**Software Requirements Specification (SRS)  
Project Name:** ProcrastiNOT

**1. Introduction**

**1.1 Purpose**

The project is being undertaken to address the challenges of managing and organizing multiple personal and collaborative lists effectively. Modern users struggle to track recommendations and to-dos across various categories such as movies, books, restaurants, or travel destinations. ProcrastiNOT aims to centralize these lists into a single intuitive platform, encouraging users to stay organized, share recommendations, and collaborate. The primary objectives are:

* Provide a user-friendly app for managing unique lists.
* Enable collaboration and sharing features for shared-lists.
* *Enable decision-making through engaging tools like random selection, voting, and progress tracking.*

**1.2 Intended Audience**

ProcrastiNOT aims at all mobile users who like to keep track of their plans using the pre-installed ‘Notes’ application, that comes with every smart phone, regardless of its operating system.  
At start, the application will be developed for, tested on and released on Android, and further down the line an iOS release is planned.  
We aim to target people who keep pushing plans ‘for later’ – whether watching a TV Show, going to a newly opened restaurant, or reading a recommended book.

**1.3 Intended Use**

*Describe how the application will be used. Include typical scenarios or use cases that demonstrate its main functionality.*

Imagine you’re at a family gathering, talking to all your relatives. All of the sudden, one of your cousins, who you’re very close to mentions a cool new bar in a nearby city. They show you photos from a recent get-together with their friends, the prices sound attractive, classical music and an overall great atmosphere. You WANT to go there with your friends on an upcoming free weekend! Asking the cousin, you write down the name of the place in your notes app and forget about it.   
As you leave the family gathering, telling yourself you’ll check it out once you get back home and see who’s interested to go out there with you.  
Few days pass, and you get lost in your busy, modern life. Losing track of time, the weekend comes around and you find yourself with your friends at the same place you always hang out at, as the next week passes, and the one after it, you forget ever wanting to go to that new place.  
**This** is where **ProcrastiNOT** comes in!  
Imagine instead of forgetting all about it, after adding that bar to your ‘Notes’ app, you’d add it to a shared list of hang out places with your friends, and they would all be notified of your addition. Then, as that same weekend would approach – voila! **ProcrastiNOT notifies every one in the group and reminds to go check that new location you added a few days prior!**

Now, imagine it not only with bars, restaurants and date spots but with movies, books, vacation spots, recipes and more!  
**ProcrastiNOT** would urge you to stop delaying, stop pushing things ‘for later’, ***stop procrastinating*** and rather ***start doing!***

**1.4 Product Scope**

*Outline the scope of the application. What are its primary features, and what boundaries or limitations does it have?*

Primary Features:

* A mobile application for creating, managing, and sharing lists.
* Integration of features like random selection, notifications, and progress tracking.
* Support for categories such as entertainment, dining, and tasks.
* Enable user collaboration through QR Codes and Links that would be used to share lists.

Boundaries and Limitations:

* Contextual suggestions (e.g., GPS for restaurant lists) not supported in the initial release.
* The initial release focuses on personal and shared lists but excludes advanced integrations with external platforms (e.g., streaming services or delivery apps) or any feature that would require scraping the net.
* Only mobile platforms (iOS and Android) are supported in the initial release, and a web version could be easily implemented via existing methods in React Native in future iterations.

**1.5 Definitions and Acronyms**

*List and define any technical terms, abbreviations, or acronyms that will appear in the document.*

**List** – a list of similarly ‘defined’ items. Such as movies, shows, restaurants, parks etc.   
 Can be created or deleted.

**Sharing** – the process of sending another user an already existing list, using a qr code (in person, for example) or a link (through a messaging application).

**Addition & Removal** – elements from each list can be removed or added.

**Collaboration** – lists can be created in a collaborative mode. The creator will be the ‘Owner’, and an invited user will be a ‘Collaborator’. Owners will grant privileges to collaborators, allowing them to add or remove new elements to a shared list. We’re also considering the option to create a queue for collaborators to send requests to, where the owner will accept or reject the additions to the list.

**2. Overall Description**

**2.1 User Needs**

ProcrastiNOT addresses the disorganization of lists scattered in general-purpose apps by providing a centralized, intuitive platform for managing personal and collaborative lists across multiple categories. It also solves the lack of shared lists, which are typically stored inside simple note files by enabling real-time shared list editing via QR codes and links, while gamified features like voting and random selection make list management fun and engaging. Users gain flexibility to create custom lists without restrictions, supported by reminders, notifications, and secure data handling, ensuring a seamless, productive, and enjoyable experience across iOS and Android platforms.

