TravelSync — Project SUMMARY (Start → End)

Purpose: Smart tourism platform — discover, plan, book eco-friendly trips, connect with locals.

1) High-level Overview

- Product: TravelSync (web). Features: Explore destinations, AI trip planner (5-step), bookings & payments, user dashboard, blogs, contact.

2) Frontend Pages (final set)

- Home
- Sign In
- Sign Up
- Explore Destinations
- Destination Details
- Trip Planner (multi-step: Step1..Step5)
- Bookings
- About Us
- Contact Us
- Blog / Travel Tips
- Blog Detail
- User Dashboard (My Bookings, Saved Destinations, Trip History, Profile Settings, Payment Methods)
- Payment Page
- Booking Confirmation / Success

3) UI & Design Summary (short)

- Fonts: Poppins / Inter. - Primary color: #3C82F6; accents: #22C55E, #F97316. - Cards: rounded-2xl, soft shadow, micro-interactions (hover, scale). - Navbar & universal Footer used across site.

4) Frontend Components (optimized, reusable)

- Navbar, Footer
- Card (generic)
- Button (primary/secondary)
- Input, DatePicker, Select
- Modal, Toast, Loader
- HeroSection (search)
- DestinationCard, DestinationGrid
- TripPlanner components (StepCard, ProgressStepper, SummaryCard)
- BookingCard, BookingForm
- AuthForm (signin/signup)
- DashboardSidebar, ProfileForm, SavedDestinationCard, TripHistoryTimeline
- BlogCard, BlogDetail

5) Frontend folder structure (recommendation)

- client/ - public/ - src/ - assets/ - components/ - common/ - home/ - planner/
- booking/ - pages/ - services/ (api calls) - context/ (auth, planner state) - utils/ -
App.jsx, main.jsx

6) Backend — Mongoose Schemas (final)

- User:

• name, email (unique), passwordHash, phone, country, dob, avatarUrl

• preferences: {emailNotifications, tripReminders, ecoFriendlySuggestions, language}

• savedDestinations: [Destination._id], tripHistory: [Trip._id], bookings: [Booking._id]

• paymentMethods: [{cardType, last4, expiry, billingAddress}]

• role (user|admin), createdAt, updatedAt

- Destination:

• name, location, description, images[], tags[], type, ecoFriendly (bool)

• rating, experiences[], highlights[], travelTips[], bestTimeToVisit, mapEmbedUrl

• reviews: [{user, rating, comment, createdAt}]

- Trip:

• user, destination, startDate, endDate, travelers, selectedHotel (object or id), selectedActivities[], itinerary[]

• notes, estimatedCost, createdAt, updatedAt

- Booking:

• user, trip, amount, currency, paymentStatus (Pending|Paid|Failed|Refunded), paymentMethod, bookingDate,
paymentResponse

• createdAt, updatedAt

- Blog:

• title, content (rich text), author, image, tags[], comments: [{user, comment, createdAt}], createdAt

- ContactMessage (optional):

• name, email, subject, message, createdAt

- PaymentRecord (optional):

• bookingId, provider (Stripe/Razorpay), providerResponse, amount, status, txnId, createdAt

7) API Routes (concise)

- Auth:

POST /api/auth/register — register user

POST /api/auth/login — login -> returns access + refresh token

POST /api/auth/logout — revoke refresh

GET /api/user/profile — get profile (auth)

PUT /api/user/profile — update profile (auth)

PUT /api/auth/password — change password (auth)

- Destinations:

GET /api/destinations — list (filters, pagination)

GET /api/destinations/:id — details + reviews

POST /api/destinations — create (admin)

PUT /api/destinations/:id — update (admin)

DELETE /api/destinations/:id — delete (admin)

POST /api/destinations/:id/review — submit review (auth)

- Saved Destinations:

GET /api/user/saved-destinations — list user saved

POST /api/user/save-destination/:id — save (auth)

DELETE /api/user/remove-destination/:id — remove (auth)

- Trip Planner / Trips:

POST /api/trip/plan — create/update trip (auth)

GET /api/user/trips — list user's trips

GET /api/trip/:id — trip detail

PUT /api/trip/:id — update trip

DELETE /api/trip/:id — delete trip

- Bookings & Payments:

POST /api/bookings — create booking + init payment (auth)

