CS551

Mobile Computing for Android Mobile Devices

FALL-ARM TERM PROJECT

TEAM MEMBERS

Kesselly Kamara

Yixuan Liang

Francis Etang

SUBMISSION DATE

08/16/2023

Table of Contents

1. Project Overview

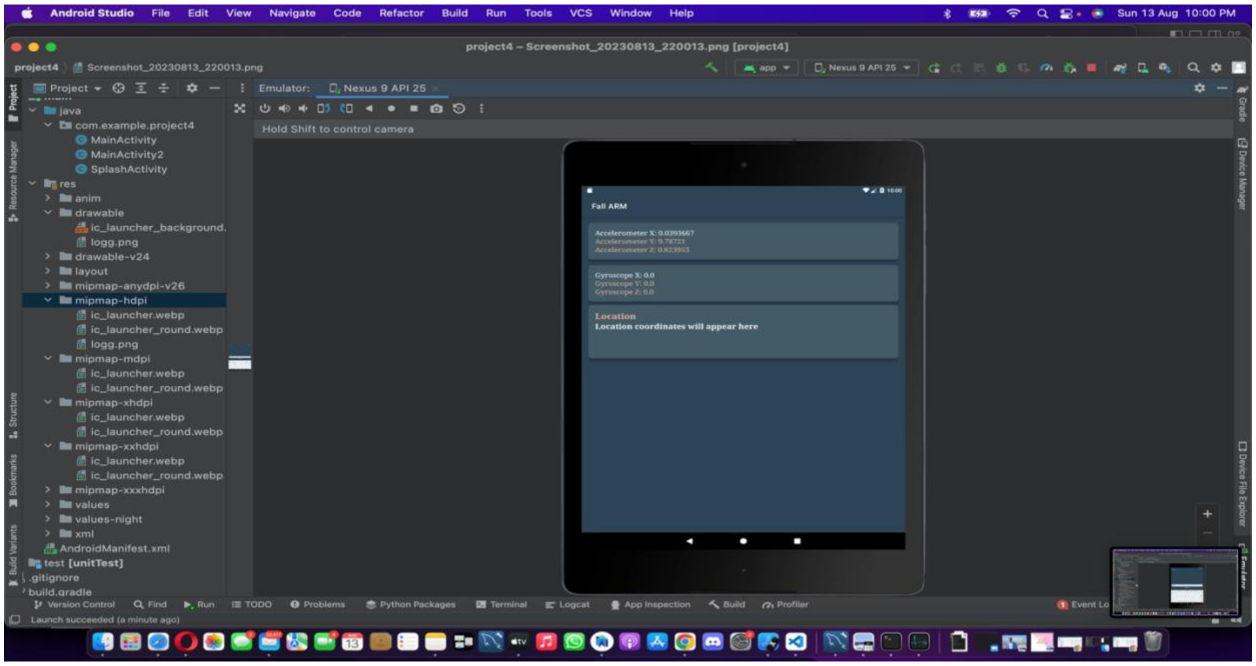
1.1 Project Description
The FallArm project aims to develop a system that detects patient falls using accelerometer and gyroscope data from an Android device, determines the type of motion, and alerts the nurse in case of a fall. The project also involves integrating geolocation data and establishing communication between the Android device and a server.

1.2 Project Objectives

The FallArm project is designed to achieve a set of well-defined objectives, each contributing to the overall functionality and success of the application. These objectives encompass a wide range of technical aspects, user interactions, and system functionalities. The primary objectives of the project are as follows:

1.2.1. Generate Dynamic Accelerometer and Gyroscope Data on an Android Emulator

The first objective of the FallArm project is to develop an Android application capable of generating dynamic accelerometer and gyroscope data. This data will simulate real-time motion and orientation changes that might be encountered by a patient wearing the device. By generating this data on an Android emulator, the project aims to create a controlled environment for testing and validation purposes. The accuracy and variability of the generated data will play a crucial role in assessing the effectiveness of the fall detection and classification algorithms.



Fall ARM

Accelerometer X: 0.0393667
Accelerometer Y: 9.78721
Accelerometer Z: 0.823953

Gyroscope X: 0.0
Gyroscope Y: 0.0
Gyroscope Z: 0.0

Location
Location coordinates will appear here

Android Studio
project4 – Screenshot_20230813_220013.png [project4]
Emulator: Nexus 9 API 25
Hold Shift to control camera
Launch succeeded (a minute ago)
Sun 13 Aug 10:00 PM

Fig 1.1 - Dynamic Accelerometer and Gyroscope Data on an Android Emulator with Patient location

1.2.2. Determine Patient Location Using Geolocation Data

The project's second objective involves leveraging geolocation data to determine the precise location of the patient. By integrating geolocation services, the application will be able to access the device's GPS data, enabling the identification of the patient's real-world coordinates. This location information is vital for accurate reporting and alerting in case of a fall. The accuracy and reliability of the geolocation data will contribute to the overall efficacy of the system.

1.2.3. Transmit Sensor Data, Patient Location, and Patient ID to a Server

Building upon the previous objectives, the project aims to establish a communication channel between the Android device and a server. This communication will facilitate the transmission of crucial data, including the generated sensor data, patient location, and patient identification. The transmission process involves handling data serialization, network protocols, and security mechanisms to ensure the safe and reliable transfer of information. A successful implementation of this objective is essential for real-time monitoring and data analysis.

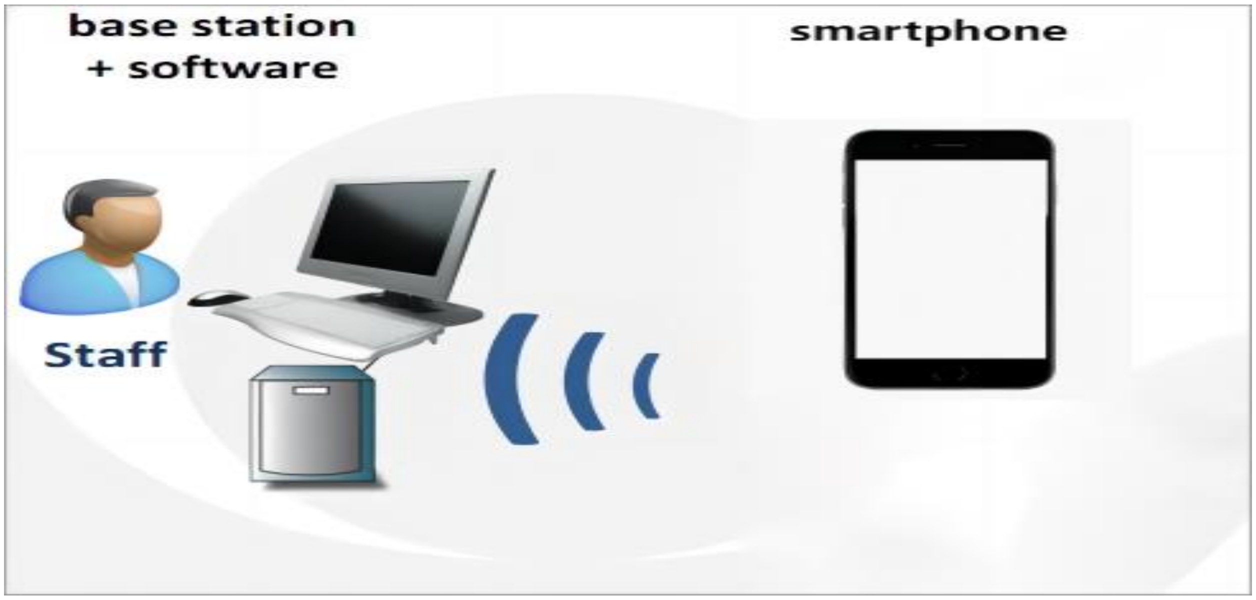


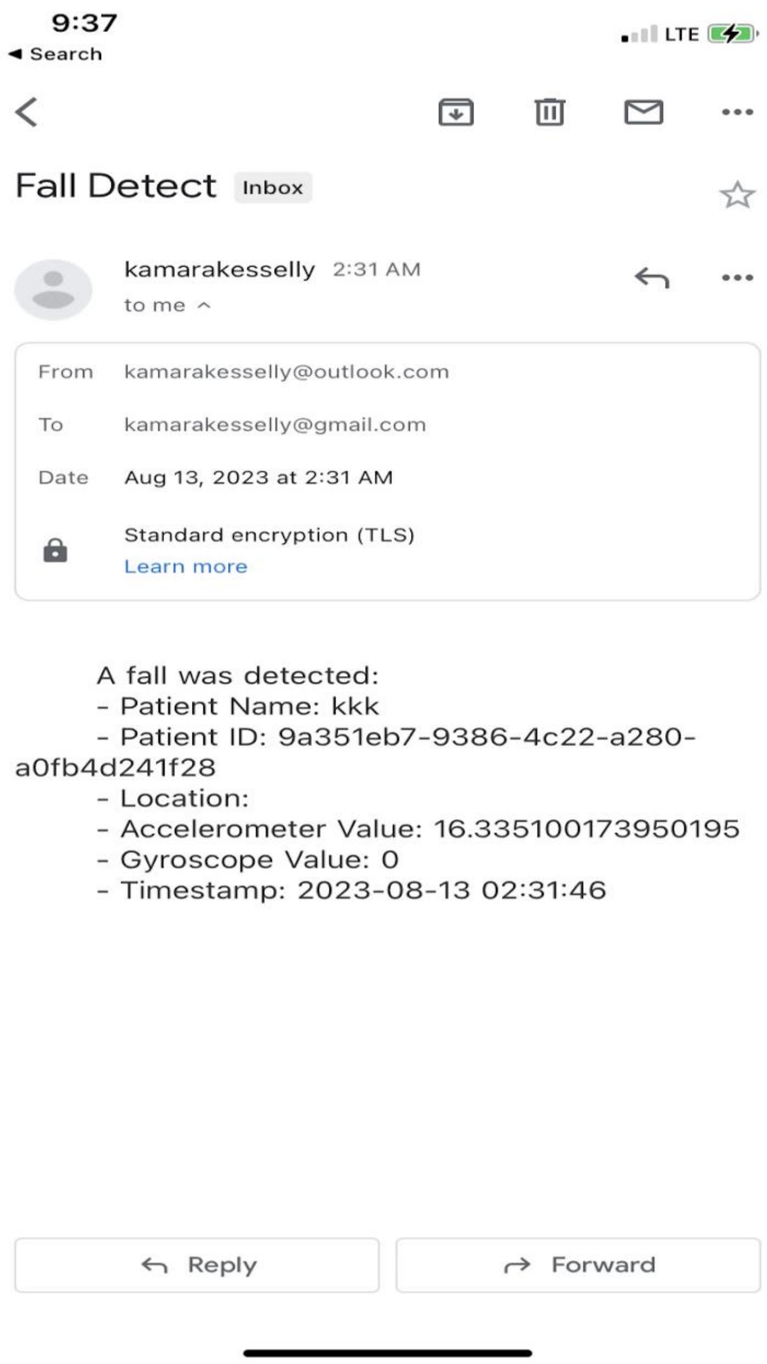
Figure 1.2 - Transmit Sensor Data, Patient Location, and Patient ID to a Server and alert nurse

1.2.4. Classify Motion Type Based on Sensor Data

One of the core functionalities of the FallArm application is motion classification. This objective entails developing algorithms that can analyze the generated accelerometer and gyroscope data to determine the type of motion being performed by the patient. This classification process may involve identifying patterns, detecting sudden changes, and applying predefined thresholds to differentiate between normal activities and potential falls. Accurate motion classification is pivotal for triggering appropriate responses and alerts.

1.2.5. Alert the Nurse via email in Case of a Fall

The project's fifth objective centers around patient safety by implementing a fall detection and alert system. If the motion classification algorithms indicate a fall, the system will promptly send an alert to the nurse responsible for the patient's care. This alert will be delivered via SMS, providing immediate notification of the fall incident and the patient's location. The timely delivery of alerts ensures swift intervention and assistance, potentially preventing further harm to the patient.



From	kamarakesselly@outlook.com
To	kamarakesselly@gmail.com
Date	Aug 13, 2023 at 2:31 AM
	Standard encryption (TLS) Learn more

```
        A fall was detected:
        - Patient Name: kkk
        - Patient ID: 9a351eb7-9386-4c22-a280-
a0fb4d241f28
        - Location:
        - Accelerometer Value: 16.335100173950195
        - Gyroscope Value: 0
        - Timestamp: 2023-08-13 02:31:46
```

Fig. 1.3 - Alert the Nurse via email in Case of a Fall

1.2.6. Convert the Android/Java Codebase to Android/Kotlin

The final objective of the project involves the technical improvement of the application by migrating the existing Android/Java codebase to Android/Kotlin. Kotlin is a modern programming language that offers enhanced expressiveness, conciseness, and safety. The migration process requires rewriting parts of the code while maintaining the existing functionality. This objective not only demonstrates technical proficiency but also positions the project for future maintenance and enhancements.

