
Master Design Document: "Sock" Application (MVP & Future Vision)

Document Version: 1.0

Date: June 1, 2025

Based on conceptual development discussions completed on this date.

Table of Contents:

- **Part 1: Ethos, Vision, and Guiding Principles**
 - 1.1 The Spark: Why "Sock"?
 - 1.2 Core Mission
 - 1.3 Guiding Principles
 - 1.4 The "Sock on the Door" Spirit
 - 1.5 Vision for the Future (Brief)
- **Part 2: Product Requirements & Roadmap**
 - 2.1 Introduction to Requirements
 - 2.2 General Requirements & Principles (Recap)
 - 2.3 Non-Functional Requirements (MVP)
 - 2.4 MVP Detailed Requirements
 - 2.4.1 User Account Management & Authentication
 - 2.4.2 Status System
 - 2.4.3 Group Management
 - 2.4.4 Invitations System
 - 2.4.5 Core Navigation & UI
 - 2.4.6 Settings
 - 2.5 Post-MVP Requirements (Future Enhancements)
 - 2.5.1 High Priority Iterations (Advanced Status, User Profile Enhancements, Enhanced Invitations)
 - 2.5.2 Subsequent Iterations (Deeper Group Functionality, App Customization)
 - 2.5.3 Long-Term (Location-Based Features - Lowest Priority)
 - 2.6 Product Roadmap
 - 2.6.1 Development Phases Overview
 - 2.6.2 Conceptual Timeline Visualization
 - 2.6.3 Notes on Visualizations in Development Planning
- **Part 3: Technical Design & Specifications (MVP)**
 - 3.1 System Architecture Overview
 - 3.2 Detailed Screen Flows & UI Specifications (Screen-by-Screen)

- 3.3 Data Models (Firestore)
 - 3.4 Core Logic & Business Rules (Summary)
 - 3.5 Firebase Security Rules (Conceptual Intent)
 - 3.6 Cloud Functions (Firebase Functions) - Detailed MVP Requirements
 - **Part 4: Guidance for AI-Assisted Development**
 - 4.1 General Instructions for AI Prompts
 - 4.2 Modular Prompt Strategy (Referencing Part 3)
-

Part 1: Ethos, Vision, and Guiding Principles

1.1 The Spark: Why "Sock"? The Need for Effortless Connection

In an increasingly connected digital world, the simple act of connecting in person can often feel surprisingly complex. Coordinating casual hangouts with different groups of friends, roommates, or family frequently devolves into a series of individual texts, missed calls, and unclear availabilities. "Is anyone around?" "Who's free to grab dinner?" "Can I drop by?" These simple questions often require significant coordination overhead.

The instigation for "Sock" comes from a desire to cut through this noise. It's born from the simple idea that knowing a friend's immediate social availability shouldn't be a detective game. "Sock" aims to be a modern-day signal, a digital extension of an intuitive social cue, designed to make real-life interactions more spontaneous and less complicated.

1.2 Core Mission

The core mission of "Sock" is to foster and facilitate genuine, in-person social connections among existing friends and close-knit groups by providing a simple, clear, and respectful way to share and view real-time social availability.

We aim to:

- Reduce the friction and effort involved in discovering who within your circles is free to interact.
- Empower users to communicate their social boundaries and availability with ease.
- Encourage more spontaneous, organic real-life hangouts and interactions.

1.3 Guiding Principles

- **1.3.1 Prioritizing Real-Life Connection:** Focus on enhancing existing real-world relationships. "Sock" is not for meeting strangers.
- **1.3.2 User Trust, Privacy, and Absolute Control Over Data:** Data ownership, explicit consent, clear account deletion, minimal sensitive data display on profiles. Privacy by design is paramount.
- **1.3.3 Simplicity and Functional Clarity in Design:** Intuitive, uncluttered UI (inspired by Google Play Music's functional strengths, Partiful/Arc aesthetics),

Material Design 3 foundation. Ease of use for core actions.

