Native Crash

1. The Developers

- **Karthik Namboori - PES1UG21CS269**
- **Leharaditya Kumar - PES1UG21CS300**

2. Project Description

This project focuses on providing a comprehensive Crash Detection and Fallback Service for mobile applications. The service aims to detect, log, and analyze crashes in real-time, offering a fallback mechanism to enhance the user experience. The stored crash logs will be accessible through a user-friendly dashboard, enabling developers to investigate and address issues effectively.

3. Mini World

The system will involve the interaction between mobile applications, the crash detection and fallback service, and developers. The service will capture crash data, provide fallback solutions, and present detailed crash reports through the dashboard for analysis.

4. Functionality

4.1 Crash Detection and Logging

- Implement a robust crash detection mechanism for mobile applications, capturing relevant data such as stack traces, device information, and timestamps.
- Log crash data securely in the database for future analysis.

4.2 Fallback Mechanism

- Develop a fallback system to gracefully handle crashes and provide users with a seamless experience whenever possible.
- Support customizable fallback strategies based on the nature of the crash.

4.3 Dashboard

- Create an intuitive web dashboard for developers to access and analyze crash reports.
- Display detailed information on crash occurrences, including timestamps, affected devices, and stack traces.

4.4 Notification System

- Implement a notification system to alert developers in real-time when critical crashes occur.
- Allow developers to set preferences for receiving notifications based on severity levels.

4.5 Data Analysis

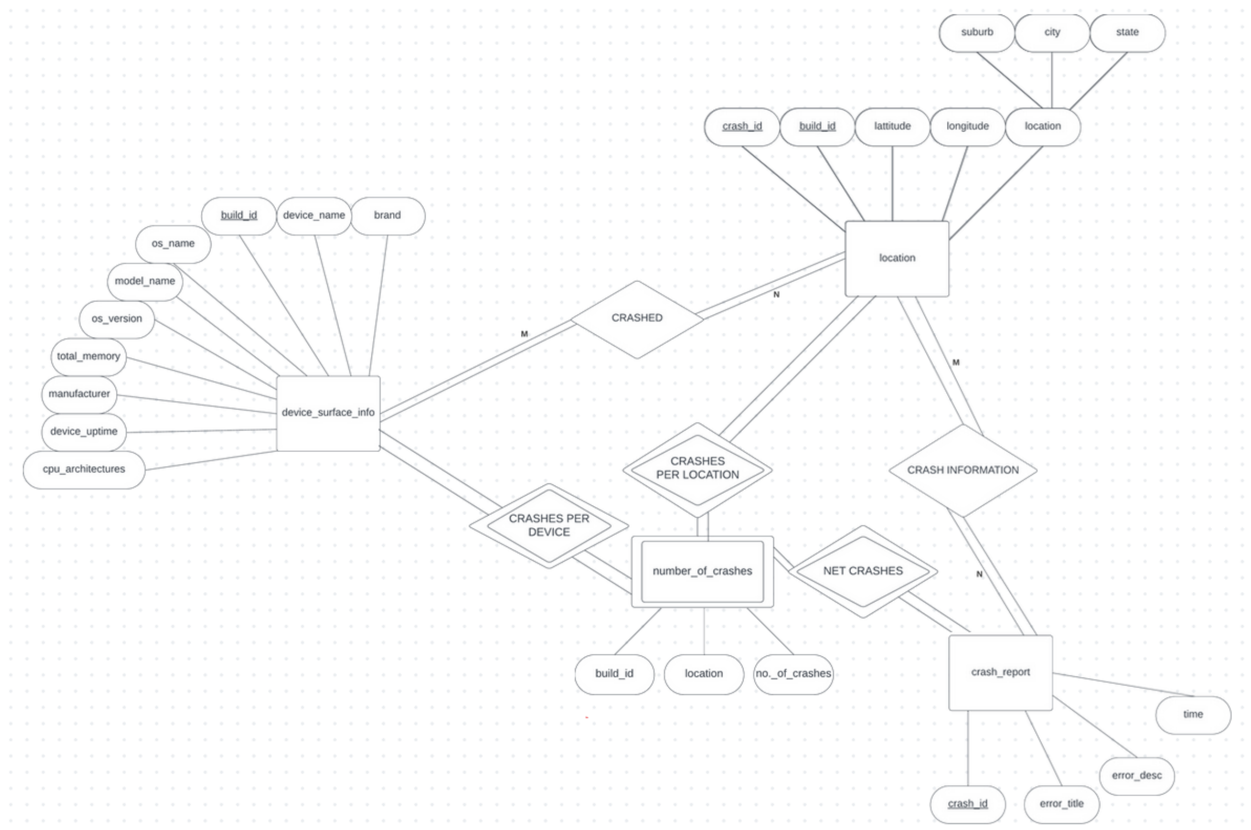
- Provide tools for in-depth analysis of crash data, including trend identification and common patterns.
- Support filtering and categorization of crash reports for efficient investigation.