This project objectives encompass a holistic approach to developing a comprehensive fall detection and alert system. The successful accomplishment of these objectives will result in an application that generates realistic sensor data, accurately determines patient location, enables secure data transmission, classifies patient motions, alerts caregivers in case of falls, and enhances codebase quality through Kotlin migration. By addressing these objectives, the FallArm project aims to contribute to patient safety and well-being while showcasing technical competence and innovation.

1.3 Project Scope

The scope of the FallArm project outlines the boundaries within which the project will operate. It defines the specific features, functionalities, and limitations of the application, setting clear expectations for what will be included and what will not be part of the final solution. The project scope encompasses various technical and functional aspects, ensuring that the project's objectives are effectively met. The following sections provide a detailed overview of the scope of the FallArm project.

1.3.1 Developing Android Applications

The project scope encompasses the development of Android application that serve as the core components of the FallArm system. This application will be responsible for generating dynamic accelerometer and gyroscope data to simulate patient movements. They will also process geolocation information obtained from the device's GPS sensor to accurately determine the patient's location. Additionally, the applications will include user interfaces that facilitate interactions with the system and display relevant information to users and caregivers.

1.3.2 Establishing Communication Between Devices and Servers

An integral aspect of the project scope involves the establishment of a communication mechanism between Android devices and servers. This mechanism will enable the secure and efficient transmission of sensor data, patient location, and other relevant information to a central server for analysis and processing. The communication protocols and security measures implemented will ensure the integrity, confidentiality, and availability of the transmitted data.

1.3.3 Motion Classification and Alert System

The project scope extends to the development of motion classification algorithms that analyze the generated sensor data. These algorithms will categorize patient motions into distinct types, facilitating the identification of potential falls. Upon detecting a fall, the application will trigger an alert mechanism. The alert system will promptly notify caregivers, such as nurses, via SMS alerts containing critical information, including the patient's identification, location, and the nature of the detected fall.

1.3.4 Limitations of the Project Scope

While the FallArm project aims to deliver a comprehensive fall detection and alert system, it is important to acknowledge certain limitations within the project scope:

Physical Hardware Integration: The project will not involve the integration of external physical hardware devices or sensors. Instead, it will rely on Android emulators and devices for generating sensor data and geolocation information.

Fully-Fledged Alert System: While the project will implement a basic SMS alert mechanism to notify caregivers, it will not encompass the development of a complete emergency response system. Further enhancements and integrations would be required to create a full-fledged alert system with advanced features.

Third-Party Services: The project will not develop custom SMS sending services or extensive geolocation services. Instead, it will leverage existing third-party libraries and APIs for these functionalities.

User Interface Complexity: The user interface of the Android applications will focus primarily on displaying relevant information and facilitating interactions. However, the project will not dive deeply into advanced user experience design or complex visual interfaces.

The project scope encompasses the development of Android applications to generate sensor data, process geolocation information, establish communication, classify motion types, and trigger alerts. The scope also clarifies the limitations, emphasizing that the project will not involve physical hardware integration or the creation of a fully-fledged emergency response system. By adhering to the defined scope, the FallArm project aims to create a functional and focused solution that addresses fall detection and patient safety.

2. Project Tasks and Actions

2.4 Step 4: Determine Motion Type

This step involves the development of algorithms that analyze the generated sensor data to classify different types of patient motions accurately. The tasks and actions are as follows:

Develop Android Code for Motion Classification: Write robust and efficient algorithms that process the accelerometer and gyroscope data. These algorithms should be capable of identifying distinct patterns associated with various motion types, such as walking, running, sitting, and falling.

Implement Motion Classification Logic: Integrate the developed algorithms into the Android application. The code should interpret the sensor data and apply the classification logic to determine the current motion type accurately.

Testing and Validation: Rigorously test the motion classification algorithms using a variety of simulated scenarios and real-world data. The accuracy of the classification should be verified against ground truth data to ensure reliable results.

2.5 Step 5: Alert Nurse in Case of Fall

This step involves implementing a real-time notification mechanism to alert nurses when a fall is detected. The tasks include:

Integrate SMS Alert System: Integrate the SMS sending functionality into the Android application. When a fall is classified, trigger the SMS alert mechanism to promptly notify the nurse.

Create Alert Content: Develop a standardized message format for the SMS alerts. The message should provide essential information, including patient identification, location details, and the nature of the fall.

Ensure Robustness: Implement error handling mechanisms to ensure that SMS alerts are sent reliably even in adverse network conditions. This involves handling exceptions, retries, and fallback mechanisms.

2.6 Step 6: Conversion to Kotlin

This step involves the migration of the existing Android/Java codebase to the Kotlin programming language. The tasks and actions are as follows:

Codebase Analysis: Thoroughly review the existing Android/Java codebase to understand its structure, functionality, and dependencies.

Kotlin Conversion: Rewrite the relevant portions of the codebase in Kotlin syntax while maintaining the same functionality. Utilize Kotlin's concise and expressive features to enhance code readability and maintainability.

Testing and Verification: Test the converted Kotlin code extensively to ensure that the migration did not introduce any bugs or errors. Verify that all existing functionalities remain intact after the conversion.

Optimization Opportunities: Take advantage of Kotlin's features to optimize code segments and potentially improve the overall performance and efficiency of the application.

After execution of tasks, the FallArm project will attain its objectives of enhancing motion detection accuracy, ensuring rapid nurse notifications, and modernizing the codebase through Kotlin adoption. This comprehensive approach ensures that the project meets its goals while prioritizing accuracy, reliability, and maintainability.3. Project Timeline

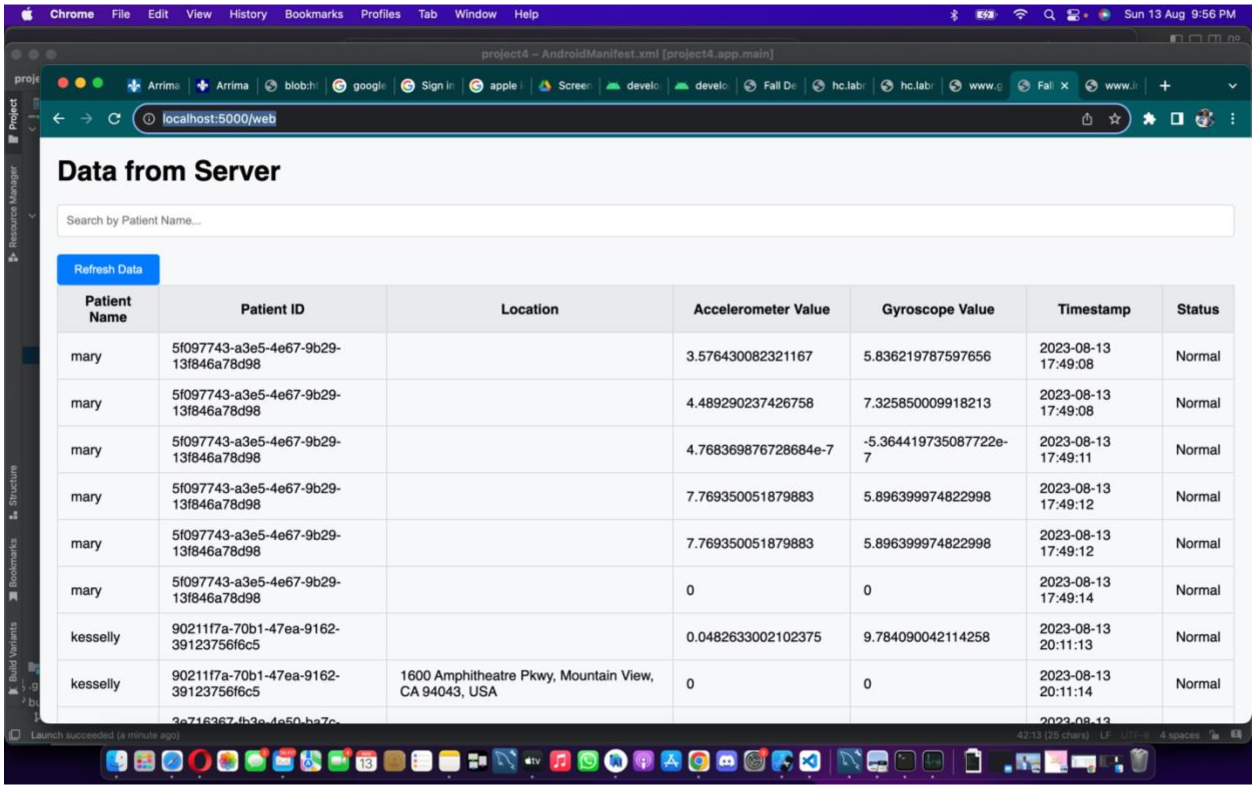
3. Project Timeline

3.1 Milestones and Deadlines

To ensure a structured and efficient project execution, the FallArm project will be divided into specific milestones, each marked by a set of tasks and corresponding deadlines. The project timeline is outlined below:

Milestone 1: Completion of Sensor Data Generation - Day 3
- Develop and integrate Android code for generating accelerometer and gyroscope data.
- Implement sensor simulation on an Android emulator.
- Test data generation with real devices for additional points.
- Deadline: By the end of Day 3.

Milestone 2: Geolocation Implementation - Day 6
- Implement a basic geolocation data application to retrieve and display location information.
- Explore advanced geolocation features (optional).
- Deadline: By the end of Day 6.



Data from Server

Patient Name	Patient ID	Location	Accelerometer Value	Gyroscope Value	Timestamp	Status
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		3.576430082321167	5.836219787597656	2023-08-13 17:49:08	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		4.489290237426758	7.325850009918213	2023-08-13 17:49:08	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		4.768369876728684e-7	-5.364419735087722e-7	2023-08-13 17:49:11	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		7.769350051879883	5.896399974822998	2023-08-13 17:49:12	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		7.769350051879883	5.896399974822998	2023-08-13 17:49:12	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		0	0	2023-08-13 17:49:14	Normal
kesselly	90211f7a-70b1-47ea-9162-39123756f6c5		0.0482633002102375	9.784090042114258	2023-08-13 20:11:13	Normal
kesselly	90211f7a-70b1-47ea-9162-39123756f6c5	1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA	0	0	2023-08-13 20:11:14	Normal
	3e716367-fb3e-4e50-ba7c-				2023-08-13	

Fig 3.1 – Retrieve Data

Milestone 3: Data Transmission and Security – Day 9
- Establish socket communication between Android client and PC server.
- Implement network security measures, including SSL.
- Test data transmission for accuracy and security.
- Deadline: By the end of Day 9.

Milestone 4: Motion Classification - Day 12
- Develop Android code for analyzing sensor data to classify motion types.
- Integrate motion classification logic into the application.
- Rigorously test the accuracy of motion classification algorithms.
- Deadline: By the end of Day 12.

Milestone 5: SMS Alert System - Day 15
- Integrate SMS alert system into the application.
- Develop standardized alert messages containing relevant fall information.
- Ensure robustness and reliability of the alert system.
- Deadline: By the end of Day 15.

Milestone 6: Conversion to Kotlin - Day 18
- Analyze the existing Android/Java codebase.
- Rewrite relevant portions of the codebase in Kotlin syntax.
- Test the converted Kotlin code for functionality and accuracy.
- Optimize code segments for enhanced performance.
- Deadline: By the end of Day 18.

By adhering to the timeline and achieving each milestone within the specified timeframe, the project aims to ensure steady progress and successful completion. The milestones and deadlines are designed to maintain a balance between thorough development, testing, and refinement, ultimately leading to a functional and reliable FallArm system.

4. Project Team

4.1 Team Leads of Major Roles

Project Design:	Yixuan Liang
Implementation:	Kesselly Kamara
Documentation:	Francis Etang

5. Resources and Budget

5.1 Required Resources
Android devices/emulators
Development environment (Android Studio)
Server resources for testing
Communication libraries (sockets, SMS APIs)

5.2 Estimated Budget
Development resources: $0
Server and testing resources: $0
Miscellaneous expenses: $0

6. Risk Management

Effective risk management is crucial for the success of the FallArm project. Identifying potential risks and implementing appropriate mitigation strategies will help minimize the impact of uncertainties and challenges. The project team has identified the following risks along with corresponding mitigation strategies:

6.1 Identified Risks

1. Sensor Data Accuracy Issues:
There is a risk that the generated sensor data may not accurately represent real-world patient motions, leading to incorrect motion classifications.

2. Server Communication Failures:
Communication between the Android client and the PC server might encounter failures, causing data transmission issues and compromising the reliability of the system.

