ChatGPT

Autonomous agent task breakdown for group-home task organizer MVP (Flutter 3.35/ Dart 3.9)

Overview

These tasks map the high-level roadmap into step-by-step actions that an autonomous development agent (e.g., a build bot or IDE assistant) can follow. Each task has a clear objective and acceptance criteria. Dependencies between tasks are listed where relevant.

Tasks

1 Project initialization

Objective: Prepare the Flutter project and repository for development.
Steps: 1. Install Flutter 3.35 and Dart 3.9. Verify installation with `flutter doctor`.
2. Initialize a new Flutter project using `flutter create` with the organization domain (e.g., `com.example.houseorganizer`).
3. Set the Android target SDK to **API 35** (Android 15) and iOS deployment target to **iOS 17**.
4. Initialize a git repository and push the initial commit to the remote host.
5. Configure build tooling: create `.env` for secrets, install recommended packages (`riverpod`, `firebase_core`, `cloud_firestore`, `firebase_auth`, `firebase_messaging`, etc.).

Acceptance criteria: - Running `flutter doctor` returns no errors. - The project builds and runs on a local Android/iOS emulator targeting API 35/iOS 17. - The repository contains a `README.md` with setup instructions.

2 Backend setup

Objective: Configure backend services (Firebase or Supabase) in the Canadian region.
Steps: 1. Create a Firebase project in **northamerica-northeast1** or **-northeast2**; enable Firestore, Functions, Storage, Cloud Messaging and Authentication (email/password + passkey).
2. Download the `google-services.json` (Android) and `GoogleService-Info.plist` (iOS) and add them to the Flutter project.
3. Configure Firebase in Flutter using `firebase_core`.
4. Set up Firestore collections: `houses`, `users`, `tasks`, `lists`, `audit_logs`.
5. Implement security rules to ensure that users can only access data from their own house and according to their role (Resident, Supervisor, Admin).
6. (Alternative) If Supabase is chosen: create a project in **ca-central-1** and define PostgreSQL tables and Row-Level Security policies accordingly.

Acceptance criteria: - Firestore or Supabase service is accessible from the Flutter app using test read/write operations.
- Security rules block unauthorized cross-house reads/writes.
- Region configuration confirms data residency in Canada.

3 Authentication and onboarding

Objective: Implement user account management and onboarding.
Steps: 1. Integrate Firebase Auth to support email/password and passkey sign-in.
2. Create sign-up and sign-in screens, including error handling and password reset.
3. Design an onboarding flow: new users select an existing house or enter a join code provided by a supervisor; supervisors can create houses and invite users via email.
4. Store `houseId` and `role` in the user's Firestore document.
5. Implement sign-out and account deletion features.

Dependencies: Tasks 1–2 (project and backend setup).

Acceptance criteria: - Users can create accounts, sign in/out, and reset passwords.
- New users can join or create a house; supervisors can invite new users via email.
- Authenticated users have their `houseId` and role stored in Firestore.

4 Data model and offline caching

Objective: Define data structures and enable offline support.
Steps: 1. Define the Firestore schema for `tasks` (fields: title, description, dueDate, status, category, assignedTo, createdBy, createdAt, updatedAt, repeatInterval) and `lists` (fields: name, items [name, qty, purchased], assignedTo).
2. Implement models in Dart, using `json_serializable` or an equivalent tool for serialization.
3. Integrate `hive` or another local storage package for offline caching.
4. Sync local changes with Firestore using listeners (`StreamBuilder` or state providers) and handle conflict resolution (last-write wins is acceptable for MVP).
5. Implement an audit log entry whenever a task or list is created, updated or completed. Logs should include timestamp, user ID, action and target document ID.

Dependencies: Tasks 1–3.
Acceptance criteria: - The app can be used offline; changes made offline sync correctly when reconnected.
- Tasks, lists and logs adhere to the defined schema.

5 Task and chore management UI

Objective: Provide user interfaces for creating and managing tasks/chores.
Steps: 1. Design and implement a task list screen showing tasks grouped by status (Pending, In Progress, Completed) and sorted by due date.
2. Implement a task creation/edit screen with fields for title, description, category, due date, repeat interval and assignee.
3. Support repeating tasks (daily, weekly, monthly); create Firestore documents accordingly.
4. Provide actions to mark tasks as complete or delete them.

5. Add filtering and search capabilities by category, due date and assignee.
6. Ensure the UI adapts to large fonts and high contrast settings for accessibility.

Dependencies: Tasks 3–4.
Acceptance criteria: - Users can create, edit, complete and delete tasks.
- Task list refreshes in real time across devices.
- UI passes basic accessibility checks (high contrast, label semantics).

6 Grocery and errand lists

Objective: Provide interfaces for creating and managing shared shopping lists and errand lists.
Steps: 1. Create a list overview screen displaying all lists (grocery, errands) with status (e.g., all items purchased or not).
2. Implement list creation and edit screens where users can add items with quantity and notes.
3. Allow users to mark items as purchased; update the state in Firestore.
4. Support assigning a list to a specific user or share it with the entire house.
5. Provide sorting and grouping of items (e.g., by aisle) as an optional enhancement.

