LAB SESSION 2

TITLE: TECHNICAL REQUIREMENTS

DATE:

Problem Definition:
Generate the technical specification report for building the project.

Software used:
Microsoft Word

Theory:
Technical requirements are detailed specifications that define the technological needs and constraints of a software project. They ensure that the software functions as intended and meets both the user's needs and the constraints imposed by the development environment. Understanding and documenting these requirements is crucial for successful project execution.

Purpose of Technical Requirements
Technical requirements define how a software system should be designed, built, and operated. They bridge the gap between the high-level goals and the practical implementation details, guiding the development team in creating a product that meets the expected standards of functionality, performance, and quality.

Categories of Technical Requirements

- **Functional Requirements**
  - Definition: Describe what the system must do. They detail specific functionalities or behaviors the system should exhibit.
  - Examples: User login, data entry forms, report generation.

- **Non-Functional Requirements**
  - Definition: Describe how the system performs its functions. These cover aspects like performance, usability, security, and maintainability.
  - Examples:
  - Performance: The system should handle 10,000 concurrent users.
  - Usability: The interface should be user-friendly and accessible.
  - Security: User data must be encrypted.
  - Maintainability: Code should be modular and well-documented.

Components of Technical Requirements

1. **Front End**
   - Definition: The part of the software that interacts with the user. It involves user interface (UI) and user experience (UX) design.
   - Considerations:
   - Examples:
   - Technologies: HTML, CSS, JavaScript frameworks (e.g., React, Angular), Flutter and etc.
   - Motivation: Provides a responsive and interactive interface.
   - Features: Cross-platform compatibility, accessibility.

2. **Back End**
   - Definition: The server-side part of the software that handles business logic, data management, and server interactions.
   - Considerations:
   - Examples:
   - Technologies: Server-side languages (e.g., Python, Java), frameworks (e.g., Django, Spring Boot).
   - Motivation: Ensures efficient data processing, security, and scalability.
   - Features: API development, database integration, user authentication.

3. **Platform Independence**
   - Definition: The ability of the software to operate across different operating systems and devices without modification.
   - Considerations:
   - Examples:
   - Technologies: Cross-platform frameworks (e.g., Electron, Java).
   - Motivation: Broadens the user base and simplifies deployment.
   - Features: Consistent behavior across different platforms.

4. **Advantages Over Other Technologies**
   - Definition: Benefits of using specific technologies or approaches compared to alternatives.
   - Examples:
   - Front End: React offers better performance and modularity compared to older libraries like jQuery.
   - Back End: Node.js provides non-blocking I/O, which can be advantageous for handling real-time data.

5. **Disadvantages**
   - Definition: Limitations or potential issues with the chosen technologies.
   - Examples:
   - o Front End: Complex state management in React may have a steep learning curve.
   - o Back End: Node.js may not be ideal for CPU-intensive tasks due to its single-threaded nature.

6. **Installation Process**
   - Definition: Steps required to install and configure the software and its dependencies.
   - Considerations:
   - Examples:
   - Environment Setup: Installing necessary tools and libraries.
   - Configuration: Setting up environment variables, configuration files.
   - Deployment: Using containerization tools (e.g., Docker) for deployment.

7. **Storage**
   - Definition: Methods and technologies used to store data within the software system.
   - Considerations:
   - Examples:
     - Databases: Choosing between relational databases (e.g., MySQL) and NoSQL databases (e.g., MongoDB)
   - File Storage: Options for storing files and documents (e.g., cloud storage solutions).

Importance of Documenting Technical Requirements

- Clarity and Communication: Provides a clear understanding of what needs to be done and facilitates communication among stakeholders.
- Guidance for Development: Serves as a blueprint for developers, ensuring alignment with project goals.
- Validation and Testing: Allows for the development of test cases to verify that the system meets the specified requirements.

Evolution and Change Management

Technical requirements may evolve due to changes in project scope, technology, or user needs. Effective change management processes are essential to handle these changes without disrupting the project.

- Change Requests: Formal process to request changes to the requirements.
- Impact Analysis: Evaluates the impact of changes on the system and project.

