

**Transformando FOMO em JOMO: A Conectividade para o Mundo Real.**

---

# Antisocial.

ECOS12A - Laboratório de Sistemas Distribuídos

Eduardo Henrique  
João Pedro Quinteiro  
Lucas Galvão  
Tales Araujo Kodama  
Thiago Rovari

# Introdução: Ideia da Aplicação

- Rede social com foco em desenvolvimento pessoal
- Cadastro/login, posts, likes, comentários, seguidores
- Visual minimalista e intuitivo
- Limite de rolagem no feed



# Backend - Express ex

- Uso do Express para estruturar a API.
- Rotas separadas por domínio (auth, users, activities...) para modularidade.
- Middlewares globais: cors e json.
- Fluxo: requisição → autenticação/validação → rota específica.
- Organização que mantém o código seguro, limpo e escalável.

```
export const app = express();
app.use(cors());
app.use(express.json());

app.use('/auth', authRoute);
app.use('/users', userRoute);
app.use('/activities', activityRoute);
app.use('/incentives', incentiveRoute);
app.use('/connections', connectionRoute);
app.use('/comments', commentRoute);

app.use(notFoundHandler);
app.use(errorHandler);
```

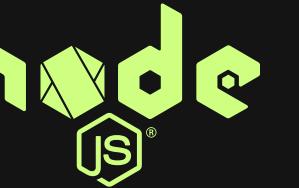
```
const activityRoute = Router();

activityRoute.use(authenticate);

activityRoute.get('/', getAllActivities);
activityRoute.get('/:id', getActivityById);
activityRoute.post('/', validateZod(createActivitySchema), createActivity);
activityRoute.put('/:id', validateZod(updateActivitySchema), updateActivity);
activityRoute.delete('/:id', deleteActivity);

export default activityRoute;
```

# Backend - NodeJs



- Node.js como ambiente de execução JavaScript no servidor.
- package.json organiza dependências, scripts e configurações do projeto.
- Scripts para desenvolvimento, build, testes, lint e formatação.
- Dependências principais: Express, Postgres, bcrypt, Zod etc.
- Facilita automação, padronização e manutenção do backend.

```
1 {  
2   "name": "anti-social",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "dev": "ts-node-dev src/index.ts",  
8     "build": "tsc",  
9     "start": "node dist/index.js",  
10    "lint": "eslint .",  
11    "format": "prettier --write .",  
12    "test": "cross-env NODE_ENV=test jest --runInBand",  
13    "test:coverage": "cross-env NODE_ENV=test jest --coverage --runInBand",  
14    "check": "pnpm lint && pnpm format"  
15  },  
16  "keywords": [],  
17  "author": "",  
18  "license": "ISC",  
19  "packageManager": "pnpm@10.4.1",  
20  "dependencies": {  
21    "bcrypt": "^5.1.1",  
22    "cors": "^2.8.5",  
23    "dotenv": "^16.4.7",  
24    "express": "^4.21.2",  
25    "http-status": "^2.1.0",  
26    "jsonwebtoken": "^9.0.2",  
27    "pg": "^8.16.0",  
28    "reflect-metadata": "^0.2.2",  
29    "typeorm": "^0.3.24",  
30    "zod": "^4.1.12"  
31  }  
32}
```

# Backend - Zod



- Biblioteca usada para validar dados de entrada na API.
- Garante que os campos tenham o tipo correto e respeitem regras (min, max, trim).
- Evita que dados inválidos cheguem aos controllers e ao banco.
- Facilita mensagens de erro claras e padronizadas.
- Aumenta a segurança e a confiabilidade das requisições.

```
import { z } from 'zod';

export const createActivitySchema = z.object({
  title: z
    .string()
    .min(1, 'O título é obrigatório')
    .max(200, 'Título deve ter no máximo 200 caracteres')
    .trim(),
  description: z
    .string()
    .min(1, 'A descrição é obrigatória')
    .max(5000, 'Descrição deve ter no máximo 5000 caracteres')
    .trim(),
});
```