3. Geolocation Accuracy Limitations:
Geolocation data might have inherent limitations in accuracy, resulting in incorrect patient location information being transmitted to the server.

6.2 Risk Mitigation Strategies

1. Extensive Testing on Multiple Devices:
To mitigate the risk of sensor data accuracy issues, the project team will conduct thorough testing using a variety of real devices. This will help identify any inconsistencies and ensure that the generated data closely matches real-world patient movements. If discrepancies are detected, adjustments to the data generation algorithms can be made.

2. Implementation of Error Handling in Communication Protocols:
To address the risk of server communication failures, the project team will implement robust error handling mechanisms within the communication protocols. This includes handling connection losses, timeouts, and other potential issues. The application will attempt to resend data in case of communication failures and provide appropriate feedback to users.

3. Research and Mitigation Strategies for Geolocation Inaccuracies:
To mitigate the risk of geolocation accuracy limitations, the team will conduct thorough research on potential sources of inaccuracies in geolocation data. Strategies will be explored to enhance the accuracy of location information, such as combining multiple data sources, using external APIs, and applying statistical techniques to filter out anomalies.

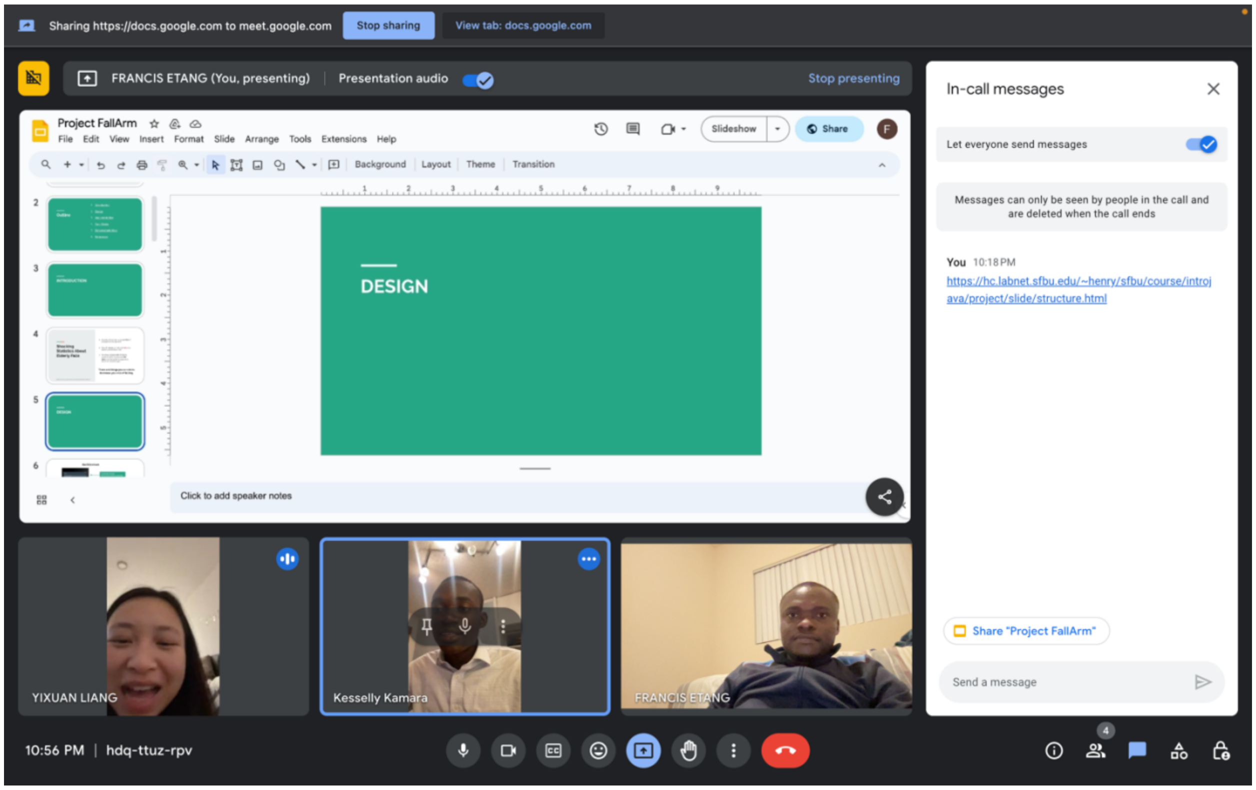
The FallArm project aims to minimize potential setbacks and ensure that the system functions reliably and accurately. Regular risk assessment and updates to the mitigation strategies will be carried out throughout the project's lifecycle to address any emerging challenges.

7. Communication Plan

Clear and effective communication is essential for ensuring that all project stakeholders are well-informed about the project's progress, challenges, and achievements. The FallArm project will have a structured communication plan to facilitate seamless communication within the project team members and with stakeholders.

7.1 Team Communication

Daily Team Meetings:
The project team will hold daily meetings to review progress, discuss ongoing tasks, and address any challenges or roadblocks. These meetings will serve as an opportunity for team members to share updates and provide insights.



Messages can only be seen by people in the call and are deleted when the call ends

You 10:18 PM
https://hc.labnet.sfbu.edu/~henry/sfbu/course/introjava/project/slide/structure.html

Fig. 7.1 – Team Meetings

Project Management Tools:
Communication within the team will also be facilitated through project management tools such as google meet and zoom. Text messages served as a platform for real-time messaging and discussions, while google docs will be used to track tasks, assign responsibilities, and monitor progress.

7.2 Stakeholder Communication

Weekly Progress Reports:
Stakeholders, including project sponsors and management, Professor Henry will receive weekly progress reports detailing the accomplishments, milestones reached, and challenges faced during that period. These reports will include key performance indicators, project metrics, and a summary of achievements.

FallArm project aims to maintain transparency, ensure alignment among team members, and keep stakeholders engaged and informed. Regular updates and open lines of communication will contribute to the project's overall success and mitigate any potential misunderstandings or delays.

8. Project Documentation and Reporting

Effective project documentation and reporting are essential for maintaining organization, tracking progress, and ensuring accountability throughout the FallArm project. The project team will implement a structured approach to manage documents and provide regular reports.

8.1 Document Management

GitHub Repository:
A dedicated GitHub repository will be established to serve as the central hub for both source code and project documentation. This repository will provide version control, collaboration tools, and a secure environment for storing project-related files.

File Naming Conventions:
To ensure easy organization and retrieval of documents, the project team will follow standardized file naming conventions. Clear and descriptive names will be used for documents, making it convenient to identify their content and purpose.

Google Docs for Documentation Management:
In addition to GitHub, Google Docs will be utilized to manage project documentation collaboratively. Google Docs offers real-time collaboration and easy sharing capabilities, allowing multiple team members to contribute and review documents simultaneously.

8.2 Reporting Structure

Weekly Progress Reports (Within the Team):
The project team will generate and share weekly progress reports detailing the tasks accomplished, challenges encountered, and goals achieved during the week. These reports will provide insights into individual and collective efforts, fostering collaboration and awareness.

Monthly Reports (For Stakeholders):
Stakeholders will receive monthly reports summarizing the project's overall progress. These reports will highlight milestones achieved, key performance indicators, and any notable developments. The reports will ensure that stakeholders are informed about the project's trajectory and its alignment with strategic goals.

By leveraging GitHub, Google Docs, and a structured reporting approach, the FallArm project aims to streamline document management, enhance collaboration, and provide clear insights into the project's status and achievements. This approach will contribute to effective communication and transparency among team members and stakeholders alike.

9. Stakeholder Engagement

9.1 Stakeholder Involvement
Regular updates through meetings and reports
Feedback solicitation for major decisions

9.2 Keeping Stakeholders Informed
Monthly progress reports
Immediate notifications for critical issues

10. Project Closure

As the FallArm project nears its conclusion, proper closure ensures that the project's outcomes are well-documented, lessons are learned, and valuable insights are captured for future reference. The project team will follow a structured approach to wrap up the project effectively.

10.1 Final Documentation

Comprehensive Documentation of Codebase and Processes:
To ensure knowledge transfer and maintainability, the project team will create comprehensive documentation of the entire codebase. This documentation will cover the architecture, code structure, algorithms, and any technical decisions made during the development process. The aim is to provide a clear understanding of the codebase for future maintenance or enhancements.

Summary of Achievements and Challenges:
The final documentation includes a summary of the project's achievements, milestones reached, and goals accomplished. Additionally, it will highlight the challenges faced and the strategies employed to overcome them. This reflection on accomplishments and setbacks provides valuable insights into the project's journey.

10.2 Lessons Learned

Identify Key Takeaways and Areas for Improvement:
The project team will conduct a thorough review to identify key lessons learned from the project. This includes analyzing what went well, what could have been done differently, and any unexpected hurdles that arose. These insights will help the team improve processes and decision-making in future projects.

The outcome of these lessons learnt are highlighted as follows:
1. Effective Collaboration: The project highlighted the importance of clear communication and collaboration within the team. Regular meetings and open discussions were key to keeping everyone aligned and informed about progress and challenges.

2. Testing and Validation: The significance of comprehensive testing became evident during the project. It emphasized the need for rigorous testing of all components to ensure accuracy and reliability, especially when dealing with critical patient data.

3. Real-World Data Variability: Dealing with real-world data showcased the variability in sensor readings and geolocation accuracy. This lesson underscores the need to account for these variations in algorithms and decision-making processes.

4. Security Considerations: Implementing network security and SSL protocols highlighted the critical nature of safeguarding sensitive patient data. The project emphasized the importance of staying up-to-date with security best practices to protect user information.

5. Agile Adaptation: The project's evolving requirements and unforeseen challenges underscored the value of an agile approach. Being adaptable and responsive to changing circumstances helped the team navigate unexpected roadblocks effectively.

6. Documentation Importance: Comprehensive documentation proved crucial for understanding code functionality and decision rationale. The lesson learned is the necessity of maintaining detailed documentation to aid future maintenance and knowledge transfer.

7. User-Centric Design: Developing an SMS alert system for nurses highlighted the importance of considering user experience and user interfaces. User-centered design principles should be applied to ensure ease of use and quick response times.

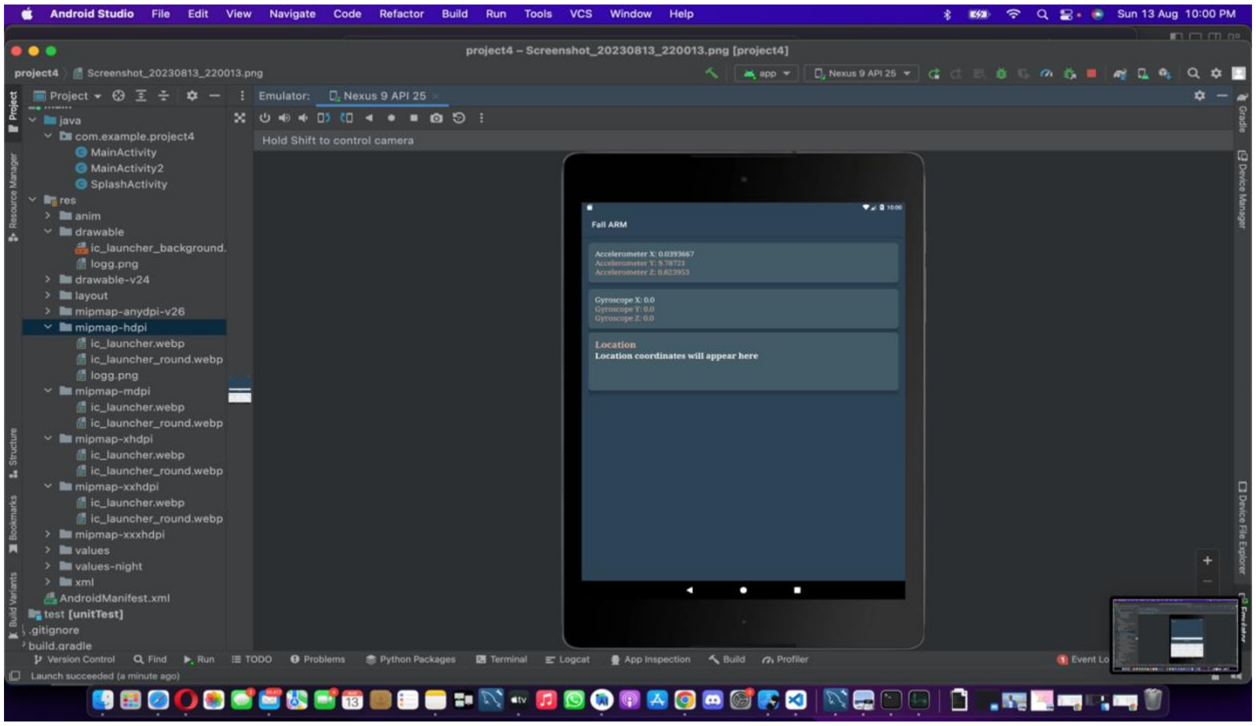
8. Resource Allocation: The project identified the significance of proper resource allocation, including time, manpower, and technology. It underscored the need for realistic project planning and management to achieve milestones effectively.

