Descrição da atividade: fazer uma aplicação back-end para persistir dados de usuários e os departamentos que eles trabalham.

Data de entrega: presencialmente na aula de 29/nov (terça-feira).

Forma de entrega: individual.

Requisitos funcionais:

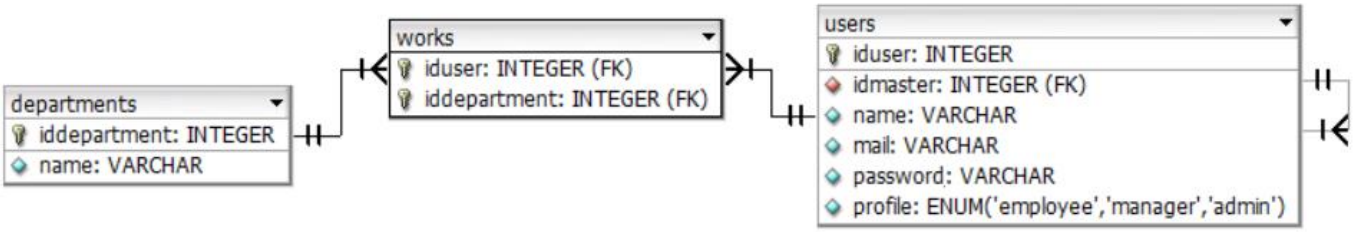
i. O usuário deverá efetuar o login usando e-mail e senha. O servidor deverá retornar um token e as demais requisições deverão utilizar esse token para a autenticação do usuário;

ii. Os usuários possuem os perfis de empregado (employee), gestor (manager) e administrador (admin). Cada perfil de usuário possui um nível de acesso;

iii. Um usuário (empregado, gestor ou administrador) pode trabalhar em vários departamentos e um departamento poderá ter vários empregados;

iv. Um usuário com perfil employee poderá ter um gestor (um usuário com perfil manager);

v. Deverá ser utilizado o seguinte modelo de dados;



vi. A aplicação deverá ter as seguintes rotas:

- POST: /login com os parâmetros mail e password;
- POST: /user com os parâmetros name, mail, password, profile, idmaster e departments (os ids dos departamentos que o empregado trabalha – não terá inserção na tabela works caso não seja fornecido ids de departamentos);
- PUT: /user com os parâmetros iduser, name, mail, password, profile, idmaster e departments. Dados do usuário a ser alterado;
- PUT: /user/password com o parâmetro password. Para atualizar a senha do usuário atualmente logado;
- DELETE: /user com o parâmetro iduser. Para excluir o usuário;
- GET: /user/:profile/:department. Os parâmetros são passados na URL. :profile pode ser employee (para listar somente os usuários com profile igual a employee), manager (para listar somente os usuários com profile manager) e admin (para listar somente os usuários com profile admin). O parâmetro department pode ser true ou false, true significa que a requisição retornará também os departamentos que o empregado trabalha;
- POST: /department com o parâmetro name. Para criar um departamento;

- PUT: /department com os parâmetros iddepartment e name. Para atualizar o departamento;
- DELETE: /department com o parâmetro iddepartment. Para excluir o departamento;
- GET: /department/:user. O parâmetro :user é fornecido pela URL. :user pode ser true ou false, true significa que a requisição retornará também os dados dos empregados trabalham no departamento.

vii. As rotas possuem as seguintes restrições de acesso:

- POST: /login sem restrição de acesso;
- POST: /user somente usuários com perfil manager;
- PUT: /user somente usuários com perfil manager;
- PUT: /user/password somente usuários logados;
- DELETE: /user somente usuários com perfil manager;
- GET: /user/:profile/:department somente usuários logados;
- POST: /department somente usuários com perfil admin;
- PUT: /department somente usuários com perfil admin;
- DELETE: /department somente usuários com perfil admin;
- GET: /department/:user somente usuários logados.

Requisitos não funcionais:

i. Deverá ser utilizada a linguagem TypeScript;
ii. Deverá ser utilizado o ORM TypeORM;
iii. Deverá ser utilizado o SGBD PostgreSQL;
iv. Deverá ser utilizado JWT para a autenticação;
v. O relacionamento entre as tabelas users e departments deverá ser bidirecional, ou seja, de ambos os lados será possível listar os relacionamentos. Utilize a anotation ManyToMany de ambos os lados, mas somente a entidade user terá JoinTable (o controle da relação bidirecional);
vi. As tabelas, relacionamentos e restrições deverão ser criadas no SGBD usando migrations.

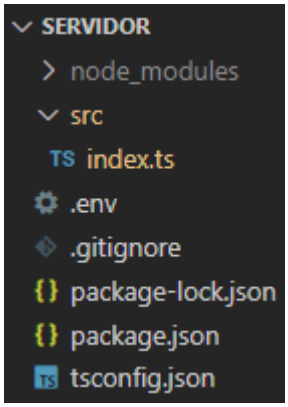
Passos para criar e configurar o projeto da aplicação sugere-se os seguinte passos:

1. Crie uma pasta de nome servidor no local de sua preferência do computador;
2. Abra a pasta servidor no VS Code;
3. Acesse o terminal do VS Code e digite o comando npm init -y para iniciar o projeto Node criando o arquivo package.json;
4. Digite npm i express e npm i @types/express -D para adicionar o pacote express e o suporte para os tipos de dados do express. O parâmetro -D indica que o pacote será instalado como dependência de desenvolvimento;
5. Digite npm i typescript -D para instalar o TypeScript como dependência de desenvolvimento;

