Introduction

DHAI Mobile App, a cutting-edge and versatile solution meticulously designed to significantly enhance the quality of life for residents of DHAI (Defence Housing Authority Islamabad). This mobile application serves as a unified platform, seamlessly connecting residents with vital services, security measures, and community engagement resources. This comprehensive documentation is crafted to provide users with a profound understanding of the app's myriad features, enabling them to harness its full potential and enrich their daily living experience.

1.1 Purpose

The DHAI Mobile App is an all-in-one solution designed to enhance the living experience of DHAI residents. The DHAI Mobile App serves as a multifaceted tool, seamlessly integrating a range of services and functionalities to cater to the diverse needs of DHAI residents. It aims to simplify daily life, enhance security, streamline complaint management, foster better community engagement, and offer a plethora of amenities and services designed to make living in DHAI even more delightful. The primary purpose is to provide a comprehensive overview of the DHAI Mobile App, its functionalities, development practices, and integration with various services. This documentation is intended for developers, maintainers, and stakeholders involved in the app's development and deployment. This elaborates the primary objectives and goals that drive the development and existence of the app to provide clarity on why the app was created and what it intends to achieve. Main objectives includes:

Enhanced Security: The DHAI Mobile App aims to enhance the security of DHAI residents by providing quick and efficient access to security services. This is achieved through features such as the Panic Alert Dashboard and Fire Alert Dashboard, which enable residents to rapidly call for help in case of emergencies, including incidents like dacoity, theft, or fire.

Efficient Grievance Redressal: An essential objective of the app is to streamline the process of grievance redressal. The Complaint Dashboard and Medical Dashboard serve as dedicated channels for residents to report complaints and medical emergencies. These features ensure that issues are promptly communicated to the relevant DHAI offices for resolution.

Community Engagement: The App fosters community engagement by keeping residents informed about community news, promotions, and updates. The News & Promotions section serves as a dynamic information hub, strengthening the sense of belonging and involvement within DHAI community.

Valued Resident Services: The app goes above and beyond by offering extraordinary facilities such as the Ambulance Facility. These services cater to the diverse needs of residents, providing convenience and peace of mind.

1.2 Scope

This documentation offers a comprehensive exploration of the DHAI Mobile App, user registration, login procedures, and a detailed breakdown of its core features including the Panic Alert Dashboard, Fire Alert Dashboard, Complaint Dashboard, Medical

Dashboard, News & Promotions, Ambulance Facility and other extraordinary facilities. It provides step-by-step instructions, best practices, and tips to ensure users can effortlessly navigate the app and fully capitalize on its diverse range of offerings.

1.3 Audience

This documentation is intended for both DHAI residents seeking to utilize the app and DHAI Islamabad administrative staff tasked with overseeing its management and operations. Whether you're a resident keen on harnessing the app's capabilities or an administrator responsible for its effective functioning, this documentation is your go-to resource.

2. Development Environment Setup

2.1 IDE Installation

IDE stands for Integrated Development Environment, which is a software application that provides a comprehensive development environment for building, testing, and deploying software applications. For developing the DHAI Mobile App, you would typically follow these steps:

2.1.1 Choose an IDE:

Before diving into development, developers need to select an Integrated Development Environment (IDE) that is well-suited for mobile app development. The choice of IDE significantly impacts productivity and coding efficiency. Common choices include Android Studio for Android app development and Xcode for iOS app development.

2.1.2 Install the IDE:

Once an IDE is selected, the next step is to install it on the development machine. Installation processes may vary depending on the chosen IDE and the operating system (e.g., Windows, macOS, Linux).

2.1.3 Configure IDE:

After installation, it's crucial to configure the IDE to align with the specific requirements of the DHAI Mobile App development environment. Configuration settings may include SDK and emulator setup, version control integration, coding style preferences and plug-in / extensions for app development.

2.2 SDK and Tools

SDK stands for Software Development Kit, which includes essential software development kits (SDKs), development tools, and third-party libraries that developers need to have in their development environment to effectively build and maintain the app.
Here's how it applies to the DHAI Mobile App:

2.2.1 Mobile Platform SDK:

The Mobile Platform SDK includes the necessary software development kits for the mobile platforms on which the DHAI Mobile App will be deployed. This typically involves SDKs for Android and iOS. Depending on whether you are developing for Android or iOS, you need to install the respective SDKs (Android SDK for Android development and iOS SDK for iOS development).

2.2.2 Development Tools:

Development tools are essential for coding, debugging, and testing the App. These tools enhance the development process and improve code quality. Install tools like Android Virtual Device (AVD) Manager for Android or Xcode for iOS to simulate and test your app on virtual devices.

