Anonymous Social Backend Roadmap — Mini Project

Prepared for: Rohan Reddy

Audience: Beginner backend developer learning Node.js, Express, MongoDB, Socket.io

Goal: Build a working, **small-scale** backend for an anonymous social platform. Focus on simplicity, clarity, and learnability rather than production-scale concerns. This document is a step-by-step roadmap, design doc, and implementation guide that will let you build a complete backend and export it as a deliverable project.

---

1. Project Overview

A minimal anonymous social backend where users can create accounts (optional), post messages anonymously to public channels or groups, like and comment on posts, and receive real-time updates using WebSockets (Socket.io). Admin moderation features are included for basic content control.

Core features (MVP): - Anonymous posting (posts are not linked to real-world identity) - Public feed and topic channels (e.g., #general, #confessions) - Post CRUD for owners and admin - Comments and likes - Real-time updates with Socket.io for new posts, comments, likes - Basic authentication (optional email/password or anonymous session token) - Admin moderation (delete posts, ban IPs/userIDs)

Non-goals for this mini project: - Horizontal scaling, distributed systems, advanced search, GDPR-level privacy, encrypted messaging. These can be added later.

---

2. Tech Stack

- **Runtime & Server:** Node.js, Express.js
- **Database:** MongoDB (use MongoDB Atlas if you want hosted DB) with Mongoose ODM
- **Real-time:** Socket.io
- **Authentication:** JWT for optional registered users; simple session tokens for anonymous flows
- **Dev tools:** nodemon, dotenv, Postman (or Insomnia)
- **Deployment (simple):** Heroku / Render / Railway / DigitalOcean App Platform

---

3. High-Level Architecture

1. **Express API** to serve REST endpoints for posts, comments, likes, users, and admin actions.

2. **Socket.io** attached to the same Express server to broadcast `new_post`, `new_comment`, `like_update` events.
3. **MongoDB** stores collections: `users`, `posts`, `comments`, `channels`, `bans`.
4. **Middleware** for authentication, rate-limiting (simple), and input validation.

Diagram (conceptual):

Client (web/mobile) ↕ REST / WebSockets Express + Socket.io (Node.js) ↕ MongoDB (Mongoose)

---

4. Database Schema (Mongoose models)

User (optional)

```
const User = new Schema({
  username: { type: String },             // optional display name
  email: { type: String },                // optional for registered users
  passwordHash: { type: String },         // bcrypt
  createdAt: { type: Date, default: Date.now },
  role: { type: String, enum: ['user','admin'], default: 'user' }
});
```

Post

```
const Post = new Schema({
  channel: { type: String, default: 'general' },
  text: { type: String, required: true },
  metadata: { ip: String, userAgent: String }, // for moderation only
  anonymousId: { type:
String }, // short generated id to allow post owner edits (optional)
  likes: { type: Number, default: 0 },
  commentCount: { type: Number, default: 0 },
  createdAt: { type: Date, default: Date.now },
  deleted: { type: Boolean, default: false }
});
```

Comment

```
const Comment = new Schema({
  postId: { type: Schema.Types.ObjectId, ref: 'Post', required: true },
  text: { type: String, required: true },
  anonymousId: { type: String },
  createdAt: { type: Date, default: Date.now },
```

```
  deleted: { type: Boolean, default: false }
});
```

Ban / Moderation

```
const Ban = new Schema({
  ip: String,
  anonymousId: String,
  reason: String,
  expiresAt: Date
});
```

---

5. API Endpoints (REST)

All endpoints prefixed with `/api/v1`.

Public

- `GET /api/v1/channels` — list available channels
- `GET /api/v1/posts?channel=general&page=1&limit=20` — fetch paginated posts
- `GET /api/v1/posts/:id` — fetch single post with last N comments

Posting & Interaction

- `POST /api/v1/posts` — create a post. Body: `{ text, channel, anonymousId? }`
- `PUT /api/v1/posts/:id` — edit post (if you have the anonymousId or are admin)
- `DELETE /api/v1/posts/:id` — delete post (admin or owner)
- `POST /api/v1/posts/:id/comments` — add comment
- `POST /api/v1/posts/:id/like` — toggle like (or simple increment)

Admin

- `POST /api/v1/admin/ban` — ban an IP or anonymousId
- `GET /api/v1/admin/stats` — basic usage stats (posts/day, active sockets)

Auth (optional)

- `POST /api/v1/auth/register` — register
- `POST /api/v1/auth/login` — login, returns JWT
- `POST /api/v1/auth/anon-session` — generate an anonymous session token (returns anonymousId)

---

6. Socket.io Events

Socket namespace: `/` (default)

Events emitted by server: - `new_post` — payload: post object - `new_comment` — { postId, comment } - `like_update` — { postId, likes }

Events client can send: - `create_post` — server validates & broadcasts - `create_comment`

Authentication: for optional registered users, send JWT with `socket.handshake.auth.token`.