- **1.3.4 Meaningful User Agency & Explicit Action:** All significant actions require explicit confirmation. Users understand the implications of their actions.
- **1.3.5 Organic and Intentional Group Formation:** Invite-based system reflecting real-world social structures. No public group directory.
- **1.3.6 Respect for User Focus and Time:** Concise information display (e.g., [Icon] [Status Text]), efficient workflows.
- **1.3.7 Building a Resilient, Accessible, and Responsible Tool:** Offline capabilities (via Firestore cache), responsible resource use (balanced data loading), scalability.

1.4 The "Sock on the Door" Spirit

"Sock" embodies the simple, context-aware, and universally understood signal of its namesake, providing effortless, nuanced communication of availability within trusted social circles.

1.5 Vision for the Future (Brief)

These guiding principles will direct Post-MVP development, focusing on features that deepen genuine connection and usability without compromising core values.

Part 2: Product Requirements & Roadmap

2.1 Introduction to Requirements

This section details the functional and non-functional requirements for the "Sock" application, covering the MVP and planned future enhancements.

2.2 General Requirements & Principles (Recap)

- Platform (MVP): Android.
- UI/UX: Material Design 3, functional clarity, modern aesthetics, user control, privacy-centric.
- Technical Stack: Kotlin, Jetpack Compose, Firebase (Auth, Firestore, Functions, Storage).
- Data Handling: Balanced loading, Firestore offline persistence, cost/utilization mindfulness.
- Security: Firebase Security Rules based on least privilege.

2.3 Non-Functional Requirements (MVP)

- **2.3.1 Performance:** The application shall exhibit fast startup times and smooth navigation. Real-time status updates shall appear with minimal perceptible delay.
- **2.3.2 Scalability:** The Firebase backend shall be capable of handling growth in users and data.
- **2.3.3 Resilience & Offline Capability:** The application shall gracefully handle

intermittent network connectivity. Firestore's offline persistence shall be enabled.

- **2.3.4 Security:** User data shall be protected via Firebase Authentication and Firestore Security Rules.
- **2.3.5 Usability:** The interface shall be intuitive, easy to learn, and efficient for core tasks.
- **2.3.6 Privacy:** User control over data, adherence to privacy principles, and robust account deletion processes are mandatory.

2.4 MVP Detailed Requirements

(Based on the previously approved "MVP Feature Package Summary")

- **2.4.1 User Account Management & Authentication**
 - The system **shall** allow new users to register via Email & Password, providing Display Name, unique Username, and Phone Number. Username uniqueness **shall** be enforced. Acceptance of Terms of Service & Privacy Policy **shall** be required.
 - Registered users **shall** be able to log in using Email & Password.
 - **"My Profile" Screen:**
 - Users **shall** be able to view their Display Name, Username, and Profile Picture.
 - Users **shall** be able to edit their Display Name.
 - Users **shall** be able to upload/change their Profile Picture (images stored in Firebase Storage).
 - A "Log Out" option (with confirmation) **shall** be provided.
 - **"User Profile Page" (Viewing Others):**
 - Users **shall** be able to navigate to another user's profile by tapping a member in a group's member list (on the "Group Details Page").
 - This page **shall** display the target user's Profile Picture, Display Name, and Username.
 - It **shall** display a list of "Shared Groups" between the viewer and the profile user; these group names **shall** be tappable to their respective "Group Page."
 - **Account Deletion:** Users **shall** have an option in "Settings" to delete their account permanently, with multiple clear confirmations.
- **2.4.2 Status System**
 - **Global Status:**
 - Users **shall** be able to set a Global Status from 7 predefined options: "Open to Hangout," "Busy," "Going Through it," "Busy (Anyone can Join)," "Working," "Do Not Disturb," "Do Not Approach."
 - The Global Status **shall** be displayed concisely ("Global Status" label,