9. Continuous Learning: The project reinforced the idea that technology and tools are constantly evolving. The lesson learned is the importance of continuous learning and staying updated with new technologies to remain competitive and efficient.

10. Stakeholder Engagement: The project highlighted the role of stakeholder involvement in shaping the project's success. It emphasized the need for regular communication and alignment with stakeholders to ensure project objectives are met.

As mentioned earlier, these lessons learned provide valuable insights that can be applied to future projects to enhance efficiency, effectiveness, and overall project success.

Screenshots of Application:

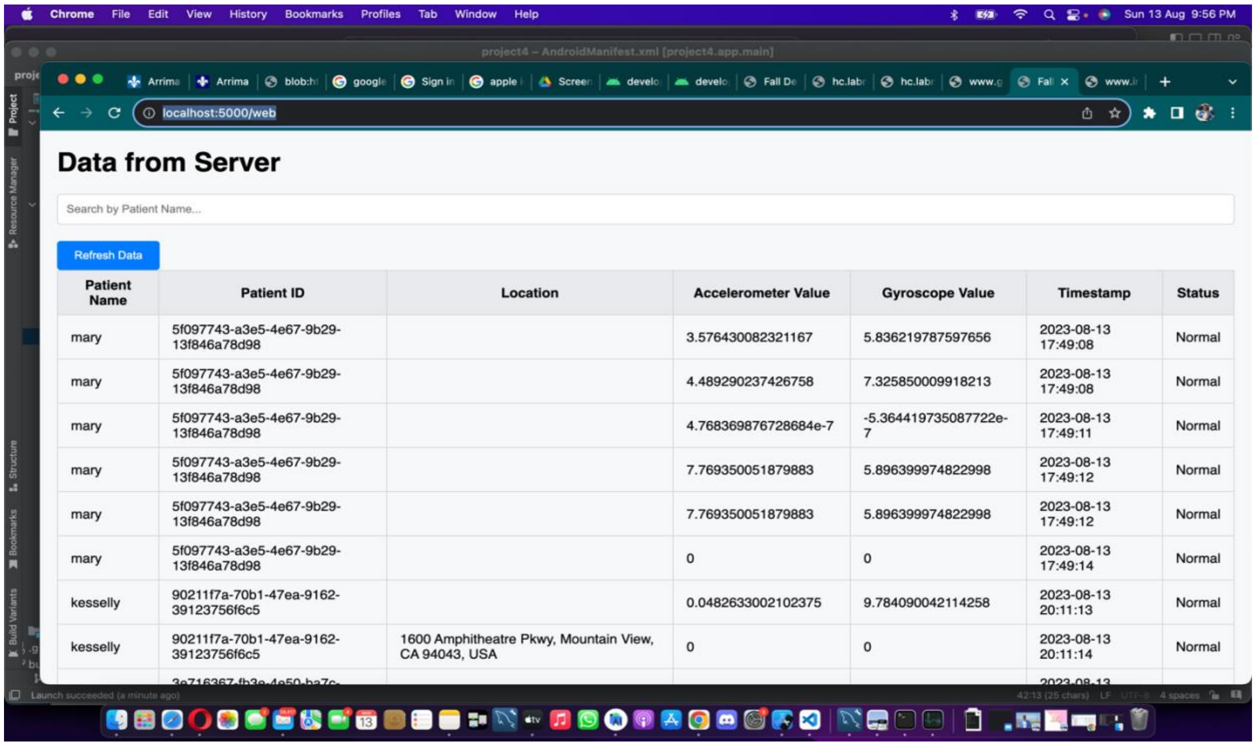


Fall ARM

Accelerometer X: 0.0393667
Accelerometer Y: 9.78721
Accelerometer Z: 0.823953

Gyroscope X: 0.0
Gyroscope Y: 0.0
Gyroscope Z: 0.0

Location
Location coordinates will appear here



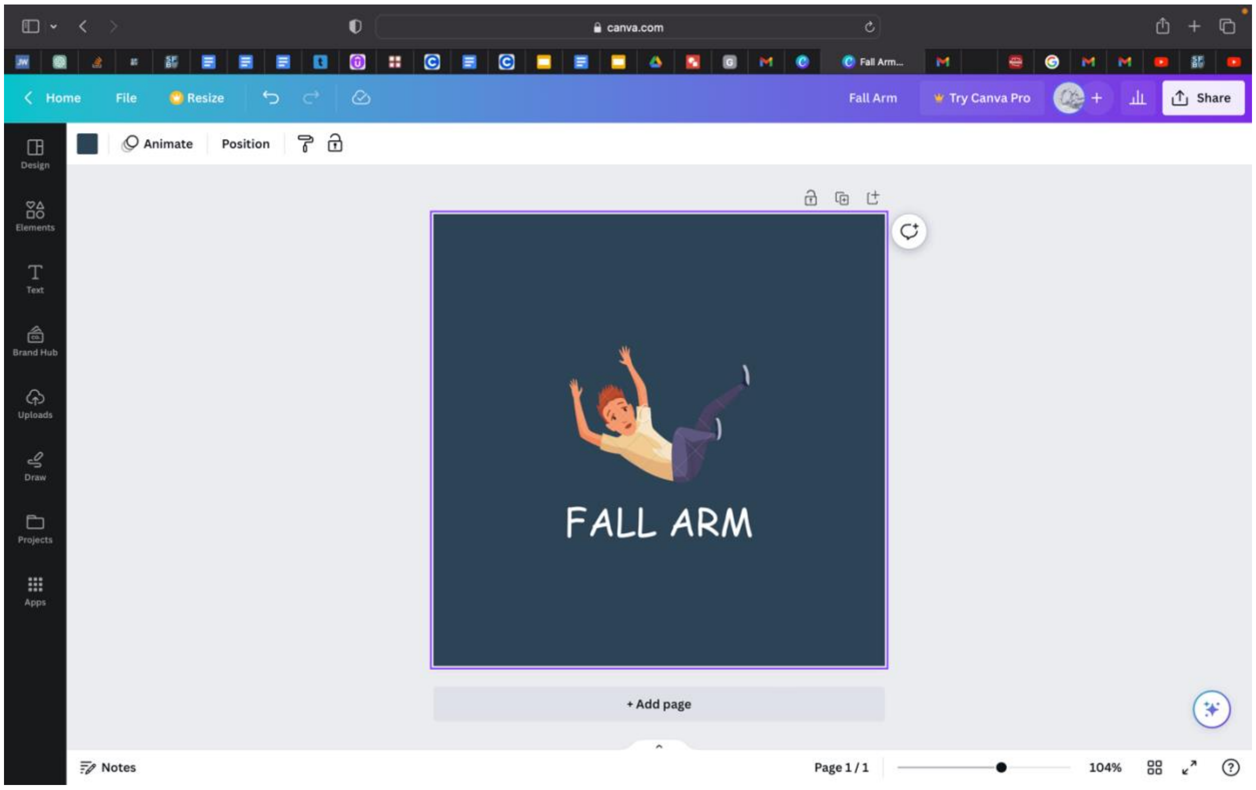
localhost:5000/web

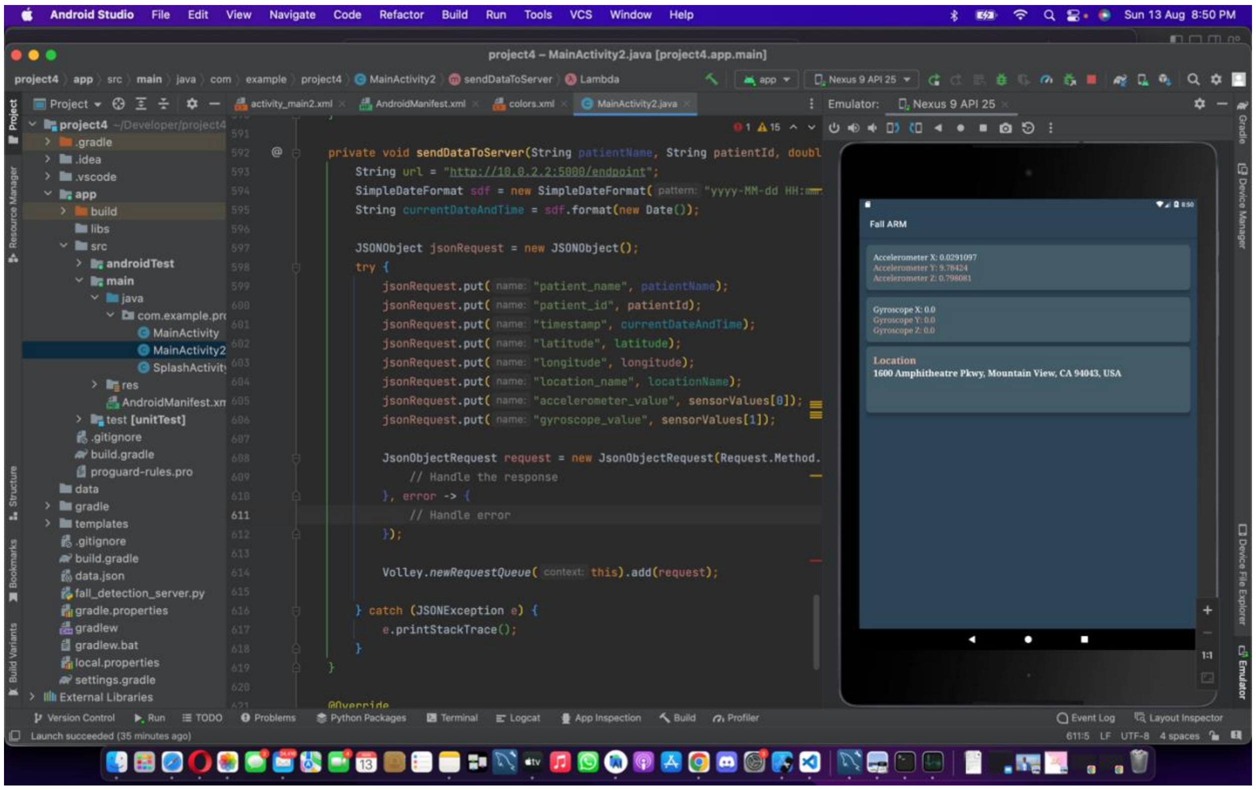
Data from Server

Search by Patient Name...

Refresh Data

Patient Name	Patient ID	Location	Accelerometer Value	Gyroscope Value	Timestamp	Status
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		3.576430082321167	5.836219787597656	2023-08-13 17:49:08	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		4.489290237426758	7.325850009918213	2023-08-13 17:49:08	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		4.768369876728684e-7	-5.364419735087722e-7	2023-08-13 17:49:11	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		7.769350051879883	5.896399974822998	2023-08-13 17:49:12	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		7.769350051879883	5.896399974822998	2023-08-13 17:49:12	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		0	0	2023-08-13 17:49:14	Normal
kesselly	90211f7a-70b1-47ea-9162-39123756f6c5		0.0482633002102375	9.784090042114258	2023-08-13 20:11:13	Normal
kesselly	90211f7a-70b1-47ea-9162-39123756f6c5	1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA	0	0	2023-08-13 20:11:14	Normal