6. Digite npm i ts-node ts-node-dev -D para instalar as bibliotecas que dão suporte a codificação em TypeScript como dependência de desenvolvimento;
7. Digite npm i pg para instalar o pacote para acessar o SGBD PostgreSQL;
8. Digite npm i cors e npm i @types/cors -D para adicionar o pacote CORS (Cross-Origin Resource Sharing) para suportar requisições de outros domínios;
9. Digite npm i jsonwebtoken para adicionar o pacote JWT para autenticação de tokens;
10. Digite npm i bcrypt e npm i @types/bcrypt -D para adicionar o pacote utilizado para codificar senhas;
11. Digite npm i typeorm para instalar o pacote do typeorm;
12. Certifique-se que o comando tsc esteja disponível no seu computador digitando tsc -v. Talvez você precise instalar o TypeScript globalmente npm i typescript -g para o tsc funcionar corretamente.
    Como estamos utilizando o TypeScript, então precisaremos inicializar ele dentro da aplicação. Digite tsc --init para gerar o arquivo tsconfig.json no projeto;
13. Substitua o arquivo tsconfig.json para ter as seguintes propriedades. As propriedades emitDecoratorMetadata e experimentalDecorators são necessárias usar o TypeORM:

```
{
  "compilerOptions": {
      "target": "es2021",
      "module": "commonjs",
      "moduleResolution": "node",
      "emitDecoratorMetadata": true,
      "experimentalDecorators": true
  }
}
```

14. Crie a pasta src para organizarmos o sistema de pastas e arquivos da aplicação. Ao lado tem-se a estrutura de pasta da aplicação;



15. Crie o arquivo .gitignore e coloque dentro dele apenas node_modules para ignorar essa pasta do projeto;
16. Creio o arquivo .env para colocarmos as variáveis de ambiente da aplicação. Dentro desse arquivo coloque a seguinte variável:

```
PORT = 3001
```

17. Digite npm i dotenv para termos acesso as variáveis de ambiente que colocaremos no arquivo .env;
18. Crie o arquivo src/index.ts e coloque o código a seguir para criarmos um servidor express:

```
import * as express from "express";
import * as cors from "cors";
import * as dotenv from "dotenv";
dotenv.config(); // precisa ser chamado no início do arquivo
const PORT = process.env.PORT || 3000; // lê a variável PORT do arquivo .env
const app = express(); // cria o servidor
```

```
app.use(express.json()); // suporta parâmetros JSON no body da requisição
app.use(cors()); // suporta requisições de qualquer domínio
// inicializa o servidor na porta especificada
app.listen(PORT, () => console.log(`Rodando na porta ${PORT}`));
```

Observação: ao rodar o projeto pode ser emitido a seguinte mensagem de erro. Ela ocorre pela propriedade esModuleInterop: true, do arquivo tsconfig.json estar setada para true, mude para false. Para mais detalhes veja https://stackoverflow.com/questions/49256040/a-namespace-style-import-cannot-be-called-or-constructed-and-will-cause-a-failu

```
src/index.ts:1:1
  1 import * as express from "express"
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  Type originates at this import. A namespace-style import cannot be called or constructed, and will
  cause a failure at runtime. Consider using a default import or import require here instead.
```

19. Substitua a propriedade scripts do arquivo de configuração do projeto package.json. Veja que foram adicionados os comandos para criar o arquivo de migração (npm run migration:generate), criar um arquivo de migração vazio (npm run migration:create), submeter o arquivo de migração para o SGBD (npm run migration:run), desfazer a última migração submetida no SGBD (npm run migration:revert) e rodar o projeto em modo de desenvolvimento (npm run dev) e em modo de produção (npm run start):

```
"scripts": {
  "dev": "ts-node-dev src/index.ts",
  "start": "ts-node src/index.ts",
  "migration:generate": "typeorm-ts-node-commonjs -d ./src/data-source.ts migration:generate
./src/migrations/default",
  "migration:create": "npx typeorm-ts-node-commonjs migration:create
./src/migrations/default",
  "migration:run": "typeorm-ts-node-commonjs -d ./src/data-source.ts migration:run",
  "migration:revert": "typeorm-ts-node-commonjs -d ./src/data-source.ts migration:revert"
},
```

Utilize o comando a seguir para testar o arquivo src/index.ts e o script de configuração do package.json:

```
PS D:\pessoal\servidor> npm run dev

> servidor@1.0.0 dev
> ts-node-dev src/index.ts

[INFO] 16:46:35 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.1, typescript ver. 4.8.4)
Rodando na porta 3001
```

20. Crie o arquivo src/data-source.ts para manter a configuração de conexão com o SGBD PostgresSQL. O código deste arquivo está disponível em https://github.com/arleysouza/api-scripts-2022-2-back. Lembre-se que o código desse exemplo faz a conexão um BD de nome bdaula que está no SGBD PostgreSQL da sua máquina. Será necessário que você altere esses parâmetros para o seu BD.

21. Organize o projeto nas pastas src/controllers, src/entitities, src/middlewares e src/routes assim como sugerido em https://github.com/arleysouza/api-scripts-2022-2-back.