**2.2 Assumptions and Dependencies**

1. ***User Assumptions***
   * Users have access to smartphones with iOS or Android operating systems.
   * Users have the need for an app that stores and manages their lists.
   * Users are familiar with basic mobile app functionalities like list creation, sharing, and notifications.
   * Users value collaboration and are likely to use shared lists for group activities or recommendations.
   * Users prioritize privacy and security when sharing lists or storing data in the app.
2. **Environment Assumptions**
   * Users have a stable internet connection to enable shared lists.
   * Users can opt for push notifications and reminders for updates, necessitating a reliable notification delivery system.

***Dependencies on External Systems, Services, or Technologies***

1. ***Cloud Services***
   * ***AWS*** *for user authentication, and server hosting.*
   * *MongoDB Atlas for database storage.*
2. ***Authentication Systems***
   * *Via AWS Cognito.*
3. ***External Libraries/Frameworks***
   * *Various React Native frameworks and libraries.*
4. ***Device-Specific Features***
   * *QR code scanning and camera access.*
   * *Notification services tied to OS-level permissions.*
5. ***Development and Testing Tools***
   * Expo
   * VSCode
   * Testing frameworks for unit and integration tests.

**3. System Features and Requirements**

**3.1 Functional Requirements**

### ***1. Private List Creation & Organization***

* + Allow users to create new lists for various categories such as movies, books, restaurants, or custom themes.
  + Enable users to add, edit, delete, and rearrange items within each list.
  + Provide options to categorize lists (e.g., entertainment, travel, tasks) for easy navigation.

### 2. **Collaboration on Lists**

* + Allow users to share lists with others using shareable links.
  + Enable real-time collaborative editing, where multiple users can add or update list items simultaneously.
  + Provide notifications when changes are made to shared lists (e.g., "John added 'Bistro 56' to the restaurant list").
  + Include permissions for collaboration (e.g., view-only or edit access).

### 3. **Random Selection Tool ("I Feel Lucky")**

* + Allow users to randomly select an item from a list (e.g., "Pick a movie to watch tonight").
  + Provide an engaging animation or visual feedback for the selection process.
  + Offer options to set parameters for random selection (e.g., exclude items marked as completed or set priorities).

### 4. **Progress Tracking**

* + Enable users to mark items on a list as "completed" or "in progress."
  + Display progress visually, such as with a progress bar or percentage for each list.
  + Allow users to set priorities for items and sort lists based on priority.

### 5. **Push Notifications**

* + Notify users of updates to shared lists (e.g., new additions or edits by collaborators).
  + Send reminders for pending tasks or items marked with due dates.
  + Provide periodic engagement notifications (e.g., "Haven't updated your book list in a while!").

### 6. **Camera Integration**

* + Support scanning QR codes for quickly joining or accessing shared lists.

**3.2 External Interface Requirements**

* **User Interface:  
  General UI Design Principles**

**Intuitive and Minimalist Design:**

* + The interface should be clean, clutter-free, and easy to navigate, catering to a broad user demographic.
  + Use familiar design patterns such as tab bars (on mobile) or swipe gestures to enhance usability.
  + Use colors and a consolidated UI, built out of components, to hint at certain actions that the user can take without actually "telling" them that these actions exist.

**Accessibility:**

* + Include support for screen readers and high-contrast themes (e.g., Dark Mode)
  + Allow customization of font sizes for visually impaired users.

**Responsive and Fast:**

* + Ensure smooth animations and quick load times to keep the experience engaging and seamless.
* **Third-party Integrations:  
  Authentication Services:**Amazon Cognito for user authentication services.

**Database and Backend:**

* + MongoDB Atlas for cloud data storage.
  + Node.js with Express for RESTful APIs.

**QR Code Integration:** Via React Native library

**Analytics and Crash Reporting:** Google Analytics and Crashlytics for usage insights and debugging.

**Backup and Storage:** AWS S3 for secure cloud-based backups.

* **Cloud Storage:**

Data Storage:

* + Use MongoDB Atlas for cloud storage (user data, lists, collaboration).
  + Cache data locally (SQLite/Secure Storage) for offline access.

**Synchronization:**

* + Offline changes sync automatically once reconnected.
  + Sync data across multiple devices.

**Security:**

* + Encrypt data in transit and at rest.
  + Implement access control for shared lists.

**Backup and Recovery:**

* + Regular automated cloud backups.
  + Restore deleted lists within a set timeframe.

**Notifications:**

* + Push notifications for list updates and changes.

**3.3 System Features**

***List Creation & Organization:***

* Users can create, categorize, and manage personal lists (e.g., movies, tasks, restaurants).
* Each value in a list can have a description.

**Collaboration:**

* Share lists with others and collaborate in real-time, allowing multiple users to add, edit, or remove items.

**Random Selection:**

* Use features like “I Feel Lucky” to randomly select an item from a list for decision-making.

**Progress Tracking:**

* Track completion status and priorities for tasks and items on the lists.

**QR Code Integration:**

* Share lists quickly via QR codes for easy collaboration.