Output/Results:

TECHNICAL REQUIREMENTS

Project Title: PublicPulse

Team Members:

- Avin Kapolkar
- Giselle Fernandes
- Joseph Fernandes

1. Front End: Flutter

- Flutter is a framework created by Google for building high-performance, visually attractive applications for mobile, web, and desktop.
- It uses the Dart programming language and offers a wide range of pre-designed widgets that make UI development faster and more customizable.
- Flutter's hot reload feature allows developers to see changes instantly, speeding up the development process.
- The framework supports reactive programming, making it easier to manage state and build dynamic UIs.
- Flutter's layered architecture allows for full customization, from the platform-specific embedder to the framework layer.
- It provides excellent documentation and learning resources, including interactive codelabs and video tutorials.
- Flutter's widget testing framework enables easy creation and running of UI tests.
- The framework has built-in support for internationalization and accessibility features.

2. Back End: Firebase

- Firebase is a comprehensive backend platform by Google that simplifies app development.
- It provides tools and services like real-time databases, user authentication, cloud storage, and analytics.
- Firebase enables developers to quickly build and deploy robust applications with real-time data sync, secure user authentication, and scalable storage.
- Its integration with other Google services and user-friendly interface make it popular for web and mobile app development.
- Firebase offers Cloud Functions, allowing developers to run backend code in response to events or HTTP requests.
- The platform includes Firebase Hosting, providing fast and secure web hosting for static and dynamic content.
- Firebase Security Rules allow for declarative, real-time security model definition for data and files.

- It offers ML Kit, bringing Google's machine learning expertise to mobile developers.
- Firebase provides A/B testing capabilities, allowing developers to test different app variants.
- The platform includes Firebase Crashlytics for real-time crash reporting and app stability monitoring.

3. Why Chosen?

a) Flutter:

- Preferred for creating high-quality, cross-platform applications from a single codebase.
- Uses Dart and provides a rich set of widgets, enabling fast and customizable UI development.
- Flutter's hot reload feature enhances productivity by allowing real-time code changes, ideal for rapid prototyping.
- Offers excellent performance due to its compilation to native code, resulting in smooth animations and responsive UIs.
- Provides a consistent look and feel across platforms, ensuring brand integrity and user experience uniformity.
- Has a growing and active community, leading to a rich ecosystem of packages and plugins.
- Supports reactive programming, making it easier to manage complex application states.
- Includes built-in testing capabilities, facilitating thorough quality assurance processes.

b) Firebase:

- Complements Flutter by providing a comprehensive backend solution.
- Offers real-time databases, user authentication, cloud storage, and analytics, allowing developers to build robust applications without managing complex backend infrastructure.
- Ensures seamless scalability and integrates well with Google services.
- Provides a serverless architecture, reducing operational overhead and allowing developers to focus on core application logic.
- Offers robust security features, including easy-to-implement user authentication and declarative security rules.
- Includes cloud functions, enabling server-side logic without the need for a separate backend server.
- Provides real-time synchronization capabilities, ideal for collaborative and live-updating applications.
- Offers built-in analytics and crash reporting, helping developers understand user behavior and quickly identify and resolve issues.

c) Synergy of Flutter and Firebase:

- The combination of Flutter and Firebase enables developers to focus on creating engaging user experiences while ensuring efficient and scalable backend support. This leads to faster time to market and improved app quality.
- Using Flutter and Firebase together provides a powerful solution for app development.
- Flutter enables rapid development of high-quality, cross-platform apps.
- Firebase offers robust backend services like real-time databases and authentication.
- This synergy allows developers to concentrate on building engaging user experiences without worrying about backend infrastructure, ensuring faster time to market, scalability, and seamless integration between frontend and backend components.
- Both technologies are developed by Google, ensuring good compatibility and consistent updates.
- The combination supports offline-first application development, improving user experience in low-connectivity situations.
- Integration between Flutter and Firebase is streamlined, with official plugins and extensive documentation available.
- This tech stack is particularly suitable for startups and MVP development due to its rapid development capabilities and low initial infrastructure costs.
- The combined use of Flutter and Firebase facilitates easier team collaboration between frontend and backend developers.