5. User Interface

Develop a user-friendly interface for the dashboard, allowing developers to navigate and analyze crash reports seamlessly. Include features such as search, filtering, and graphical representations of crash data.

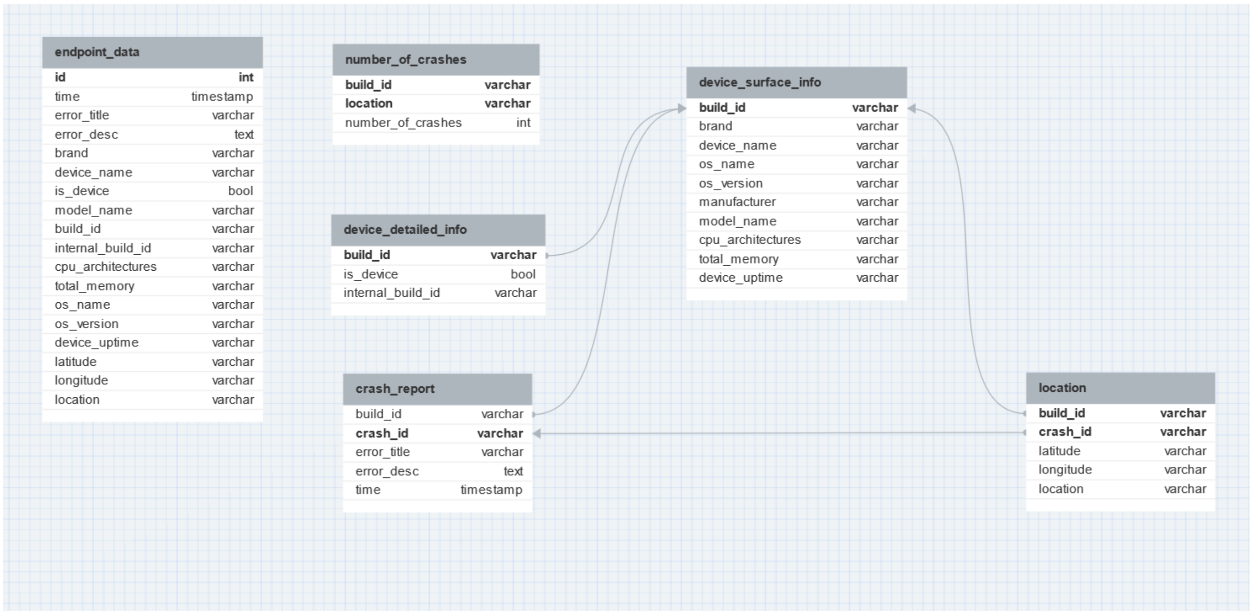
6. ER Diagram

The following is the stipulated ER diagram for this project.



7. Relationship Diagram

The following is the relationship diagram that we obtain from the said ER Diagram



endpoint_data	
id	**int**
time	timestamp
error_title	varchar
error_desc	text
brand	varchar
device_name	varchar
is_device	bool
model_name	varchar
build_id	varchar
internal_build_id	varchar
cpu_architectures	varchar
total_memory	varchar
os_name	varchar
os_version	varchar
device_uptime	varchar
latitude	varchar
longitude	varchar
location	varchar

number_of_crashes	
build_id	**varchar**
location	**varchar**
number_of_crashes	int

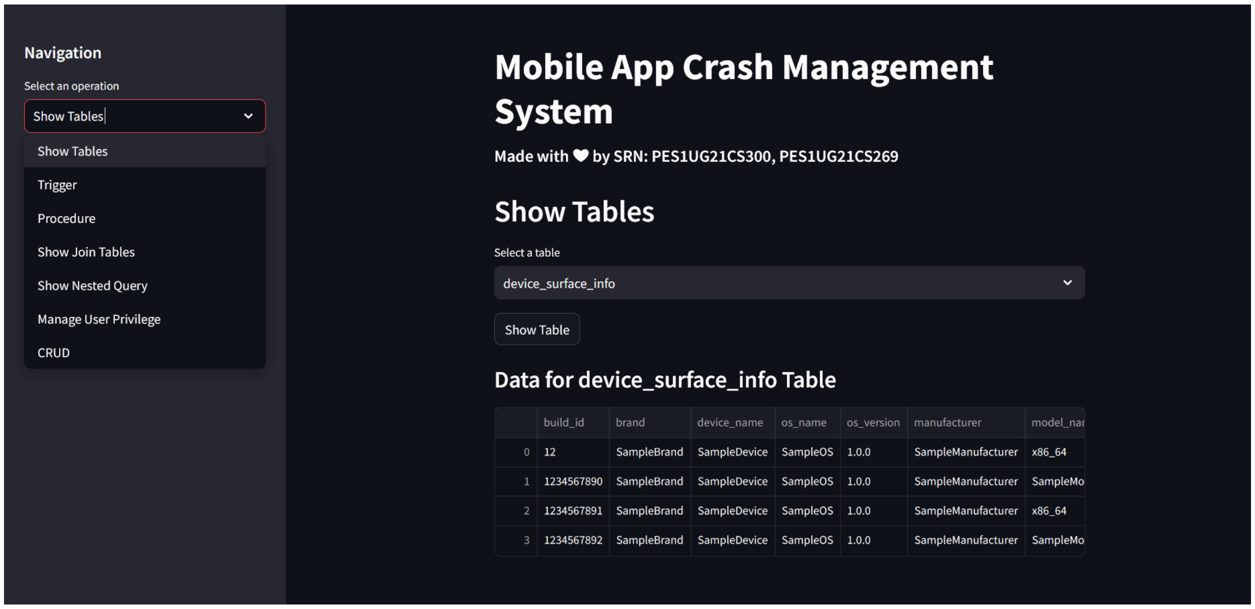
device_detailed_info	
build_id	**varchar**
is_device	bool
internal_build_id	varchar

