ChatGPT

Savitara Product Requirements Document (PRD)

Savitara is a dual-role digital platform connecting *Grihastas* (householders/seekers) with *Acharyas* (spiritual scholars). It uses a FastAPI/MongoDB backend and React Native frontends with Tailwind CSS for styling, leveraging Google SSO for authentication, Razorpay for payments, and Firebase for notifications [1]. The following sections detail comprehensive feature requirements, UI flows, system architecture, database schemas, APIs, infrastructure, and design guidelines for both the user-facing (savitara.com) and admin (admin.savitara.com) components. Citations are provided from existing Savitara design documents and specifications.

1. Feature-Level Requirements

1.1 User Flows (Grihasta/Householder)

- **Registration/Login:** Users must sign up or log in via Google Single Sign-On (SSO). On first login, users choose a role (Grihasta or Acharya) and accept terms/conditions. Google SSO is integrated using OAuth2, with the backend validating Google tokens and issuing JWTs for secure sessions [2] [3].
- **Profile Setup:** After login, Grihastas complete a profile with name, contact details, tradition/shaka, language preference, and optional profile photo. They may answer a short onboarding survey (e.g., spiritual interests) to personalize recommendations.
- **Search & Discovery:** Grihastas can browse Acharya profiles filtered by *Shaka* (Vedic branch), expertise, location, language, and ratings [4]. Each Acharya profile shows credentials, availability calendar (with Panchanga details), languages, fees, and reviews.
- **Booking Workflow:** Grihastas select an Acharya, view their availability calendar (dates/times with Panchanga context), and request a booking. The system checks for conflicts and prompts for service details (e.g., pooja type, rituals). Upon requesting, the Grihasta proceeds to payment via Razorpay, entering any coupon/referral code. Successful payment changes booking status to “confirmed” and notifies the Acharya [5]. Bookings appear on both user dashboards.
- **Chat & Communication:** After a booking is initiated or confirmed, Grihastas can chat one-on-one with the Acharya. The chat is real-time via WebSockets and stored in a `messages` collection [6]. Chats are ephemeral: all messages auto-delete 7 days after sending (via a TTL index or scheduled job), and in the app UI live chat screens disable screenshots/screen-recording (using platform secure views) to protect privacy.
- **Notifications:** Users receive push notifications (via Firebase) and in-app alerts for booking status changes, messages, upcoming sessions, and announcements. A notification center lists recent alerts, with read/unread status.
- **Ratings & Reviews:** After a completed session, Grihastas can rate the Acharya (1–5 stars) and leave a review. Reviews are visible on the Acharya profile only after admin moderation (to prevent abuse). The average rating contributes to Acharya search ranking.
- **Coupons & Referrals:** Grihastas can enter coupon or referral codes on the payment page. The system validates codes, applies discounts, and tracks usage. A referral program allows users to invite

friends (assigning referral IDs to new sign-ups). Admins manage coupon code generation and referral rewards.

- **Account Management:** Grihastas can update their profile, change settings, view booking history, and manage notifications. They also have a dashboard summarizing upcoming events, recent messages, and any pending actions (e.g., incomplete profile, pending reviews).

1.2 User Flows (Acharya/Scholar)

- **Registration/Login:** Acharyas sign up via Google SSO and choose “Acharya” as their role, then accept terms. They complete a detailed profile with fields for Shaka, specializations (e.g., rituals, discourses), region, languages, years of experience, and consultation types (online/offline) [4]. They upload verification documents (certificates, KYC IDs, photo) which enter an admin review queue.
- **Profile Verification:** New Acharya profiles start in **Pending Verification**. Admins review uploaded documents and either **Approve** or **Reject** the profile. Approved profiles are marked “Verified” and become searchable; rejected ones are flagged with feedback to the user.
- **Availability & Calendar:** Verified Acharyas set their available slots in a calendar interface (date/time ranges). The calendar displays both booked and free slots [7]. Acharyas can update availability, block dates, and define session durations or special events. The UI incorporates *Panchanga* data (tithi, nakshatra) to highlight auspicious days.
- **Booking Management:** Acharyas see incoming booking requests (with details: requester, date, service) on their dashboard. They can **Accept** or **Decline** requests. Upon acceptance, the booking is confirmed (both parties are notified and the slot is reserved); if declined, the requester is notified to choose another slot.
- **Chat & Communication:** Acharyas can chat with Grihastas involved in confirmed or pending bookings. The chat interface is identical to the Grihasta view, subject to the same 7-day auto-delete policy for messages.
- **Dashboard & Analytics:** The Acharya dashboard (web/mobile) shows upcoming sessions on a calendar, recent messages, and key metrics (e.g., total sessions, earnings). It aggregates data such as average ratings, answer response times, and earnings summaries.
- **Settings:** Acharyas can edit profile details, adjust notification preferences, view payment history (earnings, Razorpay payouts), and withdraw funds if applicable. If an Acharya’s profile is ever locked by admin (e.g., policy violation), they see a notice and cannot receive new bookings until reinstated.

