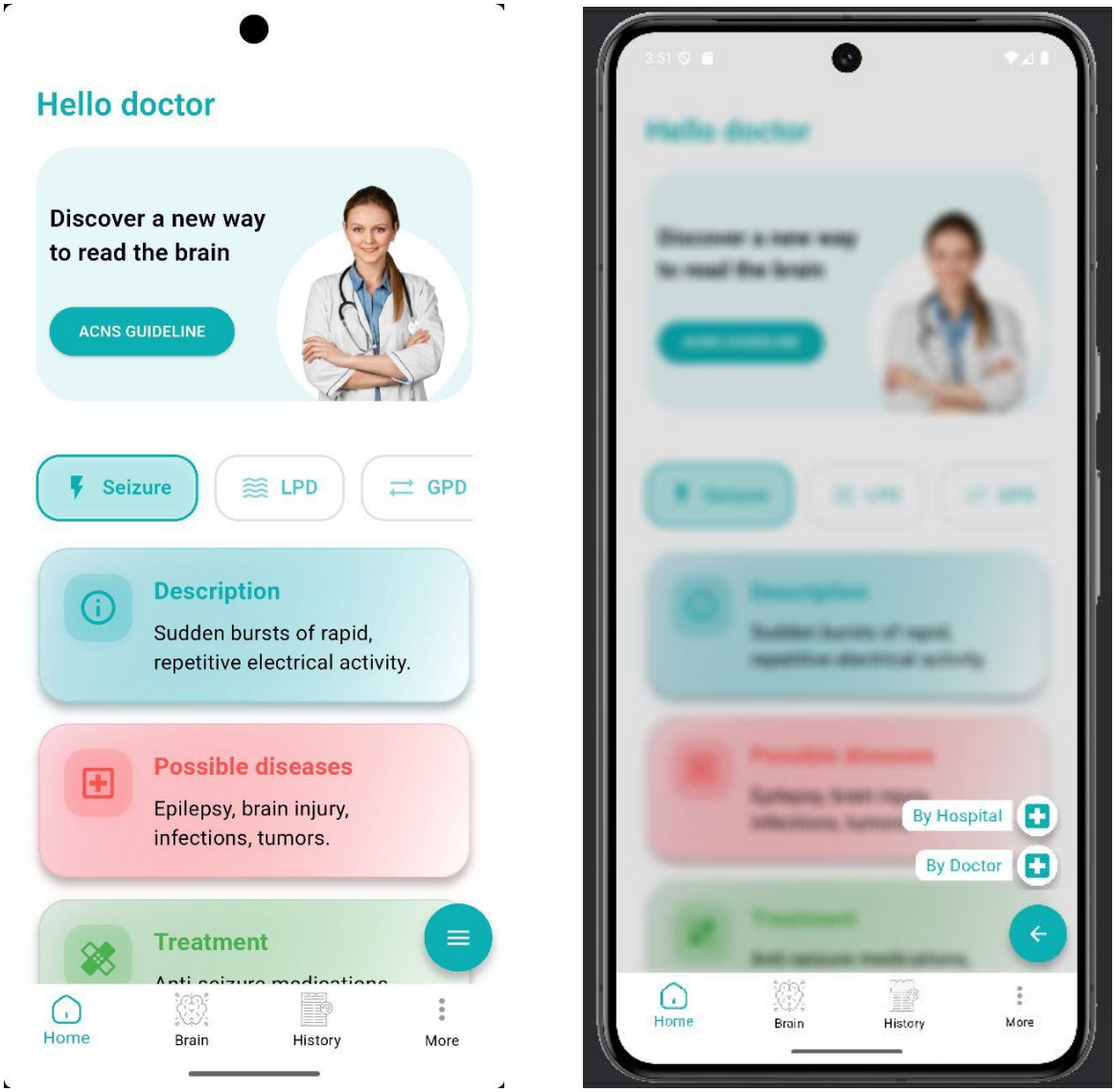


Fig.3.11 The following figures demonstrate different parts of the Home Screen Output.

- **3.6.7.1: Hospital Mode:**

```dart
class PredictionFileCubit extends Cubit<PredictionFileState> {
  final PredictFileRepoImple predictFileRepoImple;

  PredictionFileCubit({required this.predictFileRepoImple})
      : super(InitialPredictionFileState());

  Future<void> uploadEegFile(String filePath) async {
    emit(LoadingPredictionFileState());

    final result = await predictFileRepoImple.uploadEegFile(filePath);

    result.fold(
      (failure) {
        print('❌ Failure: ${failure.errorMessage}');
        emit(ErrorPredictionFileState(errormsg: failure.errorMessage));
      },
      (predictionModel) {
        print('✅ Prediction Loaded: ${predictionModel.prediction}');

        emit(LoadedPredictionFileState(prediction: predictionModel.prediction));
      },
    );
  }

  void reset() {
    emit(InitialPredictionFileState());
  }
}
```

Fig3.12 Prediction File Cubit

EEG Analysis Using CSV File Upload

The system allows clinicians to upload EEG files in CSV format for automatic analysis. Once the file is selected, it is sent to the server using the Dio library, where it is analyzed by a pre-trained AI model.

The server returns a classification result (e.g., Seizure, IPD, GPD, GRDA, IRDA, Other). The result is clearly displayed in the user interface and can be saved to a new patient's file.

The process relies on case management using Cubit, applying the Repository pattern to separate interface logic from business logic, improving code maintainability and testability.

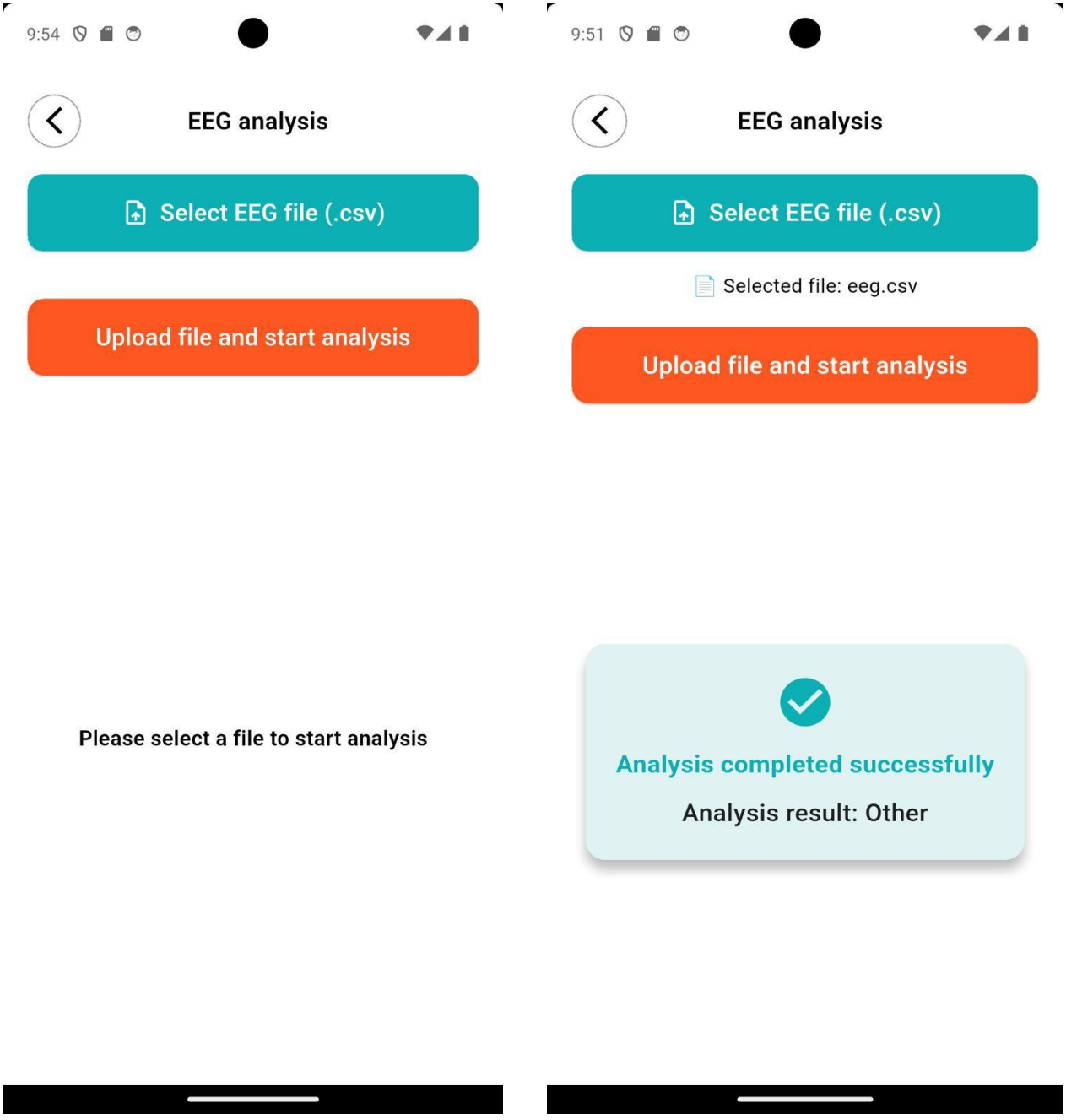


Fig3.13 Upload EEG Image.

- **3.6.7.2: Doctor Mode:**

```
RowTextWithTextFieldGetDataByDoctor(
  controller: context.read<SendDataByDoctorCubit>().p1,
  number: ' 1 ',
),
```

Fig3.14 Manual EEG Point Input by Doctor

Each row in this screen represents a numbered EEG signal input field. The values entered are collected and analyzed to detect potential abnormalities.

```
child: CustomButton(
  onTap: () {
    context.read<SendDataByDoctorCubit>().sendDataByDoctor();
  },
  text: 'Send',
  width: double.infinity,
  height: 50.h,
  textColor: ColorsApp.white,
  borderColor: ColorsApp.primary,
  color: ColorsApp.primary,
),
),
```

***Fig3.15* Triggering EEG Analys**

This button is responsible for sending the manually entered EEG signal points (20 values) to the backend server for analysis. When pressed, it triggers the sendDataByDoctor() method in the Cubit, which processes the data and returns prediction results.

```dart
class RowTextWithTextFieldGetDataByDoctor extends StatelessWidget {
  const RowTextWithTextFieldGetDataByDoctor({
    super.key,
    required this.number,
    required this.controller,
  });
  final String number;
  final TextEditingController controller;
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Text(
          ' point $number :',
          style: Theme.of(context).textTheme.bodyLarge,
        ),
        gapW(20),
        SizedBox(
          width: 80.w,
          height: 40.h,
          child: MyTextField(
            controller: controller,
            keyboardType: TextInputType.number,
            hint: '0',
            validator: (p0) {},
          ),
        ),
      ],
    );
  }
}
```

Fig3.16 Input Field Builder for EEG Points

Used 20 times in the input screen, this widget streamlines the design by creating consistent EEG input rows with minimal code repetition.

```dart
import 'package:json_annotation/json_annotation.dart';

part 'send_point_request_model.g.dart';

@JsonSerializable()
class SendPointRequestModel {
  List? arr;

  SendPointRequestModel({this.arr});
  // from json
  factory SendPointRequestModel.fromJson(Map<String, dynamic> json) =>
      _$SendPointRequestModelFromJson(json);
  Map<String, dynamic> toJson() => _$SendPointRequestModelToJson(this);
}
```

Fig3.17 EEG Points Model

This model wraps the list of 20 EEG signal points manually entered by the doctor. It is converted into JSON format and sent to the server for analysis.

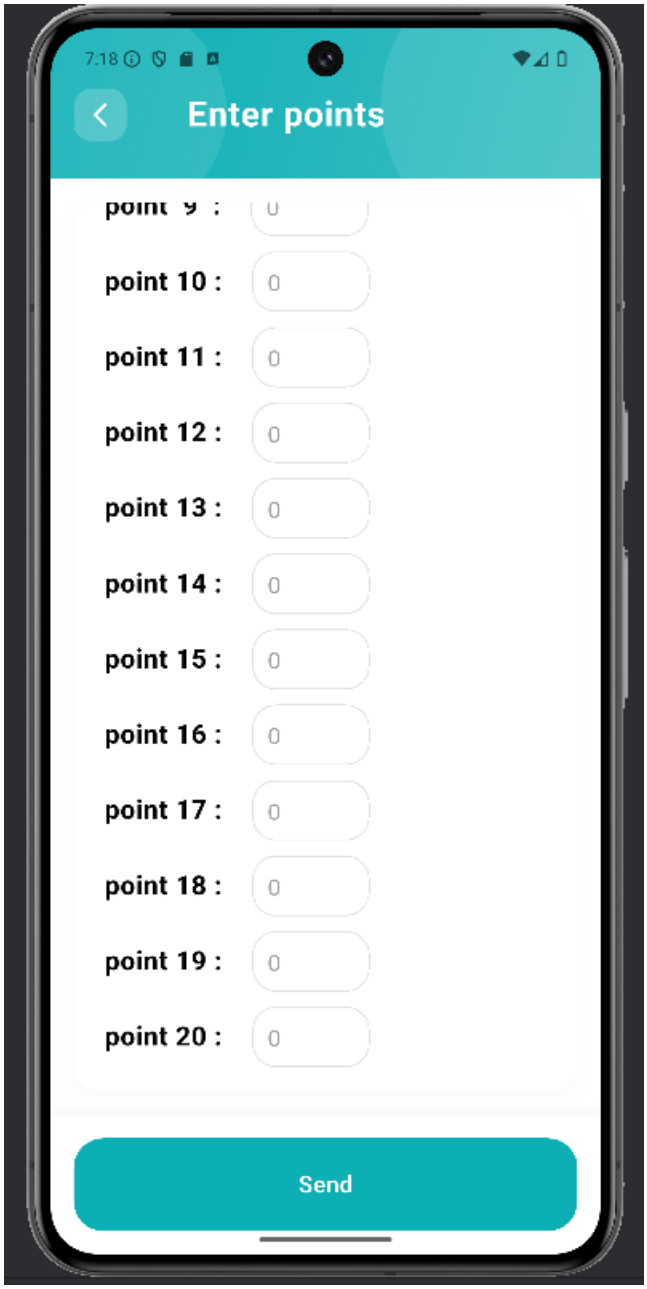


***Fig3.18* Points Entered by Doctor**

```
child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Text(
      'Patient Analysis Results',
      style: Theme.of(context).textTheme.titleLarge?.copyWith(
            color: ColorsApp.primary,
            fontWeight: FontWeight.bold,
          ),
    ),
    SizedBox(height: 20.h),
    _buildResultRow(context, 'GPD', prediction[0].toString()),
    _buildResultRow(
        context, 'GRDA', prediction[1].toString()),
    _buildResultRow(context, 'LPD', prediction[2].toString()),
    _buildResultRow(
        context, 'LRDA', prediction[3].toString()),
    _buildResultRow(
        context, 'Seizure', prediction[4].toString()),
    _buildResultRow(
        context, 'Other', prediction[5].toString()),
  ],
),
```

Fig3.19 Result Display After Analysis (Shared between Hospital and doctor)

Each _buildResultRow displays the type of signal and its corresponding confidence value. These results help the doctor understand the AI model's prediction.

Fig3.20 Analysis Result Screen

```
child: InkWell(
  onTap: () {
    if (formKey.currentState!.validate()) {
      context.read<SendDataByDoctorCubit>().addPatient(
            AddPatientRequestModel(
              age: int.parse(ageController.text),
              name: nameController.text,
              phoneNumber: phoneController.text,
              gpd: prediction[0].toDouble(),
              grda: prediction[1].toDouble(),
              ipd: prediction[2].toDouble(),
              irda: prediction[3].toDouble(),
              seizure: prediction[4].toDouble(),
              other: prediction[5].toDouble(),
            ),
          );
    }
  },
```

Fig3.21 Saving Patient Data After Diagnosis (Shared between Hospital and doctor)

This block of code is executed when the user presses the "Save" button. It first validates the input form to make sure that all fields (name, phone, age) are filled correctly. Then, it creates a AddPatientRequestModel that includes both the personal patient data and the prediction results returned by the AI system (like GPD, GRDA, Seizure, etc.). Finally, it sends this data using the addPatient() function in the Cubit.

```
AddPatientRequestModel _$AddPatientRequestModelFromJson(
       Map<String, dynamic> json) =>
   AddPatientRequestModel(
     name: json['name'] as String?,
     phoneNumber: json['phoneNumber'] as String?,
     age: (json['age'] as num?)?.toInt(),
     gpd: (json['gpd'] as num?)?.toDouble(),
     grda: (json['grda'] as num?)?.toDouble(),
     ipd: (json['ipd'] as num?)?.toDouble(),
     irda: (json['irda'] as num?)?.toDouble(),
     seizure: (json['seizure'] as num?)?.toDouble(),
     other: (json['other'] as num?)?.toDouble(),
   );

Map<String, dynamic> _$AddPatientRequestModelToJson(
       AddPatientRequestModel instance) =>
   <String, dynamic>{
     'name': instance.name,
     'phoneNumber': instance.phoneNumber,
     'age': instance.age,
     'gpd': instance.gpd,
     'grda': instance.grda,
     'ipd': instance.ipd,
     'irda': instance.irda,
     'seizure': instance.seizure,
     'other': instance.other,
   };
```

Fig3.22 Patient Saving Model

This model holds all patient-related data including name, phone number, age, and the prediction scores from the AI model. It is used to save patient data after diagnosis.

```dart
class SendPointRepo {
  SendPointRepo(this._apiService);
  final ApiService _apiService;

  Future<ApiResult<dynamic>> sendDataByDoctor(
      SendPointRequestModel request) async {
    try {
      final response = await _apiService.sendDataByDoctor(request);
      return ApiResult.success(response);
    } catch (e) {
      log(e.toString());
      return ApiResult.failure(e.toString());
    }
  }
}

class AddPatientRepo {
  AddPatientRepo(this._apiService);
  final ApiService _apiService;

  Future<ApiResult> addPatient(AddPatientRequestModel request) async {
    try {
      final response = await _apiService.addPatient(request);
      return ApiResult.success(response);
    } catch (e) {
      return ApiResult.failure(e.toString());
    }
  }
}
```

Fig3.23 Repositories for EEG Analysis and Patient Saving

- SendPointRepo is responsible for sending EEG points to the backend and getting analysis results.
- AddPatientRepo handles saving the patient's data and prediction results to the server.

```
@freezed
class SendDataByDoctorState<T> with _$SendDataByDoctorState<T> {
  const factory SendDataByDoctorState.initial() = _Initial;
  const factory SendDataByDoctorState.loadingSendDataByDoctor() =
      LoadingSendDataByDoctor;
  const factory SendDataByDoctorState.successSendDataByDoctor(T data) =
      SuccessSendDataByDoctor<T>;
  const factory SendDataByDoctorState.failureSendDataByDoctor(
      {required String message}) = FailureSendDataByDoctor;

// add patient

  const factory SendDataByDoctorState.loadingAddPatient() = LoadingAddPatient;
  const factory SendDataByDoctorState.successAddPatient(T data) =
      SuccessAddPatient<T>;
  const factory SendDataByDoctorState.failureAddPatient(
      {required String message}) = FailureAddPatient;
}
```

Fig3.24 Doctor Mode State Management

This file defines all possible states for EEG analysis and patient saving using BLoC. It includes success, failure, and loading states for both operations.

```
class SendDataByDoctorCubit extends Cubit<SendDataByDoctorState> {
  SendDataByDoctorCubit(this._sendPointRepo, this._addPatientRepo)
      : super(const SendDataByDoctorState.initial());
  final SendPointRepo _sendPointRepo;
  final AddPatientRepo _addPatientRepo;
  TextEditingController p1 = TextEditingController();
  TextEditingController p2 = TextEditingController();
  TextEditingController p3 = TextEditingController();
  TextEditingController p4 = TextEditingController();
  TextEditingController p5 = TextEditingController();
  TextEditingController p6 = TextEditingController();
  TextEditingController p7 = TextEditingController();
  TextEditingController p8 = TextEditingController();
  TextEditingController p9 = TextEditingController();
  TextEditingController p10 = TextEditingController();
  TextEditingController p11 = TextEditingController();
  TextEditingController p12 = TextEditingController();
  TextEditingController p13 = TextEditingController();
  TextEditingController p14 = TextEditingController();
  TextEditingController p15 = TextEditingController();
  TextEditingController p16 = TextEditingController();
  TextEditingController p17 = TextEditingController();
  TextEditingController p18 = TextEditingController();
  TextEditingController p19 = TextEditingController();
  TextEditingController p20 = TextEditingController();
  sendDataByDoctor() async {
    emit(const SendDataByDoctorState.loadingSendDataByDoctor());
    final response = await _sendPointRepo.sendDataByDoctor(
      SendPointRequestModel(
        arr: [ p1.text,
```

```
            p11.text,
            p12.text,
            p13.text,
            p14.text,
            p15.text,
            p16.text,
            p17.text,
            p18.text,
            p19.text,
            p20.text,
          ],
        ),
      );

      response.when(
        success: (data) {
          emit(
            SendDataByDoctorState.successSendDataByDoctor(data),
          );
        },
        failure: (error) {
          log(error.toString());
          emit(
            SendDataByDoctorState.failureSendDataByDoctor(
                message: error.toString()),
          );
        },
      );
    }
```

```
  addPatient(AddPatientRequestModel request) async {
    emit(const SendDataByDoctorState.loadingAddPatient());
    final response = await _addPatientRepo.addPatient(request);
    response.when(
      success: (data) {
        emit(SendDataByDoctorState.successAddPatient(data));
      },
      failure: (message) {
        emit(SendDataByDoctorState.failureAddPatient(message: message));
      },
    );
  }
}
```

Fig3.25 Doctor Mode Cubit Functions

The Cubit handles two key operations:

- sendDataByDoctor() takes the manually entered EEG points and sends them to the AI model.
- addPatient() saves the patient information along with the prediction results.