4. Motivation

- Using Flutter and Firebase together provides a powerful solution for app development.
- Flutter enables rapid development of high-quality, cross-platform apps.
- Firebase offers robust backend services like real-time databases and authentication.
- This synergy allows developers to concentrate on building engaging user experiences without worrying about backend infrastructure, ensuring faster time to market, scalability, and seamless integration between frontend and backend components.
- The combination reduces development time and costs by eliminating the need for separate codebases for different platforms.
- It enables smaller teams to create and maintain complex applications that would otherwise require larger development resources.
- The tech stack supports agile development practices, allowing for quick iterations and easy pivoting based on user feedback.
- Flutter's hot reload feature, combined with Firebase's real-time capabilities, facilitates rapid prototyping and testing of new features.
- The platform independence of this stack future-proofs applications, making it easier to expand to new platforms as they emerge.

- Firebase's analytics and crash reporting tools provide valuable insights into user behavior and app performance, informing data-driven decision-making.
- The combination is particularly beneficial for startups and businesses looking to validate ideas quickly with minimal initial investment.
- Flutter's rich UI capabilities paired with Firebase's backend services enable the creation of feature-rich, responsive applications that can compete with native apps.
- The stack's scalability allows applications to grow from MVP to enterprise-level without significant architectural changes.
- Using technologies from the same ecosystem (Google) ensures better long-term support and compatibility.

5. Features of Flutter and Firebase

- **Flutter:**

1. **Cross-Platform Development:** Enables creation of applications for iOS, Android, web, and desktop from a single codebase.
2. **Fast Development:** Hot reload feature allows instant visibility of changes, speeding up the development process.
3. **Expressive and Flexible UI:** Offers a wide array of customizable widgets for designing visually appealing user interfaces.
4. **High Performance:** Dart language compiles to native code, ensuring smooth performance.
5. **Extensive Libraries and Tools:** Provides a rich set of libraries and tools that simplify development tasks.
6. **Strong Community Support:** A large and active community offers numerous resources, plugins, and support.
7. **Reactive Framework:** Built-in support for reactive programming, making it easier to manage app state.
8. **Internationalization:** Built-in support for multi-language app development.
9. **Accessibility:** Includes features to make apps more accessible to users with disabilities.
10. **Rich Animation Support:** Offers a comprehensive set of animation primitives for creating complex, beautiful animations.
11. **Developer Tools:** Powerful debugging and profiling tools for optimizing app performance.
12. **Platform-Specific Code Integration:** Allows easy integration of platform-specific code when needed.

- **Firebase:**

1. **Real-time Database:** Facilitates real-time data synchronization across all clients with Firebase Realtime Database.
2. **Cloud Firestore:** A scalable, flexible NoSQL cloud database for storing and syncing data.

3. **Authentication:** Simplifies user authentication with integration options for various providers (email, Google, Facebook, etc.).
4. **Cloud Functions:** Allows for serverless backend code execution in response to events and HTTPS requests.
5. **Cloud Messaging:** Provides push notifications and messaging services for user engagement.
6. **Analytics:** Offers detailed analytics to monitor user behavior and app performance.
7. **Hosting:** Delivers fast and secure web hosting for both static and dynamic content.
8. **Crashlytics:** Real-time crash reporting tool for improving app stability.
9. **Cloud Storage:** Provides secure file uploads and downloads for Firebase apps.
10. **ML Kit:** Brings machine learning capabilities to mobile apps with on-device or cloud-based APIs.
11. **Remote Config:** Allows developers to change app behavior and appearance without publishing an app update.
12. **App Distribution:** Simplifies the process of distributing pre-release versions of your app to testers.
13. **Performance Monitoring:** Helps in gaining insight into app performance issues.
14. **Test Lab:** Provides cloud-based infrastructure for testing Android and iOS apps across a wide variety of devices.
15. **Dynamic Links:** Creates smart URLs that dynamically adapt to multiple platforms.
16. **App Indexing:** Helps surface app content in Google Search and Google Assistant.