1.3 Admin Workflows

- **Login & Access:** Admins access the platform via a separate subdomain (admin.savitara.com). They authenticate with Google SSO (with admin accounts) and see an admin dashboard after login.
- **Acharya Verification:** A priority workflow is verifying new Acharya profiles. Admins view a list of pending applications (with documents). They inspect details and click **Approve** or **Reject**. Upon approval, the Acharya’s status is updated to Verified. Rejections send an email and in-app notification with reasons.
- **User Management:** Admins can search all users (Grihastas/Acharyas), view profiles, disable suspicious accounts, and manage user roles. They can reset a user’s status or delete fraudulent profiles.
- **Content Management:** Admins manage static content and announcements. This includes adding/editing FAQ pages, terms of service, privacy policy, blog posts, and spiritual articles. Announcements (e.g. festival greetings, system updates) can be crafted and broadcast via Firebase (push notification)

or email. An announcement center allows scheduling and targeting (e.g., all users or a role-specific audience).
- **Analytics & Reporting:** The admin dashboard provides real-time analytics: user sign-ups, active bookings, revenue (via Razorpay data), popular Acharyas, and system health metrics. Charts display trends (daily/weekly active users, booking volumes). Admins can filter by time range or category (e.g., bookings by region).
- **Reviews Moderation:** Admins review flagged or new reviews. They can delete inappropriate reviews, or override star ratings if needed. A managed review queue ensures public feedback stays constructive.
- **Coupons/Referrals Management:** Admins create and manage coupon codes (set discount rates, expiry dates, usage limits) and referral campaigns. A panel shows usage stats for each code and referrals.
- **Notification Center:** Admins can draft and send targeted in-app messages or push notifications (e.g., maintenance alerts or invites to surveys). They can also view logs of sent notifications and user engagement.
- **System Settings:** Admins configure global settings: parameters like booking cancellation policy, rating weightage, and API keys (centralized via secure config). They can set rate limits (e.g., max bookings per user per week) and toggle features (like enabling/disabling the AI assistant).

1.4 Security & Special Requirements

- **Secure Chat:** The chat interface between users must be protected. On Android, use `FLAG_SECURE` to disable screenshots; on iOS, detect screenshot events and blur content. All chat data is ephemeral: messages auto-delete 7 days after sending via a MongoDB TTL index.
- **Data Privacy:** Sensitive information (phone numbers, emails) is only shared with consent. All personal data is stored securely, and communication is encrypted in transit (HTTPS everywhere) [8].
- **Rate Limiting:** Implement API rate limits per user/IP (e.g., 100 requests/minute) to prevent abuse.
- **Authentication:** Use JWT tokens in `Authorization: Bearer <token>` headers. Tokens include role claims and expire periodically. Middleware verifies token validity and enforces role-based access (e.g., only Acharyas use `/acharya/*` endpoints) [2] [9].
- **Input Validation:** All inputs (API and form data) use Pydantic/Frontend validation. Rich error messages guide users (e.g., invalid coupon code, unavailable slot).
- **Logging & Auditing:** The system logs admin actions (e.g., profile approval), authentication events, and payment transactions for audit purposes. Logs are aggregated in a secure store for monitoring and debugging.

2. Screen-by-Screen UI Flow

(References to Figma design where available; otherwise descriptions of screens and navigation.)

2.1 Grihasta Mobile App Flow

1. **Splash & Onboarding:** A splash screen with Savitara logo, followed by a welcome carousel explaining features (optional for first-time users).
2. **Login Screen:** A “Continue with Google” button (using `expo-auth-session`) triggers Google SSO [2]. After login, a **Role Selection** screen asks “Are you a Householder (Grihasta) or Scholar (Acharya)?” (only on first use; stored for future).