Dependencies: Tasks 3–4.
Acceptance criteria: - Users can create, edit and delete lists.
- Item status updates propagate in real time.
- Offline functionality mirrors that of tasks.

7 Notifications and reminders

Objective: Keep users informed of task assignments, due dates and daily summaries.
Steps: 1. Integrate `firebase_messaging` and request the `POST_NOTIFICATIONS` permission on Android.
2. Implement push notifications for: (a) task assignment; (b) task due soon (e.g., 1 day before); (c) task updated or completed; (d) new list assigned.
3. Implement scheduled local notifications for daily summaries (e.g., at 9 am) using WorkManager (Android) and `flutter_local_notifications` (iOS). Avoid using `SCHEDULE_EXACT_ALARM` unless a precise alarm is essential.
4. Provide a settings screen where users can enable/disable categories of notifications and set quiet hours.

Dependencies: Tasks 3–5.
Acceptance criteria: - Push notifications arrive reliably when triggered.
- Local notifications fire at the scheduled times.
- Users can control notification preferences.

8 Supervisor dashboard

Objective: Give supervisors cross-house visibility and audit capabilities.
Steps: 1. Build a responsive web dashboard (Flutter Web or within the mobile app) accessible only to supervisors and admins.
2. Display aggregated statistics by house: count of tasks in each status, overdue tasks, tasks completed in the last week, number of active residents.

3. Implement filters by house, resident and date range.
4. Display audit logs with filtering by action type, user and date.
5. Implement CSV export of the current view; trigger a Cloud Function to generate a CSV and store it in Cloud Storage, returning a download link.

Dependencies: Tasks 4–7.
Acceptance criteria: - Supervisors can view cross-house data and logs.
- Exported CSV matches the filtered data set.
- Access control prevents residents from accessing the dashboard.

9 Accessibility compliance

Objective: Ensure the app meets or exceeds WCAG 2.2 AA and AODA requirements.
Steps: 1. Review all screens for contrast ratios and adjust colors to meet 4.5:1 for normal text and 3:1 for large text.
2. Provide responsive text scaling; allow dynamic type sizes.
3. Add semantic labels to all interactive elements and images for screen readers.
4. Ensure all functionality is available via keyboard navigation on web and accessible via focus order on mobile.
5. Provide captions or transcripts for any audio/video content (none in MVP).
6. Conduct manual testing with TalkBack (Android) and VoiceOver (iOS).

Dependencies: All UI tasks.
Acceptance criteria: - Automated accessibility checker (axe, Flutter's accessibility scanner) passes on all screens.
- Manual testing confirms screen reader usability and keyboard navigation.
- No text or controls are truncated at large font sizes.

10 Testing and quality assurance

Objective: Validate functionality, prevent regressions and ensure stability.
Steps: 1. Write unit tests for data models, business logic and helper functions.
2. Write widget tests for key screens (authentication, task list, list management).
3. Implement integration tests to simulate user flows (sign-in, create task, receive notification).
4. Use Firebase Test Lab or local emulators to test on multiple device sizes and OS versions.
5. Continuously run tests via GitHub Actions for pull requests.

Dependencies: Prior tasks.
Acceptance criteria: - Test coverage meets internal targets (e.g., >70% line coverage for core logic).
- All tests pass before deployment.
- Critical paths (auth, task creation, notifications) are covered by integration tests.

11 CI/CD and deployment

Objective: Automate builds and deploy the app to stores.
Steps: 1. Configure GitHub Actions to build Android APK/AAB and iOS archive on push to main.
2. Automate code formatting and analysis (dart format, dart analyze).

3. Integrate signing for Android and iOS (store keystore and certificates securely).
4. Use Fastlane or similar to upload builds to Google Play Console and App Store Connect.
5. Publish the app with accurate store listings and ensure that store metadata reflects accessibility and privacy compliance.
6. Set up release channels (alpha, beta, production) and assign testers.

Dependencies: All prior tasks.
Acceptance criteria: - Successful CI builds produce signed artifacts.
- Uploaded builds pass Play Console and App Store checks (target API 35, 64-bit, etc.).
- App is available for testers in the designated release channel.

12 Documentation and compliance

Objective: Document the system for maintainers and compliance purposes.
Steps: 1. Write developer documentation covering project setup, architecture and coding standards.
2. Draft a privacy policy describing data collection, use, and storage; reference PIPEDA and Law 25 compliance and indicate data residency in Canada.
3. Draft terms of service and an accessibility statement referencing WCAG 2.2 compliance.
4. Prepare a simple user guide for residents and supervisors.
5. Conduct a Privacy Impact Assessment and document risk mitigation strategies.

Dependencies: All prior tasks.
Acceptance criteria: - Documentation is published in the repository (e.g., `/docs` folder).
- Privacy policy and terms of service are ready for store submission.
- User guide is understandable by non-technical residents and supervisors.

13 Future enhancements (backlog)

After the MVP launch, consider these backlog tasks: - Integrate grocery APIs or store scanners to auto-populate lists. - Add natural language processing for task creation (e.g., “Remind me to take out the trash every Friday”). - Implement budgeting and expense tracking for each house. - Add social features like badges or gamification to encourage task completion. - Package as a Progressive Web App (PWA) for easier distribution.