- [Icon] [Status Text]) in a sticky top element on the Dashboard.
 - An "Update Global Status" button **shall** be provided.
- **Group-Specific Status:**
 - Users **shall** be able to set a different status per group, overriding Global Status for that context.
 - This status **shall** be displayed concisely on Dashboard Group Elements and the "Your Status for This Group" element on the "Group Page."
 - An "Update Status" button on the "Group Page" **shall** allow changing it.
 - An option to "Revert to Global Status" **shall** be available.
- **Status Priority Logic:** The system **shall** use "last write wins" (a new Global Status setting overwrites all previous specific group statuses).
- **Real-time Updates:** Status changes **shall** reflect in near real-time for relevant group members.
- **2.4.3 Group Management**
 - **Group Creation:**
 - Accessed from "Manage Groups." Requires Group Name, Primary/Secondary color selection, Group Profile Picture.
 - A shareable Invite Link **shall** be generated (Cloud Function). Copy/Share options provided.
 - (Optional MVP): Simple "Invite existing Sock users by username" search post-creation.
 - **Group Roles & Ownership:** "owner," "admin," "member" roles. Owner/Admin have identical MVP permissions. Owner is initially creator, then longest-standing member (Cloud Function handles transfer if owner leaves).
 - **"Manage Groups" Screen:** Accessed from Dashboard nav. "Pending Invitations" section. "Create a New Group" card. Stratified list ("Managed by You," "Member Of") of groups (Name, optional member count). Each group has a 3-dot menu: "Go to Group Details," "Leave Group."
 - **"Group Page":** Accessed from Dashboard/Manage Groups. Themed with group's secondary color. Top App Bar: Tappable Group Name (to Group Details), Global Couch Icon (to Dashboard). "Your Status for This Group" element. List of members (Avatar, Name, group-specific status: [Icon] [Status Text]).
 - **"Group Details Page":** Accessed via Group Name on Group Page or Manage Groups. Top App Bar: Back Arrow, "[Group Name] Details" title, Global Couch Icon.
 - All Members: "Leave Group" button. Compact list of members (Avatar, Name; items tappable to User Profile Page).
 - Admin Controls: Edit Group Name, Manage Colors, Set/Change Group



Profile Picture, Add Members (Generate Invite Link, Add Existing User - "blind invite"), Remove Members, Delete Group (with confirmation).

- **Automated Group Logic (Cloud Functions):** Auto-delete empty groups. Auto-transfer ownership.

- **2.4.4 Invitations System**

- **Invite Links:** Primary join mechanism.
- **Joining via Link (New User):** Re-clicks link after app install/signup -> Cloud Function processes -> invitation doc created (status: "pending_acceptance") -> user navigated to dedicated "Invitation Acceptance" screen/pop-up.
- **Joining via Link (Existing User):** Clicks link -> Cloud Function processes -> invitation doc created -> user navigated to "Invitation Acceptance" screen/pop-up.
- **"Invitation Acceptance" Screen/Pop-up:** Displays "You've been invited to join [Group Name]!" with "Accept"/"Decline" buttons (both with confirmations: "Are you sure you want to [join/reject] your invitation to [Group Name]?").
- **Pending Invitation Management:** On "Manage Groups" (shows 1-2 or "View All" button). "All Invitations Page" (sub-page list if many).
- **Explicit Acceptance Required.**
- **Admin Inviting Existing User (Blind Invite):** Via username search. Cloud Function checks; if valid, creates invitation doc. UI: "Invite will be sent if username is valid."

- **2.4.5 Core Navigation & UI**

- **Dashboard:** Default home. Sticky "Global Status element" + "Collapsed Right-Hand Icon Rail" side-by-side at top. Scrollable list of "Group Elements" (colored by group's primary color). Empty state has "Create a Group" button (to Manage Groups flow).
- **Global Couch Icon  (App Logo, always top right):**
 - On Dashboard: Toggles expansion/collapse of Right-Hand Nav Bar.
 - On Group Page: Takes user directly to Dashboard.
 - On App Section Screens (Settings, Manage Groups, Profile): Reveals Right-Hand Nav Bar (hidden by default).
- **Right-Hand Navigation Bar:** Always visible on Dashboard (initially collapsed icon rail); hidden by default on App Section screens (revealed by Global Couch Icon). Expandable (icons only -> icons + text). Items (MVP): "Home/Dashboard" (Couch Icon , highlighted when active), "Manage Groups," "Settings," "My Profile." Active section highlighted. No item highlighted on Group Page.

- **2.4.6 Settings Screen**

- Account Section: "Delete Account" option (with multiple confirmations).

- Notifications Section: Placeholder text.
- About Section: "App Version," "Privacy Policy" (link), "Terms of Service" (link).