3. **Terms & Conditions:** The user must scroll and accept terms of service and privacy policy (checkbox) to proceed.
4. **Profile Setup (Seeker):** A series of screens (or a form) collect basic details: name, contact (auto from Google if available, editable), tradition (Shaka), preferred language, and a short bio or interests. Optionally upload a profile picture.
5. **Home/Dashboard:** The main Grihasta dashboard shows:
6. **Search Bar & Filters:** By name, tradition, expertise, location.
7. **Featured Acharyas:** Carousel or grid of top-rated Acharyas.
8. **Categories/Services:** Quick access icons (e.g., “Online Pooja”, “Astrology Consultation”).
9. **Notifications Icon:** Badge for unread alerts.
10. **Profile Icon:** Access account settings.
11. **Acharya Profile Page:** Displays Acharya’s photo, name, verified badge, rating, Shaka, languages, expertise list, bio, and calendar snippet. A **Book Now** button leads to the booking flow.
12. **Booking Screen:**
13. **Calendar View:** Shows Acharya’s available dates. Each date cell can show Panchanga info (e.g., tithi, festival).
14. **Time Slot Selection:** After date selection, shows available time slots.
15. **Session Details Form:** Choose service type from a dropdown (e.g., `Satyanarayana Pooja`) and enter any notes.
16. **Coupon/Referral Entry:** Input field for codes.
17. **Payment Summary:** Shows fee breakdown.
18. **Payment Processing:** Redirects to Razorpay checkout. On success, shows confirmation with booking details; on failure, user can retry.
19. **Post-Booking:** A confirmation screen and notification. Booking appears under **My Bookings**.
20. **Chat Screen:** List of conversations (sorted by recent message). Opening a conversation shows a message thread. The input box sends messages in real-time. (Live chat respects the 7-day TTL.)
21. **Bookings List:** A tab or section lists upcoming and past bookings. Upcoming booking cards include date/time, service, Acharya name, and status. Tapping leads to details (with “Start Chat” or “Cancel” options if allowed).
22. **Notifications Center:** Lists recent notifications (booking updates, admin announcements, system reminders). Each can be tapped to view related content (e.g., opening a chat or booking).
23. **Profile/Settings:** Shows user info, editable profile fields, logout button, and a toggle for push notifications.

2.2 Acharya Mobile/Web Flow

1. **Login & Verification Prompt:** Similar login as Grihasta. After signup and role selection, new Acharyas see a **Verification** screen prompting document uploads (photo ID, certificates). Before verification, the profile is not public.
2. **Profile Setup (Scholar):** Form for Acharya details: full name, contact, Shaka, languages, expertise tags, rates, description. Upload photos/certificates here if not already done.
3. **Pending Verification Screen:** After submission, a status indicator (e.g., “Awaiting Approval”) is shown. No booking search functions until approved.
4. **Home/Dashboard:** Once verified, the Acharya dashboard shows:
5. **Calendar Overview:** Highlights upcoming appointments and open slots. Possibly integrated with Panchanga (a “today” card with current tithi).
6. **Pending Requests:** List of recent booking requests with “Approve”/“Decline” buttons.
7. **Statistics:** Quick stats (e.g., this week’s earnings, number of sessions).

8. **Chat:** Icon or list of active conversations with seekers.
9. **Notifications:** Badge for unread messages or booking changes.
10. **Availability Management:** A screen where Acharya can add/edit availability. A calendar UI (monthly or weekly) lets them select dates/times, with fields for session duration. They can mark special event dates (e.g., workshop).
11. **Booking Details Page:** For each booking (pending or confirmed), a detail view shows requester info, service, date/time, payment status, and a "Message" button.
12. **Chat Screen:** As above, with the same look/feel as the Grihasta side (subject to screenshot policy). Conversation threads are separate per seeker.
13. **Earnings & Payments:** Screen showing booking fees collected. This may integrate with Razorpay payouts or external payment method. Includes history of deposits.
14. **Notifications Center:** Lists system messages (e.g., admin announcements, payment confirmations).
15. **Profile/Settings:** The Acharya can edit their information, change password (if applicable), and view verification status. Option to deactivate profile if needed.