2.2.3 Third-Party Libraries:

Third-party libraries are used to add functionality and efficiency to the App. These libraries provide pre-built solutions for common tasks and save development time. Identify and integrate any third-party libraries or frameworks required for specific functionalities within your app.

3. Project Structure

The Project Structure outlines how the app's source code and resources are organized. A well-structured project ensures clarity, maintainability, and efficient collaboration among developers. Understanding the file hierarchy and module organization is vital for navigating and contributing to the app's development.

3.1 File Hierarchy

The file hierarchy refers to the arrangement of files and directories within app's project. A well-structured file hierarchy can make development, maintenance, and collaboration more efficient. Organizing files logically and consistently can help make development more manageable.

3.2 Module Organization

Module organization refers to the division of app into distinct functional modules or components. This modular approach simplifies development, testing, and maintenance.

Key Modules:

Authentication:
Manages user authentication and authorization, including login and registration.

Security:
Handles security-related features like panic alerts, fire alerts, and emergency notifications.

Complaints:
Manages the complaints and grievances system, allowing users to submit and track complaints.

Medical:
Supports medical emergency services, including requesting medical assistance and ambulance facilities.

Community:
Focuses on community engagement, providing news, promotions, and event updates.

4. Programming Languages

The choice of programming languages for the DHAI Mobile App development largely depends on the targeted platforms (Android and iOS), as well as any cross-platform frameworks or technologies you may consider. Here are the primary programming languages associated with Android, iOS, and cross-platform development:

4.1 Flutter

Flutter is an open-source, UI software development toolkit created by Google. It enables developers to build natively compiled applications for mobile, web, and desktop from a single codebase. Flutter's key feature is its expressive and customizable widget-based framework, which allows for fast and visually appealing cross-platform app development.

4.2 Dart

Dart is a programming language developed by Google, designed for building web, mobile, and desktop applications. It's known for its speed, simplicity, and strong support for object-oriented programming. Dart is frequently used in conjunction with the Flutter framework to create cross-platform mobile apps with a single codebase.

5. Architecture

In the context of the DHAI Mobile App, the architecture refers to the high-level structure and organization of the app's code and components. It outlines the design patterns, component interactions, and overall architectural principles guiding the development of the app working together to achieve its functionality and maintainability.

5.1 Design Patterns (MVVM)

MVVM (Model-View-ViewModel) is a design pattern that separates an application into three interconnected components, this model separates the user interface (View) from the business logic (ViewModel) and the data (Model). It promotes separation of concerns, making the app easier to maintain and test. It also supports data binding, which allows automatic updates of the UI when underlying data changes. In DHAI Mobile App:

Model: Represents the data and business logic responsible for data retrieval, manipulation, and storage. It includes data models (user data, complaint data) and services (authentication service, complaint service) that interact with the data.

View: Represents the user interface components and screens. It includes the visual elements users interact with, such as buttons, forms, and screens like the Panic Alert, Fire Alert, Medical or Complaints Dashboards.

ViewModel: Acts as an intermediary between the Model and View. It contains the presentation logic, handling user input, data retrieval, and formatting before passing it to the View. For instance, when a user submits a complaint, the ViewModel processes it and updates the View accordingly.

5.2 Component Interaction (Widgets, Activities/Fragments, Views)

Component interaction in the DHAI Mobile App refers to how different elements of the app's user interface and functionality interact with each other:

Widgets: Widgets are UI components that provide specific functionality or display specific information. In this App, widgets are used to create interactive and informative elements within the app's user interface. Examples include buttons, text input fields, lists, and interactive elements like panic buttons and alert notifications allowing users to trigger security alerts directly from their home screen.

Activities/Fragments: In Android, activities and fragments are used to create screens and handle user interactions. Activities represent individual screens or windows within the app, while Fragments are reusable UI components that can be combined to create more complex user interfaces.

Views: Views are the user interface elements users interact with, such as buttons, text fields, and lists. In the DHAI Mobile App, views are used extensively to create the user interface components of each screen, making the app visually appealing and user-friendly.

These components work together to provide a seamless user experience. For example, a user may interact with a button (View) on the Panic Alert Dashboard (Activity/Fragment), which triggers an action handled by the ViewModel (MVVM) to initiate a security alert.

6. Database Design

The database design for the DHAI Mobile App involves the structure and organization of data storage using PostgreSQL. This design ensures efficient data management, retrieval, and integrity for various app functionalities, such as user accounts, complaints, and security incidents.