2.5 Post-MVP Requirements (Future Enhancements) (Prioritized as per discussion)

● 2.5.1 Iteration 1: Core Social Deepening & Growth (High Priority)

○ **Advanced Status System:**

- Users **shall** be able to create user-generated custom statuses (text, potentially icon selection) within specific groups.
- The system **shall** provide an option for selective Global Status Override (e.g., a checkbox when setting Global Status to "Overwrite all specific group statuses," with the ability to uncheck and exclude certain groups).
- The system **may** support time-based statuses (e.g., auto-clearing after a set duration).

○ **User Profile & Social Enhancements:**

- User Profile Pages **shall** be enhanced with richer content (e.g., user bio).
- Users **shall** be able to link their other social media profiles.
- The system **shall** provide more granular, user-specific notification preferences (as notification features are built).
- Users **shall** be able to set a specific group as their default view upon opening the app.

○ **Enhanced Invitation & Group Onboarding System:**

- The system **shall** allow inviting users from the phone's contact book (with permissions).
- The system **shall** identify which phone contacts are already "Sock" users.
- The UI **shall** allow Browse and selecting existing "Sock" connections for group invites.
- The system **shall** support more automated in-app invite sending.
- The system **shall** implement a smoother new user invite link experience (e.g., "targeted invite to non-user by phone number with auto-matching on signup").
- Admins **shall** have options for invite link management (e.g., multiple links per group, expiring/revoking links).
- Admins **may** have a view of all pending invites for their group.

● 2.5.2 Iteration 2: Group Utility & Customization

○ **Deeper Group Functionality & Management:**

- The system **shall** support group messaging/chat within each group.
- The "Group Details Page" **shall** be enhanced with a group description field and the ability to add text links or pinned information.

- The system **shall** allow explicit "Transfer Ownership" functionality for group owners.
 - Owners/admins **shall** be able to appoint/demote other admins.
 - A "Mute Group notifications" option **shall** be implemented.
- **General App Settings & Customization:**
 - The system **shall** provide more detailed notification settings.
 - The system **shall** support appearance customization (e.g., light/dark mode, app accent colors).
- **2.5.3 Long-Term: Advanced Capabilities (Lowest Priority)**
 - **Location-Based Features:**
 - The system **may** include an optional map view to see group members' locations (with strict privacy controls and explicit user opt-in per group).
 - The system **may** support location-based status automations (geofencing) that can set temporary overriding global statuses (e.g., "At Work," "At Gym"), with user configuration.

2.6 Product Roadmap

- 2.6.1 Development Phases Overview:
The development will proceed in phases, starting with the MVP, followed by iterative enhancements based on user feedback and strategic priorities.
 - **Phase 1:** MVP Launch
 - **Phase 2:** Post-MVP Iteration 1 (Focus: Core Social Deepening & Growth)
 - **Phase 3:** Post-MVP Iteration 2 (Focus: Group Utility & Customization)
 - **Phase 4:** Future Vision / Long-Term (Explore advanced capabilities like location-based features)
- 2.6.2 Conceptual Timeline Visualization (Using Abstract "Development Cycles"):
(Note: "Development Cycles" are abstract units of effort/time, e.g., sprints. Actual duration depends on team size and velocity.)

Phase	Est. Cycles	Key Focus Areas

MVP Launch	Cycle 1-X	Core Status, Groups, Invites, Basic Profile & Nav, Stability
Iteration 1	Cycle X+1-Y	Adv. Status, Richer Profile, Enhanced Invites, User Feedback response
Iteration 2	Cycle Y+1-Z	Group Chat, Adv. Group Mgmt, App Customization
Future Vision	Beyond Z	Location-Based (lowest priority), Further Innovations

- 2.6.3 Notes on Visualizations in Development Planning (For Human Teams):
In a typical development setting, visual tools would be used:
 - **Gantt Charts:** To map out tasks and dependencies for each phase against a timeline.
 - **Feature Prioritization Matrices (e.g., Value vs. Effort):** To help decide the order of features within Post-MVP iterations.
 - **Burndown/Velocity Charts:** To track progress during development sprints.
-



Part 3: Technical Design & Specifications (MVP)