GET /api/user/bookings — list bookings (auth)

GET /api/booking/:id — booking detail

PUT /api/booking/:id/status — update status (webhook/admin)

POST /api/payments/webhook — payment provider webhook (secure)

- Blogs:

GET /api/blogs — list

GET /api/blogs/:id — details

POST /api/blogs — create (admin)

PUT /api/blogs/:id — update (admin)

DELETE /api/blogs/:id — delete (admin)

- Contact:

POST /api/contact — store message or send email

8) Backend folder structure (recommended)

- server/ - controllers/ (auth, destinations, trips, bookings, blogs, contact) - models/ (mongoose schemas) - routes/ (register routers) - middleware/ (auth, role-check, error, validate) - services/ (payment, email, image upload) - utils/ (helpers) - config/ (db, env) - server.js / app.js

9) Auth & Security (must-have)

- Use JWT access token (short), refresh token (httpOnly cookie). - Passwords hashed with bcrypt (saltRounds >=
10). - Validate inputs (Joi/express-validator). Use Helmet, CORS, rate-limit. - Protect webhooks with signature verification and use HTTPS. - Store secrets in environment variables, not code.

10) Media & Storage

- Use S3 (recommended) or Cloudinary for images. Store URLs in DB. - Use multer on backend and upload to S3;
keep thumbnails and optimised sizes.

11) Payment & Booking flow (step-by-step)

1. Frontend: user clicks 'Confirm & Book' → POST /api/bookings with tripId & user info.

2. Backend: create Booking record with paymentStatus='Pending', compute amount, call payment provider (Stripe/Razorpay) to create payment session/order. Return payment URL/client secret to frontend.

3. Frontend: redirect to provider/collect card details, or use client SDK.

4. Provider: hits webhook /api/payments/webhook on success. Backend verifies signature, updates Booking.paymentStatus='Paid', store payment record.

5. Backend: send booking confirmation email and push notification. Update user bookings & tripHistory.

12) Trip Planner (mapping step → backend)

- Step1 (destination+dates): frontend saves draft to Trip model via POST /api/trip/plan.

- Step2 (experience types): update Trip.selectedActivities or tags.

- Step3 (accommodation): update Trip.selectedHotel (id or object).

- Step4 (activities): add selectedActivities[].

- Step5 (review): frontend calls /api/bookings to book (creates booking and payment).

13) Dev commands & env (quick)

- Run backend: NODE_ENV=development nodemon server.js - Run frontend: npm run dev (Vite/Create React App) -
Essential .env vars: • MONGODB_URI • JWT_SECRET • JWT_REFRESH_SECRET • S3_BUCKET, S3_KEY, S3_SECRET
• STRIPE_SECRET or RAZORPAY_KEY/SECRET • SENDGRID_API_KEY or SMTP creds

14) Testing & Deployment

- Use Postman / Insomnia for API tests; unit tests with Jest + Supertest. - CI: GitHub Actions: run lint, tests, build. Deploy: Vercel (frontend) + Render/Heroku/EC2 (backend) + MongoDB Atlas. - Use CDN for static
assets and enable gzip.

15) Monitoring & Logging

- Use Winston or Pino for backend logs, Sentry for error tracking, and Prometheus/Grafana if needed.

16) Example API: GET /api/destinations/:id (response shape)

```
{ "_id": "64a1...f", "name": "Bali", "location": "Indonesia", "description": "...", "images":
["https://.../1.jpg", "..."], "tags": ["Beach","Adventure"], "rating": 4.7, "highlights": ["Temple",
"Surfing"], "travelTips": ["Best season: Apr-Oct"], "reviews": [{ "user": { "_id":"...", "name":"Amit" },
"rating":5, "comment":"Great!" }] }
```

17) Security checklist (quick)

- Use HTTPS everywhere

- Environment variables for secrets

- Input validation & sanitization

- Rate limiting for auth and payment endpoints

- Verify payment webhooks

- Use CSP, Helmet headers

- Least-privilege IAM for storage

18) Next immediate steps (priority)

- Scaffold backend models + auth routes
- Scaffold frontend pages + shared components (Navbar/Footer/Card)
- Integrate one sample endpoint: GET /api/destinations and Destination details page
- Setup MongoDB Atlas + .env
- Integrate S3 and payment sandbox (Stripe test keys)