2.3 Admin Web Portal Flow

1. **Admin Login:** Google SSO for admin accounts, leading to a multi-panel dashboard.
2. **Admin Dashboard:** A landing page with widgets (user count, new registrations, bookings per day, revenue charts). Quick links to pending tasks (e.g., "5 Acharyas Awaiting Verification").
3. **Acharya Management:** A list/table view of all Acharyas with filters (by status, shaka, verified/unverified). Clicking an Acharya opens their profile details and documents, with buttons to **Approve/Reject**.
4. **User Management:** Lists all users. Admin can search by name/email. Each user detail page shows role, contact info, status, and actions (e.g., disable account).
5. **Booking Management:** Admin can view all bookings and filter by status or date. Clicking a booking shows full details. While admins cannot book on behalf of users, they can intervene (e.g., cancel, add notes).
6. **Analytics:** Pages with graphs and filters for metrics. For example, a "Bookings" page showing daily/weekly counts, heatmaps, or a "Revenue" page with charts. Data export (CSV) is available.
7. **Announcements:** A form to compose notifications. Options: target role (all/Grihasta/Acharya), title, message, schedule time. Sent announcements appear in a log list.
8. **Coupon Management:** Interface to create new coupon codes (define discount %, expiry date, usage limit) and list existing ones with edit/delete.
9. **Settings/Configuration:** A settings page for site-wide options (support email, contact info, cancellation policy text, etc.) and security keys (Google OAuth credentials, Razorpay keys, Firebase config – managed securely).
10. **Logout:** Available in header or profile menu.

All UI must be clean, responsive (mobile-first for React Native, and admin web responsive for various screens). Clarity, consistent spacing, and intuitive navigation are mandatory.

3. System Design & Architecture

- **Overall Architecture:** A modular, service-oriented architecture. The core backend is FastAPI (Python) with MongoDB. We use separate frontends: a React Native app for savitara.com (Grihasta/Acharya) and a React (or React Native web) admin portal for admin.savitara.com. All services are containerized (Docker) for easy deployment.

- **Repository Structure:**

```
/savitara-app
  /frontend
    /mobile (React Native code for users)
  /backend
    /api (FastAPI for user services)
    /services (optional microservices: chat, AI, payments, notifications)
  /common
    /schemas (Pydantic models, DB models)
    /middlewares
    /utils
    /config (env vars, API keys)

/savitara-admin
  /frontend (React Admin dashboard)
  /backend (FastAPI admin-specific routes)
```

  Each FastAPI application separates routers (e.g., auth, bookings, chat) and uses `middlewares` for JWT validation and rate limiting.
- **API Gateway:** All external traffic passes through an API Gateway or proxy (e.g., AWS API Gateway, Kong, or NGINX) for SSL termination, load balancing, and rate limiting. The gateway enforces security policies, forwards authentication headers, and routes to microservices. Parameterized API keys (set in environment/config) manage access to third-party services.
- **Authentication & Authorization:** Google OAuth2 for login (frontends use `expo-auth-session` to get Google ID token) [2] [3] . FastAPI backend verifies tokens with Google libraries, creates or updates a `users` record, then issues its own JWT. Role-based access control is implemented via JWT claims. Protected endpoints check the JWT and role before allowing actions [9] .
- **Security:**
- **HTTPS Enforcement:** All services run under HTTPS (via certificates or managed TLS).
- **JWT Expiry/Refresh:** Tokens have a short expiration (e.g., 1 hour) with a refresh endpoint to renew them.
- **Rate Limiting:** Middleware (e.g., via Redis) limits requests per IP or user.
- **Input Validation:** FastAPI/Pydantic validate every request body.
- **RBAC:** Users have roles (`Grihasta`, `Acharya`, `Admin`) controlling access. For example, only Acharya-role tokens can access `/acharya/*` endpoints.
- **Third-Party Integrations:**
- **Razorpay:** Payment processing via Razorpay SDK. After payment, Razorpay sends a webhook to FastAPI which verifies and updates booking/payment status in MongoDB.
- **Firebase:** Firebase Cloud Messaging (FCM) for push notifications. The backend triggers FCM events on booking updates, messages (via chat service), or announcements.
- **Google Services:** Google Cloud for OAuth credentials and (optional) analytics/events tracking.
- **Parameter Management:** Sensitive keys (Firebase server key, Razorpay key, Google client IDs) are stored in a centralized secrets store or environment variables. Configuration is parameterized (not hard-coded).

- **Performance:** Use of API caching (e.g., Redis caching for frequent reads), MongoDB indexes (see below), and aggregation pipelines to minimize round-trip calls. For example, Acharya dashboard APIs aggregate booking, message, and profile data into a single payload for faster loading.
- **Scalability:** The services are stateless; scaling occurs by adding more containers. Database and cache are separately scaled. Optionally, move to a microservices architecture: e.g., separate FastAPI instances for chat (with WebSockets), AI assistant, booking, etc.
- **Logging/Monitoring:** All services log to a centralized system (e.g., ELK stack or CloudWatch). Key metrics (API response times, error rates) are tracked. Admins can access logs via secure tools.