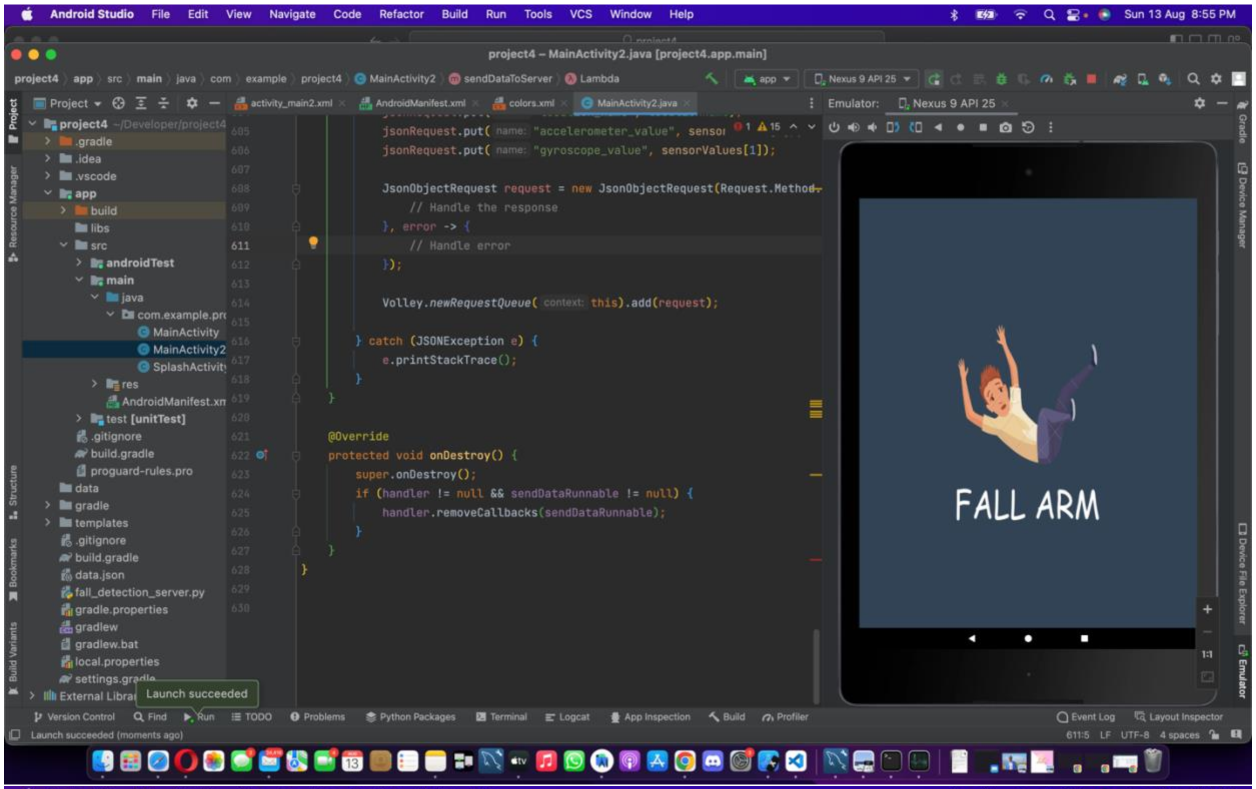
```java
private void sendDataToServer(String patientName, String patientId, doubl
    String url = "http://10.0.2.2:5000/endpoint";
    SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm
    String currentDateAndTime = sdf.format(new Date());

    JSONObject jsonRequest = new JSONObject();
    try {
        jsonRequest.put( name: "patient_name", patientName);
        jsonRequest.put( name: "patient_id", patientId);
        jsonRequest.put( name: "timestamp", currentDateAndTime);
        jsonRequest.put( name: "latitude", latitude);
        jsonRequest.put( name: "longitude", longitude);
        jsonRequest.put( name: "location_name", locationName);
        jsonRequest.put( name: "accelerometer_value", sensorValues[0]);
        jsonRequest.put( name: "gyroscope_value", sensorValues[1]);

        JsonObjectRequest request = new JsonObjectRequest(Request.Method.
            // Handle the response
        }, error -> {
            // Handle error
        });

        Volley.newRequestQueue( context: this).add(request);

    } catch (JSONException e) {
        e.printStackTrace();
    }
}

@Override
```

Fall ARM

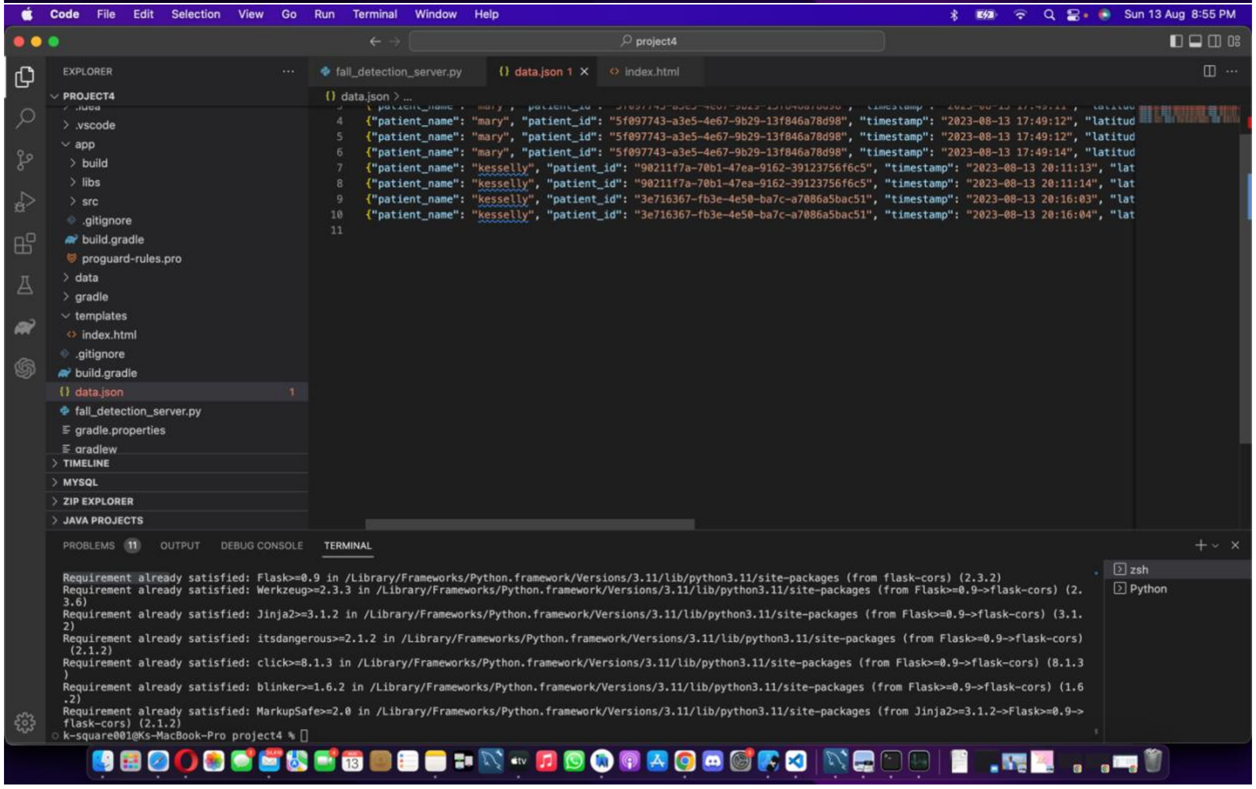
Accelerometer X: 0.0291097
Accelerometer Y: 9.78424
Accelerometer Z: 0.798081

Gyroscope X: 0.0
Gyroscope Y: 0.0
Gyroscope Z: 0.0

Location
1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA

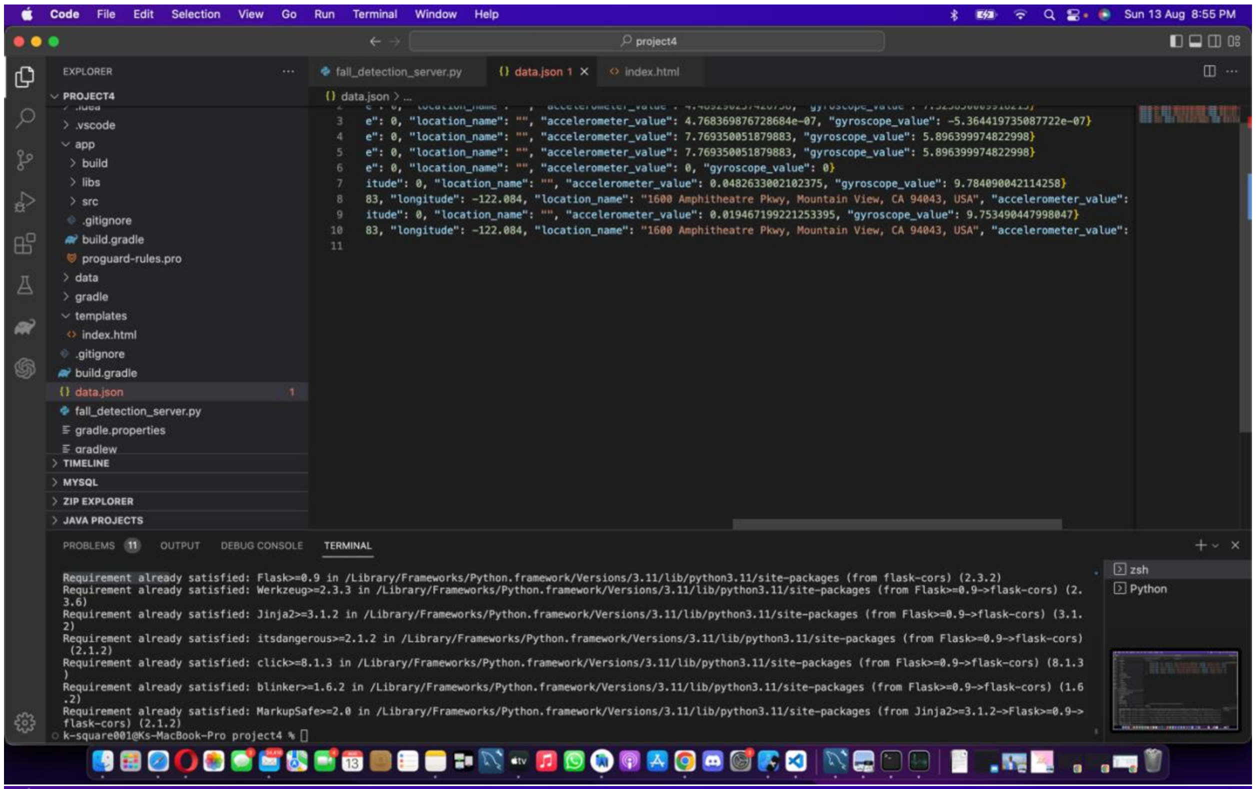


```java
            jsonRequest.put( name: "accelerometer_value", sensor
            jsonRequest.put( name: "gyroscope_value", sensorValues[1]);

            JsonObjectRequest request = new JsonObjectRequest(Request.Method
                // Handle the response
            }, error -> {
                // Handle error
            });

            Volley.newRequestQueue( context: this).add(request);

        } catch (JSONException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (handler != null && sendDataRunnable != null) {
            handler.removeCallbacks(sendDataRunnable);
        }
    }
}
```



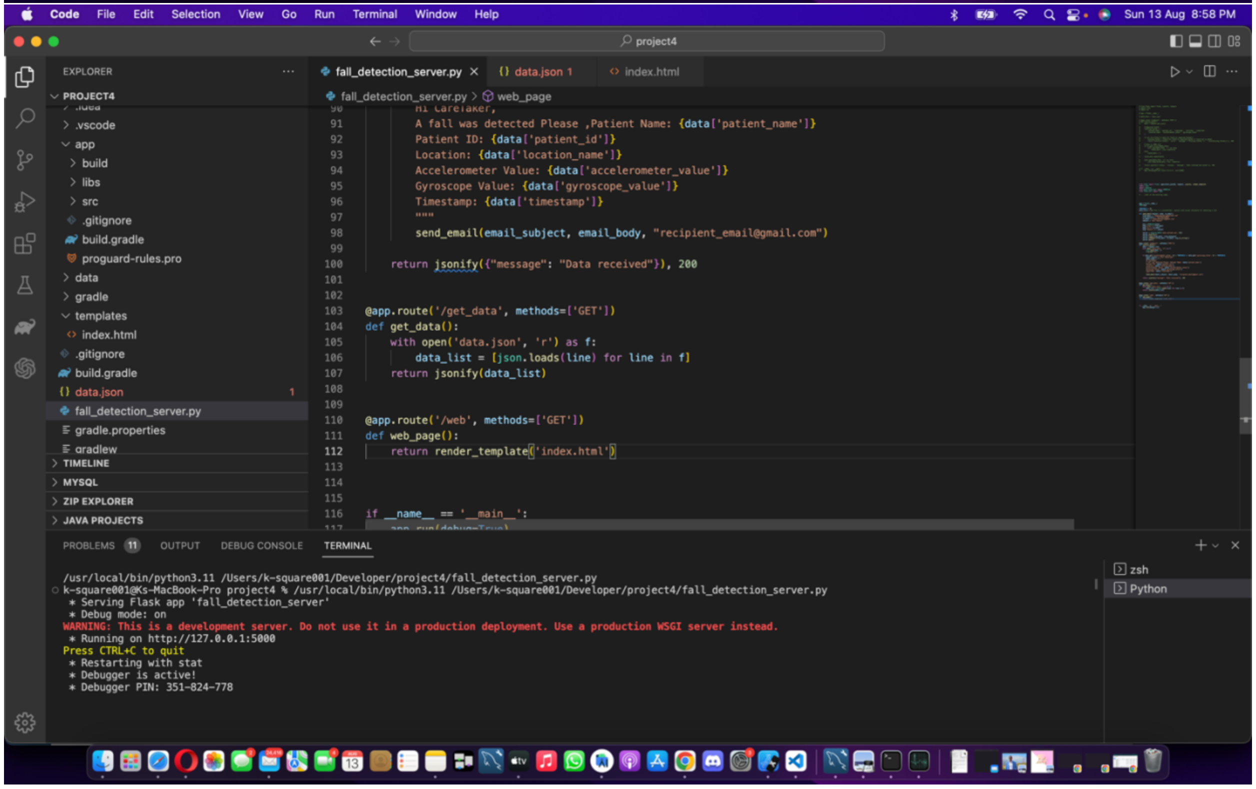
```json
{"patient_name": "mary", "patient_id": "5f097743-a3e5-4e67-9b29-13f846a78d98", "timestamp": "2023-08-13 17:49:12", "latitud
{"patient_name": "mary", "patient_id": "5f097743-a3e5-4e67-9b29-13f846a78d98", "timestamp": "2023-08-13 17:49:12", "latitud
{"patient_name": "mary", "patient_id": "5f097743-a3e5-4e67-9b29-13f846a78d98", "timestamp": "2023-08-13 17:49:14", "latitud
{"patient_name": "kesselly", "patient_id": "90211f7a-70b1-47ea-9162-39123756f6c5", "timestamp": "2023-08-13 20:11:13", "lat
{"patient_name": "kesselly", "patient_id": "90211f7a-70b1-47ea-9162-39123756f6c5", "timestamp": "2023-08-13 20:11:14", "lat
{"patient_name": "kesselly", "patient_id": "3e716367-fb3e-4e50-ba7c-a7086a5bac51", "timestamp": "2023-08-13 20:16:03", "lat
{"patient_name": "kesselly", "patient_id": "3e716367-fb3e-4e50-ba7c-a7086a5bac51", "timestamp": "2023-08-13 20:16:04", "lat
```

```
Requirement already satisfied: Flask>=0.9 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from flask-cors) (2.3.2)
Requirement already satisfied: Werkzeug>=2.3.3 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (2.
3.6)
Requirement already satisfied: Jinja2>=3.1.2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (3.1.
2)
Requirement already satisfied: itsdangerous>=2.1.2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors)
 (2.1.2)
Requirement already satisfied: click>=8.1.3 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (8.1.3
)
Requirement already satisfied: blinker>=1.6.2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (1.6
.2)
Requirement already satisfied: MarkupSafe>=2.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Jinja2>=3.1.2->Flask>=0.9->
flask-cors) (2.1.2)
k-square001@Ks-MacBook-Pro project4 %
```