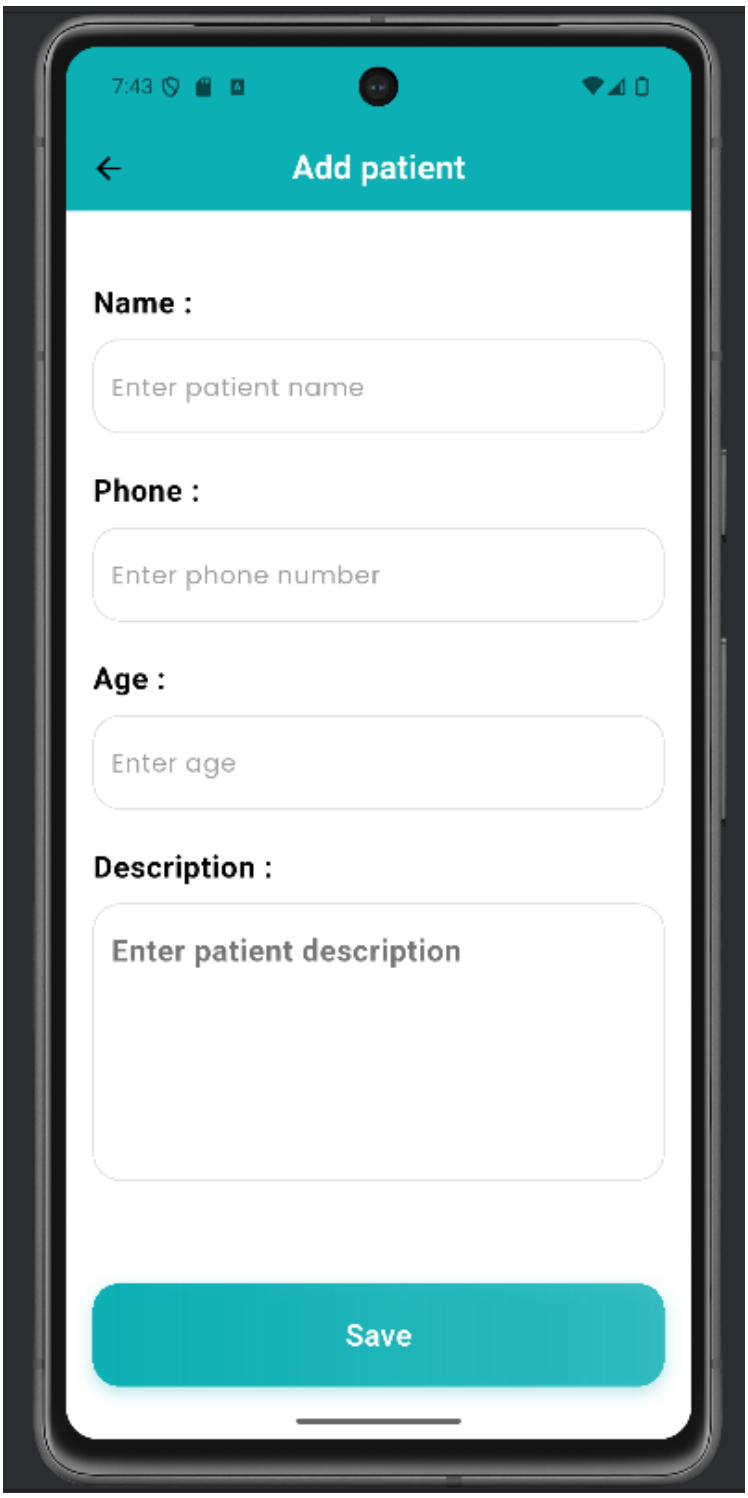


Fig3.26 Add New Patient Screen

3.6.8: History Screen:

```
@override
void initState() {
  super.initState();
  context.read<GetAllPatientsCubit>().getAllPatients();
}
```

Fig3.27 Load Patients Automatically

This line loads all patient records using the getAllPatients() function when the screen is initialized.

```
BlocConsumer<GetAllPatientsCubit, GetAllPatientsState>(
```

Fig3.28 Display and Manage States

Using BlocConsumer, the screen updates based on the cubit state: loading, success (showing patients), or error (showing a message).

```
onPressed: (context) {
  context.read<GetAllPatientsCubit>().deletePatient(patient.id.toString());
},
```

Fig3.29 Delete Patient

Each patient has a delete action. If clicked, it calls deletePatient() to remove the record from the system.

```
onTap: () {
  Navigator.push(
    context,
    MaterialPageRoute( builder: (context) {
        return PatientDetails(patientDetails: patient,);
      },
    ),
  );
},
```

Fig3.30 View Patient Details

When the user taps a patient card, the app navigates to the details screen using Navigator.push().

- Note: After analyzing EEG data and saving a patient's record, doctors can later access detailed information through the *Patient Details* screen. This includes the patient's personal information and full medical history. However, this screen is not the main focus of our documentation, as the core functionality centers around data input, AI analysis, and result display.

```dart
@JsonSerializable()
class Patient {
  final int id;
  final String name;
  final int age;
  final String phoneNumber;
  final DateTime dateOfCreation;
  final int doctorId;

  final PatientHistoryCollection history;

  Patient({
    required this.id,
    required this.name,
    required this.age,
    required this.phoneNumber,
    required this.dateOfCreation,
    required this.doctorId,
    required this.history,
  });

  factory Patient.fromJson(Map<String, dynamic> json) =>
      _$PatientFromJson(json);

  Map<String, dynamic> toJson() => _$PatientToJson(this);
}
```

Fig3.31 Get All Patients Cubit

This model represents the patient and their medical records retrieved from the backend. Each patient has a name, age, phone number, and a list of medical prediction records (like GPD, GRDA, etc.).

- Note: *get all patients repository handles the communication with the API to retrieve and delete patient data. It contains two main functions: getAllPatients() and deletePatient().* (Note: The app also supports patient deletion using BLoC, but it is not documented here to maintain focus.)

```dart
  const factory GetAllPatientsState.loadingDeletePatient() = LoadingDeletePatient;
  const factory GetAllPatientsState.successDeletePatient() = SuccessDeletePatient;
  const factory GetAllPatientsState.errorDeletePatient({required String errorMessage}) = ErrorDeletePatient;
```

Fig3.32 Get All Patients State

This Freezed state class handles the process of loading, success, and failure when retrieving all patients from the server. It is used within the BLoC to manage UI updates based on the API call results.

These functionalities provide doctors with access to patient history for reference, without shifting focus from the application's main goal: EEG analysis and smart diagnosis support.

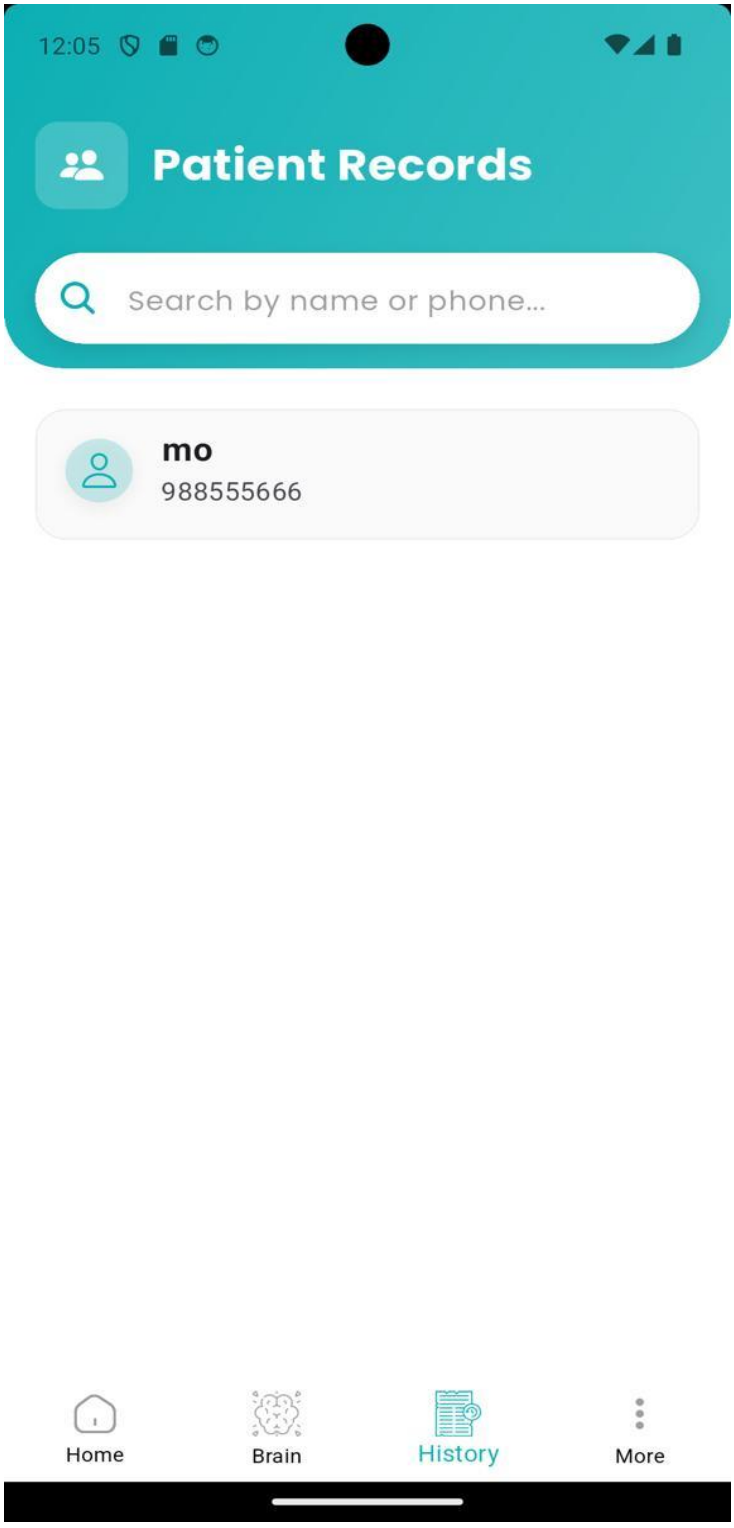


Fig 3.33 History Screen

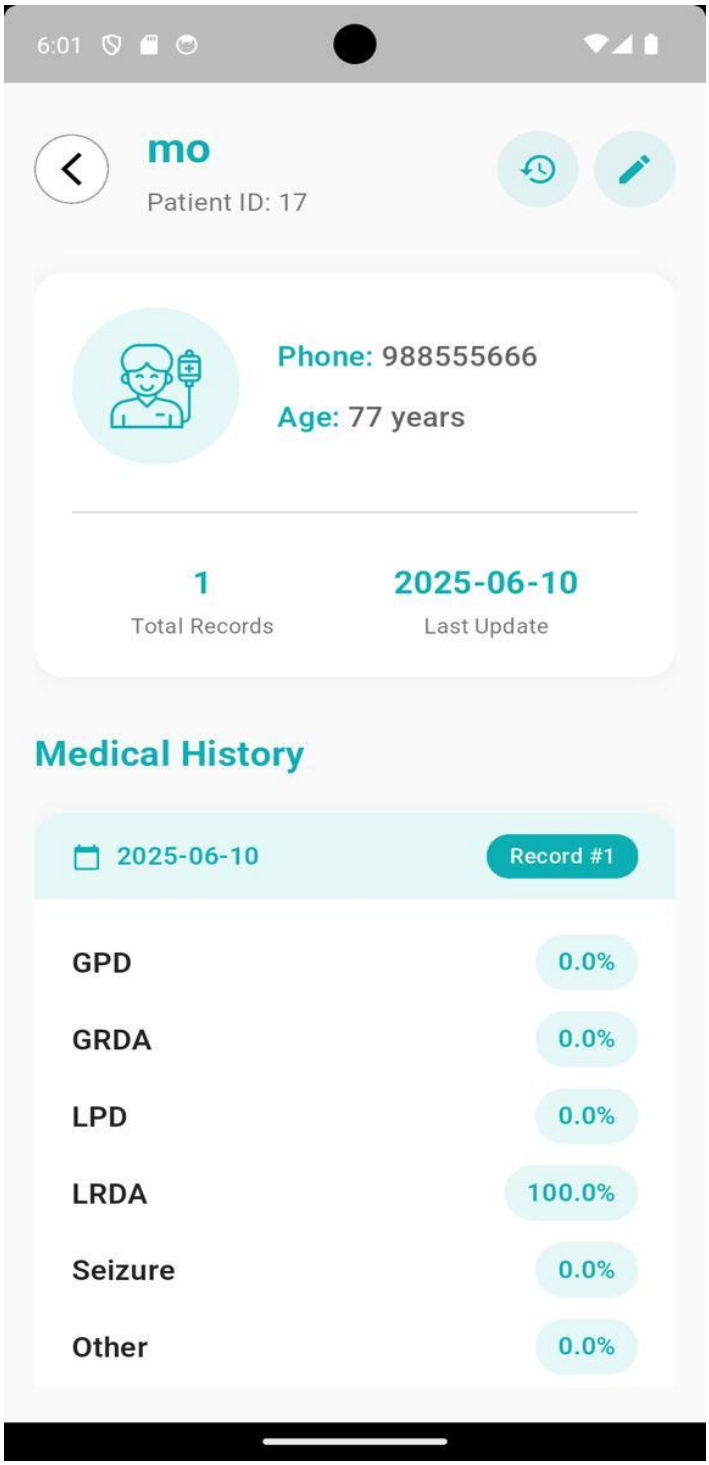


Fig3.34 Patient Details

3.6.9: Brain Screen:

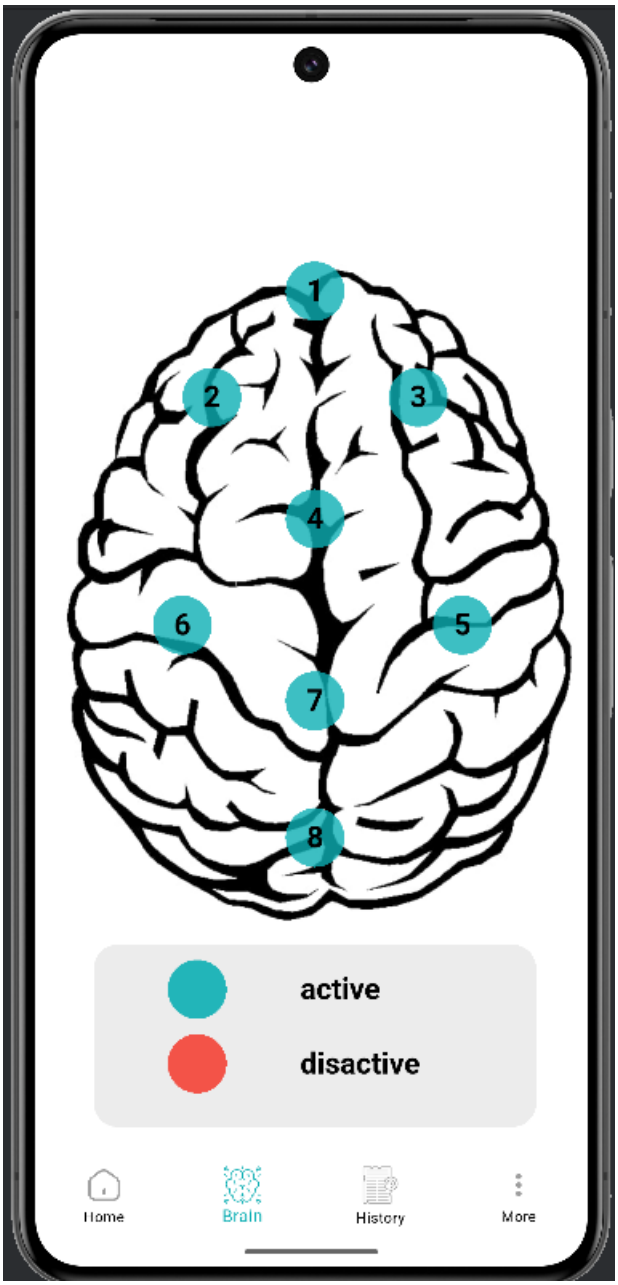


Fig3.35 Brain Screen

This screen shows the active and disactive brain points.

3.6.10: More Screen:

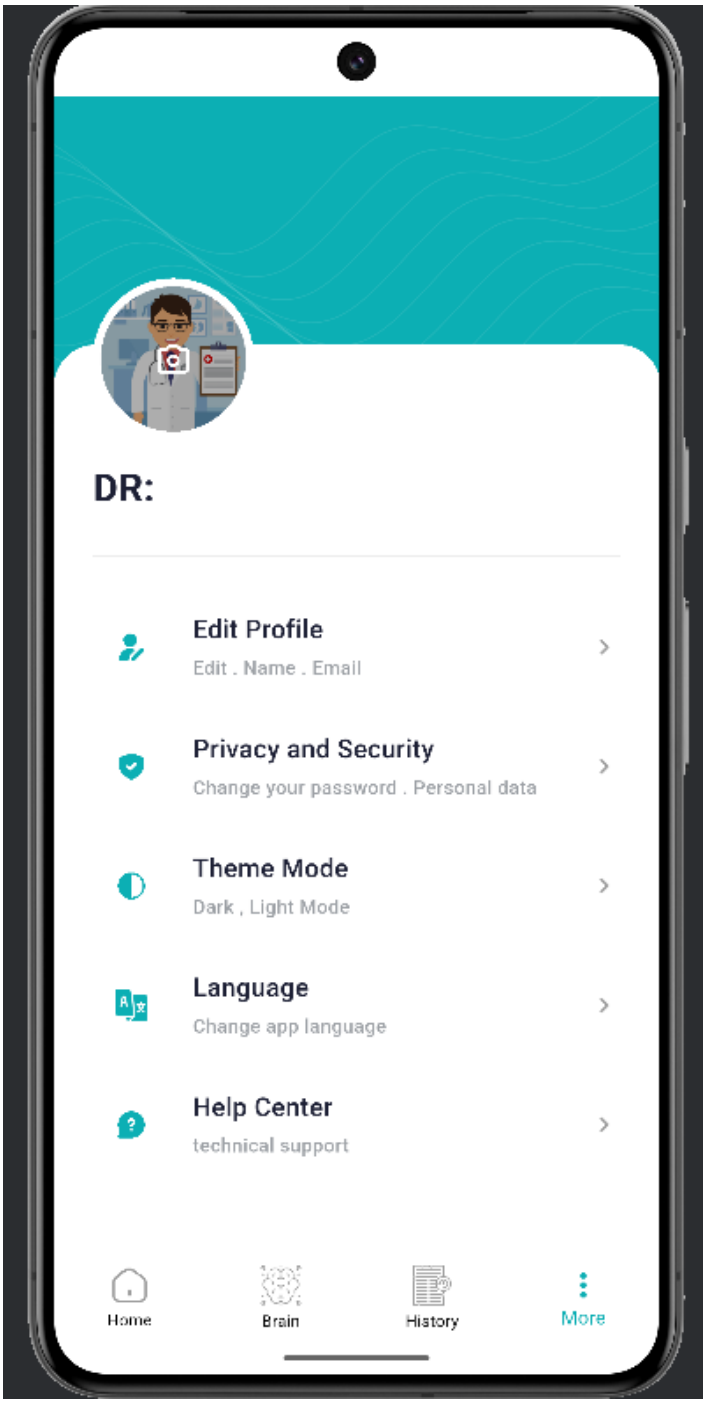


Fig3.36 More Screen

This screen groups advanced user options and app-wide settings

- My Account

  Allows doctors to view and edit their personal info (name, email, phone). Uses EditDoctorCubit to update the backend.

- Change Theme & Language

  Toggles between light/dark mode and Arabic/English using Provider and SharedPreferences.

- Privacy & Security

  Includes:

  - Edit Account Information:

    Doctors can update their name, email, and phone number. This functionality is handled using the EditDoctorCubit, which validates the form input and communicates with the backend API to save the changes.

  - Change Password:

    A secure form allows the doctor to change their password. It requires the current password and confirmation of the new one. The process is managed using ChangePassCubit and calls the backend endpoint Auth/change-Password.

  - Delete Account:

    Doctors can permanently delete their account from the system. This critical operation is performed through DeleteDoctorCubit, which calls the Auth/{id} API and removes the doctor's data after confirmation.

  ➢ These features use Cubit for state management, displaying appropriate loading, success, or error states on the UI.

  ❖ Note:

  - EditDoctorModel and ChangePasswordModel format request/response data.
  - PrivacyRepo and PrivacyRepoImple connect with the API.
  - All actions use Cubit to manage and reflect loading, success, or error states in the UI.

- Help Centre

  Opens a support/help screen for user guidance.

❖ Note:

These settings use Cubit for state management and are essential for user privacy and control, but they are not part of the core AI analysis functionality.

❖ Note:

Internal setup like Dio configuration, dependency injection via GetIt, and Flutter extensions are not covered, as they don't affect core logic or user experience.

❖ Summary:

This chapter presented the complete mobile implementation of the Brain Pulse app, covering UI, logic, Bloc/Cubit state management, and API integration.

While EEG analysis is the core focus, features like patient management and secure account settings significantly improve user experience and system prof.

