**Project name:** OpenOutings

## **1) One-line summary**

Build a responsive web app (mobile-first, desktop friendly) for finding friends and organising small group activities. Full front end, backend, and database — optionally integrated with Supabase — with complete authentication (OAuth2 compatible), cheap/free testing options before launch, image downscaling, searchable events, host reviews, interests matching, and a group availability/voting calendar that highlights optimal times. Should be with bulgarian names, mostly around or in Sofia. Clearly interlinked – you can reach any page from any other. Card layout.

## **2) Primary goals / success criteria**

* A production-quality MVP that supports event creation, joining, pictures, host reviews, search/filters, interest tagging, and a group availability poll with an automatic best-time algorithm.
* Works smoothly on mobile and desktop (responsive UI).
* All images are downscaled automatically for bandwidth/performance.
* Authentication supports email/password + OAuth2 providers (Google / Apple / Facebook / GitHub) and is pluggable.
* The app must be deployable to free/cheap tiers for testing (e.g., Supabase backend + Vercel/Netlify frontend or equivalent).
* Deliverables: Git repo, README, deployment instructions, API docs (OpenAPI/Swagger), database schema, test plan, and a separate full feature list file.

## **3) Target users & use cases (user stories)**

* As a user, I can sign up / sign in with email or OAuth and set my profile and interests.
* As a host, I can create events (title, description, location/address, optional map pin, picture(s), date/time, capacity, gender or other restrictions, activity types, paid/free flag).
* As a user, I can search events and filter by date range, interests, location radius, free/paid, activity type, host rating, and availability windows.
* As a user, I can join an event; the current participant count updates automatically and respects capacity limits.
* As a user in a group, I can create a week-long availability poll (days × slots: morning / noon / afternoon / evening) and invite group members with a shareable link; participants mark available slots and algorithm highlights best options.
* As a user, I can review hosts and events (with basic moderation).
* As an admin, I can flag/remove content and see basic analytics (events created, active users, flagged items).

## **4) Functional requirements (precise)**

### **Events**

* Create / edit / delete events (host only).
* Fields: title, body, activity type(s) (multi-select), interests tags, date/time (single or recurring option placeholder), capacity (max participants), visible gender restriction (optional), location (structured address + lat/lng), image(s) (auto downscaled + multiple sizes), free/paid flag, host id.
* Auto update attendee count; prevent overbooking (race-safe).

### **Images**

* On upload: resize to multiple sizes (thumbnail, medium, large), compress, strip EXIF by default, store URLs in DB.
* Serve optimized formats where possible (WebP fallback).

### **Authentication & Authorization**

* Email + password (secure, hashed), password reset.
* OAuth2 providers (Google, Apple, Facebook, GitHub) with pluggable config.
* JWT access tokens (short lived) and refresh tokens.
* Role system: user, host, moderator, admin.

### **Availability poll (group scheduling)**

* Create “availability link” with a week (or two) calendar grid per day × 4 slots (morning/noon/afternoon/evening).
* Participants pick available slots (or mark full-day unavailable).
* Algorithm aggregates votes and highlights top N slots by total availability and by activity preference.
* Export results (CSV) and allow creator to finalize and convert chosen slot into an event.

### **Search & filters**

* Full text search on title/description; filters for date, interests, activity types, free/paid, distance from coordinates (haversine), host rating, capacity remaining.
* Sort by relevance, date, distance, popularity.

### **Reviews & Ratings**

* Users can leave 1–5 star reviews + optional text for hosts. Rate-limited and verified joiner-only reviews recommended.
* Basic anti-abuse: one review per event per user; flagging system; admin moderation queue.

### **Interests & Recommendations**

* Users add interests to profile (tags).
* Events with overlapping interests highlighted on user feed / search results.
* Simple recommendation: boost events matching > X interest tags.

### **Notifications**

* Email notifications for: event invites, join confirmations, host messages, poll results, password resets. (Push/real-time optional; mention WebSockets / server-sent events if needed.)
* In-app notification center.

### **Admin**

* Dashboard to moderate content, ban users, view analytics (counts of events, active users, flagged items).

## **5) Non-functional & quality requirements**

* Mobile-first responsive UI (use Tailwind or similar).
* Accessibility basics (semantic HTML, alt text, color contrast).
* Performance: images cropped/resized, lazy load, client caching (service worker optional).
* Security: input validation, rate limiting, CSRF protection, secure password hashing (e.g., bcrypt/argon2), content sanitisation (XSS).
* Scalability: design DB and APIs for horizontal scaling; prepared to move from single-node to managed services.
* Logging & error reporting: Sentry or equivalent (config stubs).
* Tests: unit tests for critical backend logic (auth, poll aggregation, booking), E2E smoke tests for major flows.

## **6) Suggested tech stack (flexible; mention Supabase option)**

* **Frontend:** React (Vite/Next.js) or SvelteKit, Tailwind CSS, mobile-first design.
* **Backend / Auth / DB options:**
  + **Supabase (recommended for fast MVP):** Auth + Postgres + Storage + Edge Functions for server logic. (Plugs OAuth, offers free tier for testing.)
  + Or: Node.js (Express/Nest) + PostgreSQL + Prisma/TypeORM + Auth0 / Clerk / custom OAuth2.
* **Storage:** Supabase Storage or S3-compatible.
* **Images:** Use serverless image processing (Edge Function / Cloud Function) or client-side resizing + backend recompression.
* **Map:** Use OpenStreetMap + Nominatim (free) or a low-cost provider; integrate map tiles via Leaflet or Mapbox (note Mapbox free tier limits).
* **Deployment:** Vercel/Netlify (frontend), Supabase/Vercel Edge Functions (backend) — all offering free tiers for testing.