crash_report	
build_id	varchar
crash_id	**varchar**
error_title	varchar
error_desc	text
time	timestamp

device_surface_info	
build_id	**varchar**
brand	varchar
device_name	varchar
os_name	varchar
os_version	varchar
manufacturer	varchar
model_name	varchar
cpu_architectures	varchar
total_memory	varchar
device_uptime	varchar

location	
build_id	**varchar**
crash_id	**varchar**
latitude	varchar
longitude	varchar
location	varchar

Frontend screen shots



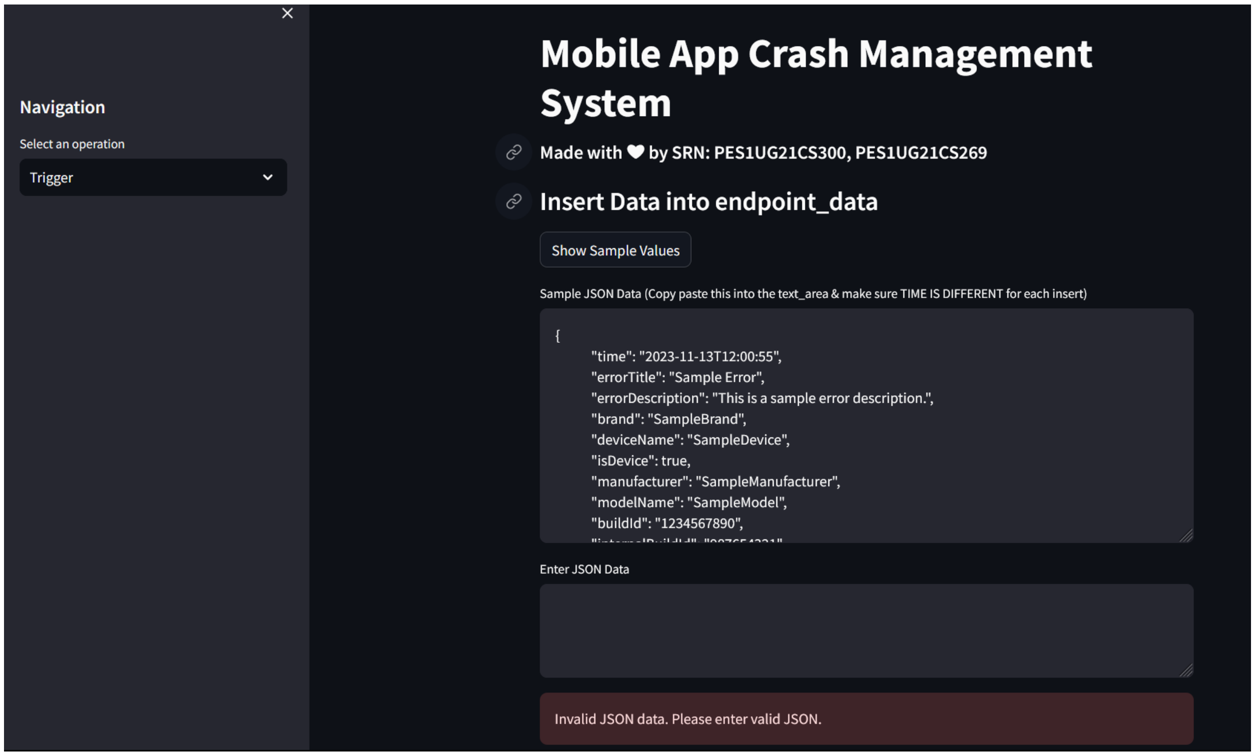
Mobile App Crash Management System

Made with ❤ by SRN: PES1UG21CS300, PES1UG21CS269

Show Tables

Data for device_surface_info Table

	build_id	brand	device_name	os_name	os_version	manufacturer	model_nar
0	12	SampleBrand	SampleDevice	SampleOS	1.0.0	SampleManufacturer	x86_64
1	1234567890	SampleBrand	SampleDevice	SampleOS	1.0.0	SampleManufacturer	SampleMo
2	1234567891	SampleBrand	SampleDevice	SampleOS	1.0.0	SampleManufacturer	x86_64
3	1234567892	SampleBrand	SampleDevice	SampleOS	1.0.0	SampleManufacturer	SampleMo