6.1 Database Type (PostgreSQL)

PostgreSQL is chosen as the database type for its robust features, reliability, and scalability. It offers support for complex data structures, transactions, and data integrity constraints, making it suitable for managing the app's data. It handles data storage and management for the app's critical features, including the Panic Alert Dashboard, Fire Alert Dashboard, Complaint Dashboard, Medical Dashboard, News & Promotions, Ambulance Facility, and extraordinary amenities for valued residents. Key Aspects of the Database Design:

Tables: The database consists of multiple tables, each responsible for storing specific types of data. Common tables in the DHAI Mobile App database include:

Users: Stores user account information, such as usernames, passwords, and contact details.

Complaints: Records user-generated complaints, including details, timestamps, and status.

Security-Incidents: Stores data related to security incidents, enabling efficient tracking and reporting.

News and Promotions: Contains news articles, promotions, and community updates.

Relationships: Tables are interconnected through relationships (e.g. foreign keys) to maintain data consistency and integrity. For example, complaints may be associated with a specific user account through a foreign key relationship.

Indexes: Indexes are created on frequently queried columns to optimize database query performance, ensuring that data retrieval is fast and efficient.

Views: Database views can be used to simplify complex queries or provide a consolidated view of data across multiple tables. For instance, a view might combine data from the Complaints and Security-Incidents tables to provide a comprehensive incident history.

Security: Access control and security measures, such as authentication and authorization, are implemented to safeguard sensitive user data.

Scalability: The database design allows for future scalability, accommodating an increasing volume of data and users as the app grows.

Backup and Recovery: Regular database backups are scheduled to ensure data resilience and facilitate recovery in case of data loss or system failures.

Data Migration: Procedures for data migration and updates are in place to manage changes to the database schema as the app evolves.

The PostgreSQL database, combined with a well-thought-out database design, plays a pivotal role in ensuring data reliability, consistency, and accessibility for DHAI Mobile App. It forms the foundation upon which the app's features and functionalities are built, offering a secure and efficient data management solution.

7. Data Storage and Management

Data storage and management in the DHAI Mobile App involve how the app elaborates on how data is organized, stored, and managed within the application. This includes data models, data retrieval, manipulation, and storage mechanisms for the app's key features. This includes the selection of a suitable database system, the establishment of data storage protocols, and the implementation of data security measures.

7.1 Data Models

Data models in the DHAI Mobile App define the structure and representation of various data entities within the system. These models encapsulate the properties, attributes, and relationships of data objects.

7.1.1 Panic Alert Dashboard Data Model

Contains records of panic alerts with attributes like timestamps, user IDs, and GPS coordinates. Stores user profile information, including emergency contact details.

7.1.2 Fire Alert Dashboard Data Model

Represents fire incidents with data points such as incident timestamps, locations, and descriptions. Contains contact information for relevant authorities or emergency services.

7.1.3 Complaint Dashboard Data Model

Manages user-submitted complaints, recording details such as descriptions, timestamps, and complaint statuses. Stores user profile data, including contact information.

7.1.4 Medical Dashboard Data Model

Records data related to medical assistance requests, including timestamps, user IDs, and request locations. Contains information about medical facilities, including contact details and services offered.

7.1.5 News & Promotions Data Model

Manages news articles, promotions, and community updates with attributes like content, publication dates, and categories. Stores user preferences for news and promotions.

7.1.6 Ambulance Facility Data Model

Records requests for ambulance services, including timestamps, user IDs, and locations. Contains information about ambulance service providers, including contact details and service coverage.

7.2 Data Retrieval and Manipulation

Data retrieval and manipulation involve the processes and methods by which the app interacts with and utilizes data. This includes querying data from the database, performing updates, and presenting data to users.

CRUD Operations: The app performs CRUD (Create, Read, Update, Delete) operations to interact with data. For example, users can create complaints, read news articles, update their profiles, and delete security incidents.

Querying: Structured queries are used to retrieve specific data from the database. For instance, users can search for their submitted complaints, filter security incidents by location, or view news articles by category.

Data Processing: Data retrieved from the database is processed as needed before being presented to users. This may include formatting, sorting, or aggregating data.

Data Validation: Input data is validated to ensure it meets required constraints and is free from errors. For instance, user registration inputs are validated to prevent invalid entries.

Real-time Updates: Some data, such as news and promotions, may be updated in real-time, providing users with the latest information.

Data Presentation: Data is presented to users through the app's user interface, allowing them to interact with and make informed decisions based on the presented information.

8 Network Communication

Network Communication in the context of DHAI Mobile App refers to the processes and protocols involved in exchanging data and information between the app and external servers or services over a network, typically the internet. Effective network communication is crucial for features like real-time updates, data synchronization, and accessing remote resources.