Chapter 4

Back-End

Chapter 4

Back-End

4.1 Back-End history:

Back-end development has been an essential part of software development since the early days of computing. In the early days, back-end development was focused on managing and processing data on mainframe computers.

As computing evolved, back-end development shifted to managing data on client-server networks. Back-end developers used programming languages such as C++, Java, C# and SQL to manage data and logic on the server-side.

In recent years, back-end development has become more complex. Back-end developers now need to manage data and processing for real-time web applications, mobile applications, and cloud-based services.

Overall, back-end development has evolved over time to meet the changing needs of computing, and it continues to be an essential part of modern software development.

4.2 Back-End definition:

Back-end refers to the part of a software application or system that is responsible for managing and processing data and logic on the server-side. It is the part of the application that is not visible to the end-user but is essential to the functionality and performance of the application.

Back-end is responsible for processing requests from the front-end, which is the part of the application that the user interacts with directly. Back-end communicates with the front-end through APIs (Application Programming Interfaces) to send and receive data. It manages critical tasks such as storing data in databases, processing business logic, ensuring security through authentication and encryption.

back-end relies on technologies like servers (Node.js or Apache), databases (SQL-Server or MongoDB), and frameworks (e.g., Django or .Net) to function effectively. It also ensures scalability and performance, enabling applications to handle large volumes of users or data.

4.3 Essential Skills Required for a Back-End:

3.1 Programming languages: Back-end developers need to be proficient in at least one server-side programming languages such as C#, PHP, Ruby, Python, Java, and Node.js.

3.2 Database management: Back-end developers need to understand how to create and manage databases, including data modeling, schema design, and optimization.

3.3 Frameworks: Back-end developers need to be familiar with popular web application frameworks such as .Net Core, Laravel, Django, Ruby on Rails, and Spring, among others

3.4 APIs: Back-end developers need to know how to work with APIs (Application Programming Interfaces) to integrate different systems and services.

3.5 Security: Back-end developers need to understand security concepts such as authentication, authorization, encryption, and secure coding practices to ensure that applications are secure and not vulnerable to attacks.

3.6 Web servers: Understanding how web servers work is crucial for back-end developers. They need to know how to set up and configure web servers.

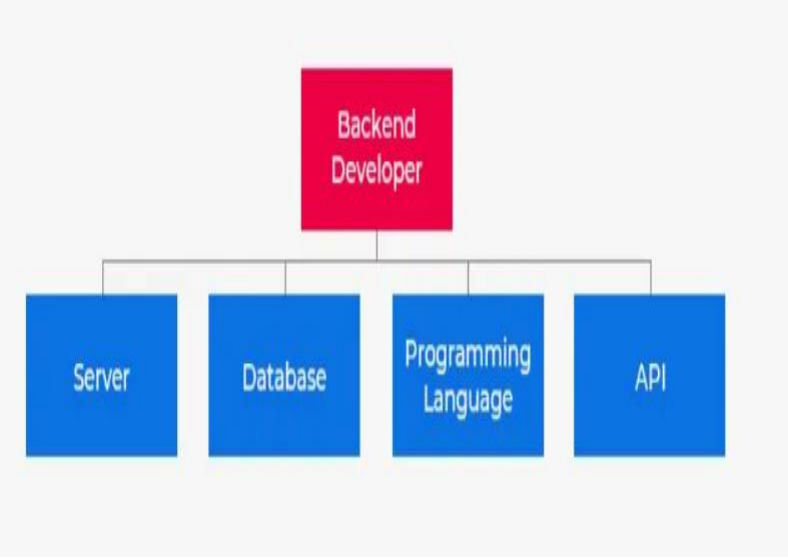


Fig.4.1 Skills of Backend Developer

Fig.4.2 Backend Languages

4.4 What is our project based on?

4.4.1 MS-SQL SERVER:

SQL Server enables organizations to manage large volumes of data efficiently through its relational database architecture, which organizes data into tables with rows and columns. These tables are linked via keys, allowing complex relationships to be defined and queried using Structured Query Language (SQL), specifically Microsoft's Transact-SQL (T-SQL) dialect. T-SQL extends standard SQL with procedural programming features, error handling, and advanced query capabilities, empowering developers to write sophisticated scripts for data manipulation, reporting, and automation. The system supports CRUD operations—create, read, update, and delete—ensuring data can be managed seamlessly for applications .

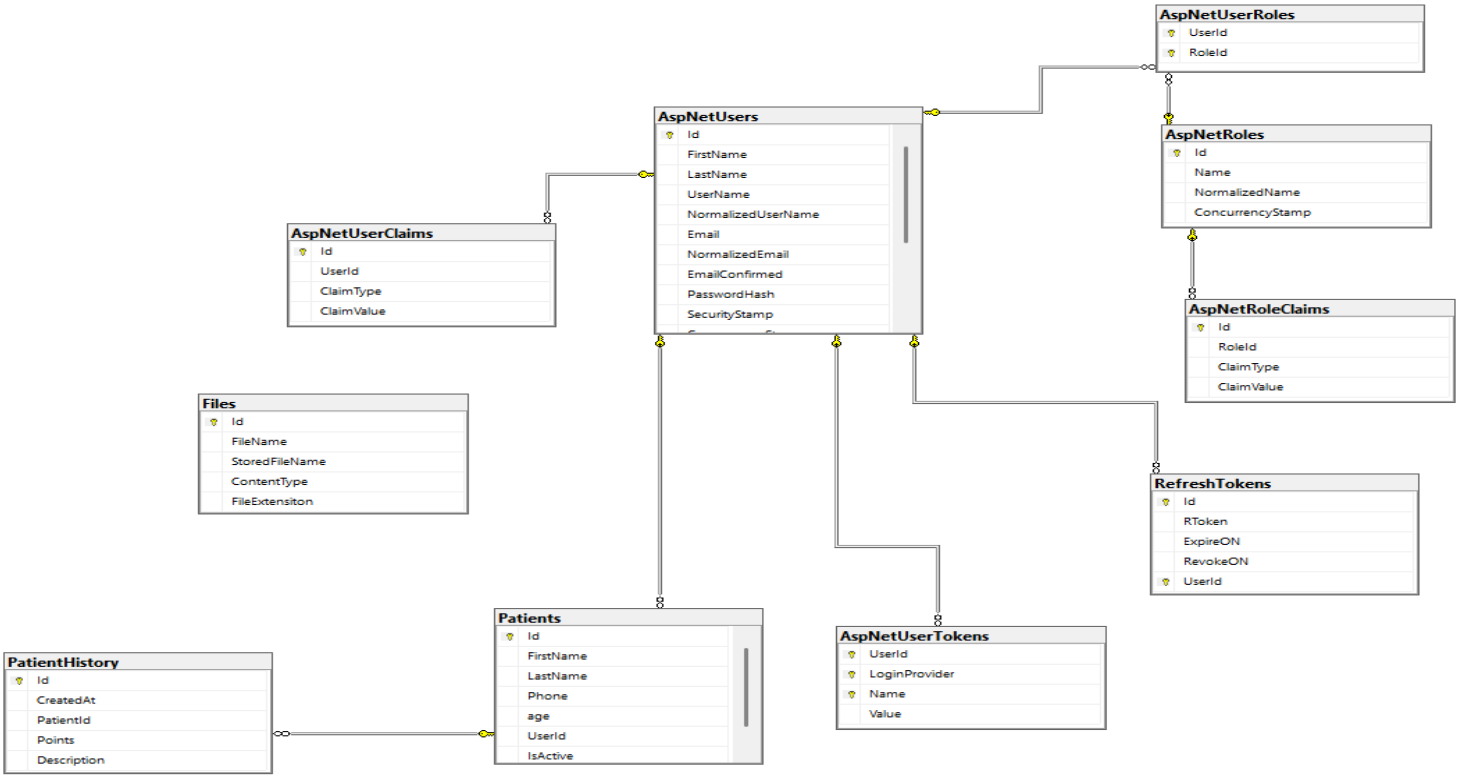


Fig.4.3 Identity Diagram

4.4.2 C#:

As we use .NET Core which is a cross-platform, open-source framework for building modern applications. So we should use C# as our server-side programming languages as it is the primary language used in .NET Core development. C# is a modern, object-oriented programming language that is used to build a wide range of applications, including desktop applications, web applications, and mobile applications. It was developed by Microsoft and is now an open-source language that is widely used in the .NET ecosystem. Here, we use .Net Core 8 that used C# version 11.0.

The language's object-oriented nature promotes modularity and reusability through concepts like classes, objects, inheritance, and polymorphism. Developers can structure code into logical units, making it easier to maintain and scale complex applications. C# also supports interfaces and abstract classes, enabling flexible design patterns that align with modern software architecture principles. Beyond object-oriented programming, C# incorporates functional programming features, such as lambda expressions.

C# prioritizes type safety, ensuring variables are used consistently to reduce runtime errors. Its static typing catches issues at compile time, while features like nullable reference types help prevent null reference exceptions, a common source of bugs. The language's garbage collection, managed by the .NET runtime, automatically handles memory allocation and deallocation, freeing developers from manual memory management while maintaining performance.

C# also supports asynchronous programming with async and await keywords, simplifying the development of responsive, non-blocking applications, such as web servers handling multiple requests concurrently.

4.4.3 LINQ:

LINQ (Language-Integrated Query) is a powerful feature of the C# programming language that allows developers to query and manipulate data from different data sources such as arrays, collections, databases, and XML files, using a unified syntax

At its core, LINQ allows developers to query data using two primary syntaxes: query syntax and method syntax. Query syntax resembles SQL, offering a declarative, readable approach with keywords like from, where, select, and join to filter, sort, and transform data. Method syntax, on the other hand, uses chained method calls, such as Where, Select, or Order-By, leveraging lambda expressions for a more functional programming style. Both syntaxes achieve the same results, with the compiler translating query syntax into method calls under the hood, giving developers flexibility to choose based on preference or readability needs. LINQ operates on any data source that implements the IEnumerable interface for in-memory collections or IQueryable for external sources like databases. This versatility enables LINQ to query arrays, lists, dictionaries, and other collections in memory, as well as connect to databases through providers like Entity Framework Core, which translates LINQ queries into SQL for execution on systems like Microsoft SQL Server.

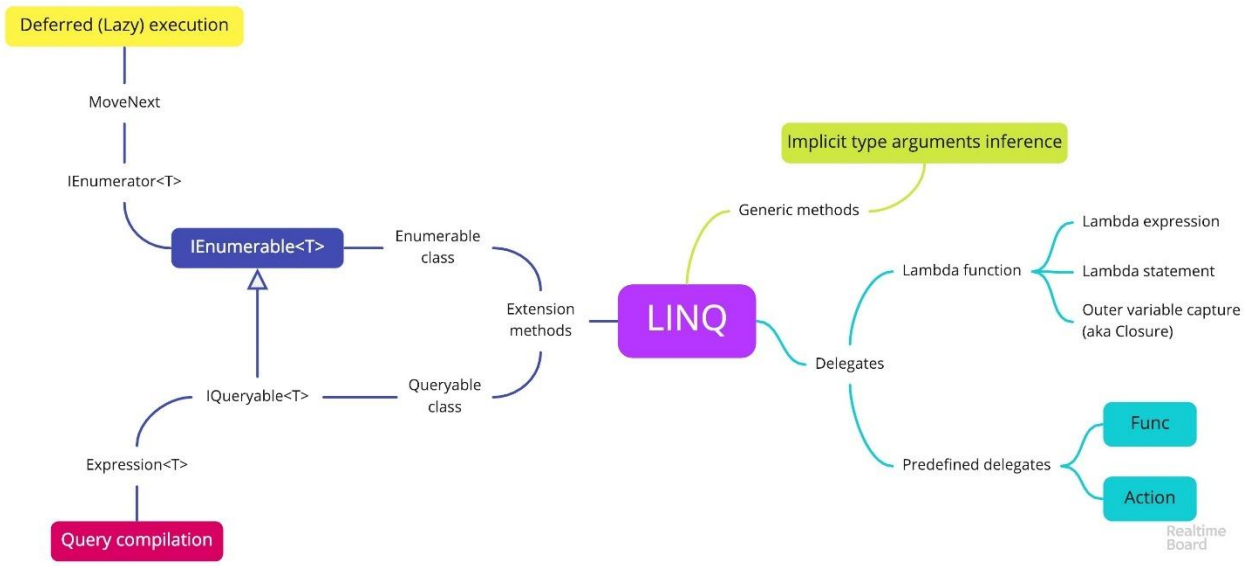


Fig.4.4 LINQ

4.4.4 EF-CORE:

Entity Framework Core (EF Core) is an open-source, lightweight, and cross-platform object-relational mapping (ORM) framework developed by Microsoft for the .NET ecosystem. It simplifies data access in C# applications by allowing developers to interact with databases using C# objects and LINQ queries, abstracting much of the underlying SQL complexity. EF Core enables developers to work with databases using a high-level, object-oriented approach, mapping database tables to C# classes, known as entities, and their relationships to object-oriented constructs.

This abstraction allows developers to write C# code instead of raw SQL to perform CRUD operations—create, read, update, and delete—streamlining back-end development for applications like web APIs, microservices, or desktop software.

EF Core offers two primary workflows for database interaction: Code-First and Database-First. In the Code-First approach, developers define C# classes and configure their relationships using attributes or a fluent API, allowing EF Core to generate or update the database schema automatically via migrations. This approach suits projects where the application model drives the database design, offering flexibility and control. Conversely, the Database-First approach reverse-engineers a database schema into C# classes, ideal for working with existing databases. Both workflows ensure developers can align the data model with application logic efficiently.

Performance is a key focus of EF Core, with features like lazy loading, eager loading, and explicit loading to control how related data is retrieved, minimizing database queries. Change tracking monitors modifications to entities, ensuring only necessary updates are sent to the database, optimizing resource usage. EF Core also supports raw SQL queries for scenarios requiring fine-tuned control, allowing developers to balance abstraction with precision.

The framework supports parameterized queries to prevent SQL injection attacks, ensuring secure data access. Migrations facilitate schema evolution, allowing developers to update database structures without losing data, a critical feature for maintaining production systems.

EF Core's combination of simplicity, power, and flexibility makes it a preferred choice for back-end developers in the .NET ecosystem. It reduces boilerplate code, enhances productivity, and supports modern development practices

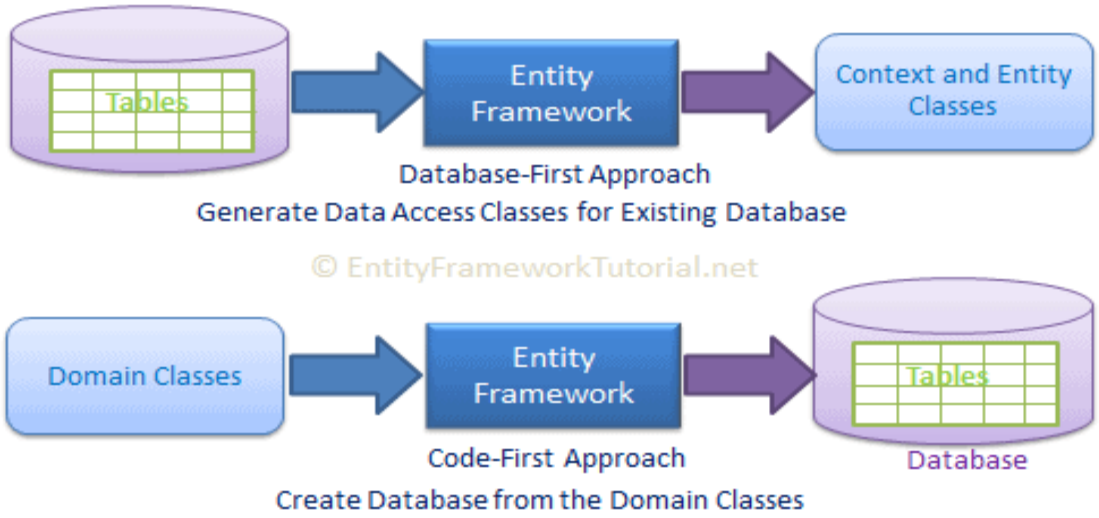


Fig.4.5 EF Core to Approaches

4.4.5 Asp.NET Core:

ASP.NET Core is a cross-platform, open-source web application framework developed by Microsoft.

It allows developers to build modern, high-performance web applications using a variety of technologies, including C#, ASP.NET Razor, and HTML/CSS/JavaScript. ASP.NET Core is designed to be lightweight and modular, making it easy to use and deploy on various platforms, including Windows, macOS, and Linux. It provides a wide range of features and capabilities for building modern web applications, including: - Cross-platform support: ASP.NET Core can be used on multiple platforms, including Windows, macOS, and Linux.

4.4.5.1 Modular architecture:

ASP.NET Core is designed to be modular, making it easy to use and deploy only the necessary components of the framework. - Middleware pipeline: ASP.NET Core uses a middleware pipeline to handle incoming HTTP requests and outgoing responses.

4.4.5.2 Dependency injection:

ASP.NET Core includes a built-in dependency injection system that makes it easy to manage dependencies and increase application scalability.

Understanding RESTful APIs in .NET:

Introduction: A RESTful API (Representational State Transfer API) is an architectural style for building web services that communicate over HTTP using standard HTTP methods (GET, POST, PUT, DELETE, etc.). In the .NET ecosystem, RESTful APIs are commonly built using ASP.NET Core Web API, which is a powerful and lightweight framework designed for creating modern web services and back-end systems.

Key Concepts of REST:

1. Statelessness: Each request from the client contains all the information needed for the server to understand and process it.
2. Resource-Based: Everything is treated as a resource (e.g., user, product, order) and is accessed through a URL (endpoint).
3. HTTP Methods: Each method corresponds to an operation:
4. GET: Retrieve data, POST: Create a new resource, PUT: Update a resource, DELETE: Delete a resource
5. JSON Format: Data is typically exchanged in JSON format for readability and interoperability.

Building a RESTful API in .NET:

In ASP.NET Core, a RESTful API is created using controllers and routing. Each controller manages a specific resource, and the routes define how HTTP requests map to specific actions in the controller.

Main components: are Controllers – Classes that define the API endpoints and logic. Routing – Maps URLs to controller actions. Models – Represent the data used by the API. Dependency Injection – Built-in support to manage services and dependencies. Entity Framework Core – Often used to interact with a database in RESTful APIs.

4.5 Project Security:

4.5.1 Getting started with authentication and authorization:

Authentication and authorization are two important aspects of security. Although these two terms are often used together, they are distinct concepts. Before we dive into the code, it is important to gain an understanding of the differences between authentication and authorization.

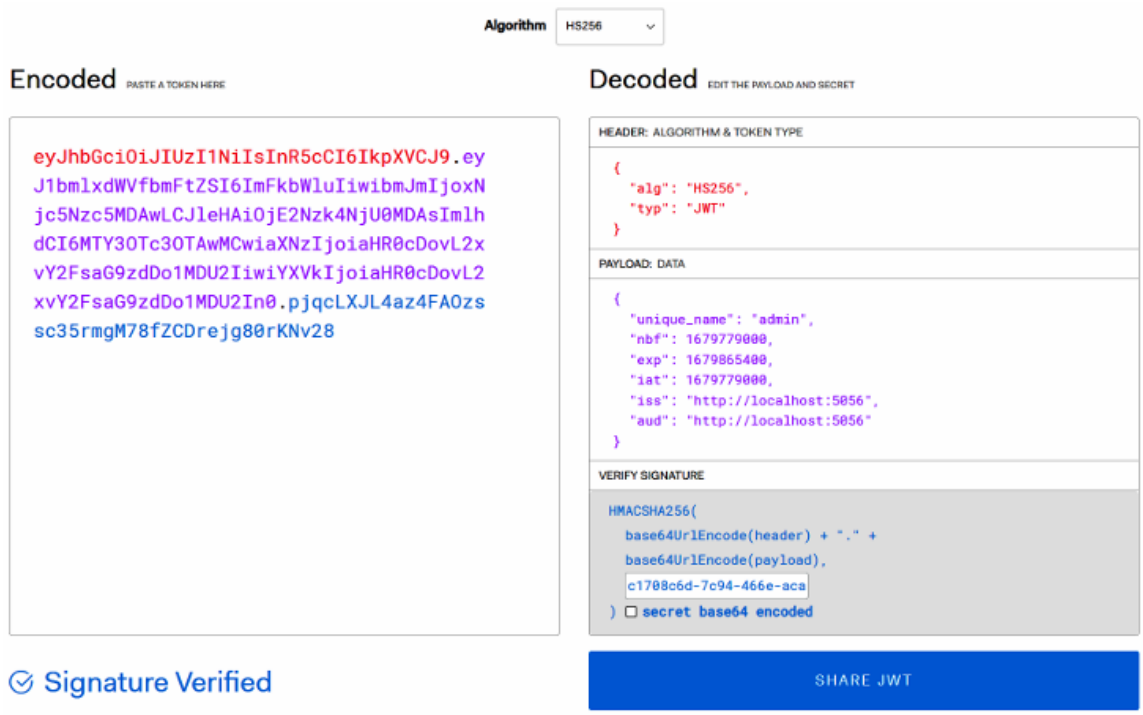
We have already built some web API applications. However, these APIs will be publicly available to anyone who knows the URL. For some resources, we want to restrict access to only authenticated users. For example, we have a resource that contains some sensitive information that should not be available to everyone. In this case, the application should be able to identify the user who is making the request. If the user is anonymous, the application should not allow the user to access the resource. This is where authentication comes into play.

For some scenarios, we also want to restrict access to some specific users. For example, we want to allow authenticated users to read the resource, but only admin users to update or delete the resource. In this case, the application should be able to check whether the user has the required permissions to execute the operation. This is where authorization is used.