# Testes - Jest e Supertest



- Setup Dedicado: Configuração de banco de dados exclusivo para rodar os testes.
- Gestão de Estado: Hooks (before/after) garantem um ambiente limpo a cada execução.
- Requisições Reais: Teste de endpoints (POST /activities) simulando um cliente.
- Verificação de Resultados: Comparação precisa dos dados retornados pela API

```
dotenv.config({
  path: process.env.NODE_ENV === 'test' ? '.env.test.local' : '.env',
});

if (process.env.NODE_ENV === 'test') {
  process.env.JWT_SECRET = process.env.JWT_SECRET_TEST;
}

beforeAll(async () => {
  await AppDataSource.initialize();
  await AppDataSource.synchronize();
});

afterAll(async () => [
  if (AppDataSource.isInitialized) {
    await AppDataSource.destroy();
  }
]);
```

```
describe('Activity Integration Tests', () => {
  let token: string;
  let userId: string;
  let activityId: string;

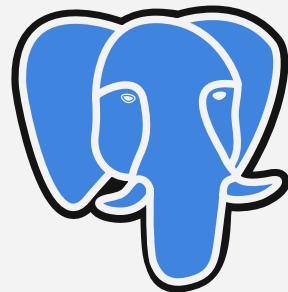
  beforeEach(async () => {
    await clearActivities();
    await clearUsers();
    ({ token, userId } = await createAndLoginTestUser());
  });

  afterEach(async () => {
    await clearActivities();
    await clearUsers();
  });

  // CREATE ACTIVITY
  describe('POST /activities', () => {
    it('Deve criar uma atividade (201)', async () => {
      const res = await request(app)
        .post('/activities')
        .set('Authorization', `Bearer ${token}`)
        .send({
          title: 'Minha Primeira Atividade',
          description: 'Esta é uma descrição detalhada da atividade.',
        });

      expect(res.status).toBe(201);
      expect(res.body).toHaveProperty('id');
      expect(res.body.title).toBe('Minha Primeira Atividade');
    });
  });
});
```

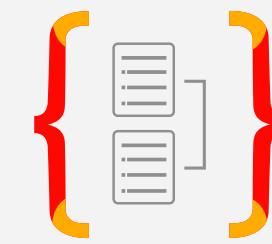
# Dados - Postgres



- Banco de dados relacional utilizado no projeto.
- Armazena as informações de forma estruturada e com integridade.
- Ideal para aplicações que precisam de consistência e transações seguras.
- Oferece boa performance e flexibilidade para consultas complexas.

A screenshot of a PostgreSQL database browser interface. The left sidebar shows a tree view of the database schema. At the top is the database name 'anti\_social\_db'. Below it are several schema objects: Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, and Schemas. The 'Schemas' node has one child, 'public', which is expanded to show Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, and Sequences. Each item in the tree has a small colored icon next to it, representing its type.

# Dados - TypeORM



- Conexão Dinâmica: Configura a conexão conforme o ambiente (Test/Prod).
- Entidades como Classes: Cada classe mapeia diretamente uma tabela.
- Gestão de Associações: Decorators simplificam relacionamentos complexos.
- Integração com TypeORM: Faz a ponte entre Node.js e PostgreSQL de forma eficiente.

```
export const AppDataSource = new DataSource({
  type: 'postgres',
  url:
    process.env.NODE_ENV === 'test'
      ? process.env.DATABASE_URL_TEST
      : process.env.DATABASE_URL,
  synchronize: true,
  logging: false,
  dropSchema: process.env.NODE_ENV === 'test',
  entities: [User, Activity, Incentive, Connection, Comment],
});
```

```
@Entity()
export class Connection {
  @PrimaryGeneratedColumn('uuid')
  id!: string;

  @ManyToOne(() => User, { eager: true })
  @JoinColumn({ name: 'user1' })
  user1!: User;

  @ManyToOne(() => User, { eager: true })
  @JoinColumn({ name: 'user2' })
  user2!: User;

  @Column({ type: 'timestamp', default: () => 'CURRENT_TIMESTAMP' })
  creationDate!: Date;
}
```

# Dados - Docker



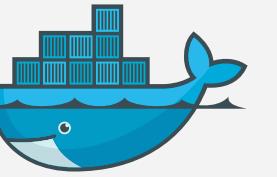
## docker-compose

- Uso do Docker para subir o Postgres de forma isolada e reproduzível.
- Containers separados para ambiente real e ambiente de teste.
- Variáveis de ambiente definem usuário, senha e nome do banco.
- Volumes garantem persistência dos dados fora do container.
- Facilita setup rápido, padronizado e sem dependências locais

```
▶ Run All Services
services:
  ▶ Run Service
  anti_social:
    image: postgres:latest
    container_name: anti_social
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB_NAME}
    ports:
      - '5434:5432'
    volumes:
      - ./data/social:/var/lib/postgresql
    restart: unless-stopped

  ▶ Run Service
  postgres_test:
    image: postgres:latest
    container_name: postgres_test
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB_TEST}
    ports:
      - '5433:5432'
    volumes:
      - ./data/test:/var/lib/postgresql
    restart: unless-stopped
```

# Dados - Docker



## docker-compose

- Ambiente Containerizado: Builds consistentes para API, Banco e Cache.
- Cluster de API: Uso de deploy: replicas: 3 para suportar mais carga.
- Roteamento Inteligente: O Nginx gerencia a entrada na porta 8080.
- Dependências: Controle de ordem de inicialização (depends\_on).