Protocol Selection: The app utilizes industry-standard communication protocols such as HTTP/HTTPS for secure data exchange.

API Endpoints: Defines and utilizes specific API endpoints for various functionalities, allowing the app to send requests and receive responses from remote servers.

Data Serialization: Data is serialized into the appropriate format (e.g. JSON or XML) before sending requests and deserialized upon receiving responses.

Secure Communication: Secure Socket Layer (SSL) or Transport Layer Security (TLS) is implemented to encrypt data transmitted between the app and servers, ensuring data privacy and security.

Authentication: Authentication mechanisms (e.g. API keys, tokens) are employed to verify the identity of the app and users when interacting with external services.

Error Handling: Robust error handling mechanisms are in place to manage network-related errors gracefully, providing informative error messages to users when necessary.

Offline Mode: The app may include features to handle network unavailability, such as caching data locally and syncing when the network becomes available.

Network Monitoring: Tools and libraries may be used to monitor network requests and responses, enabling performance analysis and issue detection.

8.1 RESTful API Integration

RESTful API Integration involves connecting DHAI Mobile App with external services or APIs that follow the principles of Representational State Transfer (REST). RESTful APIs are designed to be simple, scalable, and stateless, making them ideal for web and mobile app integration.

Endpoint Discovery: Developers identify and document the RESTful API endpoints provided by external services, understanding their functionality and data exchange format.

HTTP Methods: The app uses standard HTTP methods (e.g., GET, POST, PUT, DELETE) to interact with RESTful API endpoints, aligning requests with the intended actions (e.g., retrieving data, submitting data).

Request Construction: Requests are constructed with the required headers, parameters, and payload data following the API documentation.

Response Handling: Responses from the RESTful API are parsed and processed to extract relevant data and handle errors gracefully.

Authentication: Authentication tokens or credentials are included in requests to ensure authorized access to protected API endpoints.

Pagination: When dealing with large data sets, pagination techniques are implemented to efficiently retrieve and display data.

Rate Limiting: If RESTful API imposes rate limits, the app respects these limits to avoid overloading the external service.

Error Codes: Error codes and status messages from the API are translated into user-friendly messages and actions within the app.

Testing and Debugging: Extensive testing and debugging are conducted to ensure the correct integration of RESTful APIs, including handling edge cases and unexpected responses.

Versioning: The app accommodates API versioning to ensure compatibility with evolving API specifications.

9. Error Handling and Logging

Error handling and logging are critical components of software development that help identify, report, and manage errors, exceptions, and unexpected behaviors within the DHAI Mobile App. Proper error handling and logging enhance app stability and provide insights into issues for debugging and improvement.

Error Reporting: The app is equipped with mechanisms to detect errors and exceptions, including runtime errors and unexpected user interactions.

Error Logging: When an error occurs, relevant information about the error, including its type, location, and context, is logged to a designated log file or system. This log provides a historical record of errors for analysis.

Logging Levels: Logging is categorized into different levels (e.g. debug, info, warning, error) to distinguish the severity of issues. Each log level serves a specific purpose, from routine debugging to critical error reporting.

Exception Handling: Exceptions, which are unexpected or exceptional events in the app's execution, are caught and managed to prevent application crashes or data corruption.

Error Messages: User-friendly error messages are displayed to users when errors occur, helping them understand the issue and take appropriate actions. Error messages are informative and not overly technical.

Monitoring Tools: Tools and services may be integrated into the app to monitor error patterns, performance bottlenecks, and user-reported issues in real-time.

Security Considerations: Error handling and logging are designed with security in mind to prevent sensitive information from being exposed in error messages or logs.

Continuous Improvement: Error logs are regularly reviewed and analyzed to identify recurring issues and areas for app improvement.

9.1 Error Handling Strategies:

HTTP Status Codes: The app interprets HTTP status codes to determine the outcome of API requests and respond accordingly.

Error Messages: When errors occur, informative error messages are provided to users to help them understand the issue and take appropriate actions.

Retry Mechanism: In cases of temporary network issues, the app may include a retry mechanism to reattempt failed requests.

9.2 Logging Features:

Log Levels: Different log levels are used to categorize log entries based on their severity, allowing developers to filter and focus on specific issues.

Timestamps: Log entries include timestamps to track when events occurred, aiding in diagnosing issues.

Error Stack Traces: In the event of errors, stack traces and context information are logged to pinpoint the root cause.