Long story short, authentication is used to know who the user is, while authorization is used to know what the user can do. Together, these processes are used to ensure that the user is who they claim to be and that they have the required permissions to access the resource.

4.5.2 What is JWT?

JWT stands for JSON Web Token. It is an industry standard for representing claims securely between two parties. A JWT token consists of three parts: header, payload, and signature. So, typically, a JWT token looks like xxxxx.yyyyy.zzzzz. The header contains the algorithm used to sign the token, the payload contains the claims, and the signature is used to verify the integrity of the token.



The header contains the algorithm used to sign the token. In our case, we use the HmacSha256Signature algorithm. So, the decoded header is as follows:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

The payload contains the claims of the token and some other additional data. In our case, the decoded payload is as follows:

```
{
  "unique_name": "admin",
  "nbf": 1679779000,
  "exp": 1679865400,
  "iat": 1679779000,
  "iss": "http://localhost:5056",
  "aud": "http://localhost:5056"
}
```

There are some recommended (but not mandatory) registered claim names defined in RFC7519:

- `sub`: The `sub` (subject) claim identifies the principal that is the subject of the token
- `nbf`: The `nbf` (not before) claim identifies the time before which the token *must not* be accepted for processing
- `exp`: The `exp` (expiration time) claim identifies the expiration time on or after which the token *must not* be accepted for processing
- `iat`: The `iat` (issued at) claim identifies the time at which the token was issued
- `iss`: The `iss` (issuer) claim identifies the principal that issued the token
- `aud`: The `aud` (audience) claim identifies the recipients that the token is intended for

4.5.3 Authentication Flow:

All API endpoints are protected and require a valid token to access them.

The only exceptions to this rule are: Login, User Registration, Forgot Password These endpoints are publicly accessible because they are part of the authentication process itself.

4.5.4 Authorization Logic:

Once a user is authenticated, we enforce **role-based access control** to determine what they are allowed to do within the system. A primary example of this is with the Doctor role.

4.5.5 Doctor-Specific Access Rules:

A doctor can only access and manage data related to their own patients .A doctor cannot view or interact with other doctors' patients, ensuring: Patient privacy , Data isolation

4.6 Doctor Feature in App:

4.6.1-Login:

Input to the login endpoint

Fig.4.6 Request Record for Login

Here is the response

Fig.4.7 Response Login

4.6.2 Register:

Fig.4.8 Request Record Register

Output for Register End-Point:

Status code OK (201) That inform that the user Record Successfully Created and Confirmation Email Send or Status Code Bad-Request 400 if their email is already exist or any problem found.

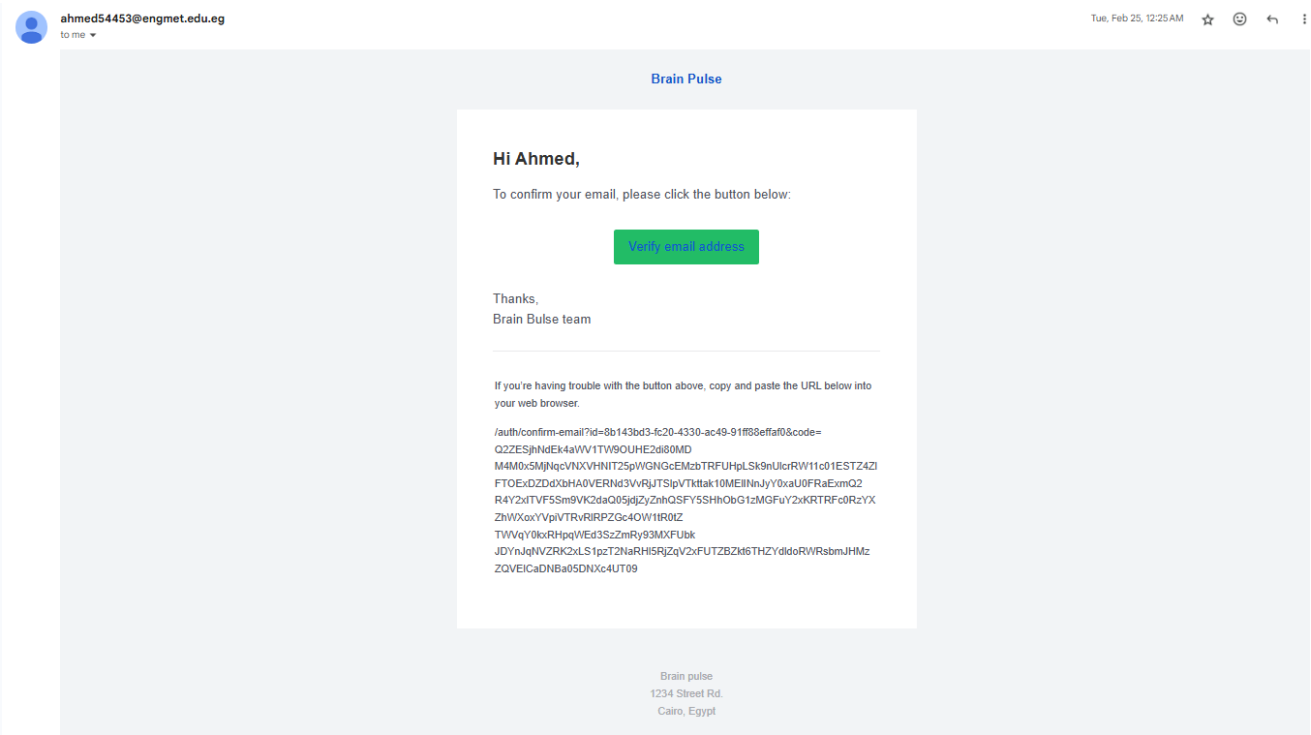


Fig.4.9 Email Send To new Doctors in System

4.6.3 Refresh Token:

A refresh token is a special kind of token that is used to obtain a new access token after the original one has expired — without requiring the doctor to log in again.

Why Use a Refresh Token?

Access tokens (JWTs) usually expire quickly (e.g., in 5 or 10 minutes) to reduce the risk if they are stolen. To avoid making the user log in again and again, the refresh token stays valid for a longer period (e.g., days or weeks).

Here is the Input and Output for Take a new Refresh Token That will make user not log out.

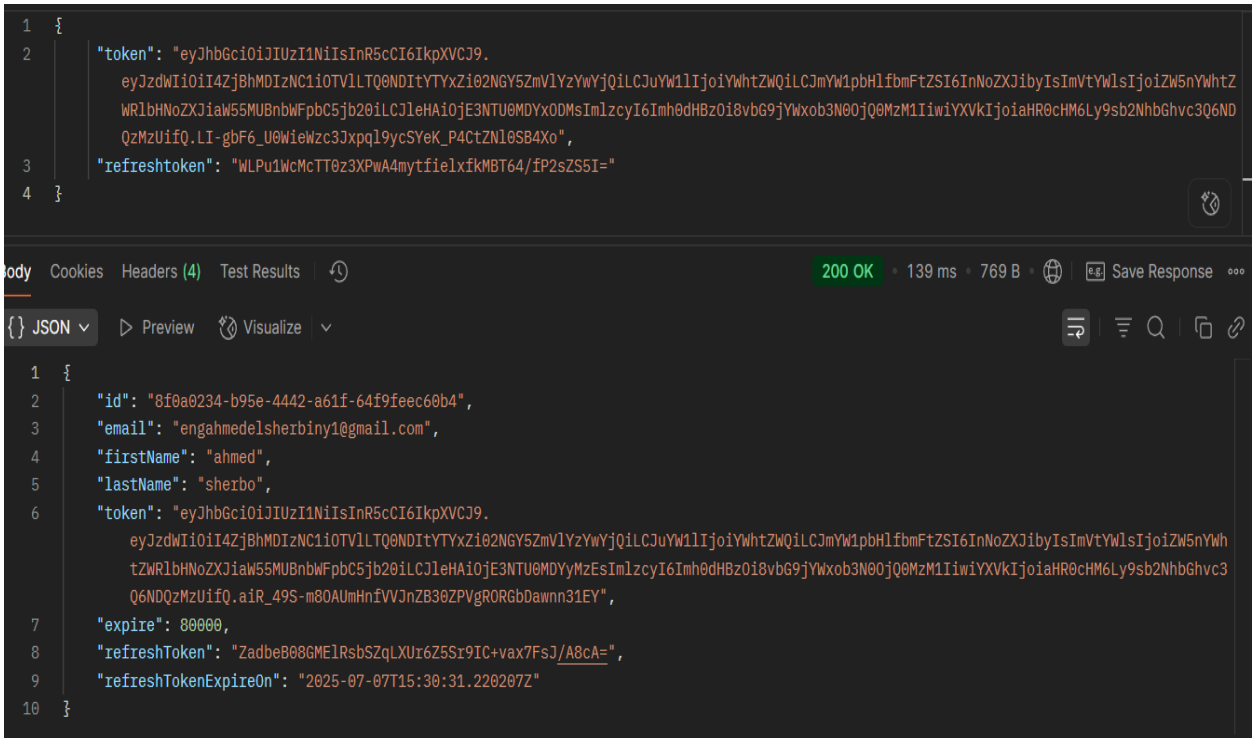


Fig.4.10 new Refresh Token Generation

4.6.4 Add Patient:

Doctor Can patient Easily To our app. The Input for Request are:

Fig.4.11 Add New Record Patient

And the Response is Status Code Ok and Response is:

Fig.4.12 Response with Patient Data

4.6.5 Add –Patient Points:

To Add Points to Patient History the Input Request Should be :

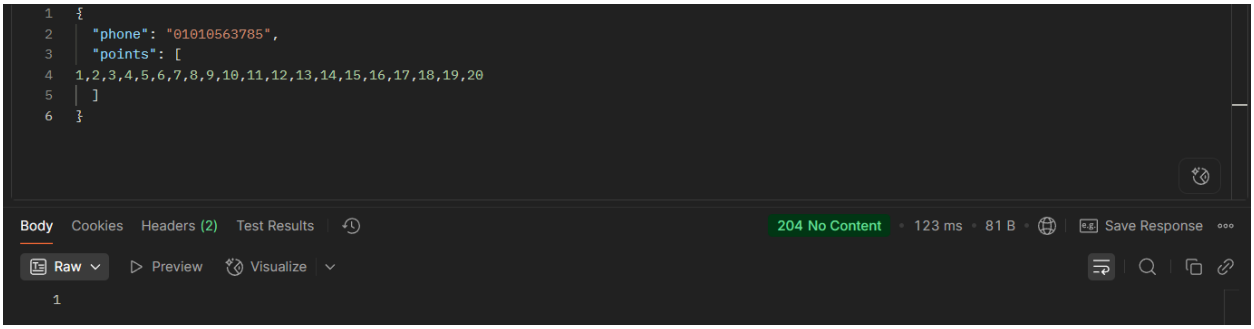


Fig.4.13 Add New Points for Patient

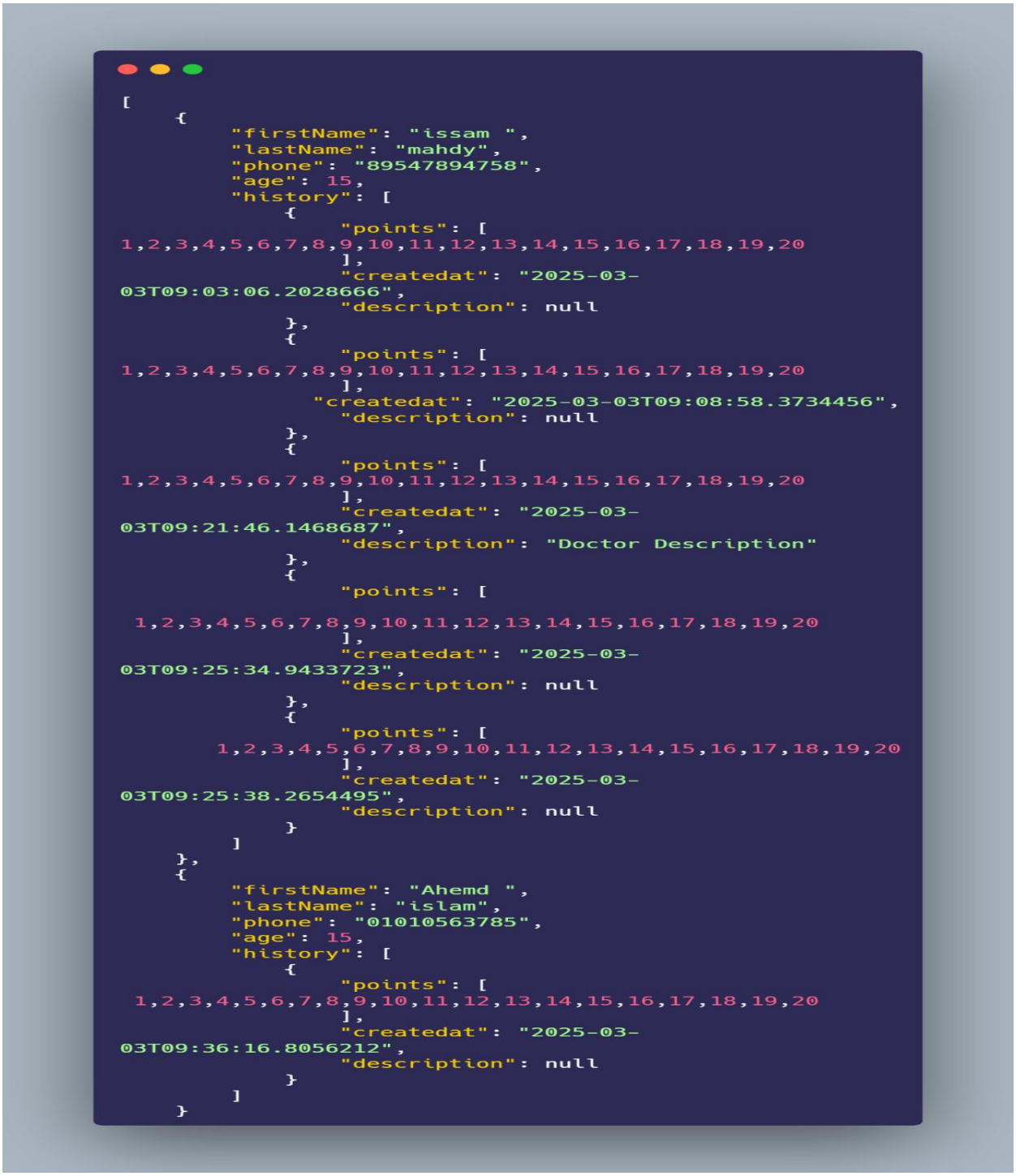
And Status code for this is NO Content 204 that means Created.

4.6.6 Get All History for Specific Patient Using Phone Number:

Fig.4.14 Get All History for Specific Patient Using Phone Number

4.6.7 Get All Patients:

Here is all Patient that related only to the doctor.



```
[
    {
        "firstName": "issam ",
        "lastName": "mahdy",
        "phone": "89547894758",
        "age": 15,
        "history": [
            {
                "points": [
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
                ],
                "createdat": "2025-03-
03T09:03:06.2028666",
                "description": null
            },
            {
                "points": [
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
                ],
              "createdat": "2025-03-03T09:08:58.3734456",
                "description": null
            },
            {
                "points": [
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
                ],
                "createdat": "2025-03-
03T09:21:46.1468687",
                "description": "Doctor Description"
            },
            {
                "points": [
 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
                ],
                "createdat": "2025-03-
03T09:25:34.9433723",
                "description": null
            },
            {
                "points": [
       1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
                ],
                "createdat": "2025-03-
03T09:25:38.2654495",
                "description": null
            }
        ]
    },
    {
        "firstName": "Ahemd ",
        "lastName": "islam",
        "phone": "01010563785",
        "age": 15,
        "history": [
            {
                "points": [
 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
                ],
                "createdat": "2025-03-
03T09:36:16.8056212",
                "description": null
            }
        ]
    }
```


Fig.4.15 Get All Patient for All Patients Related to Specific Doctor

4.6.8 Update Patient Information:

To Update Patient Profile Like name or Phone or age use this End-Point

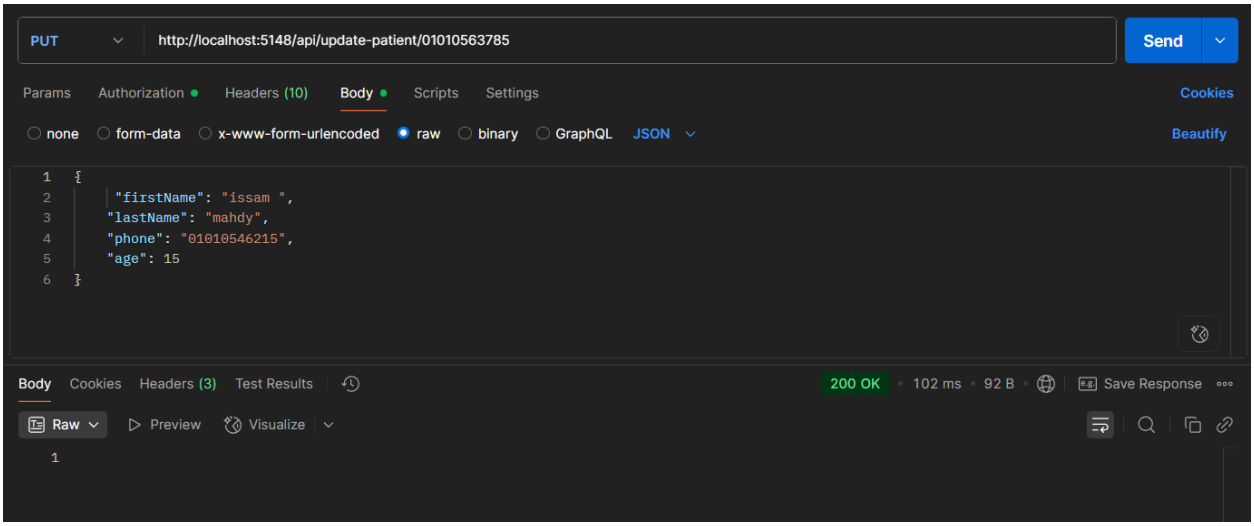


Fig.4.16 Update Patient Information

4.6.9 Doctor Can Delete Patient Profile Using Phone Number:

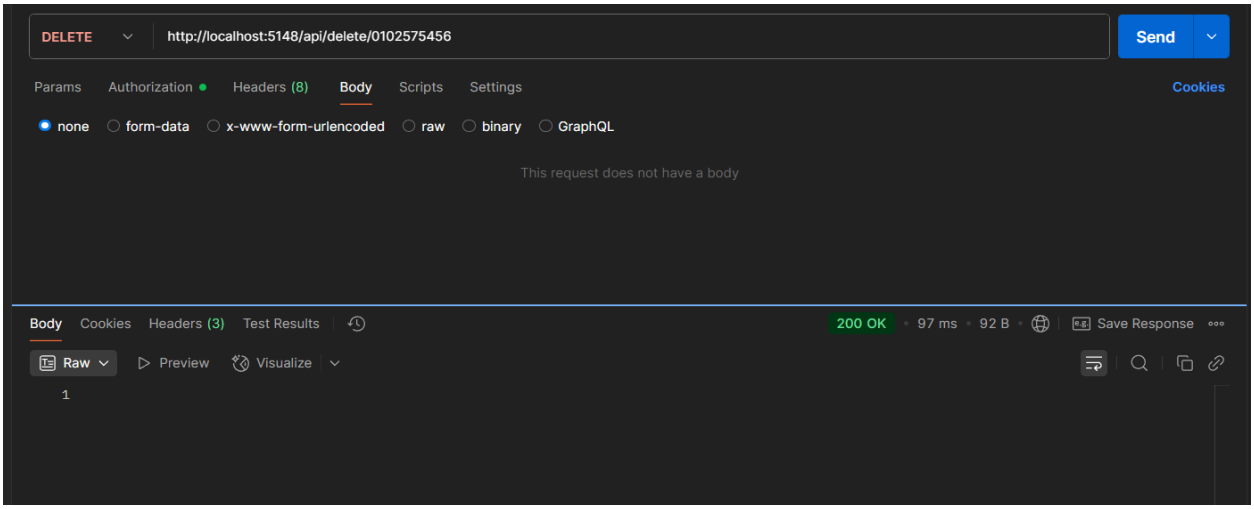


Fig.4.17 Delete Patient Request

4.6.10 Doctor Send Points array To Ai-Model and Get Prediction Result:

In our app doctor Can Send Point Or picture and Get Prediction for The Points.