4. MongoDB Schema Design

All data is stored in MongoDB with the following primary collections (example fields):

- **users:**

```
{
  "_id": ObjectId,
  "google_id": String,        // Google SSO ID
  "email": String,            // unique
  "name": String,
  "role": "Grihasta"|"Acharya"|"Admin",
  "phone": String,
  "photo_url": String,
  "terms_accepted": Boolean,
  "created_at": Date,
  "status": "Active"|"Banned",
  "verified": Boolean        // for Acharyas
}
```

- *Indexes:* unique on `email`, index on `role`.
- Acharya-specific info is stored in a separate `acharya_profiles` collection (below); `users.role` indicates the type.

- **acharya_profiles:**

```
{
  "_id": ObjectId,
  "user_id": ObjectId,         // reference to users._id
  "shaka": String,             // spiritual branch
  "expertise": [String],       // list of services (e.g., "Puja",
"Astrology")
  "languages": [String],
  "location": { city, state, country },
  "experience_years": Number,
  "bio": String,
```

```
  "fees": Number,              // default consultation fee
  "availability": [           // embedded document for slot times
    { "date": Date, "slots": [ {"from": Time, "to": Time} ] }
  ],
  "rating_avg": Number,
  "rating_count": Number,
  "verification_docs": [ String ], // URLs to uploaded docs
  "profile_picture": String,
  "created_at": Date,
  "updated_at": Date
}
```

- *Indexes:* compound index on `(shaka, location.city, languages)` for search; index on `user_id`.
- Availability could be embedded or managed via a separate `availability` collection if very dynamic.

- **seeker_profiles (optional):**
  (If extra info needed beyond `users`.)

```
{
  "_id": ObjectId,
  "user_id": ObjectId,
  "preferences": { "tradition": String, ... },
  "referred_by": ObjectId,    // reference to another user for referrals
  "created_at": Date,
  "updated_at": Date
}
```

- **bookings:**

```
{
  "_id": ObjectId,
  "acharya_id": ObjectId,    // refers to users._id or acharya_profiles
  "grihasta_id": ObjectId,   // refers to users._id
  "service": String,         // e.g., "Satyanarayana Puja"
  "scheduled_date": Date,
  "scheduled_time": String,  // e.g., "15:00"
  "duration": Number,        // in minutes
  "status": "Pending"|"Confirmed"|"Cancelled"|"Completed",
  "payment_status": "Pending"|"Paid"|"Refunded",
  "amount": Number,
  "coupon_code": String,
```

```
  "created_at": Date,
  "updated_at": Date
}
```

• *Indexes:* `(acharya_id, status)`, `(grihasta_id, status)`, and `scheduled_date`.

• Additional fields (like Zoom link if virtual) can be added in `status="Confirmed"` stage.

• **conversations:**

```
{
  "_id": ObjectId,
  "participants": [ObjectId, ObjectId], // two user IDs
  "last_message": String,
  "last_updated": Date,
  "created_at": Date
}
```

• *Indexes:* index on `participants` array (multikey index).

• **messages:**

```
{
  "_id": ObjectId,
  "conversation_id": ObjectId,
  "sender_id": ObjectId,
  "content": String,
  "sent_at": Date
}
```

• *Indexes:* index on `conversation_id`, index TTL on `sent_at` with expiry after 7 days.

• We use a TTL index on `sent_at` to auto-delete messages after 7 days to enforce ephemeral chat.

• **notifications:**

```
{
  "_id": ObjectId,
  "user_id": ObjectId,
  "type": String,       // e.g., "BookingUpdate", "Message",
"Announcement"
  "title": String,
  "message": String,
```

```
  "link": String,       // optional deep link or ID reference
  "read": Boolean,
  "created_at": Date
}
```

• *Indexes:* compound `(user_id, read)`, `created_at` for sorting.

• **reviews:**

```
{
  "_id": ObjectId,
  "booking_id": ObjectId,
  "acharya_id": ObjectId,
  "grihasta_id": ObjectId,
  "rating": Number,     // 1-5
  "comment": String,
  "admin_approved": Boolean,
  "created_at": Date
}
```

• *Indexes:* `acharya_id` for retrieving all reviews; maybe text index on `comment` for moderation search.