9.3 Exception Handling

Exception handling is a specific aspect of error handling that deals with exceptional conditions or errors that may occur during the execution of the DHAI Mobile App. It allows the app to gracefully recover from errors without crashing or compromising data integrity.

Try-Catch Blocks: Exception handling is implemented using try-catch blocks, which allow the app to attempt an operation (the "try" block) and handle any exceptions that may arise (the "catch" block).

Custom Exceptions: Custom exception classes may be created to handle specific types of errors or exceptional conditions unique to the app's functionality.

Graceful Degradation: When an exception is caught, the app may take appropriate actions to gracefully degrade the user experience rather than abruptly terminating.

Error Recovery: Exception handling routines may include error recovery strategies, such as retrying an operation, reverting to a previous state, or prompting the user for alternative actions.

Logging Exceptions: Exceptions, along with contextual information, are logged to provide a detailed record of errors for debugging purposes.

Notification: Depending on the nature of the exception, relevant stakeholders, including developers and administrators, may be notified for timely intervention.

10 Testing

The DHAI Mobile App documentation discusses the various testing strategies and methodologies employed to ensure the app's reliability, performance, and user satisfaction. Rigorous testing is a fundamental aspect of the app's development process, aimed at identifying and rectifying defects, ensuring a seamless user experience, and upholding the app's commitment to safety and convenience.

10.1 Testing Types

10.1.1 Unit Testing

Unit testing involves testing individual components or functions in isolation to ensure they perform as expected. In the context of the DHAI Mobile App, unit tests are created to verify the correctness of critical code components, including algorithms, data manipulation, and business logic. Key Aspects of Unit Testing:

Isolation: Tests are conducted on isolated code units to minimize dependencies and focus on specific functionality.

Test Frameworks: Unit tests are created using appropriate testing frameworks, such as JUnit for Android development and XCTest for iOS development.

10.1.2 Integration Testing

Integration testing assesses the interactions and interfaces between different app components, ensuring that they function harmoniously when combined. This type of testing evaluates how different parts of the app work together to deliver specific features.

Integration Testing Components:

API Integration: Ensures that API endpoints are correctly integrated into the app and that data is exchanged effectively.

Component Integration: Verifies the seamless interaction between different app components, such as user interfaces and data processing modules.

10.1.3 User Interface (UI) Testing

UI testing focuses on evaluating the app's user interface elements to ensure they are visually appealing, responsive, and user-friendly. This testing phase is critical for delivering an intuitive and enjoyable user experience.

UI Testing Aspects:

Usability Testing: Involves real users interacting with the app to provide feedback on the user interface's ease of use.

Responsiveness: Ensures that user interface elements respond promptly to user interactions, such as button presses and screen swipes.

Cross-Device Compatibility: The app is tested on various devices and screen sizes to ensure compatibility and consistent rendering.

10.2 Automated Testing

Automated testing is an integral part of the DHAI Mobile App's testing strategy. Automation frameworks are used to create and execute test cases, allowing for rapid and repeatable testing across different scenarios and devices.

Benefits of Automated Testing:

Efficiency: Automated tests can be run quickly and consistently, enabling faster feedback on code changes.

Regression Testing: Automated tests are ideal for performing regression testing, ensuring that new features or fixes do not introduce unintended issues.

Consistency: Automated tests provide consistent and reproducible results, reducing the likelihood of human error.

10.3 Manual Testing

Manual testing plays a crucial role in the app's testing process, especially for evaluating user experience aspects that require human judgment. Manual testing is typically conducted in areas such as usability, user flows, and exploratory testing.

Manual Testing Scenarios:

Usability Testing: Real users assess the app's user interface for intuitiveness and ease of navigation.

Exploratory Testing: Testers explore the app, trying various scenarios and interactions to uncover unexpected issues.

10.4 Performance Testing

Performance testing is conducted to evaluate the app's responsiveness and stability under various conditions. This type of testing helps ensure that the app can handle a wide range of user interactions and data loads without degradation in performance.

Performance Testing Aspects:

Load Testing: Simulates a high volume of user activity to assess how the app handles concurrent requests and heavy user loads.

Stress Testing: Pushes the app beyond its capacity to identify the point at which it becomes unstable or unresponsive.

Scalability Testing: Evaluates the app's ability to scale horizontally or vertically to accommodate increased user demand.

10.5 Security Testing

Security testing is paramount to safeguarding user data and maintaining the app's integrity. It involves identifying vulnerabilities and weaknesses in the app's code and infrastructure.

Security Testing Practices:

Vulnerability Scanning: Scans for known vulnerabilities and potential weaknesses in the app's codebase.