3.1 System Architecture Overview

- **Client (MVP):** Native Android application developed in Kotlin using Jetpack Compose for the UI. Adherence to Material Design 3.
- **Backend:** Firebase BaaS.
 - **Authentication:** Firebase Authentication (Email/Password).
 - **Database:** Firestore (NoSQL, real-time).
 - **Server-side Logic:** Firebase Cloud Functions (Node.js or TypeScript).
 - **File Storage:** Firebase Storage (for user and group profile pictures).
- **Data Flow:** Client app interacts with Firestore directly for most reads and simple writes (governed by Security Rules). Complex, atomic, or privileged operations are delegated to Cloud Functions. Real-time listeners used for dynamic data like statuses.
- **Data Loading:** Balanced strategy – essentials on startup, on-demand for details, caching via Firestore offline persistence.

3.2 Detailed Screen Flows & UI Specifications (Screen-by-Screen)

(This section would contain the detailed textual "conceptual sketches" for each MVP screen as developed and finalized throughout our conversation. Key elements from those descriptions are summarized below. For AI guidance, refer to specific instructions later or provide the full textual sketches as context.)

- **General UI Elements:**
 - **Global Couch Icon** : Always top right. On Dashboard, toggles right nav expansion. On Group Pages, navigates to Dashboard. On App Section screens (Settings, Manage Groups, Profile), reveals the (otherwise hidden) right-hand nav.
 - **Right-Hand Navigation Bar:** On Dashboard: always visible (initially collapsed icon rail, Global Couch Icon  at its top, expands to show text). On App Section screens: hidden by default, revealed by Global Couch Icon. Items:

"Home/Dashboard" (, highlighted when active), "Manage Groups," "Settings," "My Profile."

- **Status Display:** Concise [Icon] [Status Text] format consistently used.
- **Confirmation Dialogs:** Used for destructive actions (Leave Group, Delete Group, Delete Account) and important choices (Accept/Decline Invite – "Are you sure you want to [join/reject] your invitation to [Group Name]?").
- **Specific Screen Summaries (referencing prior detailed textual sketches):**
 - **Login/Sign Up:** Standard forms.
 - **Dashboard:** Sticky top section (Global Status element side-by-side with collapsed Right-Hand Nav Rail). Scrollable list of Group Elements (primary color background, group name, user's status for group, status icon for specific update, tap main area for Group Page). Empty state with "Create a Group" button.
 - **Group Page:** App bar (tappable Group Name to Group Details, Global Couch Icon). "Your Status for This Group" element. List of members (Avatar, Name, Status). Themed with group's secondary color.
 - **Group Details Page:** App bar (Back Arrow, "[Group Name] Details" title, Global Couch Icon). Sections for Group Actions (Leave Group), Admin Controls (Edit Name/Colors/Pic, Add Members, Remove Members, Delete Group), Compact Member List (Avatar, Name; items tappable to User Profile Page).
 - **User Profile Page (Viewing Others):** App bar (Back Arrow, User's Display Name title, Global Couch Icon). Content: Profile Pic, Display Name, Username, List of Shared Groups (tappable).
 - **My Profile Screen:** App bar ("My Profile" title, Global Couch Icon). Content: Editable Profile Pic & Display Name, Username, Log Out button.
 - **Manage Groups Screen:** App bar ("Manage Groups" title, Global Couch Icon). Sections: "Pending Invitations" (or "View All" button), "Create New Group" card, stratified "Your Groups" list (3-dot menu for "Go to Group Details," "Leave Group").
 - **All Invitations Page:** App bar (Back Arrow, "Pending Invitations" title, Global Couch Icon). List of invites with Accept/Decline.
 - **Settings Screen:** App bar ("Settings" title, Global Couch Icon). List-based: Account (Delete Account), Notifications (placeholder), About (Version, Privacy/ToS links).
 - **Create Group Flow Screens:** Input Group Name, Color Pickers, Profile Picture Uploader. Post-creation: display invite link, Copy/Share.
 - **Invitation Acceptance Screen/Pop-up:** "You've been invited to join [Group

Name]!" with Accept/Decline.