Mobile App Crash Management System

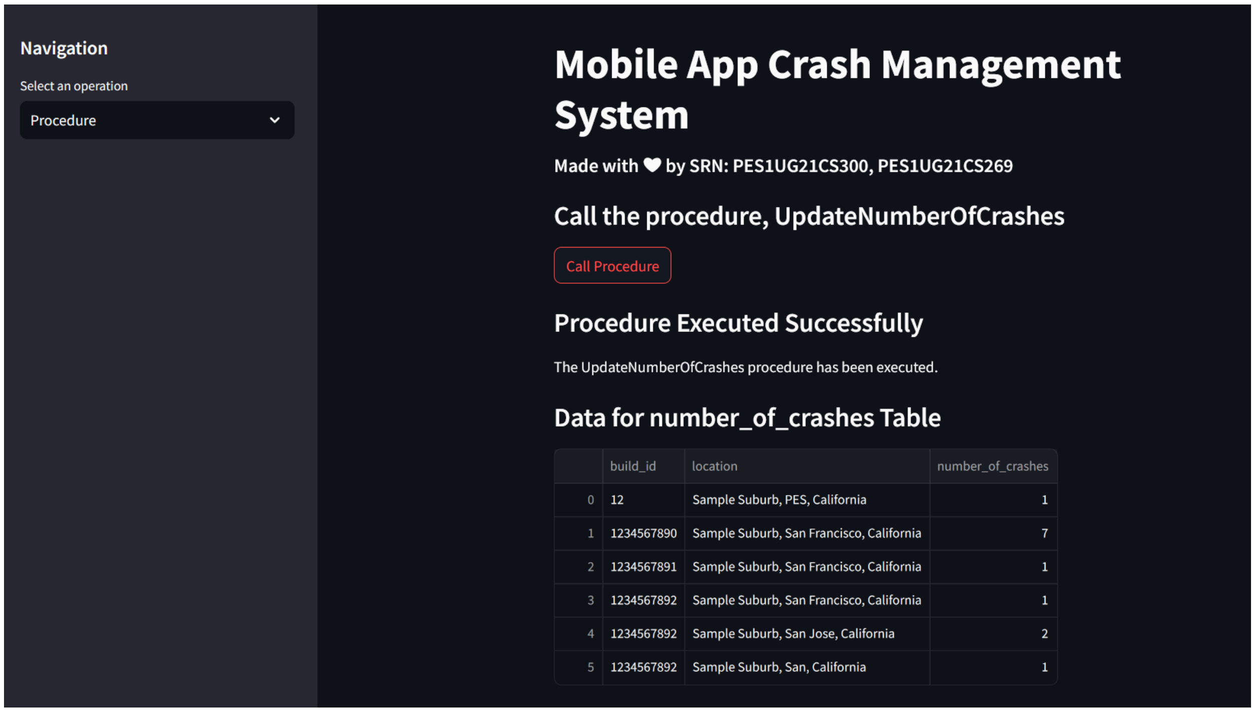
Made with ❤ by SRN: PES1UG21CS300, PES1UG21CS269

Insert Data into endpoint_data

Sample JSON Data (Copy paste this into the text_area & make sure TIME IS DIFFERENT for each insert)

```
{
        "time": "2023-11-13T12:00:55",
        "errorTitle": "Sample Error",
        "errorDescription": "This is a sample error description.",
        "brand": "SampleBrand",
        "deviceName": "SampleDevice",
        "isDevice": true,
        "manufacturer": "SampleManufacturer",
        "modelName": "SampleModel",
        "buildId": "1234567890",
```

Invalid JSON data. Please enter valid JSON.