6. Platform Independence

- Flutter:

1. Supports development on Windows, macOS, and Linux, with applications running on iOS, Android, web, and desktop platforms.
2. Provides a consistent development experience across different operating systems.
3. Allows developers to use their preferred IDEs and text editors, including VS Code, Android Studio, and IntelliJ IDEA.
4. Enables creation of desktop applications for Windows, macOS, and Linux from the same codebase.
5. Supports web development, allowing Flutter apps to be compiled to JavaScript for browser execution.
6. Offers platform-specific widgets (Cupertino for iOS, Material for Android) while maintaining a single codebase.
7. Allows easy customization of app behavior and appearance based on the platform it's running on.

- Firebase:

1. As a cloud-based service, it integrates seamlessly with applications across any platform (iOS, Android, web).

2. Provides SDKs and libraries for multiple platforms, ensuring consistent functionality across different environments.
3. Offers a unified dashboard for managing apps across all platforms, simplifying multi-platform app management.
4. Supports offline data persistence, allowing apps to function without constant internet connectivity across all platforms.
5. Provides REST APIs for platforms where native SDKs are not available, ensuring broad accessibility.
6. Offers platform-specific features when necessary (e.g., iOS-specific push notification setup) while maintaining overall platform independence.
7. Enables real-time synchronization of data across all platforms, ensuring consistency in multi-platform deployments.
8. Supports cross-platform authentication, allowing users to seamlessly switch between devices and platforms.

7. Advantages over Other Technologies

- **Flutter:**

1. **Single Codebase:** Allows for development across multiple platforms from a single codebase, reducing time and effort.
2. **Fast Development:** Hot reload feature accelerates the development process with real-time code changes.
3. **Customizable Widgets:** Provides a wide range of widgets for creating unique and expressive UI designs.
4. **Performance:** Compiles to native ARM code, ensuring performance comparable to native apps.
5. **Community and Ecosystem:** Supported by a robust community with a growing ecosystem of packages and plugins.
6. **Consistency:** Ensures consistent UI and behavior across platforms.
7. **Built-in Testing:** Comprehensive testing framework for unit, widget, and integration tests.
8. **Internationalization:** Built-in support for easy localization of apps.
9. **Rich Animation Support:** Powerful animation libraries for creating complex, smooth animations.
10. **Accessibility:** Built-in accessibility features to make apps more inclusive.

- **Firebase:**

1. **Ease of Integration:** Seamlessly integrates various services (authentication, database, analytics), reducing backend development effort.
2. **Real-time Data Sync:** Real-time database and Firestore offer live updates, ideal for dynamic apps.
3. **Scalability:** Handles a large number of users and data efficiently with scalable services.

4. **Comprehensive Suite:** Provides a wide array of tools (database, authentication, analytics, hosting) in one platform, simplifying development and maintenance.
5. **Cost-Effective:** Many Firebase services have a free tier, making it an affordable option for startups and small businesses.
6. **Serverless Architecture:** Reduces the need for backend infrastructure management.
7. **Robust Security:** Offers built-in security features and easy-to-implement authentication.
8. **Offline Support:** Provides offline data persistence and syncing when back online.
9. **Machine Learning Integration:** Easy integration of ML capabilities through Firebase ML Kit.
10. **Cross-Platform Support:** Works seamlessly across web, mobile, and desktop platforms.

8. Disadvantages

- **Flutter:**

1. **Large App Size:** Flutter apps can be larger than traditional apps, which might be an issue for users with limited storage or slower internet speeds.
2. **Limited Access to Device Features:** Accessing certain device features like camera or GPS may require extra effort or custom plugins, potentially slowing development.
3. **Performance Issues:** For tasks requiring high performance, such as real-time tracking or complex animations, Flutter may not be as efficient as platform-specific solutions.
4. **Learning Curve:** Developers need to learn Dart, which could initially slow down the development process.
5. **Ecosystem Maturity:** Flutter's ecosystem is newer, meaning fewer ready-made solutions and third-party libraries compared to more established platforms.
6. **Platform-Specific Look and Feel:** Ensuring a native look and feel on both Android and iOS can be challenging with Flutter's rendering engine.
7. **Third-Party Package Reliability:** Not all third-party packages are well-maintained, which could lead to stability issues.
8. **Integration Challenges:** Integrating Flutter with existing systems or technologies can be complex and time-consuming.
9. **Continuous Updates:** Frequent updates to the Flutter framework may require constant maintenance of apps.
10. **Limited Web Support:** While improving, Flutter's web support is not as mature as its mobile support.