• **ai_interactions:**

```
{
  "_id": ObjectId,
  "user_id": ObjectId,
  "prompt": String,
  "response": String,
  "model": String,        // e.g., "IndicBERT"
  "created_at": Date
}
```

• *Indexes:* index on `user_id` if retrieving history; possibly capped or TTL if storage concern.

• **coupons:**

```
{
  "_id": ObjectId,
  "code": String,         // unique
  "discount_percent": Number,
```

```
  "expiry_date": Date,
  "usage_limit": Number,
  "used_count": Number,
  "created_at": Date
}
```

• *Indexes:* unique on `code`.

• **referrals:**

```
{
  "_id": ObjectId,
  "referrer_id": ObjectId,
  "referred_email": String,
  "referred_user_id": ObjectId,
  "created_at": Date,
  "status": "Pending"|"Registered"
}
```

• Tracks which referrals converted to sign-ups.

• **announcements:** (admin created)

```
{
  "_id": ObjectId,
  "title": String,
  "message": String,
  "target_roles": [String], // e.g., ["Acharya", "Grihasta"]
  "scheduled_at": Date,
  "sent": Boolean,
  "created_at": Date
}
```

• Admins can schedule announcements; sent triggers FCM.

Indexing & Aggregation Strategy:
Frequent queries (e.g., login by email, calendar queries) use indexes. For example, the **users.email** field is indexed for fast lookup [10]. Booking queries use compound indexes on `(acharya_id, scheduled_date)`. Aggregation pipelines combine related data: e.g., the Acharya dashboard API joins `bookings`, `acharya_profiles`, and `conversations` to produce a single payload [10]. MongoDB's flexible schema allows embedding some denormalized info (e.g., last message in conversation) to speed reads.

5. API Documentation

Below are the main REST endpoints (all input/output in JSON). **Authentication:** Except where noted, requests require `Authorization: Bearer <JWT>`. Admin routes require an admin JWT. Sample payloads are illustrative.

- **Auth APIs:**
- `POST /auth/google` – (Grihasta/Acharya) Input: `{ "id_token": "<Google ID token>" }`. Backend verifies with Google, creates/updates user, and returns `{ "token": "<JWT>", "user": {id, name, email, role, photo_url} }`.
- `GET /auth/profile` – (Auth) Returns the current user's profile. Response: `{ "id", "name", "email", "role", "photo_url", "terms_accepted", ... }`.
- `POST /auth/refresh` – (Auth) Refreshes JWT. Input: `{ "refresh_token": "<token>" }`. Returns new JWT.

- **User Profile APIs:**

- `GET /users/me` – (User) Returns the logged-in user's data.
- `PUT /users/me` – (User) Update profile fields (name, photo, phone).
- `POST /users/me/terms` – (User) Accept terms: no body. Sets `terms_accepted = true`.
- `GET /seekers/{id}` – (User/Admin) Get a seeker's profile (by ID).
- `GET /acharyas` – (Grihasta) List Acharyas (supports query params for filtering by shaka, expertise, location). Response: list of Acharya profile summaries.
- `GET /acharyas/{id}` – (User) Get full Acharya profile details.
- `PUT /acharyas/{id}` – (Acharya) Update own Acharya profile (expertise, bio, fees, etc.).
- `POST /acharyas/{id}/verify` – (Acharya) Upload or resubmit verification docs.

- **Booking APIs:**

- `POST /bookings` – (Grihasta) Create a booking request. Body example: `{ "acharya_id":"...", "date":"2025-12-01", "time":"09:00", "service":"Satyanarayana Puja", "coupon":"CODE123" }`. Returns booking details with status "Pending".
- `GET /bookings/{id}` – (User) Get booking by ID (only participants or admin).
- `GET /bookings/user/{user_id}` – (Auth) Get all bookings for a user (param `status` optional to filter by pending/confirmed).
- `PATCH /bookings/{id}` – (Acharya) Update a booking status (approve/reject/cancel). Body: `{ "action":"confirm"|"reject"|"cancel" }`. Returns updated booking.
- `POST /bookings/availability` – (Acharya) Add availability slots. Body: `{ "date":"2025-12-01", "slots":[ {"from":"08:00","to":"12:00"}, ... ] }`.
- **Webhooks:** `POST /payments/razorpay-webhook` – Razorpay calls this with payment info. Backend verifies and updates the corresponding booking's `payment_status` and notifies users.