Mobile App Crash Management System

Made with ❤ by SRN: PES1UG21CS300, PES1UG21CS269

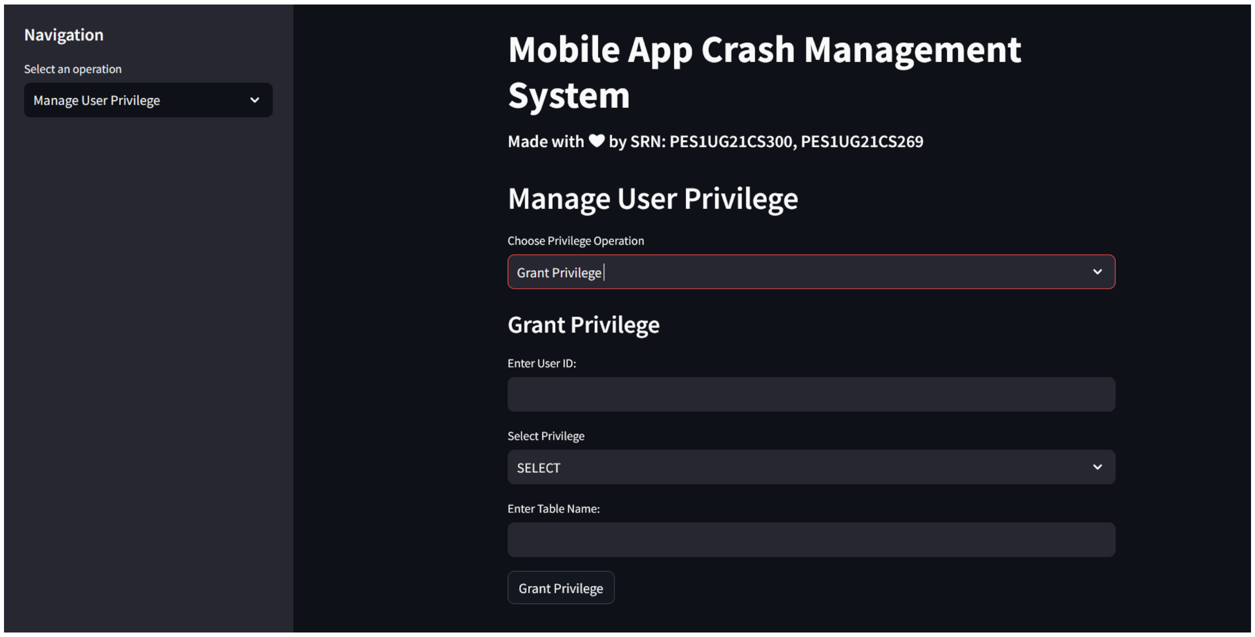
Call the procedure, UpdateNumberOfCrashes

Procedure Executed Successfully

The UpdateNumberOfCrashes procedure has been executed.

Data for number_of_crashes Table

	build_id	location	number_of_crashes
0	12	Sample Suburb, PES, California	1
1	1234567890	Sample Suburb, San Francisco, California	7
2	1234567891	Sample Suburb, San Francisco, California	1
3	1234567892	Sample Suburb, San Francisco, California	1
4	1234567892	Sample Suburb, San Jose, California	2
5	1234567892	Sample Suburb, San, California	1



Mobile App Crash Management System

Made with ❤ by SRN: PES1UG21CS300, PES1UG21CS269

Manage User Privilege

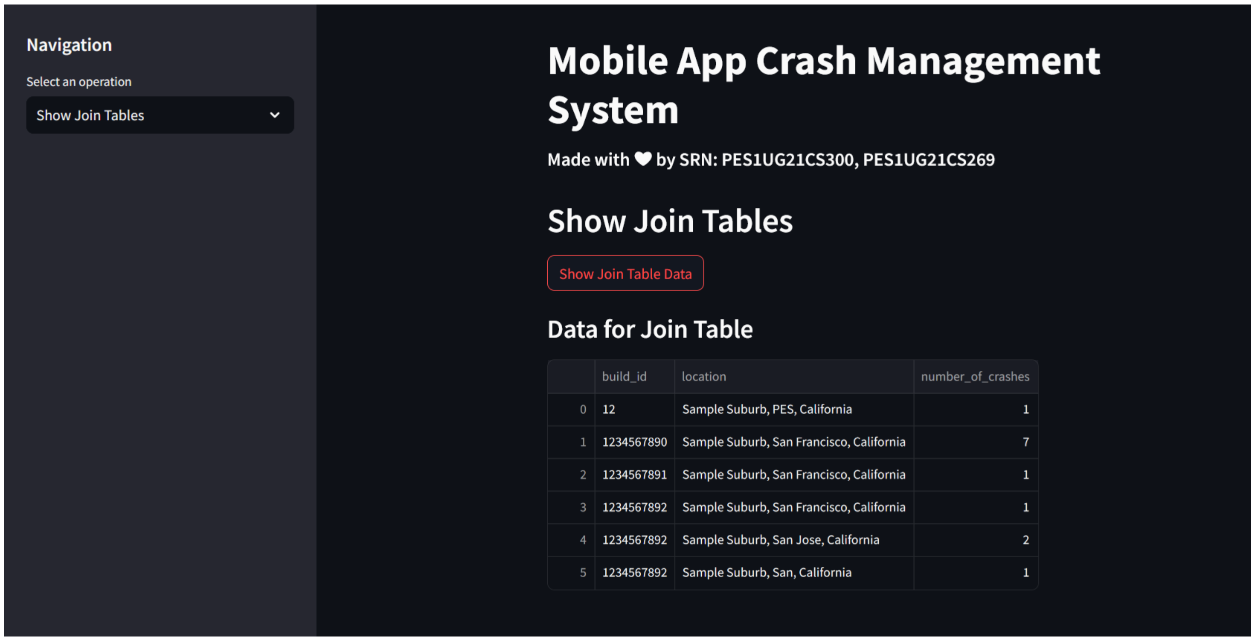
Choose Privilege Operation: Grant Privilege

