Introdução ao Desenvolvimento Backend da Solução PCD

Objetivo da aula

- Apresentar as tecnologias utilizadas no backend do projeto de empregabilidade para PCDs e iniciar a configuração do ambiente de desenvolvimento.

Visão geral do projeto

- Estamos desenvolvendo uma **API backend** para um sistema que conecta **empresas** e **candidatos PCD**, com foco em acessibilidade e inclusão.
  Tecnologias:

  - NodeJS: motor de execução do backend
  - TypeScript: linguagem segura e robusta com tipagem
  - Prisma: ORM moderno que facilita acesso ao banco de dados
  - PostgreSQL: banco relacional de dados aberto e robusto

Conceitos fundamentais das tecnologias

- Node.js
  - Ambiente de execução JavaScript fora do navegador
  - Ideal para APIs web, serviços RESTful e microserviços
  - Não é uma linguagem, mas sim uma plataforma

Conceitos fundamentais das tecnologias

- TypeScript
  - Superset do JavaScript com tipagem estática
  - Ajuda na manutenção, legibilidade e produtividade
  - Utiliza .ts no lugar de .js

Conceitos fundamentais das tecnologias

- Prisma
  - ORM (Object Relational Mapping)
  - Escreve o modelo dos dados em arquivos .prisma
  - Gera consultas SQL automaticamente
  - Facilita migrations, seeds, CRUD, e integrações com TypeScript

Conceitos fundamentais das tecnologias

- PostgreSQL
  - Banco de dados relacional open source
  - Utilizado em sistemas corporativos e críticos
  - Oferece suporte a chaves estrangeiras, relacionamentos, funções e índices complexos

Ambiente de desenvolvimento

```
# Criar o projeto
mkdir pcd-backend
cd pcd-backend
npm init -y

# Instalar dependências
npm install express cors dotenv
npm install -D typescript ts-node-dev prisma
@types/node @types/express @types/cors
npm install @prisma/client
```

Ambiente de desenvolvimento

- npm init -y

  inicializa um novo projeto Node.js automaticamente, criando o arquivo package.json com valores padrão, sem precisar responder às perguntas interativas.

```
{
  "name": "nome-do-projeto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Ambiente de desenvolvimento

- **express**

  Framework web para Node.js – permite criar rotas e servidores HTTP

- **cors**

  Middleware que habilita o CORS (Cross-Origin Resource Sharing)

- **dotenv**

  Permite carregar variáveis de ambiente do arquivo .env

Ambiente de desenvolvimento

- **typescript**

  Transpila .ts (TypeScript) para .js
- **ts-node-dev**

  Roda o projeto TypeScript com reinício automático em cada mudança
- **prisma**

  CLI do Prisma para modelagem, migração e seed do banco
- **@types/node**

  Tipagem TypeScript para recursos do Node.js (fs, path, etc.)
- **@types/express**

  Tipagem TypeScript para o Express
- **@types/cors**

  Tipagem TypeScript para o pacote CORS

Ambiente de desenvolvimento

@prisma/client

- É o cliente gerado pelo Prisma
- Permite usar o banco de dados com acesso tipado em tempo de execução
- Deve ser instalado como dependência normal, pois é usado em produção

Ambiente de desenvolvimento

```
# Inicializar TypeScript e Prisma
npx tsc --init
npx prisma init
```

Ambiente de desenvolvimento

- `npx tsc --init`
  - Cria o arquivo de configuração do TypeScript chamado `tsconfig.json`

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "rootDir": "src",
    "outDir": "dist",
    "strict": true,
    "esModuleInterop": true
  }
}
```

Campo	Função
`target`	Versão de JS gerada (ES2020, ES6 etc.)
`module`	Sistema de módulos (CommonJS para Node.js)
`rootDir`	Onde ficam os arquivos `.ts` (geralmente `src/`)
`outDir`	Para onde os arquivos `.js` compilados serão enviados
`strict`	Ativa verificação estrita de tipos
`esModuleInterop`	Permite usar imports como `import express from 'express'`

Ambiente de desenvolvimento

- npx prisma init
  - Inicializa a estrutura básica do Prisma no projeto. Ele cria:

1. prisma/
└── schema.prisma

Esse arquivo é onde você define seu modelo de dados (entidades, relacionamentos) usando a linguagem do Prisma.

2. Um arquivo .env na raiz do projeto com:

DATABASE_URL="postgresql://usuario:senha@localhost:5432/nomedobanco"

Estrutura básica da aplicação

```
src/
    controllers/
    services/
    routes/
    models/
    index.ts
prisma/
    schema.prisma
.env
```

Configuração do banco

No arquivo .env

```
DATABASE_URL="postgresql://usuario:senha@localhost:5432/pcd_db"
```

Workflow básico de desenvolvimento com Prisma

```
# Após definir o schema.prisma:
npx prisma migrate dev --name init
npx prisma generate
```

Workflow básico de desenvolvimento com Prisma

- npx prisma migrate dev --name init
  - Cria e aplica uma migração no banco de dados com base nas alterações no seu schema.prisma.
  - O que ele faz, passo a passo:
    - Cria uma pasta prisma/migrations com o nome da migração (ex: init)
    - Gera os comandos SQL para criar/alterar tabelas
    - Aplica automaticamente a migração no banco configurado na DATABASE_URL
    - Atualiza o histórico de migrações (útil para versionamento e deploy)
    - Executa o prisma generate ao final (exemplo)

Workflow básico de desenvolvimento com Prisma

```
model Empresa {
    id    Int    @id @default(autoincrement())
    nome  String
    email String @unique
}
```

Esse comando irá:

- Criar uma tabela Empresa
- Rodar o SQL no PostgreSQL
- Registrar o histórico da mudança

Workflow básico de desenvolvimento com Prisma

npx prisma generate

Gera o cliente Prisma com base no schema.prisma.

Cria pasta:

node_modules/.prisma/client

E habilita o uso em seu código com:

```
import { PrismaClient } from '@prisma/client’
const prisma = new PrismaClient()
```

Workflow básico de desenvolvimento com Prisma

- Ordem de execução ideal
  - # Gera + aplica + já executa generate
  - npx prisma migrate dev --name init

  - # ou, se você só mudar o schema e não quiser migrar:
  - npx prisma generate # Só atualiza o cliente Prisma