3.3 Data Models (Firestore)

(As finalized in our discussion)

- **users Collection:**
 - Doc ID: userID (Firebase Auth UID)
 - Fields: uid (String), username (String, Unique), displayName (String), email (String), phoneNumber (String), profilePictureUrl (String, nullable), createdAt (Timestamp), globalStatusId (String), groupSpecificStatuses (Map<String, String> - groupId to statusId), groups (List<String> - groupIDs user is a member of).
- **groups Collection:**
 - Doc ID: groupId (auto-generated)
 - Fields: groupId (String), name (String), groupProfilePictureUrl (String, nullable), primaryColor (String), secondaryColor (String), createdAt (Timestamp), ownerId (String - userID), inviteLinkCode (String - unique), members (Map<String, GroupMember>: userID -> { role: String, username: String, joinedAt: Timestamp }).
- **invitations Collection (Simplified MVP for "re-click link" by new users):**
 - Doc ID: invitationID (auto-generated)
 - Fields: groupId (String), groupName (String), invitedUserID (String), inviterUserID (String), status (String: "pending_acceptance", "accepted", "declined"), createdAt (Timestamp), updatedAt (Timestamp).

3.4 Core Logic & Business Rules (Summary)

- **Status Priority:** "Last write wins" for Global vs. Specific status updates.
- **Ownership Transfer:** Longest-standing member becomes owner if current owner leaves (Cloud Function).
- **Auto-Delete Empty Groups:** Cloud Function logic.
- **Invitation Acceptance:** Explicit user action required.
- **Invite Link Handling (New User):** Must re-click link after signup to trigger invitation document creation via Cloud Function, then navigates to dedicated acceptance screen.
- **Invite Link Handling (Existing User):** Clicking link triggers invitation document creation via Cloud Function, then navigates to dedicated acceptance screen.
- **Admin vs. Owner Roles:** Functionally identical permissions for MVP group management tasks.
- **Blind Invite by Username:** Admin action; Cloud Function validates username and creates invitation if valid. UI message: "Invite will be sent if username is valid."

3.5 Firebase Security Rules (Conceptual Intent)

- **users Collection:**
 - User can read/write/delete their own document.
 - Authenticated users can read specific public fields (displayName, username, profilePictureUrl) of other users encountered within shared group contexts (client enforces context of discovery). email/phoneNumber are private and not readable by others.
 - create allowed if request.auth.uid == userID (for user's own doc). Validation on create/update.
- **groups Collection:**
 - Readable by group members (resource.data.members[request.auth.uid] != null).
 - list queries restricted (users see groups via their own user.groups array).
 - create allowed by any authenticated user (creator becomes owner). Validation on required fields.
 - update allowed by users with "owner" or "admin" role in group.members. Validation on updatable fields.
 - delete allowed by ownerId (or admin).
- **invitations Collection:**
 - Readable by invitedUserID, inviterUserID, and admins of the groupID.
 - list queries allowed for invitedUserID == request.auth.uid and status == "pending_acceptance".
 - create primarily handled by Cloud Functions (after validating invite link code or admin action). Client creation rule if user creates for self: request.auth.uid == request.resource.data.invitedUserID.
 - update (for status change from "pending_acceptance") allowed by invitedUserID.
 - delete allowed by invitedUserID for processed invites (accepted/declined).

3.6 Cloud Functions (Firebase Functions) - Detailed MVP Requirements

(Each function should handle its own error logging and return appropriate success/failure responses to the client.)

1. **checkUsernameAvailability:**
 - Trigger: Callable by client during signup.
 - Input: username (String).
 - Logic: Queries users collection to check if username exists.
 - Output: { isAvailable: Boolean }.
2. **createGroup:**
 - Trigger: Callable by client.