Grant Privilege

Enter User ID:

Select Privilege: SELECT

Enter Table Name:



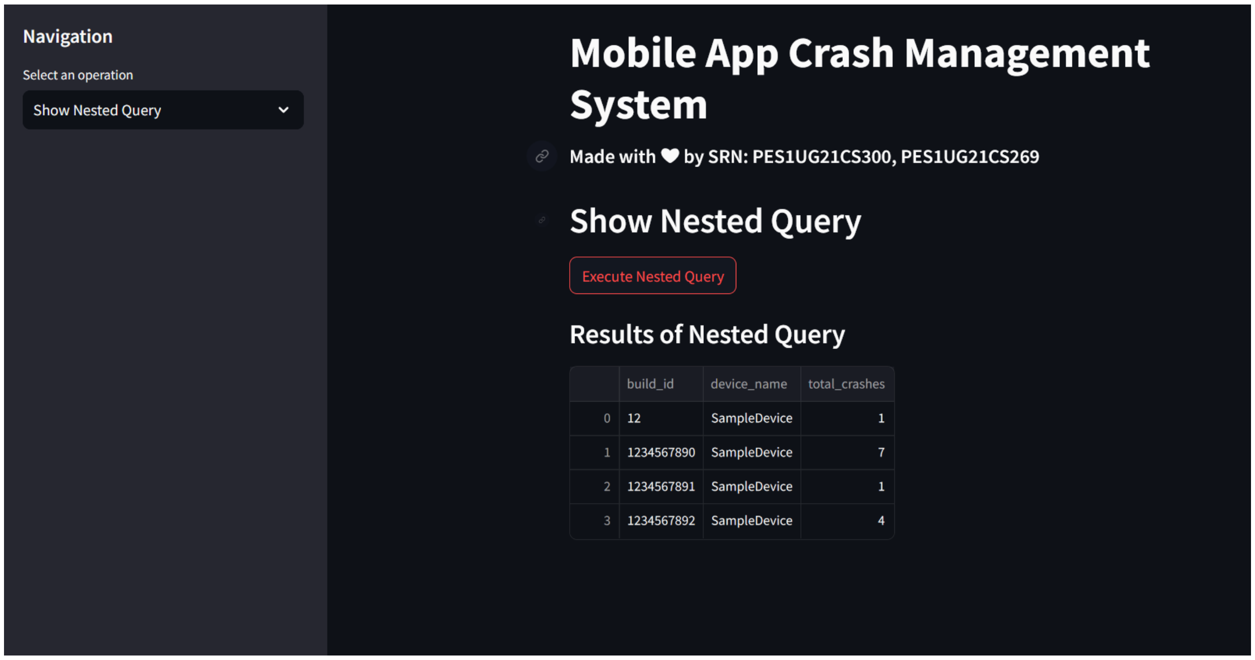
Mobile App Crash Management System

Made with ❤ by SRN: PES1UG21CS300, PES1UG21CS269

Show Join Tables

Data for Join Table

	build_id	location	number_of_crashes
0	12	Sample Suburb, PES, California	1
1	1234567890	Sample Suburb, San Francisco, California	7
2	1234567891	Sample Suburb, San Francisco, California	1
3	1234567892	Sample Suburb, San Francisco, California	1
4	1234567892	Sample Suburb, San Jose, California	2
5	1234567892	Sample Suburb, San, California	1



