documents

Anjoman-blog

Overview

A web-based platform built with TypeScript, Express, Prisma, PostgreSQL, Next.js, and Mantine for users to post articles, announcements, events, and comments.
Overview

- Backend: TypeScript-Express with Prisma ORM and PostgreSQL for data management.
- Frontend: TypeScript-Next.js with Mantine UI for a responsive interface.
- Features: User authentication, article submission/review, commenting, event/announcement management.

Installation

Clone the repository:

```
git clone https://github.com/nimankz/anjoman-blog.git
```

Install backend dependencies:

```
cd backend
npm install
```

Set up PostgreSQL and environment variables (.env):

```
DATABASE_URL="postgresql://user:password@localhost:5432/blog"
```

Run Prisma migrations:

```
npx prisma migrate dev
```

Install frontend dependencies:

```
cd ../frontend
npm install
```

Start backend and frontend:

```
sudo chmod +x run
./run
```

```
#if that didn't work

# Backend
cd backend
npm start
# Frontend (new terminal)
cd frontend
npm run dev
```

Access at http://localhost:3000.

📄 Sample Documentation: Backend Architecture

📦 Backend Architecture – Project Overview

✅ Built With

- **Node.js + Express.js** (HTTP server framework)
- **TypeScript**
- **EJS** for email previews
- **Socket.IO (optional)**
- **Custom services and controllers**

🌐 Purpose

This backend serves as the main API server for the fullstack application. It handles:

- User authentication and session management
- Organization and membership logic
- Article publishing and moderation
- Event and announcement broadcasting
- Email notifications (with preview support)

📁 Directory Overview

Folder / File	Purpose
`/controllers/api/`	Route handlers (REST API logic)
`/domain/email/`	Email sending + preview rendering

Folder / File	Purpose
`/services/`	Application logic (auth, blog, etc.)
`/config/`	Global configuration (env, secrets...)
`/views/`	EJS email templates

⚙ Server Configuration

Express Initialization:

```
const app = express();
app.use(cors(...));
app.use(json());
app.use(urlencoded({ extended: true }));
```

View Engine:

```
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
```

Used for rendering email previews on `/email-preview`.

🧩 Controllers and Services

1. `UserController`

Handles:

- Signup
- Login
- Profile updates
- Onboarding

2. `SessionController`

Handles:

- User session creation
- Token issuing

3. `AnnouncementController`

Handles:

- Creating/viewing announcements (admin + user)

4. `EventController`

Handles:

- Public event listing
- Admin event management

5. `ArticleController`

Handles:

- Article submission
- Moderation workflow (publish/reject)

6. `MembershipController` & `OrganizationController`

Handle:

- Managing organization and user roles
- Joining orgs, updating members

7. `PassResetController`

Handles:

- Password reset flows

📫 Email System

- Emails are handled using a reusable `EmailService`
- Emails are rendered using **EJS** templates
- Local preview available via:

  ```
  GET /email-preview
  ```

🔐 CORS Policy

In development:

```
cors({
  origin: '*', // Allow all
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization']
})
```

> In production, the origin should be locked down to a known frontend domain.

🛣️ Routes Summary

Route	Description
`POST /api/users`	Create user account
`POST /api/sessions`	Login
`POST /api/announcements`	Create announcement (admin)
`GET /api/announcements`	List announcements
`POST /api/articles`	Submit article
`PUT /api/articles/:id`	Moderate article
`POST /api/events`	Create event
`GET /api/events`	List public events
`POST /api/comments`	Comment on a post
`GET /email-preview`	Render sample email templates

🧠 Suggested Enhancements

- Add middleware for **request logging**
- Add rate limiting for API endpoints
- Add Swagger/OpenAPI docs

📝 Example API Request

```
POST /api/users
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "secure123"
}
```

✅ Conclusion

This backend follows a clean, layered architecture:

- **Controllers** handle routing
- **Services** encapsulate business logic
- **Domain layers** (like email) are modular and testable

It’s scalable and ready for integration with a frontend (Next.js, React, etc.) or mobile clients.

📚 API Documentation

🔗 Base URL

```
http://localhost:4000/api
```

> All routes are prefixed by `/api`

🔐 Authentication

- `decorateWithAuth` → Requires user to be logged in
- `decorate` → Optional login logic (e.g., for sign up)
- Routes not decorated → Public access

📦 Resources

- Articles
- Events
- Announcements
- Users

Thanks for sharing your full `BlogService` code! I reviewed it, and your logic is well-organized. Now you asked me to:

> ✅ **Update the body examples in the API documentation to match the actual data your service expects.**

Below is your **updated API documentation**, with correct and complete **body examples**, based on your actual Prisma and controller usage.

📘 Updated API Request Bodies (Based on `BlogService`)

📰 Articles

`POST /articles`

Create New Article

```
{
  "title": "Understanding Prisma ORM",
  "content": "Prisma is a next-gen Node.js ORM..."
}
```

`PUT /articles/:id`

Update Article (Admin only)

```
{
  "status": "PUBLISHED",
  "title": "Updated Title",
  "content": "Updated content of the article."
}
```

> 🧠 `status` must be of enum type `ArticleStatus`, like: `PUBLISHED`, `REJECTED`, `ACCEPTED`

`POST /articles/newComment`

Add a Comment to Article

```
{
  "articleId": "clx123...",
  "content": "This was a really helpful guide.",
  "title": "Nice article"    // optional
}
```

📢 Announcements

`POST /announcements`

Create New Announcement

```
{
  "title": "Server Downtime",
  "content": "The system will be down for maintenance on Friday."
}
```

`PUT /announcements/:id`

Update Announcement (Admin only)

```
{
  "title": "Updated Server Downtime",
  "content": "The downtime has been rescheduled to Saturday."
}
```

`GET /announcements/:id/comments`

→ No body needed.

`GET /announcements/newComment`

→ Not used to post, but for preview/form (can ignore).

`POST /announcements/newComment`

Add a Comment to Announcement

```json
{
  "announcementId": "clx123...",
  "content": "Thanks for the update!",
  "title": "Maintenance Notice"   // optional
}
```

📅 Events

`POST /events`

Create New Event

```json
{
  "name": "AI Conference",
  "content": "Join us for an advanced AI workshop.",
  "date": "2025-07-21T10:00:00.000Z"
}
```

`PUT /events/:id`

Update Event (Admin only)

```json
{
  "name": "Updated AI Conference",
  "content": "New speakers have been added!"
}
```

`POST /events/newComment`

Add a Comment to Event

```json
{
  "eventId": "clx123...",
  "content": "Looking forward to this event!",
  "title": "Excited!"  // optional
}
```

💬 Comments (Common for Articles, Announcements, Events)

`DELETE /comments/:commentId`

```
{
  "userId": "clxUser123"
}
```

> Only the **comment owner** or **admin** can delete.

`PUT /comments/:commentId`

```
{
  "content": "Updated comment content."
}
```