Penetration Testing: Ethical hackers attempt to exploit security flaws to identify potential risks and vulnerabilities.

Data Encryption: Ensures that sensitive data, such as user information, is encrypted and protected during transmission and storage.

10.6 Test Documentation

All testing activities, including test plans, test cases, and test results, are documented comprehensively. This documentation serves as a reference for developers, testers, and stakeholders, ensuring transparency and accountability in the testing process.

10.7 Continuous Testing

Continuous testing is integrated into the app's CI/CD pipeline, enabling automated testing to occur continuously as code changes are made. This approach ensures that the app remains stable and reliable throughout its development lifecycle.

11 Continuous Integration and Deployment (CI/CD)

Continuous Integration and Deployment (CI/CD) is a software development practice that focuses on automating and streamlining the building, testing, and deployment of applications. In the context of the DHAI Mobile App, CI/CD helps ensure that changes are integrated smoothly, tested rigorously, and deployed efficiently.

Automated Builds: The app's codebase is automatically built whenever changes are pushed to the version control system (e.g. Git). This ensures that the code is always in a buildable state.

Automated Testing: A suite of automated tests, including unit tests and integration tests, is run automatically to verify the correctness of the code. This helps catch and fix issues early in the development process.

Continuous Deployment: The CI/CD pipeline is configured to automatically deploy the app to relevant environments (e.g. staging or production) when code changes pass all tests and meet deployment criteria.

Version Control Integration: CI/CD tools are tightly integrated with version control systems, allowing developers to trigger builds and deployments through code commits and pull requests.

Deployment Strategies: Strategies such as blue-green deployments or canary releases may be employed to minimize downtime and ensure a smooth transition when deploying new app versions.

Monitoring and Rollbacks: Continuous monitoring of deployed versions helps detect issues in real-time. Automated rollback mechanisms are in place to revert to a previous version if critical errors are detected.

11.1 Build Automation (Gradle, Fastlane)

Build Automation involves automating the process of compiling, testing, and packaging the app's source code into executable artifacts. For DHAI Mobile App, build automation is typically facilitated using tools like Gradle for Android and Fastlane for iOS.

Dependency Management: Build automation tools handle the resolution and management of project dependencies, ensuring that the app's libraries and plugins are up-to-date.

Task Automation: Developers define tasks in build scripts to automate various development activities, such as compiling code, running tests, generating release builds, and packaging assets.

Customization: Build automation scripts can be customized to suit the specific requirements of the app, including the generation of signed release builds and deployment packages.

Incremental Builds: Incremental builds are supported to optimize build times. Only modified or dependent code is recompiled during subsequent builds, saving time and resources.

Integration with CI/CD: Build automation tools seamlessly integrate with CI/CD pipelines, ensuring that consistent build processes are followed during both development and deployment stages.

11.2 Versioning and Release Strategy

Versioning and Release Strategy defines how DHAI Mobile App manages version numbers and plans for the release of updates to users.

Semantic Versioning: The app follows a semantic versioning (SemVer) scheme, comprising major, minor, and patch version numbers. This helps users understand the significance of each update.

Release Planning: Release cycles are planned, and feature sets for each release are defined. This includes deciding which new features, enhancements, and bug fixes are included in a particular release.

Alpha and Beta Testing: Alpha and beta testing phases may be conducted to gather feedback from a select group of users before a full release. This helps identify and resolve issues before wider distribution.

Staging Environments: Staging environments are used to thoroughly test releases in an environment that mimics the production environment, reducing the risk of deployment issues.

Release Notes: Comprehensive release notes are provided to users, detailing what's new, improved, or fixed in each release.

Rollout Strategies: Gradual rollouts to a subset of users, also known as phased rollouts, may be employed to monitor app performance and address any unforeseen issues before a full-scale release.

User Feedback Integration: User feedback channels are established to collect user opinions and bug reports, facilitating ongoing improvements and bug fixes.

12 Documentation and Code Comments

Documentation and Code Comments are essential practices in software development that involve providing comprehensive explanations, descriptions, and comments within codebase. This helps developers, maintainers, and collaborators understand the code's functionality, purpose, and usage.

In-line Comments: Throughout the code, meaningful comments are added to explain complex logic, algorithms, or any non-trivial code segments.

Function and Method Documentation: Each function or method includes a comment block describing its purpose, input parameters, return values, and usage examples.

Class and Module Documentation: Documentation is provided at the class or module level, explaining the role and responsibilities of the class, its attributes, and how it fits into the overall system.

Usage Examples: Code comments may include usage examples to illustrate how to use specific functions, methods, or classes effectively.