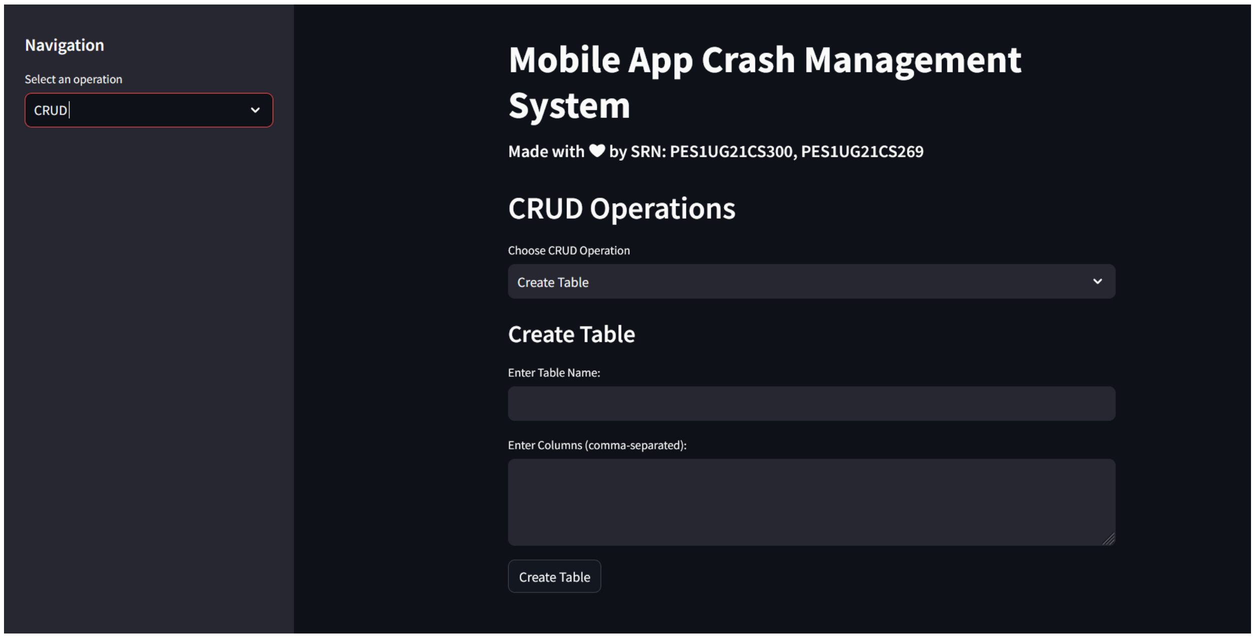
Mobile App Crash Management System

Made with ❤ by SRN: PES1UG21CS300, PES1UG21CS269

Show Nested Query

Results of Nested Query

	build_id	device_name	total_crashes
0	12	SampleDevice	1
1	1234567890	SampleDevice	7
2	1234567891	SampleDevice	1
3	1234567892	SampleDevice	4



Mobile App Crash Management System

Made with ❤ by SRN: PES1UG21CS300, PES1UG21CS269

CRUD Operations

Choose CRUD Operation: Create Table

Create Table

Enter Table Name:

Enter Columns (comma-separated):