**Follow-up on Lists:**

* Receive notifications when changes are made to shared lists, keeping users informed.
* Users can also add list values directly into their calender.

**3.4 Nonfunctional Requirements**

* **Performance:**

**App Load Time:**

* The app should load within **2-3 seconds** upon startup for a smooth user experience.

**List Creation and Update Response Time:**

* Lists should be created or updated within **1-2 seconds** after the user submits changes.

**Synchronization:**

* Real-time collaboration updates should reflect within **1-3 seconds** across all devices involved.

**Push Notifications:**

* Push notifications should be delivered within **5-10 seconds** after an update is made to a shared list.

**Offline Mode Sync:**

* When reconnected, any offline changes should sync to the cloud within **30 seconds**.

**Search and Data Retrieval:**

* Search queries for lists or items should return results within **1-2 seconds**.
* **Scalability:**
  + **MongoDB Atlas** will scale automatically to handle increased user data, supporting more users and lists.
  + The backend built with **Node.js** and deployed with AWS EC2 will scale horizontally by adding more server instances as user load increases using an AWS Elastic Load Balancer.
  + When users go online, offline changes will sync seamlessly, with minimal latency, even as the number of lists and users increases.
  + Regularly monitor system performance using tools like **AWS CloudWatch** to identify potential bottlenecks as user load grows.
* **Security:**
  + Use **AWS Cognito for user authentication.**
  + Encrypt data at rest via **AES-256** and in transit with **TLS 1.2.**
  + Implement role-based access for shared lists and use **token-based authentication.**
  + Regular **encrypted cloud backups.**
* Availability:
  + Use **multi-region cloud infrastructure** via AWS to ensure continuous availability even during server outages.
  + Implement **regular backups** to restore data quickly in case of failures. Backups should be stored via AWS S3.

**4. Specific Needs of End Users**

**4.1 User Groups**

**Primary Users:** Active Participants

* Characteristics: Indivduals with an interest in managing personal or shared lists, such as students or professionals.
* Roles:
  + Creators: Create, edit, and organize lists.
  + Collaborators: Contribute to shared lists.
  + Explorers: Use features like voting or random selection for decision-making.

**Secondary Users:** Passive Participants

* Characteristics: Casual users engaging indirectly, primarily viewing or interacting minimally with shared lists.
* Roles:
  + Viewers: Access shared lists without editing.
  + Voters: Participate in voting or decision-making.

**4.2 Unique User Requirements**

**Primary Users**

* Full feature access: list creation, sharing, collaboration, and tools like random selection.
* Custom dashboards for managing categorized lists.
* Role-based permissions for collaborators.
* Secure backups and end-to-end encryption.

**Secondary Users**

* Limited interaction: view shared lists, vote, or mark items complete.
* Guest access via QR codes or links without account creation (future updates).
* Simple, minimalistic interface for non-editing actions.

**5. Change Management**

**5.1 Change Tracking**

* Changes to the document can be viewed via the project's GitHub.

**5.2 Version Control**

* Any changes in the code and version control will be handled through GitHub.

**6. Appendices**

* **Use Case Diagrams:**

A diagram of a diagram

Description automatically generated

**Purpose:** The user wants to create a list.

• **Actors involved:** User.

• **Prerequisites:** User is logged in.

**• The steps of the process:**

1. The user clicks on the "Create List" button.

2. The user adds values and customizes the list to his liken.

3. User now has a saved list where he can interact with and edit based on his needs.

• **Successful result:** User has created a list.

• **Alternative results:**

If anything fails along the way, the proper error will be presented to inform the user on what he did wrong.

**Purpose:** The user wants to create a collaborative list.

• **Actors involved:** User.

• **Prerequisites:** User is logged in and has an existing list he would like to share with others.

**• The steps of the process:**

1. The user can press the "share this list" button.

2. User will choose whether he would want to the list to be "read-only" or would he like others to be able to fully collaborate and perform actions such as modifying the values inside the list or inviting others.

3. After configuring the terms of the collaborative list, the user (owner of the list) will can either allow others to scan the QR code from his phone or send a link for quick invitation to the newly created list.

• **Successful result:** Both users can now collaborate on a list.

• **Alternative results:**

If anything fails along the way, the proper error will be presented to inform the user on what he did wrong.

**Purpose:** A group of users want to approach a mutual decision based on a collaborative list.

• **Actors involved:** 2 or more Users.

• **Prerequisites:** All users are logged in, and are in the same collaborative list.

**• The steps of the process:**

1. One user initiates a vote.

2. Each user chooses a value that resides in the list.

3.The value that gets the most votes is declared a winner.

• **Successful result:** Users came to a decision based on a vote.

• **Alternative results:**

If anything fails along the way, the proper error will be presented to inform the user on what he did wrong.