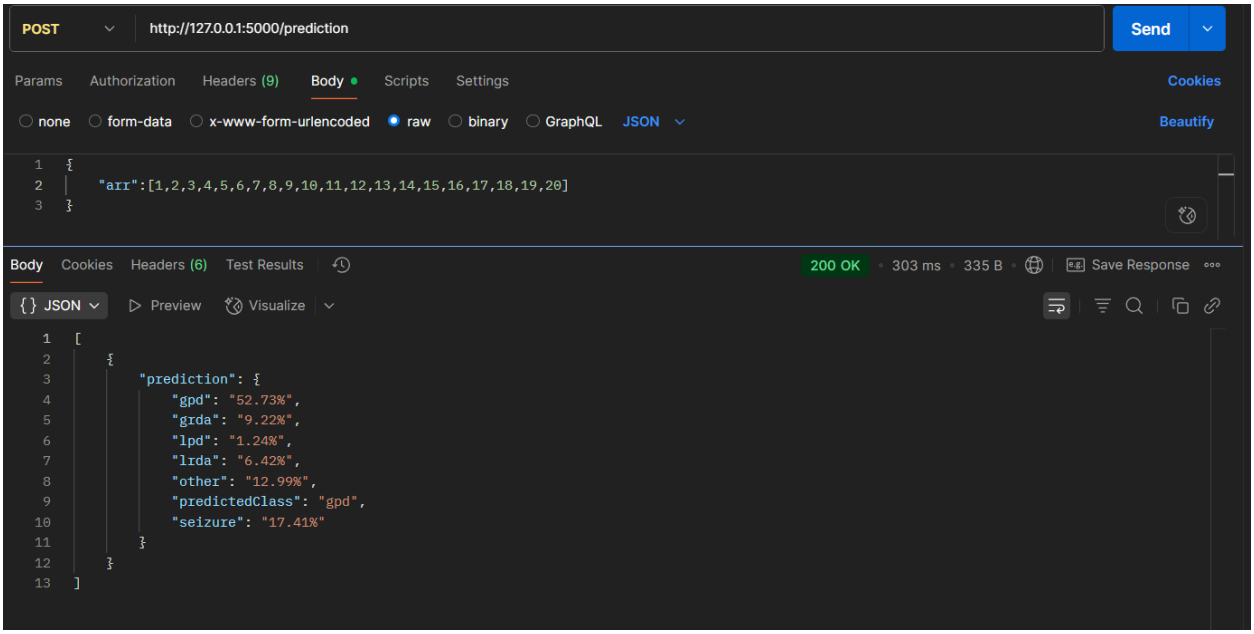


Fig.4.18 Get Prediction from Points

Status codes indicate the result of the HTTP request.

STATUS CODE	EXPLANATION
200 – OK	The request succeeded.
204 – No Content	The document contains no data.
301 – Moved Permanently	The resource has permanently moved to a different URI.
401 – Not Authorized	The request needs user authentication.
403 – Forbidden	The server has refused to fulfill the request.
404 – Not Found	The requested resource does not exist on the server.
408 – Request Timeout	The client failed to send a request in the time allowed by the server.
500 – Server Error	Due to a malfunctioning script, server configuration error or similar.

Fig.4.19 Status Codes from Apis

4.7 How Can I integrate an AI model Using Python with a C# application?

4.7.1 Using ONNX (Open Neural Network Exchange):

ONNX is an open format designed to represent machine learning models. You can export your trained AI model (from frameworks like PyTorch or TensorFlow) to the ONNX format, then load and run it directly within your C# application using libraries like Microsoft. Ml. OnnxRuntime. This approach is ideal for integrating the model fully within your application without relying on external services. It's efficient and works well for real-time or offline predictions.

```
private string Predict(DenseTensor<float> inputTensor)
        {
            // Setup inputs and outputs
            var inputs = new List<NamedOnnxValue>
            {
                NamedOnnxValue.CreateFromTensor(_modelInputName, inputTensor)
            }.AsReadOnly();

            // Run the model
            using var results = _session.Run(inputs);

            // Postprocess to get label
            var vector = results.First().AsTensor<float>().ToArray();
            int pred_idx = Array.IndexOf(vector, vector.Max());

            string label = GetLabelFromIndex(pred_idx);
            return label;
        }
        //Create new method to convert the index to label
        private string GetLabelFromIndex(int index)
        {
            Dictionary<string, int> labels = new Dictionary<string, int>
            {
                { "Seizure", 0 },
                { "GPD", 1 },
                { "LRDA", 2 },
                { "Other", 3 },
                { "GRDA", 4 },
                { "LPD", 5 }
            };

            foreach (var entry in labels)
            {
                if (entry.Value == index)
                {
                    return entry.Key;
                }
            }

            return "Unknown"; // Handle the case where the index doesn't match
any label
        }
```

Fig.4.20 Using ONNX With Points for Prediction

```
[HttpPost("predictImage")]
        public IActionResult PredictImage(IFormFile imageFile)
        {
            if (imageFile == null || imageFile.Length == 0)
            {
                return BadRequest("No image file uploaded.");
            }

            // Save the image to a temporary file
            var tempFilePath = Path.GetTempFileName() + ".png";
            using (var stream = new FileStream(tempFilePath, FileMode.Create))
            {
                imageFile.CopyTo(stream);
            }

            // Preprocess the image
            DenseTensor<float> inputTensor;
            try
            {
                inputTensor = ImagePreprocessor.ProcessSpecFromImage(tempFilePath);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $"Image processing error: {ex.Message}");
            }

            // Make the prediction
            string predictionLabel;
            try
            {
                predictionLabel = Predict(inputTensor);
            }
            catch (Exception ex)
            {
                return StatusCode(500, $"Prediction error: {ex.Message}");
            }

            // Clean up the temporary file
            System.IO.File.Delete(tempFilePath);

            return Ok(predictionLabel);
        }
```

Fig.4.21 Using ONNX With Image for Prediction

4.7.2. Using a Web API (e.g., Flask in Python):

Another common approach is to expose the AI model through a web API. You can use Flask in Python to create an API that loads your model and handles prediction requests. Your C# application then communicates with this API over HTTP.

To make this setup production-ready, you should:

Containerize the Flask application using Docker – this ensures consistency across environments. and use a requirements.txt file to define the Python dependencies and host the container on a server

This method is especially useful when the AI model needs to stay in Python or when you want to separate the model from your main application for easier updates and scalability.

Chapter 5

Artificial Intelligence

Chapter 5
Artificial Intelligence

5.1 Introduction

5.1.1 Background

Let's dive deeper into the significance of Electroencephalography (EEG). Imagine trying to understand a very complex language without a dictionary. EEG signals are that language, representing the electrical activity generated by the interaction of billions of neurons in the brain. These signals are captured via a set of electrodes placed on the scalp according to a standard international system like the 10-20 system. Each electrode records the electrical potential difference in a specific brain region.

In a clinical context, the role of EEG is not limited to initial diagnosis; it extends to continuous monitoring, especially in intensive care units (ICUs). In cases of coma or severe brain injuries, the pattern of EEG signals can provide vital indicators of a patient's neurological state, such as the presence of subclinical seizures or the progression of encephalitis.

However, the biggest challenge lies in interpreting these signals. EEG signals are essentially time-series data, which are very noisy and affected by many factors like eye movement, muscle contractions, and even electrical signals from other medical devices. Deciphering these signals requires years of experience and specialized expertise, which limits the number of clinicians capable of reading EEGs.

This is where machine learning and deep learning come in. These technologies offer a promising solution for analyzing vast amounts of complex data at a rapid pace. Instead of relying on the human eye to identify specific patterns, we can train mathematical models to automatically and objectively detect these patterns.

5.1.2 Problem Statement

To illustrate the problem further, imagine you are looking at a screen displaying EEG data for a patient in the ICU. The screen shows oscillating lines representing signals from dozens of electrodes. As a human analyst, you must distinguish specific patterns such as delta waves, which are low-frequency (0.5−4Hz) and high-amplitude (>20μV), or **spikes** that indicate epileptic activity.

The problem is twofold:

1. **Subjectivity and Variability:** The interpretation of the same signal can vary from one clinician to another. What one doctor sees as "focal activity" another might see as "noise." This variability affects diagnostic accuracy.
2. **Challenge in Subtle Distinctions**: Some patterns are very similar. For example, distinguishing delta activity caused by a pathological condition (like brain dysfunction) from natural delta activity during deep sleep can be challenging. Similarly, differentiating focal discharges from certain artifacts caused by patient movement is a significant challenge.

Therefore, there is a critical need for a system characterized by objectivity, speed, and accuracy to serve as a powerful assistive tool for clinicians.

5.1.3 Project Objectives

This project aims to develop an intelligent system, and here is how we will achieve it practically:

1. Convert Signals to Images (Spectrograms):

- EEG signals are originally time-series data, meaning they change over time.
- Convolutional Neural Networks (CNNs) have shown exceptional efficiency in analyzing images.
- Therefore, the first step is to convert the EEG data from its temporal form into a visual one that a CNN can understand. This is done using signal processing techniques such as the Short-Time Fourier Transform (STFT) or the Wavelet Transform.
- The result is a spectrogram, which is a 2D image where the horizontal axis represents time, the vertical axis represents frequency, and the color of each point represents the signal's intensity (amplitude) at that time and frequency. This way, we can "see" complex patterns like changes in frequency over time.

2. Build a CNN Model:

- After converting the signals to images, we will build a Convolutional Neural Network (CNN). This network consists of sequential layers that learn features automatically.
- The initial layers learn simple features like edges and lines, while deeper layers learn more complex features like patterns specific to each clinical category (e.g., the pattern of spikes or slow activity).
- The goal is to build a model capable of extracting these distinctive features from the spectrograms to classify them accurately.

3. Evaluate with Clinical Data:

To ensure the model's effectiveness, it must be trained and evaluated on real data from patients.

We will use a large clinical dataset that has been pre-labeled by specialists.

We will measure the model's performance using metrics such as accuracy, precision, recall, and the F1-score. These metrics provide a comprehensive view of how well the model classifies each of the six categories.

4. Assist in Diagnosis:

The ultimate goal is not just to build a model but to create a practical tool.

This system will function as a decision support tool, alerting clinicians in real-time to the presence of potentially harmful brain activity, allowing for faster intervention.

5.1.4 Importance of Study

The discussion about the project's importance is not just about the technical side. The biggest impact will be on the clinical side.

- **Reduce Workload**: Instead of spending hours reviewing long EEG recordings, a clinician can review only the segments that the system classifies as "abnormal" or "critical," which saves valuable time.
- **Minimize Human Error:** An automated system ensures consistency in evaluation, as it will apply the same criteria to all signals, reducing the impact of variability among human analysts.
- **Accelerate Diagnosis in Emergencies:** In cases like status epilepticus, every minute of delay in diagnosis and treatment can cause permanent brain damage. An automated system that can immediately alert to a continuous seizure can save a patient's life.

In summary, this study represents a practical application of Artificial Intelligence in medicine, combining advanced neurological expertise with the power of deep computing to provide innovative solutions to real-world problems.

5.2 Literature Review

5.2.1. EEG Signal Classification Techniques

Classifying EEG signals has long been a significant challenge in the field of biomedical engineering. A comparative analysis of traditional and modern methodologies is essential to understand the evolution of this domain.

5.2.1.1 Traditional Approaches

These approaches rely on **manual feature extraction**, a process that requires human expertise and is executed in two primary steps:

5.2.1.2 Feature Extraction:

- Experts manually extract statistical or frequency-based features from the signal.
- **Frequency Band Features:** The EEG signal is divided into specific frequency bands, each with its own clinical significance:
  - **Delta waves (0.5−4Hz):** Associated with deep sleep but can indicate severe brain pathologies when present during wakefulness.
  - **Theta waves (4−8Hz):** Associated with drowsiness and can point to cerebral dysfunction.
  - **Alpha waves (8−12Hz):** Associated with a state of relaxed wakefulness.
  - **Beta waves (12−30Hz):** Associated with attention and mental activity.
  - **Gamma waves (>30Hz):** Associated with complex cognitive tasks.

- **Statistical Features**: Statistical measures such as energy, entropy, variance, and mean amplitude are calculated for the signal.
- This process is time-consuming, expert-dependent, and lacks consistency across different datasets.

Classification:

- The extracted features are then fed into conventional machine learning classifiers, including:
  - **Support Vector Machines (SVM):** Aims to find the optimal hyperplane to separate data classes.
  - **k-Nearest Neighbors (k-NN):** Classifies a data point based on the most common class among its nearest neighbors.
  - **Decision Trees:** Sequentially partitions the data based on features to determine the class.
- The accuracy of these classifiers is entirely dependent on the quality of the manually extracted features.

The Shift to Deep Learning

The emergence of deep learning has brought about a paradigm shift. Instead of relying on experts for feature extraction, deep neural networks can learn features automatically from raw or minimally preprocessed data. This is known as automatic feature extraction.

The most prominent network architectures used are:

- **Convolutional Neural Networks (CNNs):**
    - These networks are highly effective at processing data with a spatial structure, such as images.
    - They use convolutional layers with learned filters to identify specific patterns and features from the data.
    - By converting the EEG signal into a spectrogram, it is transformed into an image-like representation, making it ideal for CNNs.
- **Recurrent Neural Networks (RNNs):**
    - These networks are designed to handle sequential data like time series.
    - They can capture temporal dependencies, a crucial feature in EEG signals where one pattern follows another.
    - Variants like LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) are popular for this purpose.

5.2.2 Deep Learning in Medical Applications

Deep learning has revolutionized medical diagnostics across various fields:

- **Radiology:** CNN models can detect tumors in MRI and CT scans with accuracy comparable to radiologists.
- **Cardiology:** RNNs and CNNs can analyze electrocardiograms (ECG) to identify arrhythmia patterns.
- Pathology: Analyzing tissue images to detect cancerous cells.

In the context of EEG, a primary reason for the effectiveness of CNNs is the conversion of the signal into an image. As mentioned in Chapter 1, the EEG signal is transformed into a spectrogram which visualizes the frequency content of the signal and its evolution over time. To a CNN, this spectrogram is simply an image with visual patterns it can learn. For instance, epileptic activity might appear as vertical lines (spikes) within a specific frequency band, a visual pattern that a CNN can detect.

Additionally, several key techniques enhance the robustness of deep learning models:

- **Batch Normalization:** Speeds up training and improves stability by normalizing the outputs of each layer.
- **Dropout:** Prevents overfitting by randomly dropping out neurons during training, forcing the network to learn more generalized features.
- **Data Augmentation:** Creates additional training samples by making slight modifications to existing data, which increases the dataset size and makes the model more robust.

5.2.3 Related Works

Several previous studies have explored EEG signal classification using machine learning and deep learning.

- **Zhang et al. (2021**) proposed a CNN-based model for emotion recognition from EEG signals, achieving state-of-the-art accuracy using spectrogram inputs. This work validates the feasibility of using spectrograms as inputs for CNNs.

- **Roy et al. (2020)** focused on seizure prediction using deep RNNs, highlighting the importance of temporal dependencies in EEG data. This demonstrates that RNNs can be suitable for tasks requiring the tracking of long-term changes.
- **Schirrmeister et al. (2017)** introduced DeepConvNet, a CNN model trained directly on raw EEG data for brain-computer interface tasks. This approach minimizes preprocessing steps but requires a highly specialized network architecture.

Distinctions of This Project:

While these studies demonstrated the potential of deep learning in EEG analysis, most of them focused on:

- **Limited classes:** For example, classifying only two states (normal/abnormal) or a single task like seizure prediction.
- **Synthetic or limited datasets:** Which may not reflect the variability and noise present in real-world clinical data.

This project distinguishes itself by:

1. Targeting six distinct clinical categories, making it more complex and relevant to a real clinical environment.
2. Utilizing real clinical EEG data, which enhances the model's reliability and applicability in hospital settings.
3. Combining the power of signal visualization (spectrograms) with the deep learning capabilities of CNNs to provide a comprehensive solution.

5.3: Dataset Description

5.3.1 Source of Data

The dataset utilized in this project is sourced from the HMS - Harmful Brain Activity Classification competition hosted on Kaggle. This is a crucial aspect of the study, as the data consists of real-world EEG recordings collected from clinical environments, meticulously annotated by certified neurological experts. Sourcing data from such a realistic context is paramount for developing and validating deep learning models intended for genuine medical diagnostic applications. The expert-labeled ground truth provides a high-quality foundation for training a robust and reliable model.

5.3.2 Data Format and Classes

The EEG recordings are provided in Parquet file format, with each file representing a contiguous 600-second (10-minute) segment of EEG monitoring. A key characteristic of this dataset is that the raw signals have already been pre-processed and converted into time-frequency spectrograms.

What are spectrograms?

A spectrogram is a visual representation of a signal's frequency content over time. While the original signal is a 1D waveform (amplitude vs. time), the spectrogram is a 2D image where the x-axis represents time, the y-axis represents frequency, and the pixel intensity (or color) at each point indicates the signal's amplitude at that specific time and frequency. This visual transformation makes the data highly suitable for processing by Convolutional Neural Networks (CNNs), as the network can "see" complex patterns, such as frequency shifts or bursts of activity, within a visual context.

The dataset is annotated into six distinct clinical classes, each representing a specific pattern of brain activity:

1. **GPD (Generalized Periodic Discharges):** Repetitive, generalized discharges occurring across all brain regions.
2. **GRDA (Generalized Rhythmic Delta Activity):** Widespread rhythmic activity in the delta frequency band.
3. **LPD (Lateralized Periodic Discharges):** Periodic discharges confined to one hemisphere of the brain.
4. **LRDA (Lateralized Rhythmic Delta Activity):** Rhythmic delta activity confined to one hemisphere.
5. **Other:** A catch-all class for non-specific or mixed patterns that do not fit into the other categories.
6. **Seizure:** Discrete seizure episodes identified within the EEG signal.

Distinguishing between some of these patterns, such as LPD and LRDA, can be highly challenging even for human experts, which underscores the complexity of the automated classification task.

5.3.3 Challenges in the Dataset

These challenges are inherent to real-world clinical data and are essential to address for model reliability.

- A. Class Imbalance:

This is a common issue in medical datasets. The occurrence of a "Seizure" or "LPD" event is significantly less frequent than "GRDA" or "Other" patterns. If not addressed, a model trained on this data would become biased towards the more prevalent classes, leading to poor performance in classifying rare but critical events like seizures. This necessitates specific techniques during training to mitigate bias.

- B. Signal Complexity:

EEG spectrograms are non-stationary, meaning their statistical properties change over time. Patterns like GPD and LPD may have similar frequency content but differ in their spatial distribution across the scalp, making discrimination challenging. This complexity requires a model with a high capacity for learning intricate patterns through multiple convolutional layers.

- C. Spectrogram Size Variation:

Although all recordings have a consistent duration of 600 seconds, the dimensions of the resulting spectrograms may vary slightly due to factors like frequency binning or sampling parameters. This dimensional inconsistency can disrupt the input pipeline of a neural network, as CNNs require uniform input sizes. Therefore, standardizing the size of all spectrograms is a necessary preprocessing step.

- D. Clinical Noise:

Data collected in a hospital environment is prone to artifacts. These can be caused by patient movement (e.g., blinking, muscle contractions), machine noise from adjacent equipment, or electrode displacements. Such noise can obscure the underlying brain activity and negatively impact model performance, necessitating robust preprocessing or model architecture to handle it.

5.3.4 Preprocessing Needs

To address the aforementioned challenges, the dataset underwent several critical preprocessing steps:

- Stratified Sampling:

This technique is vital for tackling class imbalance. Instead of random sampling, it ensures that the proportion of each class in the training and validation sets is preserved, mirroring its distribution in the original dataset. This ensures the model is trained on a representative sample of all classes, including rare ones.

- Normalization of Spectrogram Pixel Values:

The pixel values within the spectrograms are scaled to a uniform range (typically 0 to 1). This standardizes the contrast and brightness across all images, which significantly speeds up the neural network training process and improves performance.

- Padding and Resizing:

To resolve the issue of varying spectrogram sizes, all images are unified to a fixed dimension, such as 300x100 pixels. This is achieved by either resizing the image or adding empty pixels (padding) around it to reach the target size, ensuring consistent inputs for the neural network.

- Label Encoding:

The class labels (e.g., "Seizure," "GPD") are originally text strings. For the neural network to process them, they are converted into numerical IDs (e.g., "Seizure" = 0, "GPD" = 1), which is a standard requirement for model compatibility.

5.4: System Design and Methodology

5.4.1 Data Preprocessing Pipeline

Before the deep learning model could process the data, the raw EEG signals had to be transformed into a format suitable for image-based learning. This pipeline is a crucial step to ensure high-quality inputs, which directly influences the model's performance.

A. Spectrogram Conversion:

The raw EEG signals, which are 1D time-domain waveforms, were converted into 2D time-frequency spectrograms.

- **How it works:** This is typically done using the Short-Time Fourier Transform (STFT). The STFT breaks the long signal into short, overlapping time segments and applies a Fourier Transform to each segment. The result is a frequency spectrum for each time slice.
- **The Output:** These spectra are then stacked together to form a visual image. The horizontal axis represents time, the vertical axis represents frequency (e.g., delta, theta, alpha bands), and the color intensity of each pixel represents the signal's energy at that specific frequency and time.
- **Why it's essential:** This transformation allows us to leverage the power of CNNs to recognize visual patterns—like the vertical lines of "spikes" or the colored blobs of "delta" activity—which represent critical clinical features.

C. Padding and Resizing:

As noted in the previous chapter, the dimensions of the generated spectrograms can vary slightly.

- **How it works:** All spectrograms were resized to a consistent dimension of 300x100 pixels. If an image was smaller, padding was applied around it with empty pixels (usually black or zero-valued). If it was larger, it was cropped or resized to fit the target dimensions.
- **Why it's essential:** CNNs with a fixed architecture require uniform input sizes for efficient batch processing. This step is a prerequisite for feeding data into the model.

D. Stratified Train-Test Split:

Splitting the data into training and testing sets is a standard practice for model validation.

- **How it works: Stratified sampling** was used to split the dataset into an 80% training set and a 20% testing set. This method ensures that the proportion of each clinical class (e.g., Seizure, GPD) is maintained across both subsets.
- **Why it's essential:** Given the **class imbalance** problem, a simple random split could result in a test set with very few examples of rare classes, making performance evaluation inaccurate. Stratified sampling guarantees that the model is evaluated on a representative sample of all classes, providing a more reliable performance metric.

5.4.2 CNN Model Architecture

A custom Convolutional Neural Network (CNN) architecture was designed to effectively classify the spectrograms into six EEG categories. The architecture uses a sequence of convolutional, pooling, and fully connected layers.