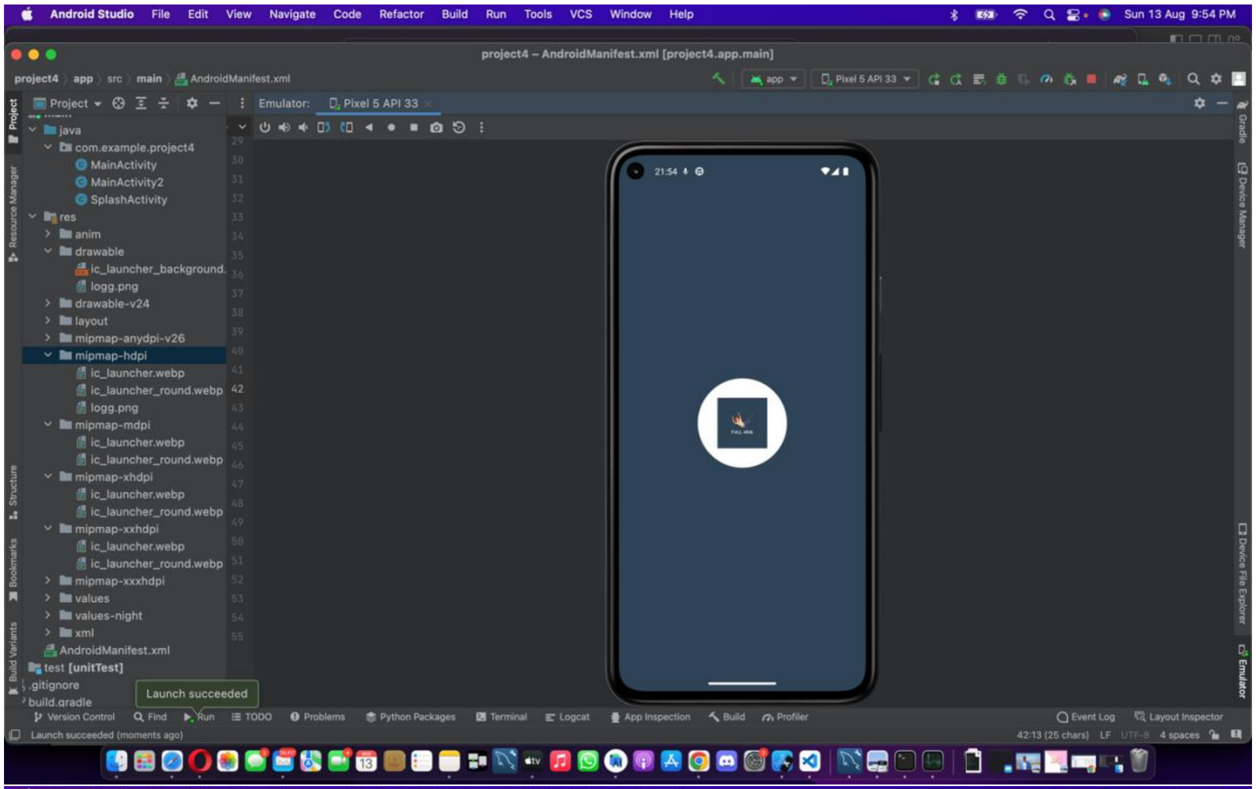
```
e": 0, "location_name": "", "accelerometer_value": 4.768369876728684e-07, "gyroscope_value": -5.364419735087722e-07}
e": 0, "location_name": "", "accelerometer_value": 7.769350051879883, "gyroscope_value": 5.896399974822998}
e": 0, "location_name": "", "accelerometer_value": 7.769350051879883, "gyroscope_value": 5.896399974822998}
e": 0, "location_name": "", "accelerometer_value": 0, "gyroscope_value": 0}
itude": 0, "location_name": "", "accelerometer_value": 0.0482633002102375, "gyroscope_value": 9.784090042114258}
83, "longitude": -122.084, "location_name": "1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA", "accelerometer_value":
itude": 0, "location_name": "", "accelerometer_value": 0.019467199221253395, "gyroscope_value": 9.753490447998047}
83, "longitude": -122.084, "location_name": "1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA", "accelerometer_value":
```

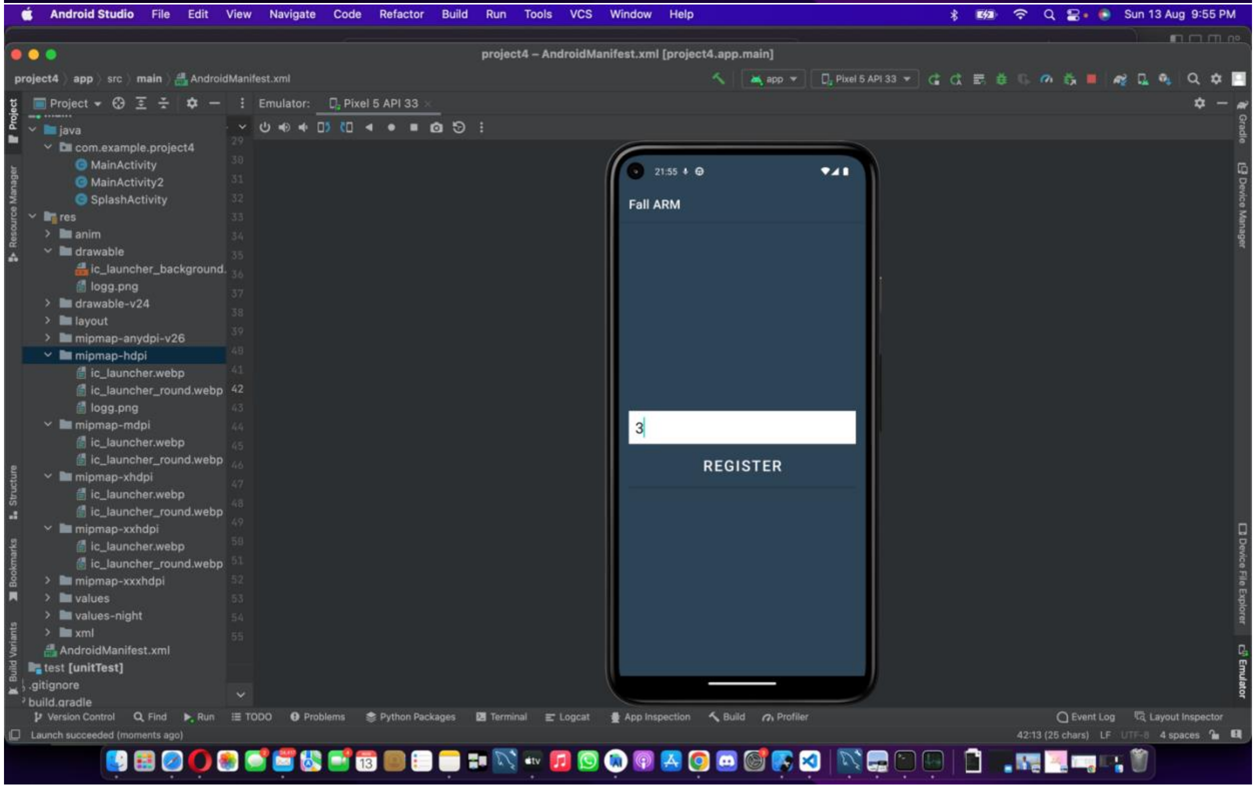
```
Requirement already satisfied: Flask>=0.9 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from flask-cors) (2.3.2)
Requirement already satisfied: Werkzeug>=2.3.3 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (2.
3.6)
Requirement already satisfied: Jinja2>=3.1.2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (3.1.
2)
Requirement already satisfied: itsdangerous>=2.1.2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors)
 (2.1.2)
Requirement already satisfied: click>=8.1.3 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (8.1.3
)
Requirement already satisfied: blinker>=1.6.2 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Flask>=0.9->flask-cors) (1.6
.2)
Requirement already satisfied: MarkupSafe>=2.0 in /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages (from Jinja2>=3.1.2->Flask>=0.9->
flask-cors) (2.1.2)
k-square001@Ks-MacBook-Pro project4 %
```