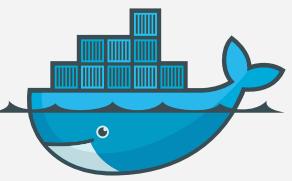
```
▷ Run Service
redis:
  image: redis:alpine
  container_name: anti_social_redis
  ports:
    - '6379:6379'
  networks:
    - app-network

▷ Run Service
api:
  build: .
  deploy:
    replicas: 3
  environment:
    - PORT=3000
    - POSTGRES_HOST=anti_social
    - POSTGRES_PORT=5432
    - POSTGRES_USER=${POSTGRES_USER}
    - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    - POSTGRES_DB_NAME=${POSTGRES_DB_NAME}
    - REDIS_HOST=redis
    - JWT_SECRET=${JWT_SECRET}
  depends_on:
    - anti_social
    - redis
  networks:
    - app-network

▷ Run Service
nginx:
  image: nginx:latest
  container_name: load_balancer
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
  ports:
    - '8080:80'
  depends_on:
    - api
  networks:
    - app-network

networks:
  app-network:
    driver: bridge
```

# Dados - Docker



## dockerfile

- Ambiente Padronizado: Garante a mesma versão do Node para todos os devs.
- Construção em Camadas: Estratégia eficiente para reinstalar dependências apenas se necessário.
- Automação: Configura todo o ambiente com um único comando.
- Execução: Define o comando de entrada (npm run dev) automaticamente.

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

# Expõe a porta 3000
EXPOSE 3000

# Comando para iniciar
CMD ["npm", "run", "dev"]
```

# Balanceamento de Carga



- Atua como Proxy Reverso e Balanceador de Carga na frente da aplicação.
- Distribui as requisições HTTP recebidas na porta 8080 entre as 3 réplicas da API.
- Utiliza o algoritmo Round-Robin por padrão para alternar entre os servidores.
- Garante transparência de acesso: o Frontend não sabe qual servidor respondeu.
- Aumenta a disponibilidade e permite escalabilidade horizontal.
- Alta Disponibilidade: se um container cair, os outros continuam respondendo.

```
nginx.conf
You, 3 days ago | 1 author (You)
1 events { worker_connections 1024; }
2
3     http {
4         upstream backend_servers {
5             server api:3000;
6         }
7
8         server {
9             listen 80;
10
11         location / {
12             proxy_pass http://backend_servers;
13             proxy_set_header Host $host;
14             proxy_set_header X-Real-IP $remote_addr;
15             proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
16         }
17     }
18 }
```

You, 3 days ago • first commit ...

# Cache Distribuído



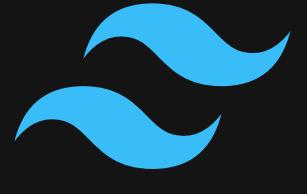
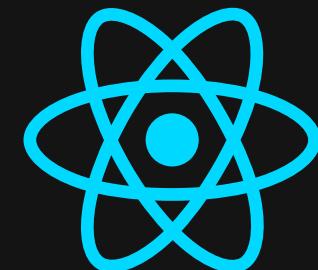
- Banco de dados em memória (In-memory) utilizado para cache de alta performance.
- Implementa a estratégia Cache-Aside: a aplicação verifica o cache antes do banco.
- Reduz drasticamente a latência de leitura do Feed.
- Diminui a carga (load) sobre o banco de dados principal (PostgreSQL).
- Possui mecanismo de Invalidação de Cache para manter a consistência dos dados.

```
const cachedFeed = await redis.get('feed_activities');
if (cachedFeed) {
  console.log('⚡ Retornando do Cache (Redis)');
  return res.status(status.OK).json(JSON.parse(cachedFeed));
}

console.log('🌐 Buscando no Banco de Dados...');
const activities = await activityService.getAllActivities();

await redis.set('feed_activities', JSON.stringify(activities), 'EX', 30);
```

# Frontend



- Single Page Application (SPA): Desenvolvido em React com TypeScript para alta interatividade.
- Desacoplamento: Totalmente independente do backend, comunicando-se apenas via API REST.
- Transparência de Acesso: O cliente desconhece a infraestrutura distribuída; ele aponta para o Gateway (NGINX na porta 8080).
- moderna e responsiva construída com Tailwind CSS.

```
src > services >  api.ts > ...
You, 3 days ago | 1 author (You)

1 import axios from "axios";
2
3 const api = axios.create({
4   baseURL: "http://localhost:8080",
5 });
6
7 api.interceptors.request.use((config) => {
8   const token = localStorage.getItem("token");
9
10  if (token) {
11    config.headers.Authorization = `Bearer ${token}`;
12  }
13
14  return config;
15 });
16
17 export default api;
18
```

Muito  
obrigado!