---

7. Security & Moderation (Basic)

- **Input validation & sanitization**: use `express-validator` or Joi. Strip HTML.
- **Rate limiting**: `express-rate-limit` — e.g., 10 posts per hour per IP.
- **Basic abuse controls**: ban list (IP + anonymousId), maximum text length, profanity filter (optional package).
- **CORS**: restrict origins in production.
- **Store minimal identifying metadata**: IP and user-agent only for moderation; keep retention policy.
- **Passwords**: bcrypt hashing.
- **JWT secret** in `process.env.JWT_SECRET`.

---

8. Implementation Plan & Milestones (1–2 weeks plan, beginner-friendly)

Day 1 — Setup & Models

- Initialize project: `npm init -y`
- Install: `express`, `mongoose`, `socket.io`, `dotenv`, `cors`, `express-rate-limit`, `bcrypt`, `jsonwebtoken`, `nodemon`, `express-validator`
- Create Mongoose models: `Post`, `Comment`, `User`, `Ban`

Day 2 — Basic REST APIs

- Implement `GET /posts` and `POST /posts`
- Implement simple validation & sanitization

Day 3 — Socket.io integration

- Attach Socket.io to server
- Emit `new_post` after creating post
- Test with simple HTML client or Postman websockets

Day 4 — Comments & Likes

- Implement comment endpoints and broadcasting
- Implement like endpoint and realtime updates

Day 5 — Authentication (optional) & Anonymous Sessions

- Implement `anon-session` endpoint that issues an `anonymousId` token (signed or random)
- Implement JWT auth for registered users (optional)

Day 6 — Admin & Moderation

- Implement ban endpoint, middleware to check ban list
- Admin delete post

Day 7 — Polish, Tests, README, Deploy

- Add README, Postman collection, scripts
- Deploy to Heroku/Render

---

9. Sample Folder Structure

```
/anonymous-backend
  /src
    /controllers
    /models
    /routes
    /middleware
    /utils
    server.js
  .env
  package.json
  README.md
```

---

10. Example Code Snippets (short & clear)

server.js (basic)

```
require('dotenv').config();
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');
const mongoose = require('mongoose');
```

```
const app = express();
const server = http.createServer(app);
const io = new Server(server, { cors: { origin: '*' } });

app.use(express.json());

// connect to MongoDB
mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true,
useUnifiedTopology: true });

io.on('connection', socket => {
  console.log('socket connected', socket.id);
  socket.on('disconnect', () => console.log('socket disconnected', socket.id));
});

const PORT = process.env.PORT || 3000;
server.listen(PORT, () => console.log('Server running on', PORT));
```

Create post controller (simplified)

```
const Post = require('../models/Post');

exports.createPost = async (req, res) => {
  const { text, channel, anonymousId } = req.body;
  const post = await Post.create({ text, channel, anonymousId, metadata: { ip:
req.ip, userAgent: req.headers['user-agent'] } });
  req.app.get('io').emit('new_post', post);
  return res.status(201).json(post);
};
```

(When wiring, store `io` on `app.set('io', io)` after creation.)

---

11. Testing & Debugging

- Unit test with Jest (optional)
- Manual testing with Postman
- Simple web client to test Socket.io events
- Logging: `morgan` middleware for HTTP logs

---

12. Deployment (simple)

- Add `Procfile` if using Heroku: `web: node src/server.js`

- Set environment variables in host (MONGO_URI, JWT_SECRET)
- Use a managed MongoDB (Atlas) for ease

---

13. Tips for a Beginner

- Keep endpoints small and focused.
- Write and test one feature at a time.
- Use Postman to exercise endpoints and WebSocket events.
- Keep secrets out of source code.
- Add comments and README; explain architecture and how to run locally.

---

14. Deliverables for the Mini Project

- GitHub repo with code and README
- Postman collection
- Short `demo.md` showing how to run locally and how to test sockets
- OPTIONAL: small frontend HTML file to demo real-time posts

---

15. Next steps (if you later want to scale)

- Add pagination with range queries
- Add Redis for session storage and socket adapter
- Introduce microservices for heavy features
- Add full-text search (ElasticSearch)

---

16. Appendix — Environment Variables (example)

```
PORT=3000
MONGO_URI=mongodb+srv://<user>:<pass>@cluster0.mongodb.net/anon-db
JWT_SECRET=your_jwt_secret
RATE_LIMIT_WINDOW=60
RATE_LIMIT_MAX=100
```

If you'd like, I can: - Generate a ready-to-run starter repo scaffold (with `package.json`, basic routes, and models), or - Produce a PDF export of this document.

> I have created a full, clear, and official roadmap document in the canvas next to this chat. I can also produce and provide a downloadable PDF now.