Code Snippets

```
// Create trigger to handle procedure calls after endpoint_data insertion
const triggerQuery = `
CREATE TRIGGER IF NOT EXISTS InsertYourEndpointData
AFTER INSERT ON endpoint_data
FOR EACH ROW
BEGIN
CALL InsertDeviceSurfaceInfo(
      NEW.build_id, NEW.brand, NEW.device_name, NEW.os_name, NEW.os_version,
      NEW.manufacturer, NEW.cpu_architectures, NEW.model_name,
      NEW.total_memory, NEW.device_uptime
      );

      CALL InsertLocationInfo(
        NEW.id, NEW.build_id, NEW.latitude, NEW.longitude, NEW.location
        );

          CALL InsertDetailedInfo(
      NEW.build_id, NEW.is_device, NEW.internal_build_id
      );

      CALL InsertCrashReport(
        NEW.id, NEW.error_title, NEW.error_description, NEW.time
      );

      CALL UpdateNumberOfCrashes();

      END;
      `;

  con.query(triggerQuery, (err, results) => {
    if (err) throw err;
    console.log("Trigger InsertYourEndpointData created successfully");
  });
});
```

```
// Create procedure to insert or update device_surface_info
  con.query(
    `CREATE PROCEDURE IF NOT EXISTS InsertDeviceSurfaceInfo(
      IN p_build_id VARCHAR(255),
      IN p_brand VARCHAR(255),
      IN p_device_name VARCHAR(255),
      IN p_os_name VARCHAR(255),
      IN p_os_version VARCHAR(255),
      IN p_manufacturer VARCHAR(255),
      IN p_cpu_architectures VARCHAR(255),
      IN p_model_name VARCHAR(255),
      IN p_total_memory VARCHAR(255),
      IN p_device_uptime VARCHAR(255)
    )
    BEGIN
      INSERT INTO device_surface_info (
        build_id, brand, device_name, os_name,
        os_version, manufacturer, model_name,
        cpu_architectures, total_memory, device_uptime
        )
        VALUES (
          p_build_id, p_brand, p_device_name, p_os_name,
          p_os_version, p_manufacturer, p_cpu_architectures,
          p_model_name, p_total_memory, p_device_uptime
          )
          ON DUPLICATE KEY UPDATE
          brand = p_brand,
          device_name = p_device_name,
          os_name = p_os_name,
          os_version = p_os_version,
          manufacturer = p_manufacturer,
          model_name = p_model_name,
          cpu_architectures = p_cpu_architectures,
          total_memory = p_total_memory,
          device_uptime = p_device_uptime;
      END;`,
    (err, results) => {
      if (err) throw err;
      console.log("Procedure InsertDeviceSurfaceInfo created successfully");
    }
  );

  // Create procedure to insert crash location information
  con.query(
    `CREATE PROCEDURE IF NOT EXISTS InsertLocationInfo(
    IN p_crash_id INT,
    IN p_build_id VARCHAR(255),
    IN p_latitude VARCHAR(20),
    IN p_longitude VARCHAR(20),
    IN p_location VARCHAR(255)
  )
  BEGIN
    INSERT INTO location (
      crash_id, build_id, latitude, longitude, location
    )
    VALUES (
      p_crash_id, p_build_id, p_latitude, p_longitude, p_location
    );
    END;`,
    (err, results) => {
      if (err) throw err;
      console.log("Procedure InsertLocationInfo created successfully");
    }
  );

  // Create procedure to insert or update device_detailed_info
  con.query(
    `CREATE PROCEDURE IF NOT EXISTS InsertDetailedInfo(
    IN p_build_id VARCHAR(255),
    IN p_is_device bool,
    IN p_internal_build_id VARCHAR(255)
    )
    BEGIN
    INSERT INTO device_detailed_info (
      build_id, is_device, internal_build_id
      )
    VALUES (
      p_build_id, p_is_device, p_internal_build_id
      )
      ON DUPLICATE KEY UPDATE
      is_device = p_is_device,
      internal_build_id = p_internal_build_id;
      END;`,
    (err, results) => {
      if (err) throw err;
      console.log("Procedure InsertDetailedInfo created successfully");
    }
  );

  // Create procedure to insert crash report information
  con.query(
    `CREATE PROCEDURE IF NOT EXISTS InsertCrashReport(
    IN p_crash_id INT,
    IN p_error_title TEXT,
    IN p_error_description TEXT,
    IN p_time TIMESTAMP
    )
    BEGIN
    INSERT INTO crash_info (
      crash_id, error_title, error_description, time
    )
    VALUES (
      p_crash_id, p_error_title, p_error_description, p_time
      );
      END;`,
    (err, results) => {
      if (err) throw err;
      console.log("Procedure InsertCrashReport created successfully");
    }
  );
// Create procedure to count the number_of_crashes table using location and
con.query(
  `CREATE PROCEDURE IF NOT EXISTS UpdateNumberOfCrashes()
  BEGIN
    -- Delete existing data in number_of_crashes table
    DELETE FROM number_of_crashes;

    -- Insert new counts into the number_of_crashes table
    INSERT INTO number_of_crashes (build_id, location, number_of_crashes)
    SELECT
    l.build_id,
    l.location,
    COUNT(DISTINCT ci.crash_id) AS number_of_crashes
    FROM
        location l
    JOIN
        device_surface_info ds ON l.build_id = ds.build_id
    LEFT JOIN
        crash_info ci ON l.crash_id = ci.crash_id
    GROUP BY
        l.build_id, l.location;

  END;`,
  (err, results) => {
    if (err) throw err;
    console.log("Procedure UpdateNumberOfCrashes created successfully");
  }
);
```

```
# Function to execute a nested query and retrieve the result
def execute_nested_query():
    query = """
    SELECT
        ds.build_id,
        ds.device_name,
        (SELECT COUNT(DISTINCT l.crash_id) FROM location l WHERE l.build_id = ds.build_id) AS total_crashes
    FROM
        device_surface_info ds;
    """
    data = pd.read_sql(query, conn)
    return data
```

```
# Function to execute a join query and retrieve the result
def execute_join_query():
    query="""
    SELECT
    l.build_id,
    l.location,
    COUNT(DISTINCT ci.crash_id) AS number_of_crashes
    FROM
        location l
    JOIN
        device_surface_info ds ON l.build_id = ds.build_id
    LEFT JOIN
        crash_info ci ON l.crash_id = ci.crash_id
    GROUP BY
        l.build_id, l.location;
    """
    data = pd.read_sql(query, conn)
    return data
```

```
# Function to create a role with the given name
def create_role(role_name):
    cursor = conn.cursor()

    # Build the CREATE ROLE query
    create_role_query = f"CREATE ROLE {role_name}"

    # Execute the query
    cursor.execute(create_role_query)
    conn.commit()
    cursor.close()

# Function to grant a privilege to a user on a table
def grant_privilege(user_id, privilege, table_name):
    cursor = conn.cursor()

    # Build the GRANT query
    grant_query = f"GRANT {privilege} ON {table_name} TO {user_id}"

    # Execute the query
    cursor.execute(grant_query)
    conn.commit()
    cursor.close()

# Function to revoke a privilege from a user on a table
def revoke_privilege(user_id, privilege, table_name):
    cursor = conn.cursor()

    # Build the REVOKE query
    revoke_query = f"REVOKE {privilege} ON {table_name} FROM {user_id}"

    # Execute the query
    cursor.execute(revoke_query)
    conn.commit()
    cursor.close()
```