- **Calendar APIs:**

- `GET /calendar/availability?acharya_id=...&month=YYYY-MM` – (Grihasta) Returns all available dates and times for the given Acharya in that month, including Panchanga metadata if needed.
- `GET /calendar/my` – (Auth) Returns upcoming booked dates for the user (for quick display on dashboard calendar).

- **Chat & Messaging APIs:**

- `GET /conversations` – (Auth) List all conversations for current user with metadata (last_message, timestamp).
- `POST /conversations` – (Auth) Create a conversation (if not auto-created). Body: `{ "participant_id": "<other_user_id>" }`.
- `GET /conversations/{conv_id}/messages` – (Auth) Get all messages in a conversation (sorted by time).
- `POST /conversations/{conv_id}/messages` – (Auth) Send a message. Body: `{ "content": "Hello" }`. This broadcasts via WebSocket to the other user and stores the message.
(Note: Chats use WebSockets for real-time delivery but also support REST for history sync.)

- **Notification APIs:**

- `GET /notifications` – (Auth) List the user's notifications (optionally filter unread).
- `PATCH /notifications/{id}` – (Auth) Mark a notification as read.

- **Review APIs:**

- `POST /reviews` – (Grihasta) Submit a review. Body: `{ "booking_id": "...", "rating": 5, "comment": "Great guidance!" }`. Initially `admin_approved=false`.
- `GET /acharyas/{id}/reviews` – (User) List approved reviews for an Acharya (only returns those `admin_approved=true`).

- **Coupon/Referral APIs:**

- `POST /coupons/validate` – (Grihasta) Validate a coupon code. Body: `{ "code": "XYZ" }`. Returns discount details if valid.
- `POST /referrals` – (Grihasta) Claim a referral (on signup) with a referrer's code.

- **Admin APIs:** (Require Admin JWT)

- `GET /admin/acharyas/pending` – List Acharyas awaiting verification.
- `PATCH /admin/acharyas/{id}` – Approve or reject Acharya. Body: `{ "action": "approve"|"reject", "notes": "..." }`.

- `GET /admin/users` – List all users (with filters).
- `GET /admin/bookings` – List all bookings.
- `GET /admin/analytics` – Get stats (parameters: metric type, date range).
- `POST /admin/announcements` – Create/send announcement. Body: `{ "title": "...", "message": "...", "target_roles": ["Grihasta"], "scheduled_at": "..." }`.
- `POST /admin/coupons` – Create a new coupon. Body: `{ "code":"", "discount_percent": 10, "expiry_date":"2025-12-31", "usage_limit":100 }`.
- `GET /admin/logs` – (Optional) Fetch system logs.

Sample requests and responses should be included in developer docs (not fully detailed here). All responses include status codes (200, 201 for created, 400 for bad request, 401 unauthorized, etc.) and error messages for invalid actions.

6. Infrastructure Planning

- **Environment:** All services run in Docker containers (Docker Compose or Kubernetes). Separate containers for the FastAPI app, the MongoDB database, and any microservices (e.g., a WebSocket/chat service). The React Native app is deployed via app stores; the admin web app is hosted on a CDN or cloud (behind Nginx or similar).
- **Database:** MongoDB is hosted on a managed cluster (for high availability). Backups run daily. For chat caching and rate limiting, use Redis (e.g., AWS Elasticache). As noted, Redis is used for caching sessions and chat context [11].
- **Autoscaling:** Use a load balancer (e.g., AWS ELB) in front of the backend. Multiple FastAPI replicas handle requests. Horizontal scaling triggers on CPU or request load. Static assets (frontend builds) are served via CDN.
- **Third-Party Services:**
- **Firebase:** Used for push notifications; no extra backend needed. Firebase Cloud Messaging (FCM) keys are configured, and the backend calls FCM APIs when needed.
- **Razorpay:** Integrated via their official SDK. A secure webhook endpoint handles payment callbacks and verifies signatures.
- **Google OAuth:** OAuth credentials (client ID/secret) are stored securely (not in code). The FastAPI backend uses Google's OAuth2 library to verify tokens. Google also provides analytics/tracking (e.g., Google Analytics for app usage).
- **Logging & Monitoring:** Use a centralized logging service (e.g., ELK or Datadog). Application logs include INFO events (user login, booking created) and ERROR traces. Setup alerts for critical issues (e.g., payment failures, high error rates). Admins can view basic logs via the admin portal.
- **CI/CD:** Code is version-controlled (Git). Automated pipelines build Docker images and run tests. Upon merge to main, code deploys to staging; after approval, promotes to production. Environment variables differentiate between dev/staging/prod.
- **Data Privacy Compliance:** Ensure infrastructure complies with data protection (e.g., GDPR, if applicable). Use appropriate firewall rules and security groups.
- **Disaster Recovery:** Have failover replicas for MongoDB, and plan for quick restore from backups. Keep infrastructure-as-code for rapid redeployment (Terraform/CloudFormation).
- **Analytics Tracking:** In addition to admin analytics, integrate event tracking (e.g., via Google Analytics or a custom solution) for user actions: log events like `booking_requested`, `login`, `payment_success`, etc., to analyze user behavior and funnel conversion.