Code Guidelines: Comments adhere to a consistent style and follow established coding guidelines for clarity and readability.

Change Logs: Documentation may include change logs that track modifications, enhancements, and bug fixes for each code segment.

Dependency Documentation: External dependencies and libraries used in the app are documented, including version information and usage instructions.

12.1 Code Documentation Standards

Code Documentation Standards refer to the conventions and guidelines established for documenting code within DHAI Mobile App. These standards ensure consistency and readability across codebase.

Comment Style: Define a consistent comment style, such as using a specific syntax for in-line comments (e.g. "//" for single-line comments, "/* */" for multi-line comments).

Documentation Blocks: Specify the format for documentation blocks, including what information to include (e.g. function descriptions, parameters, return values).

Naming Conventions: Establish conventions for naming variables, functions, classes, and modules, ensuring that names are descriptive and meaningful.

Code Organization: Define how code should be organized, including the placement of comments within the code structure.

Documenting Edge Cases: Guidelines for documenting edge cases, error handling, and exceptional conditions to ensure robustness and understanding code behavior.

Version Control Integration: Encourage inclusion of code documentation in version control commits and pull requests to maintain up-to-date documentation.

12.2 API Endpoint Documentation

API Endpoint Documentation is crucial for DHAI Mobile App, as it involves describing the endpoints and functionalities exposed by the app's backend APIs. API documentation helps developers, both internal and external, understand how to interact with the app's services programmatically.

Endpoint Descriptions: Each API endpoint is documented with a clear and concise description of its purpose and expected behavior.

Request and Response Formats: Documentation specifies the expected request formats (e.g. JSON, XML) and provides examples. It also defines the structure of the response and its possible status codes.

Authentication and Authorization: Explain how authentication and authorization are handled for API access, including required tokens or credentials.

Endpoint Usage Examples: Include practical usage examples for each endpoint, showcasing how to make requests and interpret responses.

Rate Limiting: If applicable, communicate rate limiting policies to ensure fair usage of the API.

Error Handling: Document error responses and their meanings, helping developers troubleshoot issues effectively.

Versioning: If multiple API versions are supported, clarify how versioning is implemented and how developers can specify the desired version.

Change Log: Maintain a change log for API endpoints, documenting updates, additions, and deprecations to keep developers informed about changes.

13 Conclusion

DHAI Mobile App is an integrated and user-centric application designed to enhance the living experience of DHAI (Defence Housing Authority Islamabad) residents. With a primary focus on safety, convenience, and community engagement, the app offers a range of critical features and services. Throughout this documentation, we have detailed the app's key components and functionalities to provide a comprehensive understanding of its capabilities. This multifaceted app serves as a digital bridge between residents and DHAI Islamabad, fostering improved security, efficient complaint management, and enhanced community engagement. DHAI Mobile App offers the following key benefits and features:

Panic Alert Dashboard: Empowers users to trigger emergency alerts and request assistance in critical situations.

Fire Alert Dashboard: Provides rapid response mechanisms for fire-related emergencies.

Complaint Dashboard: Streamlines the process of submitting and tracking complaints related to various community issues.

Medical Dashboard: Offers access to medical assistance and ambulance services within the DHAI community.

News & Promotions: Keeps residents informed about community news, events, and promotions.

Ambulance Facility: Provides quick access to ambulance services when needed.

Data Security: Robust security measures are in place to safeguard user data and privacy.

Scalability and Flexibility: The app is designed to accommodate growth, allowing for the addition of new features and services in the future.

Error Handling and Logging: Comprehensive error handling and logging mechanisms ensure app stability and facilitate rapid issue resolution.

Exception Handling: The app gracefully manages unexpected errors and exceptional conditions, preventing crashes and data corruption.

The DHAI Mobile App is a testament to the commitment of DHAI Islamabad in prioritizing the comfort, safety, and convenience of its residents. The documentation presented here serves as a comprehensive guide for users, developers, and stakeholders, facilitating a deeper understanding of the app's functionalities and the technology that drives it. Whether you are a resident looking to make the most of the app's features or a developer contributing to its enhancement, this documentation is a valuable resource for all stakeholders.

With continuous improvement and user feedback, the DHAI Mobile App aims to fulfill its vision and provide an exceptional living experience for all DHAI residents. Your comfort is truly their vision.