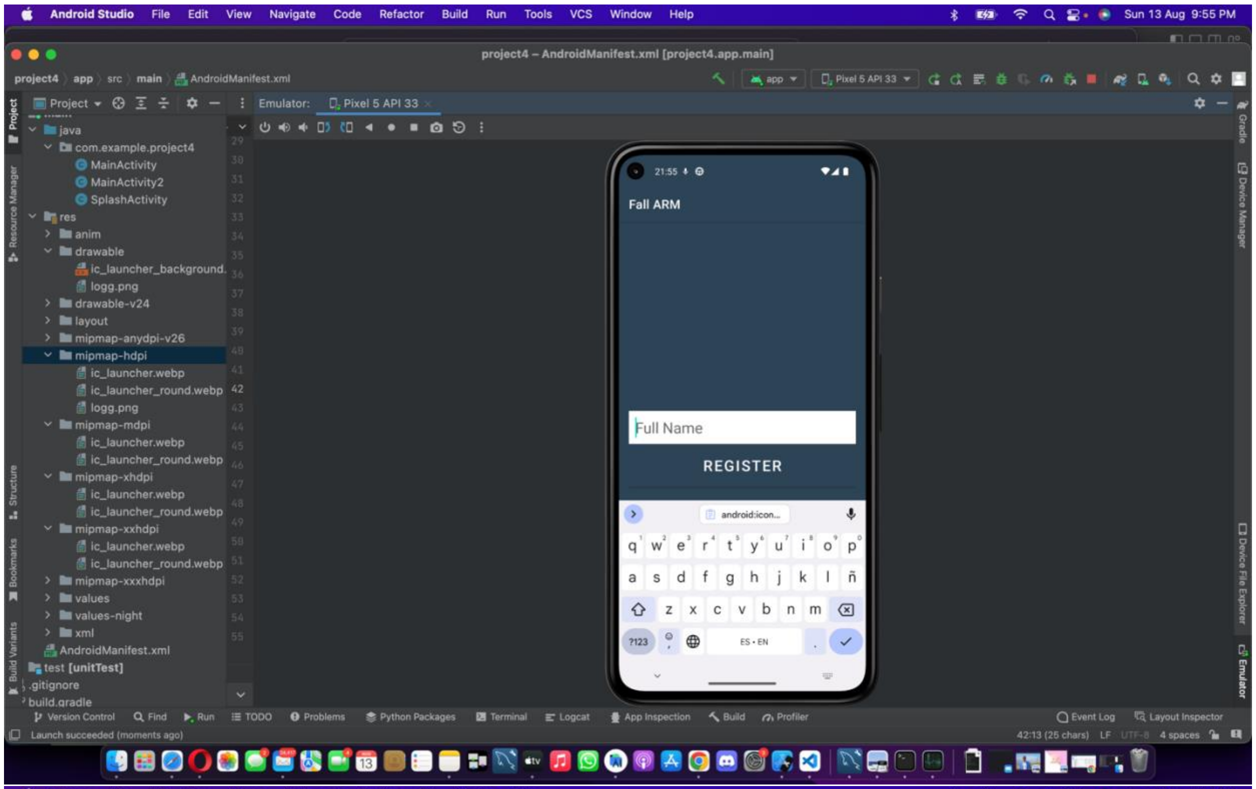


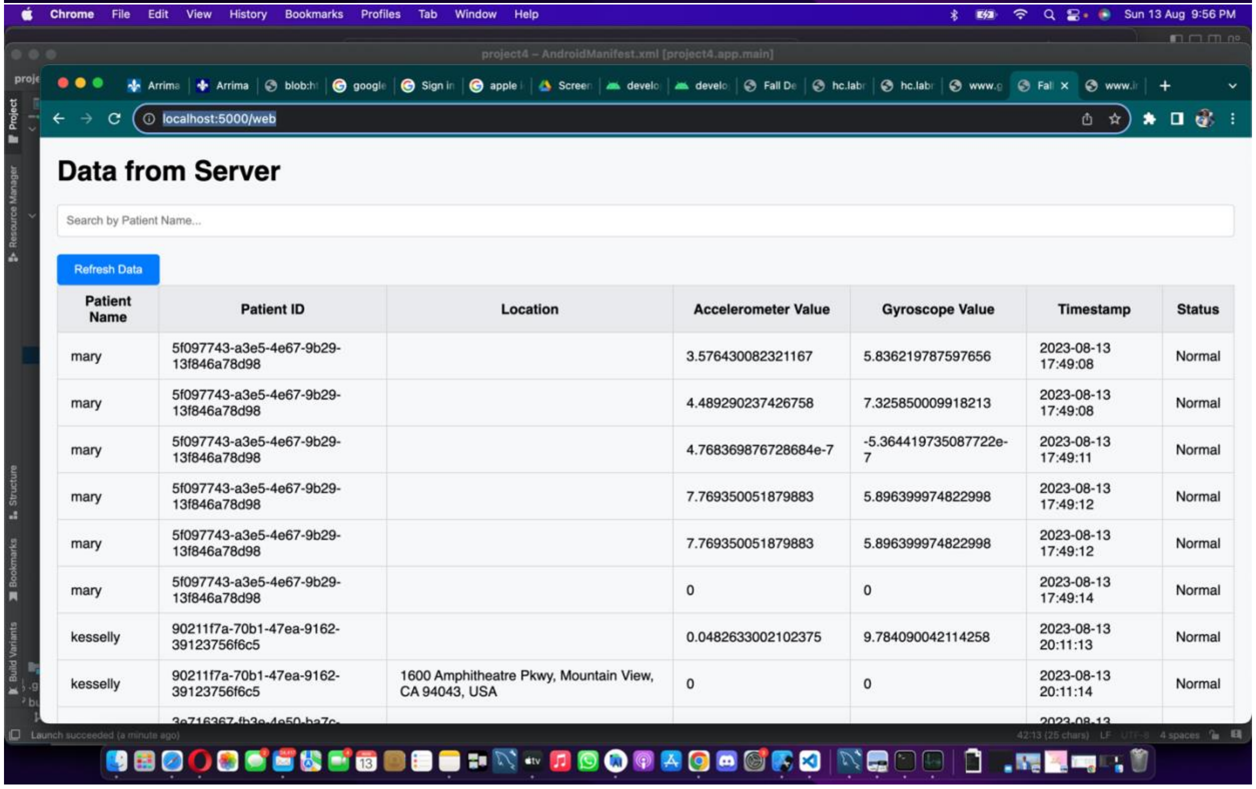
```python
        A fall was detected Please ,Patient Name: {data['patient_name']}
        Patient ID: {data['patient_id']}
        Location: {data['location_name']}
        Accelerometer Value: {data['accelerometer_value']}
        Gyroscope Value: {data['gyroscope_value']}
        Timestamp: {data['timestamp']}
        """
        send_email(email_subject, email_body, "recipient_email@gmail.com")

    return jsonify({"message": "Data received"}), 200


@app.route('/get_data', methods=['GET'])
def get_data():
    with open('data.json', 'r') as f:
        data_list = [json.loads(line) for line in f]
    return jsonify(data_list)


@app.route('/web', methods=['GET'])
def web_page():
    return render_template('index.html')



if __name__ == '__main__':
```

```
/usr/local/bin/python3.11 /Users/k-square001/Developer/project4/fall_detection_server.py
k-square001@Ks-MacBook-Pro project4 % /usr/local/bin/python3.11 /Users/k-square001/Developer/project4/fall_detection_server.py
 * Serving Flask app 'fall_detection_server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 351-824-778
```


Android Studio File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Sun 13 Aug 9:54 PM
project4 – AndroidManifest.xml [project4.app.main]
Emulator: Pixel 5 API 33
Launch succeeded
Launch succeeded (moments ago)


Android Studio File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
Sun 13 Aug 9:55 PM
project4 – AndroidManifest.xml [project4.app.main]
Emulator: Pixel 5 API 33
Fall ARM
3
REGISTER
Launch succeeded (moments ago)


project4 – AndroidManifest.xml [project4.app.main]
Emulator: Pixel 5 API 33
Fall ARM
Full Name
REGISTER
Launch succeeded (moments ago)



Data from Server

Patient Name	Patient ID	Location	Accelerometer Value	Gyroscope Value	Timestamp	Status
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		3.576430082321167	5.836219787597656	2023-08-13 17:49:08	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		4.489290237426758	7.325850009918213	2023-08-13 17:49:08	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		4.768369876728684e-7	-5.364419735087722e-7	2023-08-13 17:49:11	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		7.769350051879883	5.896399974822998	2023-08-13 17:49:12	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		7.769350051879883	5.896399974822998	2023-08-13 17:49:12	Normal
mary	5f097743-a3e5-4e67-9b29-13f846a78d98		0	0	2023-08-13 17:49:14	Normal
kesselly	90211f7a-70b1-47ea-9162-39123756f6c5		0.0482633002102375	9.784090042114258	2023-08-13 20:11:13	Normal
kesselly	90211f7a-70b1-47ea-9162-39123756f6c5	1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA	0	0	2023-08-13 20:11:14	Normal
	3e716367-fb3e-4e50-ba7c-				2023-08-13	

Code Implementation:

Codes:

```
package com.example.project4;
import android.Manifest;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.os.Bundle;
import android.os.Handler;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import com.android.volley.Request;
import com.android.volley.toolbox.JsonObjectRequest;
import com.android.volley.toolbox.Volley;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.tasks.OnSuccessListener;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Locale;
public class MainActivity2 extends AppCompatActivity implements
SensorEventListener {
private static final long TWO_HOURS_IN_MILLIS = 7200000;
private static final int LOCATION_REQUEST_CODE = 1;
private static final String PREFERENCE_NAME = "your_preference_name";
private SensorManager sensorManager;
private FusedLocationProviderClient fusedLocationClient;
private long lastDataSentTime = 0;
private TextView accelerometerXTextView, accelerometerYTextView,
accelerometerZTextView;
private TextView gyroscopeXTextView, gyroscopeYTextView, gyroscopeZTextView;
private TextView locationText1;
private Geocoder geocoder;
private SharedPreferences sharedPref;
private String patientName, patientId;
private double latitude, longitude;
private Handler handler;
private Runnable sendDataRunnable;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main2);
geocoder = new Geocoder(this, Locale.getDefault());
initializeSharedPrefs();
initializeSensors();
```

```
initializeTextViews();
initializeLocationClient();
initializeHandler();
requestLocationPermissionIfNeeded();
handler.postDelayed(sendDataRunnable, TWO_HOURS_IN_MILLIS);
getLastLocation();
}
private void initializeSharedPrefs() {
sharedPref = getSharedPreferences(PREFERENCE_NAME, MODE_PRIVATE);
patientName = sharedPref.getString("patient_name_key", "Default Name222");
patientId = sharedPref.getString("patient_id_key", "Default ID");
latitude = Double.parseDouble(sharedPref.getString("latitude_key", "0.0"));
longitude = Double.parseDouble(sharedPref.getString("longitude_key", "0.0"));
}
private void initializeSensors() {
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
registerSensor(Sensor.TYPE_ACCELEROMETER);
registerSensor(Sensor.TYPE_GYROSCOPE);
}
private void registerSensor(int sensorType) {
Sensor sensor = sensorManager.getDefaultSensor(sensorType);
if (sensor != null) {
sensorManager.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_NORMAL);
}
}
private void initializeTextViews() {
accelerometerXTextView = findViewById(R.id.accelerometerX);
accelerometerYTextView = findViewById(R.id.accelerometerY);
accelerometerZTextView = findViewById(R.id.accelerometerZ);
gyroscopeXTextView = findViewById(R.id.gyroscopeX);
gyroscopeYTextView = findViewById(R.id.gyroscopeY);
gyroscopeZTextView = findViewById(R.id.gyroscopeZ);
locationText1 = findViewById(R.id.locationText);
}
private void initializeLocationClient() {
fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
}
private void initializeHandler() {
sendDataRunnable = new Runnable() {
@Override
public void run() {
getLastLocation();
sendDataToServer(patientName, patientId, latitude, longitude, "", new
float[]{0, 0, 0});
handler.postDelayed(this, TWO_HOURS_IN_MILLIS);
}
};
handler = new Handler();
}
private boolean detectFall(float[] sensorValues) {
return sum(sensorValues) > 14.800000;
}
private float sum(float[] values) {
float total = 0;
for (float value : values) {
total += value;
}
return total;
```

```
}
@Override
public void onSensorChanged(SensorEvent event) {
float[] sensorValues = event.values;
int sensorType = event.sensor.getType();
if (sensorType == Sensor.TYPE_ACCELEROMETER) {
accelerometerXTextView.setText("Accelerometer X: " + event.values[0]);
accelerometerYTextView.setText("Accelerometer Y: " + event.values[1]);
accelerometerZTextView.setText("Accelerometer Z: " + event.values[2]);
} else if (sensorType == Sensor.TYPE_GYROSCOPE) {
gyroscopeXTextView.setText("Gyroscope X: " + event.values[0]);
gyroscopeYTextView.setText("Gyroscope Y: " + event.values[1]);
gyroscopeZTextView.setText("Gyroscope Z: " + event.values[2]);
}
if (detectFall(event.values)) {
sendDataToServer(patientName, patientId, latitude, longitude, "", sensorValues);
lastDataSentTime = System.currentTimeMillis();
} else if (System.currentTimeMillis() lastDataSentTime >=
TWO_HOURS_IN_MILLIS) {
sendDataToServer(patientName, patientId, latitude, longitude, "", sensorValues);
lastDataSentTime = System.currentTimeMillis();
}
}
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
super.onRequestPermissionsResult(requestCode, permissions, grantResults);
if (requestCode == LOCATION_REQUEST_CODE) {
if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
getLastLocation();
} else {
locationText1.setText("Location permissions denied.");
}
}
}
private boolean checkLocationPermissions() {
return ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED
&& ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) ==
PackageManager.PERMISSION_GRANTED;
}
public String updateUIWithLocation(Location location) {
String addressString = "Unknown Location";
try {
List<Address> addresses = geocoder.getFromLocation(location.getLatitude(),
location.getLongitude(), 1);
if (addresses.size() > 0) {
Address address = addresses.get(0);
addressString = address.getAddressLine(0);
}
} catch (IOException e) {
e.printStackTrace();
}
```

```
locationText1.setText(addressString);
return addressString;
}
private void requestLocationPermissionIfNeeded() {
if (!checkLocationPermissions()) {
ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.ACCESS_COARSE_LOCATION},
LOCATION_REQUEST_CODE);
}
}
private void getLastLocation() {
if (checkLocationPermissions()) {
fusedLocationClient.getLastLocation().addOnSuccessListener(this, new
OnSuccessListener<Location>() {
@Override
public void onSuccess(Location location) {
if (location != null) {
latitude = location.getLatitude();
longitude = location.getLongitude();
String address = updateUIWithLocation(location);
sendDataToServer(patientName, patientId, latitude, longitude, address,
new float[]{0, 0, 0});
}
}
});
}
}
```