- **Continuous Monitoring:** Use uptime monitoring tools (e.g., Pingdom) on key endpoints (auth, booking) to alert if the system is down.

7. Design Language & Branding

- **Color Theme:** A saffron/kesari-orange palette (Shakti-inspired) with accent golds and neutral whites. Example primary color: `#F57C00` (saffron) with a complementary deep `#BF360C` (dark orange). Use shades for hierarchy: lighter saffron for highlights, dark for buttons/headers. All buttons and links use the saffron accent to reinforce brand identity.
- **Typography:** A harmonious blend of Devanagari-inspired and modern English fonts. For headings, a stylized serif or calligraphic font that evokes Sanskrit calligraphy (e.g., *Mukta Malar*, *Purusheela*). For body text, a clean sans-serif (e.g., *Mukta Mahee* or *Noto Sans Devanagari*) for readability. This creates a Sanskrit-English fusion look. Letter spacing and line height ensure clarity.
- **UI Style:** Use Tailwind CSS for rapid, consistent styling. The UI is spacious with clear hierarchy: generous padding around cards, consistent margin between elements, and bold color accents for interactive elements. Icons (e.g., lotus, book) with a line-art style match the spiritual theme.
- **Imagery & Icons:** Use subtle religious/spiritual motifs (temple silhouettes, simple lotus). Illustrations (if any) should be flat and minimalistic in the saffron palette. Avoid excessive imagery to keep the UI clean.
- **Responsiveness:** All layouts must adapt to different screen sizes. The mobile app (React Native) is inherently responsive, but ensure components scale (e.g., grid of Acharya cards). The admin web portal uses a responsive grid: side menus collapse on smaller widths, tables become scrollable.
- **Modern UX:** Navigation is simple (tab bars for user app; side nav or top nav for admin). Feedback is immediate: show spinners on load, disable buttons on submission, and display success/error toasts. Use familiar UX patterns (pull-to-refresh on lists, swipe-to-delete notifications).
- **Accessibility:** High contrast text on backgrounds (e.g., white text on saffron headers), support for screen readers (accessible labels on icons), and proper touch target sizes (>=44px). Localization-ready for potential multi-language support (e.g., Hindi, Sanskrit, English).
- **Consistency:** All buttons, inputs, and cards use a unified border-radius and shadow style. Fonts and colors are defined in a global theme file. Interactive elements have a consistent hover/active state (e.g., darker shade on press).
- **Brand Elements:** Incorporate the Savitara logo (likely a symbolic icon) at key locations (splash screen, admin header). Ensure the saffron theme ties to spiritual significance (e.g., saffron robes of monks). The design should feel warm and trustworthy, aligning with both modern app standards and cultural aesthetics.

Color Palette Example (RGB/Hex): Saffron: `#FF9800` / `rgb(255, 152, 0)`; Deep Orange: `#F57C00`; Gold accent: `#FFC107`; Neutral backgrounds: off-white `#FFF8E1`, light gray `#FAFAFA`.

In sum, Savitara's UI combines a clean, intuitive interface with culturally resonant design choices. As per the reference design guidance, elements like the calendar or chat should reflect a modern look, while the overall theme remains spiritually inspired.

Sources: Design decisions (fonts, colors) draw on the uploaded style reference and Tailwind defaults. Core feature and architecture requirements are based on Savitara's system overview and planning documents [1] [12], ensuring alignment with existing plans.

---

1 2 3 4 5 6 7 8 9 10 12 cd03fcdc-042d-4d06-a29e-4c66ba789f9a.pdf

file://file_00000000613c720bb889bf19daaf25ec

11 399049e2-af8d-49b5-a6f7-09da7c88a584.docx

file://file_000000003118720b9bfa47defd6c38cc