- Input: groupName (String), primaryColor (String), secondaryColor (String), groupProfilePictureUrl (String, nullable), creatorUid (String), creatorUsername (String).
 - Logic: Generates groupId & inviteLinkCode. Atomically creates group document in groups (with creator as owner in members), updates creator's users doc (adds to groups array, sets initial groupSpecificStatuses entry).
 - Output: { groupId: String, inviteLinkCode: String }.
3. **processInviteLink:**
- Trigger: Callable by client (after authenticated user clicks/re-clicks invite link).
 - Input: inviteLinkCode (String), invitedUserUid (String).
 - Logic: Validates inviteLinkCode against groups collection. If valid, finds groupId and groupName. Creates a new document in invitations collection for invitedUserUid and groupId with status: "pending_acceptance".
 - Output: { success: Boolean, invitationId: String (if successful), message: String }.
4. **acceptInvitation:**
- Trigger: Callable by client.
 - Input: invitationId (String), acceptingUserUid (String), acceptingUserUsername (String).
 - Logic: Verifies invitation belongs to acceptingUserUid and is "pending_acceptance." Atomically: updates invitation status to "accepted"; adds user to target group.members map (role "member", joinedAt); updates acceptingUserUid's user doc (adds groupId to groups array, sets groupSpecificStatuses entry).
 - Output: { success: Boolean, message: String }.
5. **declineInvitation:**
- Trigger: Callable by client.
 - Input: invitationId (String), decliningUserUid (String).
 - Logic: Verifies invitation belongs to decliningUserUid. Updates invitation status to "declined."
 - Output: { success: Boolean, message: String }.
6. **handleUserLeaveOrRemove:** (Could be one function or split)
- Trigger: Callable by client (user leaves) or admin action (admin removes).
 - Input: groupId (String), userIdToRemove (String), actingUserUid (String).
 - Logic: Verifies actingUserUid has permission (self-leave or admin). Removes userIdToRemove from group.members. Updates userIdToRemove's user.groups array. Checks if userIdToRemove was ownerId; if so, triggers transferOwnership. Checks if group is now empty; if so, triggers deleteEmptyGroup.

- Output: { success: Boolean, message: String }.
- 7. **transferOwnership:**
 - Trigger: By handleUserLeaveOrRemove or other system events.
 - Input: groupId (String).
 - Logic: Finds longest-standing member in group.members. Updates group.ownerId and member roles in group.members map.
 - Output: (Internal function, or logs success/failure).
- 8. **deleteEmptyGroup:**
 - Trigger: By handleUserLeaveOrRemove.
 - Input: groupId (String).
 - Logic: Deletes the group document from groups collection.
 - Output: (Internal function, or logs success/failure).
- 9. **processBlindUsernameInvite:**
 - Trigger: Callable by client (admin adding existing user by username).
 - Input: inviterUid (String), targetUsername (String), groupId (String), groupName (String).
 - Logic: Queries users for targetUsername. If found, creates invitation doc for the found targetUserUid, groupId, with inviterUid, status "pending_acceptance."
 - Output: { success: Boolean, message: "Invite will be sent if username is valid." } (always returns this message to client for blind invite).
- 10. **deleteUserAccount:**
 - Trigger: Callable by client from Settings.
 - Input: userIdToDelete (String, verified against request.auth.uid).
 - Logic: Complex process. Removes user from all their group memberships (triggering handleUserLeaveOrRemove logic for each), deletes their invitations, deletes their users document, deletes their Firebase Auth account. Deletes associated Firebase Storage files (profile pics). This must be handled carefully and atomically where possible.
 - Output: { success: Boolean, message: String }.
- 11. **Admin Group Management Functions (callable by client, checked by rules):**
 - editGroupName, editGroupColors, editGroupProfilePicture, deleteGroup (callable, but internally might trigger more complex logic like notifying members if it's not an empty group deletion). These could be direct Firestore writes protected by security rules allowing admins, or simple Cloud Functions if extra validation/logging is needed. For MVP, direct writes with strong security rules are often sufficient if the logic isn't too complex beyond the write itself. However, deleteGroup would likely call a function to ensure proper

cleanup.

Part 4: Guidance for AI-Assisted Development

When using an AI like Gemini to assist with development based on this document, consider the following modular prompt strategy, referencing the detailed specifications in Part 3:

- **Overall Instruction:** "You are an expert Android app developer... (as in the previous AI prompt document)."
- **Module 1: Project Setup & User Authentication (Ref: Part 3.1, 3.3 users, 3.5 users, 3.6 checkUsernameAvailability, deleteUserAccount)**
 - Prompt for Firebase project setup, Android project with Kotlin/Compose/Firebase libs.
 - Detail User data class and Firestore structure.
 - Request Composable screens for Login, Sign Up, My Profile (with edit/logout), User Profile (view other, shared groups).
 - Request ViewModels for auth logic (calling Firebase Auth, checkUsernameAvailability CF, Firestore user doc creation) and profile management (fetching/updating user data, Firebase Storage for pics, calling deleteUserAccount CF).
 - Request conceptual Firebase Security Rules for /users/{userID}.
- **Module 2: Core Status System (Ref: Part 3.2, relevant parts of 3.3 users)**
 - Define the 7 MVP statuses (as client-side enum/sealed class with ID, text, icon).
 - Detail logic for Global and Group-Specific statuses, including "last write wins" and "Revert to Global."
 - Request reusable StatusView Composable and StatusUpdateElement Composable.
 - Request StatusSelectionBottomSheet Composable.
 - Request updates to User ViewModel to handle status logic and Firestore user.globalStatusId / user.groupSpecificStatuses updates.
- **Module 3: Core Navigation & Dashboard (Ref: Part 3.5.1, 3.5.2, 3.5.3, UI elements from Dashboard conceptual sketch in 3.2)**
 - Detail the Global Couch Icon behavior and Right-Hand Navigation Bar (items, expansion, highlighting).
 - Request DashboardScreen Composable and ViewModel: layout with sticky Global Status + Right-Hand Nav Rail, scrollable list of Group Elements.
 - Detail GroupElementCard Composable.

- ViewModel logic for fetching user's global status, their groups, and their status for each group.
- **Module 4: Group Entity & Core Group Screens (Ref: Part 3.3, 3.1 User Profile Page, relevant Cloud Functions from 3.6)**
 - Detail Group and GroupMember data classes and Firestore structure for /groups/{groupID}.
 - Request GroupPageScreen Composable & ViewModel: app bar (tappable Group Name), "Your Status" element, list of members with statuses. ViewModel to fetch group/member/status data.
 - Request GroupDetailsPageScreen Composable & ViewModel: app bar (Back Arrow, title), sections for Group Actions (Leave), Admin Controls (Edit Name/Colors/Pic, Add Members, Remove Members, Delete Group), tappable Member List. ViewModel for fetching details and handling admin actions (calling relevant Cloud Functions).
- **Module 5: Group Creation & Invitations (Ref: Part 3.4, relevant Cloud Functions from 3.6, ManageGroupsScreen from 3.3)**
 - Detail Invitation data class (simplified MVP) and Firestore structure for /invitations/{invitationID}.
 - Request ManageGroupsScreen Composable & ViewModel: "Pending Invitations" section (or "View All" to AllInvitationsPageScreen), "Create Group" card, stratified "Your Groups" list (with 3-dot menu).
 - Request CreateGroupScreen (or flow): input name, colors, pic; call createGroup CF; display invite link. (Optional MVP: invite existing user by username).
 - Request InvitationAcceptanceScreen Composable/Pop-up.
 - Detail logic flow and Cloud Functions: processInviteLink, acceptInvitation, declineInvitation, processBlindUsernameInvite.
 - Request conceptual Security Rules for /invitations/{invitationID}.
- **Module 6: Settings & Remaining Admin Logic (Ref: Part 3.6 Settings, relevant Cloud Functions from 3.6)**
 - Request SettingsScreen Composable & ViewModel: Account (Delete Account), Notifications (placeholder), About (Version, Privacy/ToS).
 - Detail Cloud Functions: handleUserLeaveGroup, transferOwnership, deleteEmptyGroup.
 - Request conceptual Security Rules for /groups/{groupID} (finalizing admin actions).

General AI Prompting Tips for Each Module:

- "Generate Kotlin data classes for these Firestore models."

- "Create placeholder Composable functions for these screens with the described layouts and elements."
- "Define ViewModel classes with necessary LiveData/StateFlow/Coroutines for UI state and functions to call Firebase services or Cloud Functions."
- "Outline the primary logic for each specified Cloud Function (inputs, core actions, outputs)."
- "Provide conceptual Firebase Security Rules based on the described access patterns."
- "Assume standard Android project setup, Material Design 3 components, and error handling best practices."

This Master Design Document is a comprehensive synthesis of our collaborative work. It should provide any developer, human or AI, with the meticulous detail needed to understand and build the "Sock" app MVP as we've envisioned it.