Backend Documentation

Table of contents

1. Overview

This document describes the server-side API for the web application (TypeScript / Node.js). The backend exposes a REST API with resources for users, startups, founders, investors, events, partners, news, and messages. The app uses Prisma as the ORM to connect to a MySQL database and issues JWT tokens for authentication.

2. Tech stack

- Node.js + TypeScript
- Express
- Prisma (MySQL datasource)
- JWT for authentication
- bcrypt for password hashing
- Cors
- Nodemon / ts-node for development

3. Project structure (important files/folders)

```
Backend/
├─ src/
│   ├─ app.ts                # main app (start server)
│   ├─ routes/               # endpoint routes
│   ├─ controller/           # controllers
│   ├─ middleware/           # auth middleware
│   ├─ prisma/               # prisma schema and migrations
│   ├─ config/               # database config, other config utilities
│   ├─ utils/                # helper utilities (e.g. SyncDb.utils.ts)
│   └─ apis/                 # external API wrappers (JebApi)
├─ .env.exemple              # example env vars (see below)
├─ Dockerfile
├─ package.json
└─ tsconfig.json
```

4. Environment variables

See `Backend/.env.exemple` — these are expected (example names and meaning):

- `DATABASE_URL` — Prisma-compatible MySQL URL (e.g. `mysql://USER:PASSWORD@localhost:PORT/DBNAME`)
- `API_KEY` — Jeb API key
- `PORT` — server port (default in repo set to `4000` in example)
- `VITE_BACKEND_URL` — frontend base URL (used by frontend config)

Make sure to set a production-ready `DATABASE_URL` and keep secrets out of source control.

5. Installation & run

Local (development)

1. Install dependencies:

```
cd Backend
npm install
```

2. Create a `.env` file based on `.env.exemple` and set `DATABASE_URL`.

3. Run Prisma migrations (if needed):

```
npm run migrate
```

4. Build (optional) and run (or use ts-node / nodemon for development):

```
npm run build
npm start
```

5. The server listens on `PORT` (default from `.env`), and base path.

6. Database & Prisma

The project uses Prisma with a MySQL datasource. The Prisma schema is located at `src/prisma/schema.prisma` and includes the following main models: `User`, `Message`, `Event`, `Founder`, `Investor`, `Partner`, `NewsDetail`, and `StartupDetail`.

Migrations live under `src/prisma/migrations/` and can be applied with the `migrate` script.

There is a `SyncDb.utils.ts` utility in `src/utils/` which is invoked at app startup (`syncDB();`) — review it to understand behavior for schema sync / seed.

7. Authentication & Authorization

- Authentication is implemented with JWT. Login issues a JWT (see `Auth.controller`).
- Middleware `requireAuth` protects routes that require a valid token.
- Middleware `authorizeRoles(...)` restricts endpoints to roles such as `admin` and `founder` where necessary.
- Passwords are hashed with `bcrypt` on registration (see auth controller logic).

Ensure you keep the JWT secret and related configuration secure (look for `process.env` usages in `src/controller/Auth.controller.ts` or auth middleware).

8. API endpoints (summary)

All resource routes are mounted in `app.ts`. Each listed route below is mounted at its base path. For example, the `EventRouter` is mounted at `/event`, so `POST /event/create` creates an event.

Auth (`/auth`)

- `POST /auth/register` — register a new user (public), but not used in the front
- `POST /auth/login` — login, returns JWT (public)

User (`/user`)

- `POST /user/create` — create user (requires auth and `admin` role)
- `GET /user/get/:id` — get user by id
- `GET /user/get` — list users
- `DELETE /user/delete/:id` — delete user (requires `admin`)
- `PUT /user/update/:id` — update user (requires auth)

Event (`/event`)

- `POST /event/create` — create event (requires `admin` or `founder`)
- `GET /event/get/:id` — fetch event by id
- `GET /event/get` — fetch all events
- `DELETE /event/delete/:id` — delete (requires `admin` or `founder`)
- `PUT /event/update/:id` — update (requires `admin` or `founder`)

Founder (`/founder`)

- `POST /founder/create`
- `GET /founder/get/:id`
- `GET /founder/get`
- `DELETE /founder/delete/:id`
- `PUT /founder/update/:id`

Investor (`/investor`)

- `POST /investor/create`
- `GET /investor/get/:id`
- `GET /investor/get`
- `DELETE /investor/delete/:id`
- `PUT /investor/update/:id`

Partner (`/partner`)

- `POST /partner/create`
- `GET /partner/get/:id`
- `GET /partner/get`
- `DELETE /partner/delete/:id`
- `PUT /partner/update/:id`

Startup (`/startup`)

- `POST /startup/create` — create startup (requires auth)
- `GET /startup/get/:id`
- `GET /startup/get`
- `DELETE /startup/delete/:id`

- `PUT /startup/update/:id` — update (requires `admin` or `founder`)

News (`/news`)

- `POST /news/create`
- `GET /news/get/:id`
- `GET /news/get`
- `DELETE /news/delete/:id`
- `PUT /news/update/:id`

Message (`/message`)

- `POST /message/create`
- `GET /message/get/:id`
- `GET /message/get`
- `DELETE /message/delete/:id`
- `PUT /message/update/:id`
- `GET /message/received/:userId` — get messages received by user
- `GET /message/sent/:userId` — get messages sent by user

> Note: This is a summary — for full parameter lists, request/response shapes, and validation rules, see bellow

9. Data models (Prisma) — summary

The schema defines the following models (fields abbreviated):

- **User**: `id`, `email` (unique), `password`, `name`, `role`, optional relations to `Founder` and `Investor` and messages (sent/received).
- **Message**: `id`, `sender_id`, `receiver_id`, `content`, `sent_at`.
- **Event**: event fields (id, title, description, date/time, organizer relation, etc.).
- **Founder**: founder-specific fields and relation to `StartupDetail`.
- **Investor**: investor fields.
- **Partner**: partner fields.
- **NewsDetail**: news fields.
- **StartupDetail**: startup fields including `name`, `email`, `website_url`, `project_status`, `sector`, `maturity`, and relation to `Founder[]`.

(See `src/prisma/schema.prisma` for complete field lists and types.)

10. Error handling & validation

- Controllers typically send standard HTTP status codes (200, 201, 401, 403, 404, 422, 500) depending on validation and authorization.

- Input sanitization and permitted fields are enforced inside controllers (there are patterns of `authorizedFields` and filtering update payloads).
- Server-side logging is done with `console.error`/`console.log` in the current codebase; consider integrating a structured logger (e.g., `pino` or `winston`) for production.

Where to look in the code for details

- `src/controller/*` — controller logic, request/response shapes
- `src/routes/*` — route definitions and required middleware
- `src/middleware/*` — authentication and authorization middleware
- `src/prisma/schema.prisma` — DB models
- `src/utils/SyncDb.utils.ts` — DB sync/seed logic
- `src/apis/JebApi/` — external API wrappers