Developer End Project Documentation:

Folder Structure:

- **main class** (home: Splash())
- **assets** (contains images and icons)
- **lib** (contains frontend and backend code)
  - **anim** (for animations)
  - **model** (API and services)
  - **view** (frontend)
    - **screen**
      - **complaint**
        - new_complaint
          - Functions:
            - getCategoriesFromSharedPreferences()
            - storeListApi()
            - fetchAndSaveMembershipIds()
            - getLocation()
        - track_complaint
          - Functions:
            - fetchComplaintListDetailModel()
            - fetchStatusData()
      - **contacts**
        - official_contacts
          - Functions:
            - fetchEmergencyContactData()
      - **Dashboard**
        - **screens**
          - complaint
            - Functions:
              - fetchComplaintStatus()
          - dashboard
            - Functions:
              - loadSessionData() // Load data from SharedPreferences
              - getLocation()
              - fetchPages()
          - news
            - Functions:
              - fetchNews()
        - **bottombar**
          - Widgets:
            - Dashboard() // Initialize Dashboard without passing parameters
            - Complaint()
            - News()
      - **login**
        - class login
          - Functions for Android and iOS
      - **mosque**
        - announcement

    - Functions:
        - fetchNewsListData()
  - mosque_time
    - Functions:
      - fetchDataAndUpdateList()
- **pet**
  - add_pet
    - Functions:
      - fetchGenderData()
      - fetchPetTypes()
  - manage_pet
    - Functions:
      - fetchManagePetData()
- **Reset**
  - change_password
    - Functions:
      - reset_password()
      - logout()
  - reset_password
    - Functions
- **Signup**
  - class signup
    - Functions:
      - fetchPages()
      - registerUser()
- **sos**
  - class sos
    - Functions:
      - fetchStatusData()
      - fetchSosListModel()
      - ApifetchAndStoreSOSList()
- **Splash**
  - splash
    - Functions:
      - isUserLoggedIn()
- **Staff**
  - add_staff
    - Functions:
      - _getTokenFromSharedPreferences()
      - sendStaffData()
      - uploadDocuments()
  - manage_staff
    - Functions:
      - fetchManageStaffData()
- **website**
  - view_pay_bill
    - Functions:
      - _downloadAndOpenPDF()

- loadPdfFromNetwork(String url)
- **class**
  - check
    - Functions:
      - fetchMasjidTimingData()
- **widgets**

  Custom Widgets:
  - AnnouncementHorWidegt
  - CardImageWidget
  - ColorWidget
  - CompalintDetailWidget
  - ComplaintFilterSearch
  - ContactFilterSearch
  - CustomAppBar
  - CustomButton
  - CustomDialog
  - CustomDrawer
  - CustomTextField
  - ManageListWidget
  - NewsWidget
  - PetListWidget
  - PetFilterSearch
  - PickDate
  - Shimmer
  - ShowMoreDetail
  - SosFilterSearch
  - SosListWidget
  - SOSHistoryList
  - StaffFilterSearch
  - TrackListWidget
  - VanishSection

- **view model** (handles API communication)
  - **utils** (contains commonly used custom widgets)
  - **complaint_state_model**
  - **Models**:
    - complaint_list_detail_model
    - complaint_list_long_detail_model
    - complaint_state_model
    - fetch_api_data
    - new_model
    - sos_history_model
    - bill_model
    - dialogue_model
    - mosque_timing
- **constants** (contains variables like colors and sizes)

- **pubspec**.yaml (lists project dependencies)

- cupertino_icons: ^1.0.2
- motion_tab_bar_v2: ^0.3.0
- shrink_sidemenu: ^2.0.0+2-null-safety
- pie_chart: ^5.3.2
- dropdown_textfield: ^1.0.8
- http: ^1.1.0
- provider: ^6.0.1
- fluttertoast: ^8.1.1
- date_format: ^2.0.7
- shared_preferences: ^2.2.0
- marqueer: ^1.4.0
- scroll_snap_list: ^0.9.1
- custom_clippers: ^2.0.0
- flutter_dash: ^1.0.0
- dotted_border: ^2.0.0+3
- overlay_loader_with_app_icon: ^0.0.3
- flutter_html: ^3.0.0-beta.2
- html: ^0.15.4
- intl: ^0.18.1
- awesome_dialog: ^3.1.0
- image_picker: ^1.0.4
- file: ^6.1.4
- geolocator: ^7.0.3
- file_picker: ^5.5.0
- url_launcher:
- dio:
- flutter_widget_from_html: ^0.10.4
- shimmer: ^3.0.0
- animated_text_kit:
- flutter_phone_direct_caller:
- flutter_rating_bar:
- group_button: ^3.3.1
- intl_phone_field:
- syncfusion_flutter_pdfviewer: ^23.1.38
- path: ^1.8.3

Appendix

Glossary

References

Contact Information