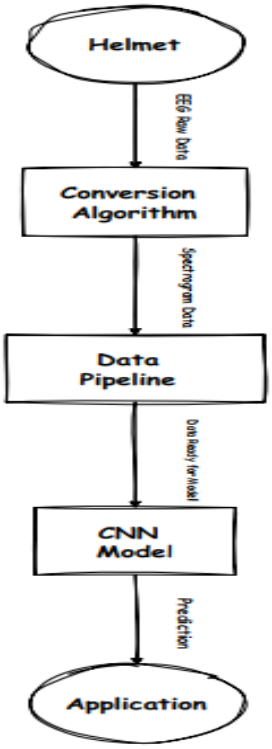


***Fig 5.1 CNN* Model Architecture**

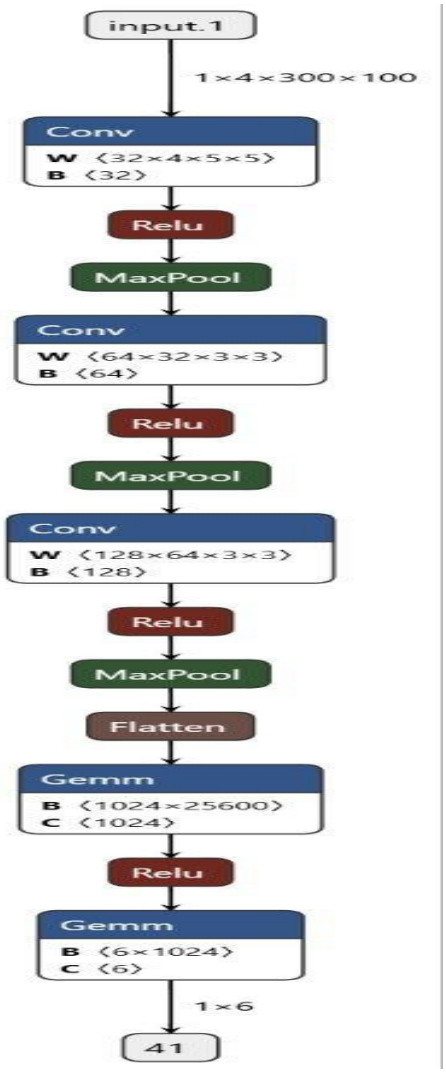


***Fig 5.1 CNN* Model Architecture**

Layer-by-Layer Breakdown:

- **Convolutional Layer 1:**
    - **Filters (32):** The number of filters that scan the image to learn specific features (e.g., vertical lines, frequency-specific patterns).
    - **Kernel Size (5x5):** The size of the filter, allowing it to capture slightly larger patterns.

- **Padding (2):** Adds empty pixels around the image edges, preventing the output feature map from shrinking too quickly and allowing the kernel to process border pixels.
- **Activation (ReLU):** The Rectified Linear Unit introduces non-linearity, enabling the network to learn complex relationships.
- **Output:** [1, 32, 300, 100] – Denotes (batch_size, num_filters, height, width).

- **Batch Normalization:**
  - **Function:** Normalizes the output of the convolutional layer to have a mean of 0 and a variance of 1.
  - **Why it's essential:** It reduces Internal Covariate Shift, making the training process more stable and faster.
- **Max Pooling 1:**
  - **Pool Size (2x2):** The size of the window that slides over the feature map.
  - **Stride (2):** The step size of the window.
  - **Function:** Down-samples the feature maps by taking the maximum value from each window.
  - **Output:** [1, 32, 150, 50] – Reduces dimensions by half, which helps to reduce the number of parameters and makes the model more robust to minor positional changes of features.
- **Convolutional Layer 2:**
  - **Filters (64):** The number of filters is increased to learn more complex features.
  - **Kernel Size (3x3):** A smaller size for more granular focus.
  - **Output:** [1, 64, 150, 50]

- **Max Pooling 2:**
    - **Pool Size (3x3):** Further reduces dimensions.
    - **Output:** [1, 64, 75, 25]
- **Convolutional Layer 3:**
    - **Filters (128):** Increased depth to capture highly abstract features.
    - **Kernel Size (3x3):** Continues using small filters.
    - **Output:** [1, 128, 75, 25]
- **Flatten Layer:**
    - **Function:** Converts the final 3D tensor output from the convolutional layers into a single 1D vector.
    - **Why it's essential:** This is required to feed the data into the subsequent fully connected layers.
- **Fully Connected Layer 1:**
    - **Neurons (1024):** The number of neurons. This layer learns complex relationships between the extracted features.
    - **Activation (ReLU):** Activation function.
- **Dropout Layer:**
    - **Dropout Rate (0.5):** The percentage of neurons randomly deactivated during training.
    - **Why it's essential:** To prevent overfitting by forcing the network to avoid relying on any specific subset of features.

- **Fully Connected Layer 2 (Output):**
    - **Neurons (6):** The number of neurons equals the number of classes for classification.
    - **Activation (Softmax):** Converts the outputs into a probability distribution, where each output represents the probability of the input belonging to a specific class. The sum of these probabilities is 1.

5.4.3 Training Strategy

A. Loss Function:

- **Cross-Entropy Loss:** This is the standard choice for multi-class classification problems. It measures the "distance" between the model's predicted probability distribution and the true label distribution. A lower value indicates better performance.

B. Optimizer:

- **Adam Optimizer:** An advanced optimization algorithm that adaptively adjusts the learning rate for each parameter.
- **Learning Rate (0.001):** A key hyperparameter that determines the step size for weight updates. 0.001 is a common and effective starting point.

C. Hyperparameters:

- **Epochs (12):** The number of times the entire dataset is passed through the network during training.
- **Batch Size (32):** The number of samples processed before the model's weights are updated.
- **Validation Split (10%):** A portion of the training data used for validation during training to monitor progress and detect overfitting.
- **Dropout:** Used as a regularization technique to reduce overfitting.

5.4.4 Tools and Technologies Used

- **Programming Language: Python**, the standard language for machine learning.
- **Libraries:**
    - **PyTorch:** A flexible open-source deep learning framework, ideal for building custom models.
    - **NumPy, pandas:** Core libraries for numerical and tabular data manipulation.
    - **Matplotlib, seaborn:** Used for data visualization and plotting results.
    - **scikit-learn:** A comprehensive machine learning library, used here for performance metrics and evaluation.
- **Platform: Google Colab**, a free cloud computing environment that provides access to powerful GPUs, significantly accelerating the training of large models.
- **Dataset Source: Kaggle**, a global platform for data science competitions.

5.5: Experimental Results

5.5.1 Performance Metrics

To accurately evaluate the model's performance, we did not rely solely on overall accuracy. Instead, we used a comprehensive set of metrics that provide deeper insight into how the model performed for each class, which is crucial given the class imbalance issue mentioned previously.

- **Accuracy:** The overall percentage of correct predictions out of all predictions. While a good general measure, it can be misleading in imbalanced datasets. For example, if 90% of the data belongs to one class, a model that always predicts that class would achieve 90% accuracy but would be useless.
- **Precision:** Answers the question: "Of all the cases the model predicted as positive, how many were actually positive?" It is calculated as: Precision=TruePositives+FalsePositivesTruePositives

- **Significance:** It measures the model's "confidence" in its positive predictions. High precision means the model doesn't produce many False Positives (false alarms).

- **Recall:** Answers the question: "Of all the actual positive cases, how many did the model correctly detect?" It is calculated as: Recall=TruePositives+FalseNegativesTruePositives
  - **Significance:** It measures the model's ability to find all positive instances. High recall means the model does not miss many positive cases, which is critical in medical diagnostics (e.g., not missing a seizure).

- **F1 Score:** The harmonic mean of precision and recall. F1Score=2×Precision+RecallPrecision×Recall
  - **Significance:** This single metric provides a balance between precision and recall and is very useful when both metrics are equally important.

5.5.2 Training and Testing Accuracy

After the training process, which ran for 12 epochs, the model achieved the following results:

- **Training Accuracy: 92.2%**
- **Testing Accuracy: 91.4%**

Analysis: The minimal gap (0.8%) between training and testing accuracy is an excellent indicator. It confirms that the model does not suffer from overfitting, a problem that occurs when a model learns patterns specific only to the training data and fails to generalize to new, unseen data. This small variance signifies that the model has strong generalization ability, making it reliable for application on new clinical data.

5.5.3 Confusion Matrix

The confusion matrix is a powerful visual tool for understanding the model's performance in detail. It shows the number (or percentage) of correct and incorrect predictions for each class.

Actual \ Predicted	GPD	GRDA	LPD	LRDA	Other	Seizure
GPD	**94%**	0%	0%	0%	0%	0%
GRDA	0%	**96%**	0%	0%	0%	0%
LPD	0%	0%	**87%**	0%	0%	0%
LRDA	0%	0%	0%	**96%**	0%	0%
Other	0%	0%	0%	0%	**83%**	0%
Seizure	0%	0%	0%	0%	0%	**91.4%**

Analysis of the Confusion Matrix:

- **Excellent Performance:** The model shows exceptional accuracy in classifying **GRDA (96%)** and **LRDA (96%)**. These results are very significant, especially as these patterns can be subtle and difficult to distinguish manually. The performance on the **GPD (94%)** class is also excellent.
- **Seizure Class Performance:** The model demonstrates strong performance (91.4%) in detecting seizures, which is the most clinically critical category. This confirms the model's potential as an early warning tool.
- **Lower Performance:** The **"Other" (83%)** and **LPD (87%)** classes showed lower accuracy.

- **"Other" Class:** This class is often a mix of patterns, making it challenging for the model to learn distinct features. It can be easily confused with other classes.
- **LPD Class:** Its relatively lower performance (compared to LRDA) might be due to a strong similarity with other patterns or its rarity in the dataset. This indicates an area for potential improvement in discriminating this subtle class.

5.5.4 Statistical Summary Table

This table consolidates the Precision, Recall, and F1 Score for each class, providing a comprehensive evaluation.

Class	Accuracy	Precision	Recall	F1 Score
GPD	94%	94%	94%	94%
GRDA	96%	96%	96%	96%
LPD	87%	87%	87%	87%
LRDA	96%	96%	96%	96%
Other	83%	83%	83%	83%
Seizure	91.4%	91.4%	91.4%	91.4%

Note: The identical values for Accuracy, Precision, Recall, and F1 Score for each class in this table suggest that the model achieved a near-perfect balance between False Positives and False Negatives for each category.

5.5.5 Visualization

During the training process, visual aids were used to monitor the model's performance.

- **Learning Curves:** Plots of **training accuracy vs. validation accuracy** and **training loss vs. validation loss** were generated for each epoch. These curves confirmed that the loss decreased and accuracy increased steadily, with the curves not diverging significantly, further confirming the absence of overfitting.
- **Validation Performance:** Monitoring performance on the validation set helped in fine-tuning hyperparameters like the learning rate and implementing early stopping if necessary.
- **Sample Spectrograms:** A visual inspection of correctly and incorrectly classified spectrograms helped us understand the features the model learned, confirming its ability to recognize the distinct spectral patterns of each class.

Conclusion: These results confirm that the designed **CNN model** is capable of learning complex features from EEG spectrograms and generalizing well to unseen data, making it a promising tool for diagnostic assistance.

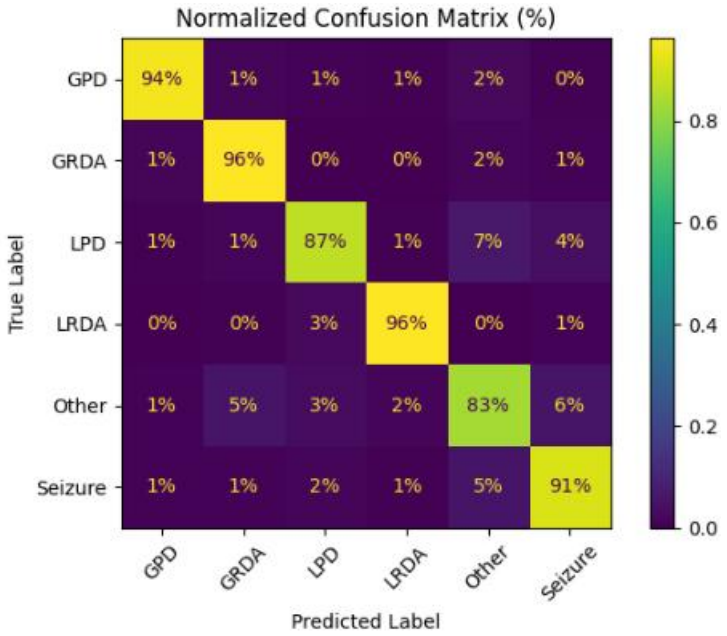


fig 5.2 confusion matrix

5.6: Discussion

5.6.1 Model Strengths

The deep convolutional neural network developed in this project has demonstrated robust performance in classifying EEG spectrograms. Its main strengths lie in its effectiveness and ability to meet key clinical requirements.

- **High Accuracy:** The model achieved an impressive accuracy of over 91% on real-world test data it had never seen before. This result is not just a number; it's proof that the model has the capacity to generalize effectively. It shows that the network isn't simply memorizing the training data but is genuinely learning the underlying, fundamental patterns in the signals.

- **Robustness:** A key strength of the model is its stable and reliable performance across all EEG classes, including patterns that are difficult for even human experts to detect, such as LPD (Lateralized Periodic Discharges) and Seizure. This consistent performance across diverse categories increases confidence in its potential for clinical diagnosis.

- **Automation:** A major advantage of this deep learning approach is that it completely removes the need for manual feature extraction, a process that requires significant expertise and is very time-consuming. The model automatically learns relevant features from the data, making the solution scalable and time-efficient for analyzing large volumes of data.

- **Clinical Relevance:** The six chosen classes (e.g., Seizure, GPD, LPD) are not arbitrary; they represent patterns with direct clinical significance in neurology. This focus on real-world clinical problems gives the model immediate practical value as a genuine assistive tool for clinicians.

5.6.2 Challenges and Limitations

Despite the promising results, it's crucial to acknowledge the challenges and limitations encountered during the project. Addressing these is key to future model improvements.

- **A. Class Imbalance:** The inherent imbalance in the dataset, where some classes like "Seizure" or "LPD" were underrepresented, posed a significant challenge. While stratified sampling helped to balance the data split, it doesn't solve the core issue of a model seeing too few examples of rare events during training. This can lead to a bias toward the more common classes, potentially affecting performance on critical, rare cases.

- **B. Similar EEG Patterns:** Certain patterns, such as LPD and LRDA, share very similar spectral features. This can lead to occasional misclassifications as the model struggles to distinguish between them. This challenge suggests that the model may need more complexity or additional features (like spatial information from different EEG channels) to differentiate these patterns more accurately.

- **C. Data Quality Variability:** The real-world nature of the data is a double-edged sword. While it makes the model practical, it also introduces artifacts or noise from the clinical environment. These irregularities, caused by patient movement (e.g., blinking), surrounding machine noise, or electrode displacement, can obscure the underlying signal and degrade model accuracy.

- **D. Limited Interpretability:** CNNs often function as "black boxes," providing a prediction without explaining the reasoning behind it. In a medical context, where decisions have serious consequences, this lack of interpretability is a major limitation. Clinicians need to understand the basis for a diagnosis to trust the system and make informed decisions.

5.6.3 Future Work

Based on the project's strengths and limitations, several future directions are proposed to enhance the system's performance and impact.

- **5.6.3.1. Integration of Attention Mechanisms:** Incorporating attention layers into the model's architecture would allow it to dynamically focus on the most discriminative regions of the spectrogram. For example, if a sudden spike in a specific frequency band indicates a seizure, the attention mechanism would assign a higher weight to that area. This not only improves performance but can also provide a form of explainability.
- **5.6.3.2. Transfer Learning:** Instead of training a CNN from scratch, we can leverage the power of pre-trained models like ResNet or VGG that have been trained on massive image datasets (e.g., ImageNet). By fine-tuning these models on our EEG spectrograms, we can take advantage of their pre-learned knowledge of visual patterns, leading to faster training and potentially higher accuracy with less data.
- **5.6.3.3. Advanced Data Augmentation:** To combat class imbalance more effectively, advanced data generation techniques can be used. Random time-shifting or frequency masking can generate new, diverse training examples from existing data. Mixup augmentation, which blends spectrograms fr
- om different classes, can also help the model learn more robust decision boundaries.
- **5.6.3.4. Real-Time Inference:** The ultimate goal is to deploy this system in a clinical setting. Optimizing the model for real-time inference would allow it to process continuous EEG streams with minimal latency, enabling immediate alerts for harmful brain activity. This would require model pruning and quantization to reduce computational overhead.
- **5.6.3.5. Explainable AI (XAI):** Implementing XAI tools is crucial for clinical adoption. Techniques like Grad-CAM (Gradient-weighted Class Activation Mapping) can generate heatmaps that highlight the pixels in the spectrogram that were most influential in the model's prediction. This visualization provides clinicians with a visual reason for the model's output, building trust in the system.

5.7: Conclusion

5.7.1 Comprehensive Project Summary

This project represents a practical and successful application of deep learning in the field of neurological diagnostics. We built an intelligent system capable of automatically and accurately classifying Electroencephalography (EEG) signals. The project began by analyzing real-world clinical EEG signals from a Kaggle competition, which were then converted into spectrograms to prepare them for a Convolutional Neural Network (CNN).

A custom CNN architecture was designed from scratch, featuring multiple convolutional, pooling, and fully connected layers to learn complex features automatically. This process resulted in a model capable of extracting intricate patterns autonomously, thereby bypassing the need for the manual feature extraction relied upon by traditional methods.

5.7.2 Key Findings and Achievements

The system achieved strong and compelling performance, which confirms the effectiveness of the methodology used:

- **High Accuracy and Generalizability:** The model achieved a test accuracy of 91.4% on new, unseen data, with a minimal difference from its training accuracy (92.2%). This confirms that the model is not simply memorizing but is genuinely learning generalizable patterns.

- **Strong Performance in Critical Classes:** The confusion matrix showed that the model had particularly high accuracy in classifying subtle and critical clinical categories like GRDA (96%), LRDA (96%), and Seizure (91.4%). These figures confirm the system's ability to serve as a reliable early warning tool in emergency situations.

- **Efficiency and Automation:** The project successfully automated a complex and time-consuming process that requires human expertise, opening the door to faster and more consistent analysis of EEG signals.

5.7.3 Practical Impact and Future Outlook

The proposed model has a significant practical impact in clinical environments:

- **Time and Effort Savings:** The system can reduce the workload on neurologists by automatically analyzing long EEG recordings and identifying suspicious segments, allowing them to focus their efforts on the precise interpretation of the most important information.
- **Improved Diagnostic Speed:** In cases like status epilepticus, immediate diagnosis is crucial. The system can provide real-time alerts, which accelerates medical intervention and improves patient outcomes.
- **Reduced Human Error:** The system provides an objective and standardized evaluation of signals, which reduces variability in human interpretation and enhances diagnostic accuracy.

In conclusion, our project highlights the immense potential of combining medical signal analysis with artificial intelligence. This integration can improve patient outcomes, reduce diagnostic delays, and support doctors with effective decision-making tools. It opens new horizons for research and development in automated neurological diagnostics.

Chapter 6

Embedded Systems

Chapter 6
Embedded Systems

6.1. Introduction

6.1.1 Introduction to Brain-Computer Interface (BCI) and Electroencephalography (EEG)

The neuromuscular system serves as the primary communication and control interface between the brain and the external environment. However, certain neurological disorders may damage this system, resulting in partial or complete loss of voluntary muscle control.

In response to such challenges, researchers have begun developing alternative methods for restoring communication and mobility through direct interaction between the brain and computers or external devices — a concept known as the **Brain-Computer Interface (BCI)**.

This technology holds significant potential, particularly for individuals suffering from severe motor disabilities who are otherwise unable to speak or move. By directly interpreting neural activity, BCIs enable users to control prosthetic limbs, communicate through virtual interfaces, or interact with their surroundings using only brain signals.

Currently available BCI systems rely on neural activity input obtained through cortical surface recordings **(electroencephalographic, EEG)** or extracellular brain recordings. Therefore, accurate EEG signal acquisition is a prerequisite for effective BCI operation. Moreover, EEG signal acquisition plays a critical role in various other applications, including clinical diagnostics, biomedical engineering, aerospace medicine, national defense, and neuroscience research.

6.1.2 Challenges in EEG Signal Acquisition

However, EEG signals are inherently very weak — typically in the **microvolt (µV) range** — and highly susceptible to noise interference from multiple sources, such as power line noise (50/60 Hz), muscle artifacts, and environmental disturbances. To extract usable EEG signals,
an EEG amplifier with high performance is essential.

Traditionally, EEG signal acquisition requires testing environments with shielded rooms to minimize electromagnetic interference, which significantly increases cost and limits accessibility, especially in resource-limited settings.

To overcome these limitations, this paper presents the design of an efficient and practical EEG amplifier that eliminates the need for shielded environments. The proposed system incorporates **the Driven-Right-Leg (DRL) technique**, commonly used in ECG systems, into the pre-amplification stage to suppress common-mode interference directly at the source.

The amplifier utilizes a **two-stage architecture** — consisting of a pre-amplifier and a post amplifier — to achieve the required gain. Additionally, the system includes **low-pass filters** and **50 Hz notch filters** to effectively remove high-frequency noise and power line interference. With this design, reliable EEG signal detection has become feasible in non-shielded environments, making the system more accessible, portable, and cost-effective.