//Sending the Data to the Server

```
private void sendDataToServer(String patientName, String
patientId, double latitude, double longitude, String locationName,
float[] sensorValues) {
String url = "http://10.0.2.2:5000/endpoint";
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss", Locale.US) I ;
String currentDateAndTime = sdf.format(new Date());
JSONObject jsonRequest = new JSONObject();
try {
jsonRequest.put("patient_name", patientName);
jsonRequest.put("patient_id", patientId);
jsonRequest.put("timestamp", currentDateAndTime);
jsonRequest.put("latitude", latitude);
jsonRequest.put("longitude", longitude);
jsonRequest.put("location_name", locationName);
jsonRequest.put("accelerometer_value", sensorValues[0]);
jsonRequest.put("gyroscope_value", sensorValues[1]);
JsonObjectRequest request = new
JsonObjectRequest(Request.Method.POST, url, jsonRequest,
response > {
// Handle the response
}, error > {
// Handle error
});
Volley.newRequestQueue(this).add(request);
} catch (JSONException e) {
e.printStackTrace();
}
}
@Override
protected void onDestroy() {
super.onDestroy();
if (handler != null && sendDataRunnable != null) {
handler.removeCallbacks(sendDataRunnable);
}
}
}
```

//Sending the Data to the Server explain

This is the explanation of this code:
1. Function sendDataToServer:
Purpose: This function is responsible for sending data to a server endpoint via an HTTP POST request.
Parameters:
patientName: Name of the patient.
patientId: ID of the patient.
latitude and longitude : Geographical coordinates.
locationName : Name of the location.
sensorValues : Array with sensor data, presumably accelerometer and gyroscope values.
Variables Inside Function:
url : This is the server endpoint where data will be sent.
sdf : An object of SimpleDateFormat to format the current date and time.
currentDateAndTime : Stores the current date and time formatted as "yyyy-MM-dd HH:mm:ss".
Building the JSON Request :
An instance of JSONObject named jsonRequest is created.
Various key-value pairs are then put into this jsonRequest to construct the JSON body of the POST request.
Sending the Request using Volley :
A JsonObjectRequest is created, which represents an HTTP request for a JSON object.
If the request is successful, the first lambda function (response > {}) can be used to handle the response.
If there's an error in the request, the second lambda function (error > {}) is triggered.
Finally, the request is added to Volley's request queue to be executed.
2.onDestroy Method :
Purpose : This method is an overridden method from the Android Activity lifecycle. It's called when the activity is being destroyed.
Inside the Method :
The method checks if the handler and sendDataRunnable are not null.
If they are not null, it removes any pending callbacks of sendDataRunnable from the handler . This is likely done to prevent memory leaks and to ensure that no background tasks continue after the activity is destroyed.
In a nutshell, this code is designed to send patient and sensor data to a server in a structured JSON format. Additionally, it ensures that any background tasks (presumably set up elsewhere in your code with handler and sendDataRunnable) are cleaned up when the activity is destroyed.

Complete Server Code

```
from flask import Flask, appcontext_pushed, request, jsonify, render_template
import json
import smtplib
from email.mime.text import MIMEText
from flask_cors import CORS
# ... [rest of the existing code]
app = Flask(__name__)
CORS(app)
THRESHOLD = 10
data_store = []# This is a placeholder; replace with actual threshold for detecting a
fall
def send_email(subject, body, to_email):
from_email = "kamarakesselly@outlook.com"
from_password = "password"
to_email = "kamarakesselly@gmail.com"
subject = "Fall Detect"
msg = MIMEText(body)
msg['From'] = from_email
msg['To'] = to_email
msg['Subject'] = subject
server = smtplib.SMTP('smtp.outlook.com', 587)
server.starttls()
server.login(from_email, from_password)
server.sendmail(from_email, to_email, msg.as_string())
server.quit()
@app.route('/endpoint', methods=['POST'])
def receive_data():
data = request.json
with open('data.json', 'a') as f:
json.dump(data, f)
f.write('\n')
if data.get('accelerometer_value', 0) > THRESHOLD or data.get('gyroscope_value', 0)
> THRESHOLD:
email_subject = "Fall Detected!"
email_body = f"""
Hi CareTaker,
A fall was detected Please ,Patient Name: {data['patient_name']}
Patient ID: {data['patient_id']}
Location: {data['location_name']}
Accelerometer Value: {data['accelerometer_value']}
Gyroscope Value: {data['gyroscope_value']}
Timestamp: {data['timestamp']}
"""
send_email(email_subject, email_body, "recipient_email@gmail.com")
return jsonify({"message": "Data received"}), 200
@app.route('/get_data', methods=['GET'])
def get_data():
with open('data.json', 'r') as f:
data_list = [json.loads(line) for line in f]
return jsonify(data_list)
@app.route('/web', methods=['GET'])
def web_page():
return render_template('index.html')
if __name__ == '__main__':
app.run(debug=True)
```

Data Retrieve From the Server Database

This is a Flask-based Python web application, and have the capability to receive sensor data, detect potential falls based on the received data, and send email notifications when falls are detected.

Here is a breakdown of the code:

1. Imports :

Flask : Main library to create the web application.

appcontext_pushed, request, jsonify, render_template : Utilities and methods from Flask to handle context, HTTP requests, JSON responses, and templates rendering.

json : Module to handle JSON operations.

smtplib, MIMEText : Used to send emails.

flask_cors, CORS : To handle Cross-Origin Resource Sharing (CORS) for the Flask app.

2. Flask App and CORS Initialization :
This initializes the Flask application and sets it up to handle CORS. This is particularly useful if your frontend application is hosted on a different domain or port than your backend.
3. Global Variables:
THRESHOLD: A constant that determines the value beyond which a reading is considered to indicate a fall.
data_store: A list that is meant to store the received data.
4. send_email function:
This function sends an email using the Outlook SMTP server. It takes in `subject`, `body`, and `to_email` as parameters but currently overwrites the `to_email` and `subject`.
5. Endpoint: /endpoint:
This is an HTTP POST endpoint.
It receives data in JSON format, appends it to a file named `data.json`.
If the received data indicates a fall (values exceed the threshold), it sends an email with the details of the fall.
Finally, it responds with a JSON object `{"message": "Data received"}`.
6. Endpoint: /get_data :
This is an HTTP GET endpoint.
When accessed, it reads the `data.json` file and loads all the data in it.
It then sends this data back in JSON format.
7. Endpoint: /web :
This is an HTTP GET endpoint.
It serves an HTML page (`index.html`) from the templates directory when accessed.
8. Main Execution :
if __name__ == '__main__':
app.run(debug=True)
This block checks if this script is the main module being run, and if so, it starts the Flask app in debug mode.
Important Considerations:
1. Security : This code contains email login credentials in plaintext. This is a significant security risk. In a real-world scenario,
2. Thresholds : The `THRESHOLD` is set to 10, but it's commented that it's a placeholder. Adjustments may be required based on real-world data to accurately detect falls.
3. Email Overwriting : The `send_email` function currently overwrites the `to_email` and `subject` passed to it, which makes passing these parameters redundant. This might be an oversight and should be reviewed.
This is the overview of what the code does.

NOTICE TO NURSE:

Code:

```
app = Flask(__name__)
CORS(app)
THRESHOLD = 10
data_store = []# This is a placeholder; replace with actual threshold for detecting a
fall
def send_email(subject, body, to_email):
from_email = "kamarakesselly@outlook.com"
from_password = "0641305"
to_email = "kamarakesselly@gmail.com"
subject = "Fall Detect"
msg = MIMEText(body)
msg['From'] = from_email
msg['To'] = to_email
msg['Subject'] = subject
server = smtplib.SMTP('smtp.outlook.com', 587)
server.starttls()
server.login(from_email, from_password)
server.sendmail(from_email, to_email, msg.as_string())
server.quit()
@app.route('/endpoint', methods=['POST'])
def receive_data():
data = request.json
with open('data.json', 'a') as f:
json.dump(data, f)
f.write('\n')
if data.get('accelerometer_value', 0) > THRESHOLD or data.get('gyroscope_value',
0) > THRESHOLD:
email_subject = "Fall Detected!"
email_body = f"""
Hi CareTaker,
A fall was detected Please ,Patient Name: {data['patient_name']}
Patient ID: {data['patient_id']}
Location: {data['location_name']}
Accelerometer Value: {data['accelerometer_value']}
Gyroscope Value: {data['gyroscope_value']}
Timestamp: {data['timestamp']}
```

send_email(email_subject, email_body, "recipient_email@gmail.com")

return jsonify({"message": "Data received"}), 200

```
@app.route('/get_data', methods=['GET'])
def get_data():
with open('data.json', 'r') as f:
data_list = [json.loads(line) for line in f]
return jsonify(data_list)
@app.route('/web', methods=['GET'])
def web_page():
return render_template('index.html')
if __name__ == '__main__':
app.run(debug=True)
```

The email functionality in the provided code serves to send notifications when a certain condition (like detecting a fall) is met. Let's break down the email-related aspects of the code:

1. Dependencies:

import smtplib

from email.mime.text import MIMEText

smtplib: This is the Python library used for sending emails using the Simple Mail Transfer Protocol (SMTP). SMTP is the most commonly used protocol for sending emails.

MIMEText: This helps in creating MIME-formatted messages. MIME (Multipurpose Internet Mail Extensions) is a standard that extends the format of email to support text in character sets other than ASCII, as well as attachments.

2. send_email function:

def send_email(subject, body, to_email):

This function encapsulates the logic needed to send an email:

It starts by defining some hard coded variables like the sender's email (`from_email`), the sender's password (`from_password`), the recipient's email address (`to_email`), and a subject (`subject`). Note that the `to_email` and `subject` parameters passed to the function are immediately overwritten with hardcoded values. This could be an oversight.

A `MIMEText` object is created using the provided email body (`body`).

The email metadata (from, to, and subject) are set on the `MIMEText` object.

The code then sets up a connection to the Outlook SMTP server (`smtp.outlook.com`) on port 587. This is a common port for sending emails with TLS encryption.

starttls()` initiates the encrypted connection.

The code logs into the SMTP server using the provided email credentials.

It then sends the email and finally closes the connection with the SMTP server.

3. Usage of the send_email function:

In the `/endpoint` route, if the accelerometer or gyroscope value surpasses a certain threshold (`THRESHOLD`), indicating a potential fall, the `send_email` function is called to notify a recipient:

python

if data.get('accelerometer_value', 0) > THRESHOLD or data.get('gyroscope_value', 0) > THRESHOLD:

email_subject = "Fall Detected!"

email_body = f""

send_email(email_subject, email_body, "recipient_email@gmail.com")

Security and Best Practices:

Sensitive Data: The sender's email address and password are hardcoded directly in the code. This is a security risk, especially if the code is shared or stored in a public location. In a production environment, it's advisable to use environment variables, configuration files, or secret management tools to store such sensitive data.

- Error Handling: The current email sending mechanism lacks error handling. In real-world scenarios, sending emails can fail for various reasons. Implementing try-catch blocks and handling potential exceptions can provide a more resilient solution.

References:

Course Materials:
1.
https://hc.labnet.sfbu.edu/~henry/sfbu/course/capstone/android/slide/exercise_android.html
2.
https://hc.labnet.sfbu.edu/~henry/sfbu/course/capstone/android/slide/exercise_android.html#basic_geo
3.
https://hc.labnet.sfbu.edu/~henry/sfbu/course/capstone/android/slide/exercise_android.html#pp
4.
https://hc.labnet.sfbu.edu/~henry/sfbu/course/capstone/2022_spring/slide/topic_frame.html#sms

Online Resources:
1. Android Developers. (n.d.). Sensors Overview.
https://developer.android.com/guide/topics/sensors/sensors_overview

2. Google Developers. (n.d.). Location API Overview.
https://developers.google.com/android/reference/com/google/android/gms/location/Location

3. Vogel, L. (2021). Android Socket Programming with Examples. CodeLearn.
https://codelearn.org/android-tutorial/socket-programming-in-android

4. Volley. (n.d.). Volley: Easy, Fast Networking for Android.
https://developer.android.com/training/volley

5. Google. (n.d.). Best Practices for Performance. https://developer.android.com/training/best-performance

6. Kotlin. (n.d.). Get Started with Kotlin on Android. https://developer.android.com/kotlin/get-started

Books:
1. Phillips, B., Stewart, C., & Marsicano, K. (2020). Android Programming: The Big Nerd Ranch Guide. Big Nerd Ranch.

2. Obugyei, E. A., & Dunn, A. (2019). Learning Kotlin by Building Android Applications. Apress.

3. Griffiths, D., & Griffiths, D. (2017). Head First Android Development: A Brain-Friendly Guide. O'Reilly Media.