Prompts

I have listed the initial prompts I used which I generated using chatGPT, but I had to tweak a lot of things as I went on.

1. Project Setup & Tooling

   1. "Generate a package.json for a Discord bot using Node.js 18+, with dependencies: discord.js@14, dotenv, mongoose, axios, winston, @discordjs/builders, and dev-dependencies: eslint, prettier, jest. Include scripts: start, dev, deploy-commands, test, lint, format."
   2. "Produce an ESLint configuration (.eslintrc.js) that enforces Airbnb style plus Discord.js best practices. Add a Prettier config (.prettierrc) harmonized with ESLint."
   3. "Create a .gitignore ignoring node_modules, .env, dist (if building), logs, and coverage."
   4. "Write a .env.example containing placeholders for BOT_TOKEN, MONGODB_URI, OPENWEATHER_API_KEY, LOG_CHANNEL_ID, ENV=development|production."

2. Configuration Loader & Validation

   1. "Implement src/config/index.js that loads and validates environment variables. Use joi or zod to ensure required keys are present and of correct type. Export a typed config object."
   2. "Add a startup health check: if validation fails, log a clear error with winston and exit process."

3. Database Integration

   1. "Create src/database/mongoose.js that connects to MongoDB via mongoose, listens to connected, error, disconnected events, and auto-reconnects."
   2. "Define Mongoose schemas & models in src/models/:
      - Punishment (userId, guildId, type, reason, moderatorId, timestamp)
      - MutedMember (userId, guildId, expiresAt, moderatorId)
      - WelcomeSetting (guildId, channelId, messageTemplate)
      - RolePicker (guildId, messageId, roleIds)

- SuggestionChannel (guildId, channelId)
- CommandUsage (userId, commandName, timestamp)
  Each schema should include timestamps and indexes for efficient queries."

4. Bot Client & Command Deployment

   1. "Write src/index.js:
      - Initialize Client with intents for guilds, members, messages
      - Import config, database, logger
      - Dynamically load commands and events
      - On ready, deploy slash commands to Discord (global vs guild-scoped based on ENV)."
   2. "Implement scripts/deploy-commands.js to read all command JSON from src/commands and push to Discord's API. Support --global and --guild <guildId> flags."
   3. "Create src/handlers/eventHandler.js: scan src/events, register each file (once or on)."

5. Logging & Utilities

   1. "Implement src/utils/logger.js using Winston: console and file transports, timestamped logs, error stack traces in production."
   2. "Create src/utils/embedBuilder.js: helper to build uniform embeds with configurable color, footer, timestamp, author."
   3. "Write a CommandRateLimiter class in src/utils/rateLimiter.js to block users who spam commands. Configurable per-command cool-downs."

6. Core Event Listeners

   1. "guildMemberAdd in src/events/guildMemberAdd.js: fetch welcome settings, format the stored template (support {user}, {guild}, {count}), and send embed."
   2. "interactionCreate in src/events/interactionCreate.js:
      - If not a command, ignore
      - Defer replies for long-running commands
      - Log each usage to CommandUsage model

- Catch and report errors in an ephemeral reply"

3. "messageReactionAdd and messageReactionRemove in src/events/messageReaction.js to handle reaction-roles via RolePicker model."

7. Slash Command Modules

Organize under src/commands/{category}/{commandName}.js. Each export: { data: SlashCommandBuilder, execute(interaction) }.

7.1 Debug & Info

- "/ping: reply with pong and latency (API vs client). Include code to measure Date.now()."
- "/botinfo: embed with uptime, memory usage, Discord.js version, node version, total guilds, total users (cache) and shard count if any."
- "/serverinfo: embed fields-name, ID, creation date, member counts (bots vs humans), region, verification level, roles count, emojis count."
- "/userinfo <user>: embed-username, discriminator, ID, avatar thumbnail, joined guild at, account created at, roles list (up to 10)."

7.2 Fun & Utility

- "/joke: fetch a random joke from https://official-joke-api.appspot.com/random_joke, send question + punchline."
- "/weather <location>: use OpenWeatherMap; parse args to support city name or zip code; embed with temperature, humidity, conditions, icon; handle invalid location errors."

7.3 Moderation

- "/ban <user> [reason]: check permissions, banMember, save to Punishment model, log embed to mod-log channel (configurable in DB or .env)."
- "/kick <user> [reason]: similar to ban, but member.kick."
- "/mute <user> [duration] [reason]:
    - Create or find a 'Muted' role with denied SEND_MESSAGES & ADD_REACTIONS.
    - Add role, insert into MutedMember with expiresAt.

        - Schedule unmute (in-memory scheduler or better use node-cron or database-driven polling).
        - Log action."
- "/unmute <user>: remove role, delete from MutedMember, log."
- "/record <user>: fetch Punishment entries, paginate if >10, display in embeds."

7.4 Role & Reaction Roles

- "/giverole <user> <role> and /takerole <user> <role>: check hierarchy, add/remove, confirm via ephemeral reply, log."
- "/rolepicker create <channel> <message> <roles...>: post an embed listing roles with emoji reactions, save messageId & roleIds in RolePicker model."
- "In messageReactionAdd/Remove: cross-reference RolePicker, give/take roles."

7.5 Welcome & Suggestions

- "/setwelcomechannel <channel>: upsert WelcomeSetting for guild."
- "/setwelcomemessage <message>: store template."
- "/suggest <text>: post in suggestion channel (from DB or global), add 👍 👎 reactions, store suggestion metadata (suggestionId, userId, channelId, messageId)."
- "/suggestclose <suggestionId> [status: accepted|rejected] [reason]: edit suggestion embed to show closed status and reason, log action."

8. Scheduling & Jobs

1. "Implement src/utils/scheduler.js: on startup, load all pending MutedMember entries with future expiresAt, schedule unmute jobs. Use node-schedule or bree."
2. "Add a daily job to purge old logs (Punishment older than 1 year)."

9. Testing & Quality

1. "Write Jest unit tests for:
    - Config validation (valid & invalid env)
    - Database connection mock

    - Each command’s data structure (correct options/description)
    - A few command execute flows using a mock interaction object”

  2. “Configure coverage thresholds (80%+), add a test:watch script.”

10. Monitoring & Error Tracking

    1. “Integrate Sentry: initialize in index.js, capture unhandledRejection and uncaughtException, wrap command execution in try/catch sending errors to Sentry.”
    2. “Set up a simple health-check HTTP endpoint (Express) on port 3000 returning bot status and DB status for uptime monitoring.”