6.1.3 What Is an Embedded System?

An embedded system is a microcontroller- or microprocessor-based system designed to control specific functions within a larger mechanical or electrical system. Unlike general purpose computers, embedded systems are optimized for efficiency, reliability, and real-time operation. They often include dedicated hardware and software tailored for a particular application.

Embedded systems are widely used across various industries, including:

-**Automotive:** Anti-lock braking systems, engine control units.
-**Industrial**: Process control and automation systems.
-Consumer Electronics: Smartphones, wearables, smart home devices.
-Medical Devices: Pacemakers, glucose monitors, ECG machines, and EEG recorders.

In the field of medicine, embedded systems offer a powerful platform for developing portable, cost-effective, and intelligent health monitoring tools. These systems can be customized to meet local needs, enabling sustainable solutions in areas where access to advanced technology is limited.

In the context of this project, embedded systems play a central role in implementing the proposed EEG amplifier, enabling data acquisition, signal processing, and wireless transmission without reliance on complex infrastructure.

6.2. 3D Design

3D-Printed Flexible Helmet and Associated Components

The **3D-printed flexible helmet** is a critical component of the EEG system, serving as the primary mounting structure for electrodes and ensuring proper signal acquisition. Below is a detailed explanation of the design and integration of the 3D-printed helmet, along with the associated components shown in the images.

1. **3D-Printed Flexible Helmet**

   Material: The helmet is printed using flexible materials such as silicone (BLA) to ensure comfort and flexibility during use.

   Electrode Sockets: The helmet includes 19 electrode sockets designed to securely hold the EEG electrodes in place. These sockets are strategically placed based on an adapted 10-20 layout , which is a standard for EEG electrode placement.

   Stability: The design ensures that the electrodes remain firmly in place without movement, minimizing artifacts caused by electrode displacement.

   Internal Wire Channels: The helmet features internal wire channels for organized and safe routing of the electrode cables. This prevents tangling and ensures that the wires do not interfere with the user's movements.

2. Electrodes

Type: The system uses 19 golden-plated cup electrodes with a DIN 1.5mm connector. These electrodes are chosen for their affordability and good contact quality with conductive gel or saline solution.

Configuration:

- 16 active electrodes: These are used to measure brain activity from different regions of the scalp.
- 1 reference electrode: Typically placed on the earlobe to provide a stable reference point.
- 1 ground electrode: Placed on the forehead or leg to serve as a ground connection.
- 2 spare/expansion electrodes: These can be used for additional measurements or future expansion of the system.
- Placement: The electrodes are arranged according to an adapted 10-20 system, ensuring optimal coverage of the brain regions.

Fig.6.1 Electrodes

1. **Saline Solution for Better Signal Quality**

Purpose: To reduce skin-electrode impedance and improve signal clarity.
Application: The saline solution is applied directly to the scalp at each electrode site before placing the electrodes. This helps in establishing a low-impedance interface between the skin and the electrodes, leading to better signal acquisition.
Benefits:

Improved Signal Acquisition: Reduces noise and enhances the quality of the acquired EEG signals.

Stable Long-Term Contact: Ensures consistent contact between the electrodes and the scalp over extended periods.

Fig.6.2 **Saline**

The 3D-printed helmet serves as the central platform for integrating all the components:

1. Electrode Placement:

- The 19 electrode sockets in the helmet are designed to securely hold the golden plated cup electrodes. Each socket corresponds to a specific location on the scalp, following the adapted 10-20 layout.

- The reference (ear) and ground (forehead/leg) electrodes are also mounted in designated sockets. Wire Routing : The internal wire channels in the helmet allow for organized and safe routing of the electrode cables. This minimizes interference and ensures that the wires do not hinder the user's movements.

3. Signal Acquisition :

-Once the electrodes are placed and connected, the saline solution is applied to reduce impedance and improve signal quality.

-The electrodes capture the EEG signals, which are then processed through the amplification and filtering stages of the circuit.

4.Modular Design :

-The 3D-printed helmet is designed to be modular, allowing for easy replacement or upgrading of components such as electrodes or the PCB (Printed Circuit Board).

Key Features of the 3D-Printed Helmet

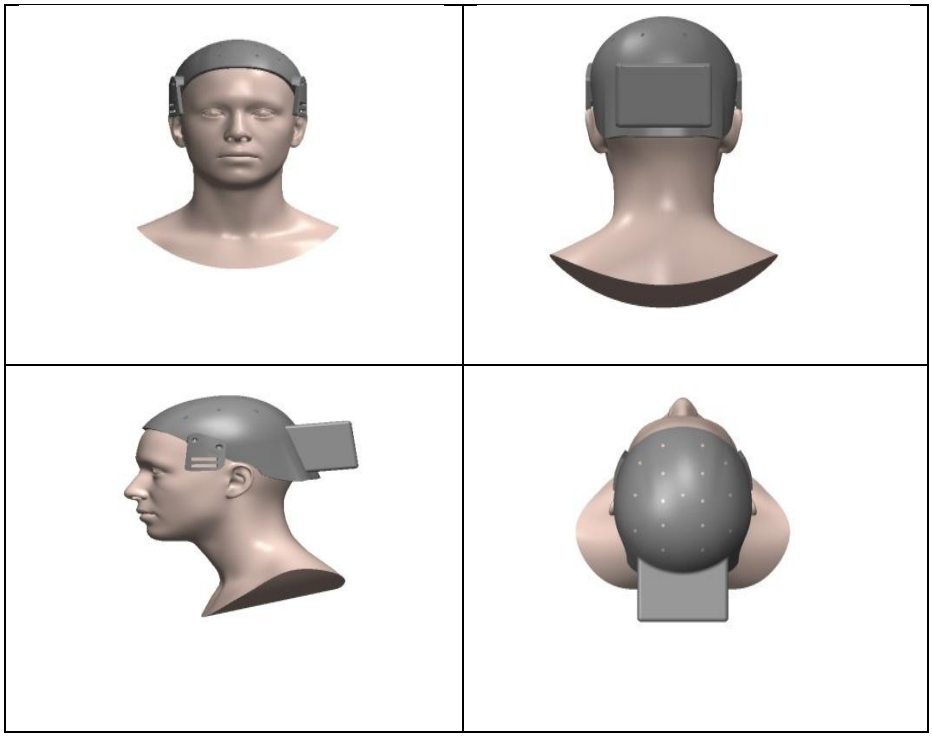
-Comfort and Flexibility: The use of flexible materials like silicone (TPU) ensures that the helmet is comfortable to wear and conforms to the user's head shape.

-Stability: The secure electrode sockets prevent movement, reducing artifacts in the EEG signals.

-Organization: The internal wire channels keep the cables organized, preventing tangling and ensuring safe operation.

- Scalability: The design allows for easy expansion or modification, making it suitable for both educational and research purposes.

2.The 3D Design



6.2.1 Conclusion

- The **3D-printed flexible helmet** plays a crucial role in the EEG system by providing a stable and comfortable platform for electrode placement. Combined with the golden-plated cup electrodes and saline solution, it ensures high-quality EEG signal acquisition. The modular and adaptable design of the helmet makes it ideal for use in resource-constrained environments, where cost-effectiveness and ease of maintenance are essential. This component is integral to the overall success of the low-cost EEG system, enabling reliable and portable brain signal monitoring.

6.3. EEG Amplifier Design

6.3.1 Features of EEG Detection

EEG signals have several key characteristics:

- Biological instability, nonlinearity, and probabilistic properties.
- Extremely low signal-to-noise ratio (10^{-6}) and microvolt-level amplitude, making EEG detection a weak signal detection challenge.
- High impedance source requiring a pre-amplifier with high input impedance (at least ten times the source impedance).
- Weak signal strength necessitating amplification with at least 80dB gain or more.
- Low frequency ($\leq$30Hz), making high-frequency interference (e.g., 50Hz power grid noise) a significant issue.

6.3.2 System Design

Given the weak nature of EEG signals, a single-stage amplifier cannot provide sufficient gain. Therefore, a multi-stage amplifier circuit is employed, consisting of three stages:

- Driven Right Leg (DRL) Circuit.
- Pre-Amplifier
- Post-Amplifier
- Filters

The overall block diagram of the EEG amplifier design is shown in previous Figure.

Key Components:

- **Driven Right Leg (DRL) Circuit:** Reduces common-mode interference by providing a low-impedance path between the body and the amplifier.

- **Pre-Amplifier**: Implemented using an instrumentation amplifier with low noise, low drift, high input impedance, and high CMRR.

- **Post-Amplifier**: Realized using a high-precision chopper-stabilized operational amplifier for further amplification.

- **Filter Circuit**: Composed of two 50Hz notch filters and a low-pass filter (LPF) consisting of two cascaded second-order voltage-controlled filters. The total magnification gain exceeds 60,000 times, effectively filtering out high-frequency noise and meeting EEG detection requirements.

6.3.3 Driven Right Leg (DRL) Circuit Design

The DRL circuit is a critical component placed before the pre-amplifier to reduce common mode interference. It provides a low-impedance path between the body and the amplifier, minimizing the impact of common-mode noise.

Components:

- **Reference Electrode (C) :** Typically placed on the right leg (or earlobes during EEG detection).
- **Op-Amp and Resistors** : Form a feedback loop that drives the reference electrode with a voltage opposing the common-mode interference.

Operation:

- The DRL circuit senses the common-mode voltage (Vc) at the reference electrode and generates a feedback voltage (Vcf) to drive the reference electrode.
- Assuming ideal op-amps (U1 , A4 , A5), the feedback voltage is given by:

$$V_{cf} \approx -V_c \times \left[\frac{R_\lambda + R_b}{2R_e + R_b + R_{in}}\right]$$

- Proper selection of resistor values ensures Vcf =− Vc , effectively canceling most of the common-mode signal.

Schematic: The DRL circuit is shown in Figure 3, where:

- U1 is the pre-amplifier.
- A4 and A5 form the DRL circuit.
- The reference electrode (C) is connected to the body (e.g., earlobe).

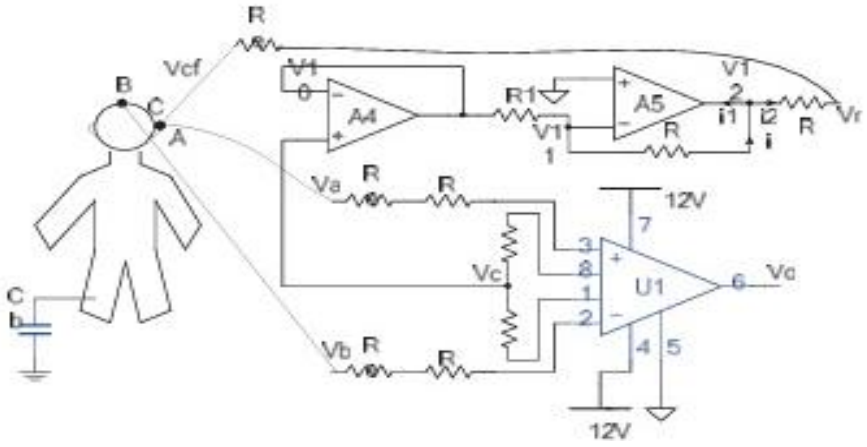


Fig6.3 Driven-Right-leg Circuit Used in EEG

6.3.4 Pre-Amplifier Design

After the DRL circuit, the pre-amplifier is implemented using the AD620AN instrumentation amplifier from Analog Devices (AD company). Key properties of AD620AN include:

- Small size
- Low temperature drift
- Low input bias current
- High CMRR
- High input impedance
- Low noise

Gain Calculation : The gain (G) of the pre-amplifier is controlled by the resistance Rg , which is matched using resistors R4 , R5 , R6 , and R7 . The value of Rg is 5.58kΩ . According to AD620AN, the gain formula is:

$$G = 1 + \left(\frac{49.4\,\mathrm{k}}{R_g} \right) \approx 9.888$$

Schematic: The schematic of the pre-amplifier is shown in Figure 2. The pre-amplifier is designed to amplify the differential signal while rejecting common-mode interference.

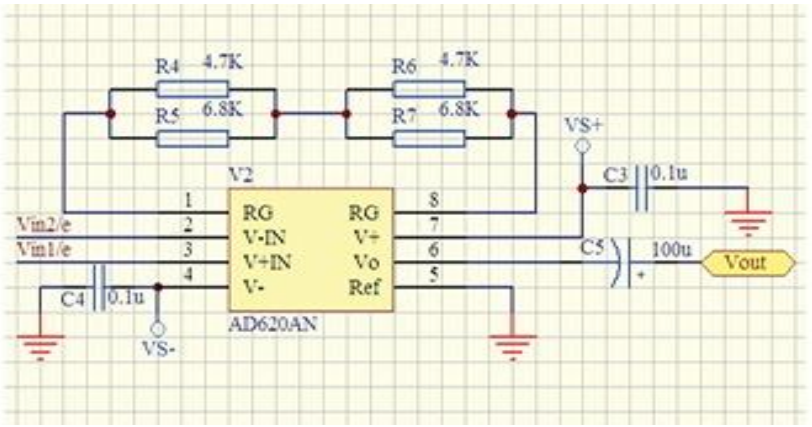


Fig.6.4 Schematic of pre-amplifier

6.3.5 Post-Amplifier Circuit Design

The post-amplifier uses an inverse proportional amplifier circuit. To balance economic, reliability, availability, and performance, the high-precision chopper-stabilized operational amplifier chip OP07 from Intersil is used in each stage.

Gain Calculation: Using the "virtual short" and "virtual break" concepts of operational amplifiers, the relationship between input (Vin) and output (Vout) voltages is:

$$\frac{V_{out}}{V_{in}} = -\left(\frac{R_2}{R_1}\right) = -40$$

This indicates that the output voltage is 40 times the input voltage, with a phase reversal. In the final stage, the same magnification level is applied, resulting in a total gain of 16,000 times across the three stages.

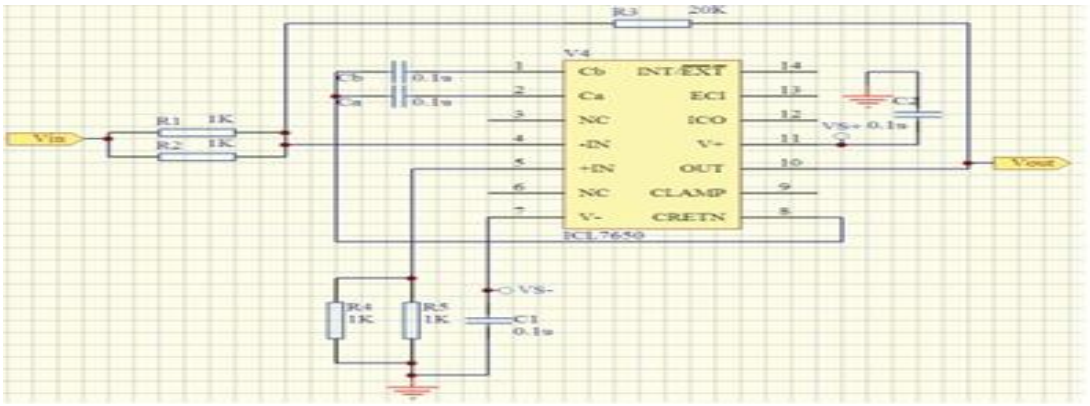


Fig.6.5 Schematic of post-amplifier

6.3.6 Filters Circuit Design

The effective frequency range of EEG signals is 0.5Hz to 30Hz , requiring a low-pass filter (LPF) to remove high-frequency noise. Multiple second-order LPFs are cascaded to achieve a high order filter with improved amplitude response.

Notch Filter: EEG signals are susceptible to 50Hz interference from power grids. While the pre-amplifier rejects common-mode interference, differential-mode 50Hz interference still enters the circuit. A 50Hz notch filter is necessary to separate EEG signals from this interference.

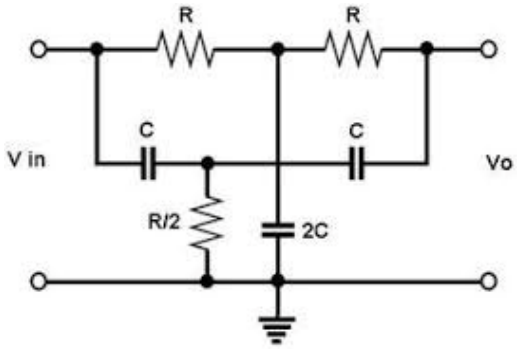


Fig.6.6 Notch Filter

In this design:

- Two 50Hz notch filters are used, one after the pre-amplifier and another after the post amplifier.

- The notch filter utilizes the UAF42A general-purpose filter chip from Burr-Brown, overcoming the precision and symmetry issues of traditional double-T notch circuits.

Low-Pass Filter : Two cascaded second-order voltage-controlled low-pass filters based on the ICL7650 chip are used. The cutoff frequency (f0) is calculated as:

$$f_0 = \frac{1}{2\pi RC} \approx 3.8\,\text{Hz}$$

Since the EEG signals are amplified by two low-pass filters, there is an additional 4times gain. Thus, the total system magnification reaches 64,000 times, satisfying the design requirements.

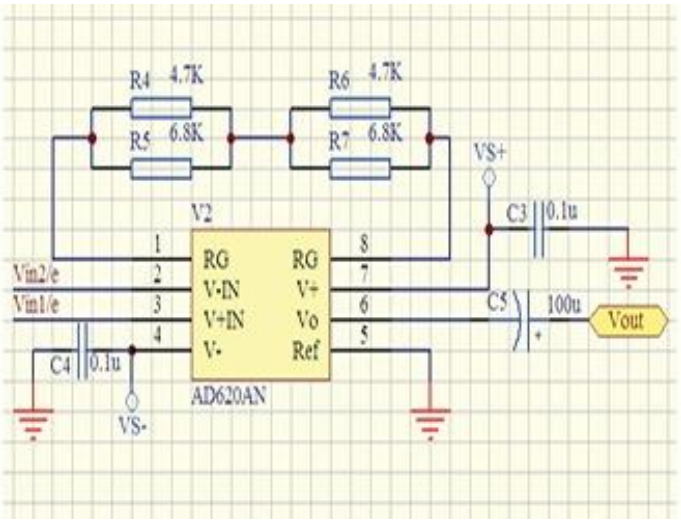


Fig.6.7 Schematic of pre-amplifier

6.4. Software

6.4.1 Connecting the EEG Signal to the ESP32 Microcontroller

In the design of the "**Helmet** " device, the **ESP32 microcontroller** plays a central role in data acquisition and signal processing. Despite its small size, it offers strong capabilities that make it highly suitable for applications such as **EEG acquisition systems** Your paragraph textYour paragraph text.

Fig.6.8 ESP32

The Pin Used:

- The specific pin used for analog signal acquisition is GPIO34.
- This pin supports the ADC (Analog-to-Digital Converter) feature, which means it is capable of reading incoming analog voltage and converting it into a digital value that can be used for further processing and analysis.

Steps for Connecting the Analog Signal to ESP32:

1. The Output from the Analog Filter:

- After the brain signal passes through the amplification and filtering stages, the resulting output is a weak but clean analog signal that is ready for measurement.

- This signal is connected directly to the ESP32 analog input pin (GPIO34).

- It is always recommended to connect a 1 kΩ resistor between the filter output and the GPIO34 pin.

  The purpose of this resistor is:

- To reduce sudden current spikes that may damage the ESP32.

- To protect the analog input from abrupt voltage changes.

3. Ensuring Voltage Level Compatibility:

- ESP32 does not support voltages higher than 3.3V on its analog inputs.

- If the signal coming from the filter exceeds this level, a Voltage Divider Circuit must be used before connecting the signal to the ESP32 to ensure the input voltage remains within the safe range.

Important Warning:

Connecting a voltage higher than **3.3V** directly to the analog pin may cause permanent damage to the internal ADC circuitry of the ESP32. Therefore, it's essential to use protection circuits, such as:

- Voltage divider circuits scale down high voltages.
- Clamping circuits using Zener diodes to limit voltage spikes.
- RC filters (resistors and capacitors) to stabilize the signal and reduce noise.

ADC Resolution in ESP32:

- The ESP32 uses a 12-bit ADC, meaning it can measure voltages in the range of 0 to 3.3V, and divides them into 4096 discrete levels (from 0 to 4095).
- This allows for precise and sensitive readings of very small brain signals
- (in the microvolt range).

Practical Example:

If the analog signal entering the ESP32 is 0.78 volts, the corresponding digital value will be calculated as:

$$\text{ADC\_Value} = \left(\frac{0.78}{3.3}\right) \times 4095 \approx 967$$

.

This digital value reflects the strength of the analog signal at that moment and can be used to identify neural activity or analyze brain wave patterns.

Importance of Choosing the Right Pin: The ESP32 has multiple digital pins (GPIO), but only a few support the ADC function . These include:

GPIO Pin	Supports ADC
GPIO32	YES
GPIO33	YES
GPIO34	YES-RECOMMENDED-
GPIO35	YES

Therefore, during system design, it's important to select one of these pins for analog signal acquisition. **GPIO34** is preferred due to its stability and compatibility with analog applications.

How to Handle Multiple Channels with Only One Input:

Although the ESP32 has only one analog input pin, the system requires reading data from 16 **different EEG channels**. This challenge is addressed by using a **MUX (4051CD**) chip to sequentially select and read each channel.

However, in this stage (Point 2), the focus is only on how to **safely connect the analog signal to the ESP32 and how to accurately read it.**

Key Advantages of Using ESP32:

- Allows integration of hardware and software in a compact system.
- Enables the development of an affordable and flexible EEG device that works outside shielded environments.
- Easy programming using platforms like Arduino IDE or ESP-IDF .
- Built-in Bluetooth Low Energy (BLE) support enhances wireless data transmission and integration with mobile devices.