- **Firebase:**

1. **Scalability Issues:** Firebase may struggle with very high traffic, leading to slower response times and potential downtime as "PublicPulse" grows.
2. **Complex Queries:** Firebase's NoSQL database can be limiting for advanced queries needed for data analysis and reporting.

3. **Vendor Lock-In:** Using Firebase means committing to Google's ecosystem, making it difficult to switch to other platforms later.
4. **Data Privacy and Compliance:** Firebase's data storage may not meet all regional privacy laws, posing challenges for compliance.
5. **Real-Time Limitations:** Handling a large volume of real-time updates and notifications might not be optimal with Firebase.
6. **Cost Management:** Firebase can become expensive as usage scales, making it less cost-effective over time.
7. **Limited Customization:** Firebase's out-of-the-box functionality has limited customization options for specific backend needs.
8. **Offline Capabilities:** Firebase's offline functionality may not be robust enough for users in areas with poor internet connectivity.
9. **Limited Data Migration:** Moving data in or out of Firebase can be challenging, especially for large datasets.
10. **Lack of SQL Support:** Firebase's NoSQL structure may not be suitable for applications requiring complex relational data models.

9. Installation

- **Android Installation:**

1. Download Flutter SDK
2. Extract Flutter SDK
3. Set Environment Variables
4. Install Android Studio
5. Configure Android Studio
6. Accept Android Licenses
7. Install Flutter Plugin
8. Create a New Flutter Project
9. Set Up an Emulator
10. Run the App

- **macOS Installation:**

1. Download Flutter SDK
2. Extract Flutter SDK
3. Set Environment Variables
4. Install Xcode
5. Install Android Studio (optional)
6. Run Flutter Doctor
7. Create and Run a Flutter App

- **Linux Installation:**

1. Download Flutter SDK
2. Extract Flutter SDK
3. Set Environment Variables
4. Install Android Studio (required)
5. Run Flutter Doctor
6. Create and Run a Flutter App

10. Storage Requirements for Installing Flutter and Firebase

➢ **Flutter Installation:**

- **Flutter SDK:** Approximately 1.5 GB to 2 GB after extraction. Ensure at least 3 GB of free space.
- **Additional Tools:**
  - Android Studio: About 3.5 GB to 5.5 GB
  - Xcode (for macOS): Around 12 GB
- **Emulators:**
  - Android Emulator: 1 GB to 3 GB per virtual device
  - iOS Simulator (macOS only): Included with Xcode
- **Dart SDK:** Included with Flutter, no additional space required
- **Total Storage Estimate:** 5 GB to 10 GB, depending on tools and emulators installed.

➢ **Firebase Installation:**

- **Firebase Configuration Files:** 10 KB to 100 KB each
- **Firebase SDK for Flutter:** A few megabytes depending on services used
- **Firebase CLI:** About 200 MB
- **Additional Storage for Firebase:**
  - Minimal local storage impact
  - Cloud storage for data (varies based on app usage)
- **Total Storage Estimate:**
  - Local storage: Firebase integration itself is minimal; the main storage requirements are for the SDK packages, which are a few megabytes
  - Cloud storage: Depends on the application's data storage needs

➢ **Development Machine Requirements:**

- **RAM:** Minimum 8 GB, recommended 16 GB or more
- **CPU:** Multi-core processor, 2 GHz or faster
- **Available Storage:** At least 20 GB free space for comfortable development with Flutter and Firebase

Conclusion:

Technical requirements play a vital role in software development by defining the technological needs and constraints of a project. They guide the development process, ensure alignment with project goals, and provide a basis for validation and testing. By addressing both functional and non-functional aspects, and documenting all relevant components, technical requirements help in creating a robust, efficient, and user-friendly software system.