*Note:* If you want me to pick exact providers and configuration for production cost/limits, say so and I’ll produce a comparison.

## **7) API surface (example endpoints)**

* POST /api/auth/signup
* POST /api/auth/login
* GET /api/users/:id
* PUT /api/users/:id
* POST /api/events
* GET /api/events (search + filters)
* GET /api/events/:id
* POST /api/events/:id/join
* DELETE /api/events/:id/join
* POST /api/polls (create availability poll)
* GET /api/polls/:id/results
* POST /api/reviews
* POST /api/upload (image upload)
* Webhooks or endpoints for OAuth callbacks

Include OpenAPI/Swagger spec as part of the deliverables.

## **8) Data model (core tables / collections)**

* **users**: id, email, display\_name, avatar\_url, interests[], oauth\_providers[], bio, role, created\_at
* **events**: id, host\_id, title, description, activity\_types[], interest\_tags[], start\_datetime, end\_datetime, capacity, gender\_restriction, price\_info, location {address, lat, lng}, image\_urls[], created\_at, status
* **event\_participants**: event\_id, user\_id, joined\_at, role (attendee/guest)
* **reviews**: id, reviewer\_id, host\_id, event\_id (optional), rating, text, created\_at
* **polls**: id, creator\_id, start\_date, end\_date, slots[][], results (user\_id -> selected slots)
* **images**: id, owner\_id, storage\_path, sizes {thumb, medium, large}, created\_at
* **flags/moderation**: id, target\_type, target\_id, reporter\_id, reason, status

## **9) Availability poll algorithm (spec)**

* Input: grid of dates × 4 slots, N participants with binary availability per slot.
* Score each slot = count(participants available) + interest\_weight (if participants also indicate preferred activities → small boost).
* Output: ranked list of slots, visual heatmap, top K highlighted.
* Tie-breaker: earliest date or slot with least conflicts across participants.
* Export: CSV with aggregated counts and participant notes.

## **10) Testing & cheap/free testing plan (no cost assumptions)**

* Use free tiers: Supabase free tier for DB/auth/storage; Vercel / Netlify for frontend; free SMTP for dev emails (Mailtrap/dev mode); GitHub Actions for CI (free for public / limited for private).
* Testing steps:  
  1. Unit tests for critical backend logic (auth, join/capacity, poll aggregator).
  2. Integration tests for API endpoints (Postman collection + automated runs).
  3. E2E smoke tests for main user flows (signup, create event, join, poll).
  4. Manual QA on a variety of mobile screen sizes.
* Include a test matrix and acceptance tests file in the repo.

## **11) Deliverables (explicit)**

* Full source code in a public/private GitHub repo (one repo or mono-repo) with clear commit history.
* README with setup, env variables, deployment steps.
* Dockerfile(s) and docker-compose for local dev where applicable.
* OpenAPI / Swagger spec and Postman collection.
* Database migration scripts and schema.
* CI pipeline (GitHub Actions) for tests + deploy to preview envs.
* Basic analytics dashboard (counts) and admin moderation UI.
* File FEATURES\_AND\_SPECS.md — a single, separate file that lists every functionality and implementation detail (the user requested a separate file).
* Optional: simple UI mockups (Figma links or static images) or basic component library.

## **12) Acceptance criteria (what QA will verify)**

* Users can sign up and login via email and one OAuth provider; tokens issued.
* Host can create event, upload image(s) and images are downscaled; participant count increments when users join; capacity enforced.
* Search returns relevant events using filters; interest-matching boosts results.
* Group poll: participants register availability; system highlights best top slots correctly.
* Host reviews can be submitted and appear in host profile; flagged reviews appear in admin queue.
* App renders correctly on iPhone/Android viewport and a desktop viewport.

## **13) Implementation roadmap (stepwise, no time estimates)**

1. Project skeleton: repo, linting, prettier, CI
2. Auth + user model + profile UI
3. Event model + create/edit UI + image upload + downscaling pipeline
4. Join/participant flow + capacity safeguards
5. Search API + basic filters
6. Poll creation + aggregation algorithm + shareable link
7. Reviews & moderation
8. Notifications (email) + in-app notifications
9. Admin dashboard + analytics
10. Tests + documentation + deployment to free preview envs
11. Polish UX, accessibility, and production hardening

## **14) What I expect from whoever receives this prompt**

* Produce a working MVP that satisfies acceptance criteria and deliverables.
* Use Supabase if requested and document any limitations.
* Provide clear instructions to run locally and to deploy to the recommended free tiers.
* Produce the separate file FEATURES\_AND\_SPECS.md that enumerates every endpoint, UI screen, DB table, and business rule.

## **15) Extra notes / optional features (nice to have)**

* Real-time chat for event participants (Socket or Supabase Realtime).
* Calendar sync export (ICS) or Google Calendar integration.
* Map clustering for many events in same area.
* Paid events workflow (payment provider integration stub: Stripe).
* Gamification badges for active users/hosts.

## **16) Final meta: format for the response you should deliver to me**

When you (developer/AI) respond to this prompt, return:

1. A short plan that confirms which stack you will use (or why you won’t use Supabase).
2. A repository skeleton link or ZIP with at least the README and FEATURES\_AND\_SPECS.md.
3. The OpenAPI spec (or a promise that it’s included in the repo).
4. A checklist showing the acceptance criteria with current completion status.