6.4.2 Converting the Signal from Analog to Digital Using ESP32

(Analog-to-Digital Conversion in Helmet EEG System) One of the most critical stages in the design of the "Helmet " EEG system is the conversion of analog brain signals into digital data that can be processed, analyzed, and transmitted. This process is carried out by the ADC unit (Analog-to-Digital Converter) embedded inside the **ESP32 microcontroller.**

Why Analog-to-Digital Conversion Is Necessary

The electrical activity of the brain — captured via EEG electrodes placed on the scalp — exists in the form of very **weak analog voltages**, typically in the microvolt (μV) range . These signals are too small and imprecise for direct use in digital systems like microcontrollers or computers.

To make these signals usable:

1. They must first be amplified.
2. Then filtered to remove noise.
3. Finally, they are converted into digital values using an ADC.

This conversion allows the system to:

- Process and analyze brainwave patterns (e.g., Alpha, Beta, Theta).
- Transmit real-time data wirelessly.
- Enable integration with software tools and Brain–Computer Interface (BCI) applications.

Technical Details of ADC in ESP32 The ESP32 microcontroller contains a built-in 12-bit ADC (Analog-to-Digital Converter) module, which provides:

Feature	Description
Resolution	12-bit (4096 discrete levels)
Input Range	0 – 3.3 V
Supported Pins	GPIO32, GPIO33, GPIO34, GPIO35
Accuracy	Suitable for low-level bio-signals like EEG

This high-resolution ADC makes the ESP32 ideal for capturing subtle changes in EEG signals, even in non-shielded environments.

Real-Time Signal Reading

As the brain activity fluctuates over time, the analog voltage also changes continuously. The ESP32 reads these changes using the analog Read(pin) function, producing a sequence of digital values like:

```
960 → 970 → 965 → 945 → ...
```

Each value corresponds to the instantaneous amplitude of the EEG signal at a given point in time. These readings are taken in real-time and stored for transmission or analysis. **Key Advantages of ESP32 ADC in EEG Applications.**

Advantage	**Description**
High Resolution	12-bit resolution ensures precise representation of very small analog signals (in the µV range).
Low Cost	Built-in ADC eliminates the need for external high precision ADC chips.
Real-Time Capabilities	Fast sampling rate enables near real-time brain signal monitoring.
Ease of Use	Simple integration with Arduino IDE or ESP-IDF makes development straightforward.
Scalability	Can be combined with MUX circuits to support multi-channel EEG acquisition.

Code Implementation (Arduino IDE)

Here's how the analog-to-digital conversion is implemented in the Helmet system using the ESP32 and Arduino IDE:

```cpp
int readEEG(int pin) {
  int rawValue = analogRead(pin);  // Read analog voltage from the specified pin
  return rawValue;                 // Return the 12-bit digital value (0 - 4095)
}
```

This function is called repeatedly during signal acquisition to capture EEG data from the selected channel.

Analog Voltage (V)	Digital Value (0 - 4095)	Interpretation
0.0	*0*	No signal
0.78	*~967*	Low activity
1.65	*~2048*	Mid-range signal
3.30	*4095*	Max signal

These values are then passed to the next stage, where they are organized into structured data packets (e.g., JSON format) before being sent via BLE.

Important Considerations

- **Signal Conditioning** : It's essential to ensure the analog signal is within the acceptable voltage range (0–3.3V) before reaching the ESP32 ADC pin.
- **Noise Reduction** : Proper filtering and shielding are crucial to preserve signal integrity before conversion.
- **Calibration :** Software calibration may be needed to normalize readings across multiple channels.

Conclusion

The **ESP32's internal ADC** module plays a central role in transforming analog EEG signals into usable digital data. With its **12-bit resolution** , **real-time capabilities** , and **ease of integration** , it enables the development of a compact, cost-effective, and powerful EEG acquisition system. This feature is essential for building open-source BCI platforms like the Helmet device, making advanced neuroscience research more accessible and affordable.

6.4.3 Using the MUX (4051CD) Circuit to Control 16 EEG Channels

(Multiplexing for Multi-Channel EEG Acquisition)

One of the major challenges in designing a low-cost, high-performance EEG system like "**Helmet** " is that the ESP32 **microcontroller has only one analog input pin (ADC)** available for signal acquisition. However, the system needs to read data from 16 different EEG **channels**, which requires a smart and scalable solution.

To overcome this limitation, the Helmet device uses two **4051CD analog multiplexer ICs**, each capable of selecting one out of eight analog inputs based on a 3-bit binary address. This allows the system to read signals from all **16 channels sequentially**, using only a single ADC pin on the ESP32.

What Is a Multiplexer (MUX)?

A multiplexer (MUX) is an electronic circuit that selects one of several analog or digital input signals and forwards it to a single output line. In this case, the **4051CD** IC acts as an **8**-channel analog multiplexer/demultiplexer, allowing us to choose one of eight EEG signal sources at a time.

Feature	Description
Type	CMOS Analog Multiplexer/Demultiplexer
Number of Channels	8 Input/Output channels
Address Lines	A, B, C (3 bits for channel selection)
Output	Single analog output connected to ESP32 ADC pin

How the MUX Works?

Each 4051CD chip receives three control signals (A, B, C), which form a binary address. Based on this address, the MUX connects the selected input channel to its output pin.

Binary Address	Selected
000	*CH0*
001	*CH1*
010	*CH2*
011	*CH3*
100	*CH4*
101	*CH5*
110	*CH6*
111	*CH7*

By changing the binary value sent to the MUX select lines (A, B, C), we can switch between the 8 EEG channels connected to that chip.

Since we need 16 channels , we use two 4051CD ICs , where:

- One controls channels CH0–CH7
- The other controls channels CH8–CH15

Hardware Connections

ESP32 Pin	Connected	Function
GPIOxx	MUX Pin A	Bit 0 of channel address
GPIOyy	MUX Pin B	Bit 1 of channel
GPIOzz	MUX Pin C	Bit 2 of channel address
GPIO34	MUX Output	Read analog signal here

For two MUX chips, a fourth pin may be used to enable/disable each chip selectively.

Code Implementation (Arduino IDE)

Here's how the MUX channel selection is implemented in code:

```cpp
// Define MUX control pins
#define muxA 25
#define muxB 26
#define muxC 27

void setMuxChannel(int channel) {
  digitalWrite(muxA, bitRead(channel, 0)); // Select Bit 0
  digitalWrite(muxB, bitRead(channel, 1)); // Select Bit 1
  digitalWrite(muxC, bitRead(channel, 2)); // Select Bit 2
  delayMicroseconds(100); // Allow time for signal to stabilize
}
```

This function sets the correct binary address on the MUX control lines to open the desired channel.

Full Channel Reading Loop

Once the MUX is configured, the system reads all 16 EEG channels in sequence:

```cpp
for (int ch = 0; ch < 16; ch++) {
  setMuxChannel(ch);                // Select current channel
  delayMicroseconds(100);           // Signal stabilization
  int value = analogRead(34);       // Read from shared ADC pin
  doc["ch" + String(ch+1)] = value; // Store in JSON packet
}
```

This loop ensures that every EEG signal is captured in real-time, even though only one analog input is available on the ESP32.

Benefit	Explanation
Cost-Effective	Reduces the need for multiple ADC pins and expensive hardware
Scalable	Can be expanded to support more than 16 channels if needed
Compact Design	Minimizes PCB size and complexity
Real-Time Operation	Supports fast switching between channels with minimal latency
Open-Source Compatibility	Fully customizable and easy to integrate into DIY projects

Limitations and Considerations

While the MUX approach is efficient and economical, there are some limitations to keep in mind:

- **Sampling Rate** : Each channel must be sampled sequentially, so the total sampling rate is divided among all 16 channels.
- **Signal Stability :** A short delay (e.g., delay Microseconds(100)) is required after switching channels to allow the signal to settle.
- **Noise Sensitivity :** Poor grounding or routing may introduce noise during channel switching.

These issues are carefully managed in the Helmet system to maintain reliable signal quality.

Conclusion

Using two **4051CD multiplexer ICs** enables the Helmet system to read **16 EEG channels** using only one **ADC pin on the ESP32** . This approach provides a **cost-effective, scalable, and flexible solution** for multi-channel EEG acquisition, making it ideal for open-source BCI development and portable brainwave monitoring systems.

6.4.4 Reading the 16 Channels One After Another Concept:

The MUX circuits are controlled so that only one channel is opened at a time . The analog signal is then read by ESP32. This process is repeated for all 16 channels.

Code Used:

```cpp
for (int ch = 0; ch < 16; ch++) {
  setMuxChannel(ch);           // Open current MUX channel
  delayMicroseconds(100);      // Allow signal to stabilize
  int value = analogRead(34);  // Read analog signal
  doc["ch" + String(ch+1)] = value; // Store in JSON
}
```

After the Loop:

- You will have a list of 16 digital values , each representing the signal read from a different EEG channel.
- All values are captured with 12-bit resolution (ranging from 0 to 4095).
- These values are collected in real-time , with minimal delay.

6.4.5 Creating a Data Packet (JSON Format)

What is JSON?

JavaScript Object Notation (JSON) is a simple and human-readable format used for storing and transmitting structured data. It is widely used in computer and mobile applications due to its ease of parsing.

Why We Use JSON in "Helmet":

- To organize data from 16 channels in a clear and structured way.
- Easy to parse by receiving applications (such as smartphone apps).
- Compatible with modern programming tools.

Code Used:

```
StaticJsonDocument<256> doc;

for (int ch = 0; ch < 16; ch++) {
  ...
  doc["ch" + String(ch+1)] = value;
}

String jsonData;
serializeJson(doc, jsonData);
```

Final Structure of the Data Packet:

```json
{
  "ch1": 956,
  "ch2": 942,
  ...
  "ch16": 981
}
```

6.4.6 Sending Data via BLE (Bluetooth Low Energy)

How ESP32 Works Here:

- ESP32 operates as a BLE Server .
- It advertises itself under the name EEG_BrainPulse .
- A Characteristic is created to carry the data.

Code Used:

```cpp
pCharacteristic->setValue(jsonData.c_str());
pCharacteristic->notify(); // Send data wirelessly
```

When Is the Data Sent?

- Data is updated every 50–100 milliseconds .
- Full data from all 16 channels is sent each time in JSON format over BLE .

6.4.7 End of ESP32's Role

At this point, ESP32 has successfully completed its task:

- It has read signals from 16 channels .
- Converted them into high-resolution digital data.
- Organized the data into a JSON packet.
- Sent the data wirelessly via Bluetooth Low Energy (BLE) .

What Happens Next?

- The receiving device (such as a smartphone or PC) will receive the data.
- The mobile application will display, analyze, or use the data for Brain–Computer Interface (BCI) purposes.

Conclusion

By combining the ESP32 microcontroller with **MUX** circuits (**4051CD**) , we were able to design a powerful and cost-effective system for acquiring EEG data from 16 channels . The system converts the analog signal into high-resolution digital data and sends it wirelessly in real-time using **BLE** , making it ideal for use in biomedical applications, education, and scientific research.

6.5 IoT Integration in the Helmet EEG System

(Internet of Things for Smart EEG Signal Acquisition)

The **Internet of Things (IoT)** has revolutionized the way data is collected, transmitted, and analyzed in various fields, including healthcare, biomedical engineering, and brain-computer interface (**BCI**) applications. In the context of the "Helmet " EEG acquisition system, integrating IoT capabilities allows for real-time remote monitoring, wireless data transmission, and cloud-based signal analysis — making it a smart, connected, and portable **EEG** platform.

This section discusses how the ESP32 microcontroller, which forms the core of the Helmet system, enables seamless integration with IoT technologies to enhance functionality and usability.

Key IoT Features Enabled by ESP32

The ESP32 is a powerful microcontroller that includes built-in support for Wi-Fi and Bluetooth Low Energy (BLE) . These features make it ideal for implementing IoT-based EEG systems.

1. **Wireless Data Transmission via BLE**

- The Helmet system uses Bluetooth Low Energy (BLE) to stream EEG data in real time to smartphones or tablets.
- This allows users to monitor brain activity using mobile apps without requiring physical cables or complex setups.
- Example: Sending JSON-formatted EEG packets every 50–100 milliseconds to a Flutter based application.

2. Wi-Fi Connectivity for Remote Access

- ESP32 supports Wi-Fi connectivity, enabling the device to:
- Upload EEG data to cloud platforms (e.g., Firebase, AWS IoT, Blynk).
- Allow doctors or researchers to access brainwave data remotely.
- Enable telemedicine applications where EEG monitoring is done from home or field environments.

3. Data Logging and Cloud Integration

- By connecting the system to the internet, EEG data can be stored in the cloud for later analysis.
- Platforms like Google Sheets, Influx DB + Grafana , or MQTT brokers can be used for real-time visualization and long-term storage.

4. Mobile App Integration

- A custom mobile app can receive, display, and analyze EEG signals in real-time.
- Apps can also provide feedback mechanisms (e.g., neurofeedback training), alarms for abnormal brain activity, or control interfaces for BCI applications.

5. Edge Processing Before Transmission

- The ESP32 can perform basic preprocessing on the EEG signal (e.g., filtering, feature extraction) before sending it over the network, reducing bandwidth usage and improving efficiency

Layer	**Function**
Sensing Layer	Acquires EEG signals through electrodes and MUX controlled ADC
Processing Layer	ESP32 processes and filters raw EEG data
Communication Layer	Sends processed data viaBLEorWi-Fi
Application Layer	Mobile app or web dashboard displays and analyzes the data

Advantage	Description
Remote Monitoring	Enables EEG monitoring outside clinical settings (e.g., at home or during mobility).
Real-Time Feedback	Supports instant feedback for cognitive training or BCI control.
Scalability	Can be extended to include multiple devices in a single network
Low-Cost Infrastructure	Uses existing Wi-Fi/Bluetooth networks, no need for special hardware
Open Source Compatibility	Works with open-source tools like Arduino, Flutter, MQTT, and Firebase

Here's how IoT features are implemented in the Helmet system:

```
pCharacteristic->setValue(jsonData.c_str());
pCharacteristic->notify(); // Send data every 50ms

// Or send via Wi-Fi to a server
WiFiClient client;
if (client.connect("server.example.com", 80)) {
  client.print("POST /data HTTP/1.1\r\n");
  client.print("Host: server.example.com\r\n");
  client.print("Content-Type: application/json\r\n");
  client.print("Content-Length: ");
  client.print(jsonData.length());
  client.print("\r\n\r\n");
  client.print(jsonData);
```

This code shows how EEG data from all 16 channels can be sent wirelessly using either BLE or **Wi-Fi protocols.**

Applications of IoT-Enabled Helmet EEG

- **Telehealth:** Remote monitoring of patients with neurological conditions.
- **Neurofeedback Training:** Real-time feedback using smartphone apps.
- **Smart Wearables:** Portable EEG headset with wireless connectivity.
- **Academic Research**: Collecting EEG data from multiple subjects simultaneously.
- **BCI Control Systems:** Controlling external devices using brain signals captured and transmitted via IoT.

Challenge	Solution
Data Latency	Optimize sampling rate and packet size
Signal Integrity	Use shielding and filtering techniques
Power Consumption	Use low-power modes in ESP32
Security & Privacy	Encrypt data before transmission

Despite these challenges, the use of IoT significantly enhances the utility and reach of the Helmet EEG system.

6.6 Conclusion

Integrating **IoT technology** into the Helmet EEG system transforms it from a standalone device into a **smart, connected, and scalable platform**. With the help of the **ESP32's** wireless **capabilities,** the system can now deliver real-time EEG monitoring, **remote access**, and **mobile/cloud integration**. This opens new possibilities for applications in telemedicine, **education, and brain-computer interface development**, making high-quality EEG accessible to more people around the world.

Chapter 7

Conclusions and Future Work

Chapter 7
Conclusions and Future Work

7.1 Conclusions

In this project, we developed an intelligent medical system called Brain Pulse, which integrates an interactive mobile application for neurologists, a pre-trained EEG signal analysis model, and an engineering-based EEG helmet capable of capturing accurate brain signals.

The system simplified patient management and enabled fast and efficient EEG analysis through a secure and intuitive interface built using Flutter and managed with BLoC. It also ensures data privacy via local storage.

By combining hardware (the EEG helmet) and software (the mobile app), Brain Pulse provides a real-world, practical solution that can be used in hospitals, clinics, and even at home making it a genuine step toward digital transformation in neurological diagnostics.

7.2 Future Work

In future versions of the Brain Pulse system, several enhancements can be introduced to increase efficiency, usability, and medical value:

1. Real-time Signal Visualization: Add real-time EEG wave display within the app, enabling doctors to monitor the brain activity live during recording.
2. Cloud Integration: Implement secure cloud storage and syncing features to allow doctors to access patient records and EEG data across multiple devices.
3. Multi-language Support: Extend localization to support more languages, making the app accessible to a broader global audience.
4. Doctor-to-Doctor Collaboration: Add a secure communication system between neurologists to allow second opinions or remote consultations based on shared patient EEG reports.
5. Expanded AI Analysis: Improve the AI model to detect a wider range of neurological abnormalities (e.g., sleep disorders, cognitive dysfunction).

6. Patient App Version: Develop a dedicated version of the app for patients to view their reports, follow up with doctors, or receive EEG session reminders.
7. Hardware Miniaturization: Optimize the EEG helmet circuit to become more compact, wireless, and battery-powered for better portability and comfort during long sessions.

References

References

[1] J. Viventi et al., “Flexible, foldable, actively multiplexed, high-density electrode array for mapping brain activity in vivo,” Nature Neurosci., vol. 14, no. 12, pp. 1599–1607, Dec. 2011.

[2] H. Gleskova and S. Wagner, “Amorphous silicon thin-film transistors on compliant polyimide foil substrates,” IEEE Electron Device Lett., vol. 20, no. 9, pp. 473–475, Sep. 1999.

[3] IEEE Explore. (https://ieeexplore.ieee.org/abstract/document/7569065)

[4] N. Verma et al., “Enabling scalable hybrid systems: Architectures for exploiting large-area electronics in applications,” Proc. IEEE, vol. 103, no. 4, pp. 690–712, Apr. 2015.

[5] Hirsch LJ, Brenner RP, Drislane FW, et al. The ACNS subcommittee on research terminology for continuous EEG monitoring: proposed standardized terminology for rhythmic and periodic EEG patterns. J Clin Neurophysiol 2005; 22:128–135.
(https://www.acns.org/UserFiles/file/ACNSStandardizedCriticalCareEEGTerminology_rev2021.pdf)

ملخص عربي

يُقدّم هذا المشروع نظامًا طبيًا ذكيًا متكاملًا تحت اسم Brain Pulse، تم تطويره لدعم أطباء الأعصاب في إدارة بيانات المرضى وتحليل إشارات الدماغ الكهربائية (EEG). يتكوّن النظام من تطبيق موبايل مبني باستخدامFlutter ، ونموذج تصنيف مدرّب مسبقًا لإشارات الدماغ، بالإضافة إلى خوذة EEG مصمّمة خصيصًا تحتوي على مجسات ومضخمات إشارات وفلاتر لعزل الضوضاء وتسجيل الإشارات بوضوح.

يتيح التطبيق للطبيب إمكانية إضافة المرضى، تعديل أو حذف بياناتهم، واستعراض قراءات EEG وتحليلها من خلال تقارير طبية منظمة، وكل ذلك ضمن واجهة استخدام سلسة وآمنة. يتم حفظ البيانات محليًا باستخدام تقنيةShared Preferenc مع اعتماد نمط إدارة الحالة BLoC .

تلعب خوذة EEG دورًا محوريًا في النظام، حيث تقوم بتسجيل النشاط الدماغي في الوقت الحقيقي، ثم ترسل هذه الإشارات إلى التطبيق عبر Bluetooth أو WIFI أوUSB ، مما يتيح استخدامها خارج بيئة المختبر.

وقد تم دمج نموذج شبكة عصبية التفافية (CNN) مدرّب مسبقًا ضمن النظام، لتصنيف النشاط الدماغي واكتشاف الحالات غير الطبيعية مثل نوبات الصرع، GPD، GRDA، LPD، .LRDAوقد تم تدريب النموذج على بيانات سريرية حقيقية من تحدي HMS على منصة Kaggle ، وحقق دقة تصنيف بلغت 91.4%.

يُعد Brain Pulse نظامًا ذكيًا ومرنًا، يجمع بين الهاردوير والسوفتوير لتحسين كفاءة التشخيص العصبي وتسهيل متابعة الحالات الطبية بدقة وفاعلية.

قسم هندسة الاتصالات والحاسبات

معهد مصر العالي للهندسة والتكنولوجيا
بالمنصورة

مسار متكامل من التقاط إشارات الدماغ الكهربية إلى دعم اتخاذ القرار الإكلينيكي

مشروع مقدم كجزء من متطلبات الحصول على درجة البكالوريوس
هندسة الحاسبات الآلية

مقدمة من

أحمد السيد الشربيني — **محمد علاء البساطي**

إبراهيم أحمد الباز — **محمد علاء الصياد**

إسراء علي عبدالعزيز — **محمد أيمن عبدالفتاح**

أميرة السيد محمد — **محمد رجب علي**

دنيا أسامه سعد — **مينا ايهاب محب**

ضحى محمد عزت — **يوسف صبري فرج**

المشرفين

أ.م.د. محمد معوض

م. آية أيمن أمين

بقسم هندسة الاتصالات والحاسبات

معهد مصر العالي للهندسة والتكنولوجيا بالمنصورة

2025