

Criando o projeto:

1. Pasta: **escola**
2. Comando: **npm init -y**
3. Comando: **npm i express dotenv mongoose**
4. Comando **npm i -D @types/express**
5. Comando **npm i -D ts-node ts-node-dev typescript**
6. Comando **tsc --init**
7. Arquivo: **.gitignore** ignorar a pasta **node_modules**
8. Arquivo: **.env** variável de ambiente na pasta raiz:
PORT=3001
9. Propriedades no arquivo **package.json**:


```
"scripts": {
  "start": "ts-node ./src",
  "dev": "ts-node-dev ./src"
```

10. Arquivo **connection.ts**: conexão com BD do MongoDB.

Arquivo: **src/models/connection.ts**

```
import mongoose from "mongoose";

// A URI indica o IP, a porta e BD a ser conectado
const uri = "mongodb://127.0.0.1:27017/bdaula";

export default function connect() {
  // Configura manipuladores de eventos para diferentes estados de conexão
  // cada mensagem de log indica um estado específico da conexão.
  // É opcional configurar os manipuladores de estado,
  // mas é interessante para sabermos sobre a conexão
  mongoose.connection.on("connected", () => console.log("connected"));
  mongoose.connection.on("open", () => console.log("open"));
  mongoose.connection.on("disconnected", () => console.log("disconnected"));
  mongoose.connection.on("reconnected", () => console.log("reconnected"));
  mongoose.connection.on("disconnecting", () => console.log("disconnecting"));
  mongoose.connection.on("close", () => console.log("close"));
  // Utiliza o método connect do Mongoose para estabelecer a conexão com o MongoDB, usando a URI
  mongoose
    .connect(uri, {
      serverSelectionTimeoutMS: 5000,
      maxPoolSize: 10,
    })
    .then(() => console.log("Conectado ao MongoDB"))
    .catch((e) => {
      console.error("Erro ao conectar ao MongoDB:", e.message);
    });
}

// o sinal SIGINT é disparado ao encerrar a aplicação, geralmente, usando Ctrl+C
process.on("SIGINT", async () => {
  try {
    console.log("Conexão com o MongoDB fechada");
    await mongoose.connection.close();
    process.exit(0);
  } catch (error) {
    console.error("Erro ao fechar a conexão com o MongoDB:", error);
    process.exit(1);
  }
});
```

11. arquivo **src/index.ts**: ervidor express/conexão com uma instância do MongoDB

Arquivo: src/index.ts

```
import express from "express";
import routes from './routes';
import dotenv from "dotenv";
import connect from "./models/connection";
dotenv.config();

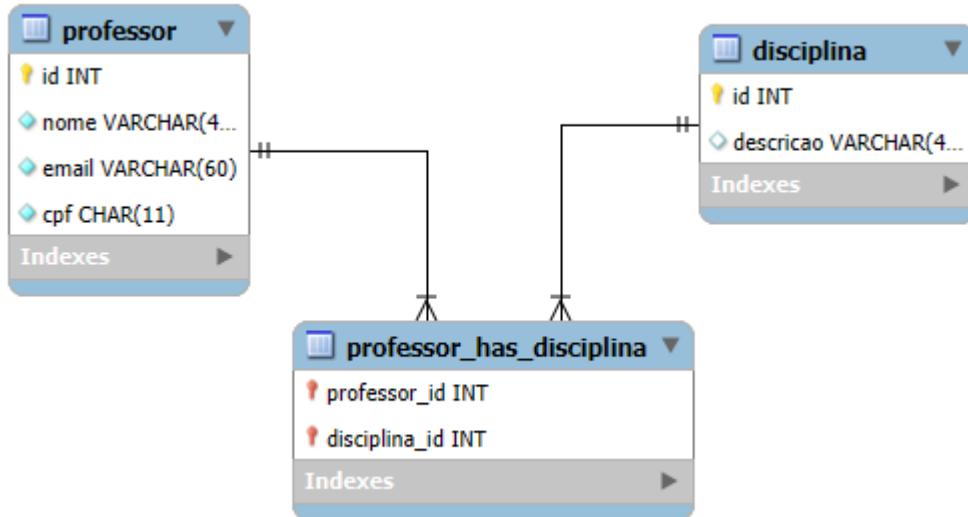
// será usado 3000 se a variável de ambiente não tiver sido definida
const PORT = process.env.PORT || 3000;
const app = express(); // cria o servidor e coloca na variável app
// suportar parâmetros JSON no body da requisição
app.use(express.json());

// conecta ao MongoDB no início da aplicação
connect();

// inicializa o servidor na porta especificada
app.listen(PORT, () => {
  console.log(`Rodando na porta ${PORT}`);
});

// define a rota para o pacote /routes
app.use(routes);
```

12. Arquivo **models/index.ts**: esquemas e modelos. Usaremos o seguinte MER para criarmos as coleções no MongoDB:



Arquivo: src/models/index.ts

```

import mongoose from "mongoose";
const { Schema } = mongoose;
import { isValidCPF } from "./ValidaCPF";

// define os schemas

const ProfessorSchema = new Schema({
  nome: {
    type: String,
    maxlength: [45, "O nome do professor pode ter no máximo 45 caracteres"],
    required: [true, "O nome do professor é obrigatório"],
  },
  email: {
    type: String,
    maxlength: [60, "O e-mail pode ter no máximo 60 caracteres"],
    unique: true,
    required: [true, "O e-mail é obrigatório"],
    validate: {
      validator: function (value: string) {
        // expressão regular para validar o formato do e-mail
        const regex = /^[^s@]+@[^\s@]+\.[^\s@]+$/;
        return regex.test(value);
      },
      message: (props: any) =>
        `${props.value} não é um formato de e-mail válido`,
    },
  },
  cpf: {
    type: String,
    trim: true,
    minlength: [11, "O CPF precisa ter no mínimo 6 caracteres"],
    maxlength: [11, "O CPF precisa ter no máximo 10 caracteres"],
    required: [true, "O CPF é obrigatório"],
    validate: {
      validator: function (value: string) {
        return isValidCPF(value);
      },
      message: (props: any) =>
        `${props.value} não é um CPF válido`,
    },
  },
});

const DisciplinaSchema = new Schema({
  descricao: {
    type: String,
    maxlength: [45, "A descrição da disciplina pode ter no máximo 45 caracteres"],
    required: [true, "A descrição da disciplina é obrigatória"],
  },
});

const Professor = mongoose.model("Professor", ProfessorSchema, "professores");
const Disciplina = mongoose.model("Disciplina", DisciplinaSchema);

const Professor_has_DisciplinaSchema = new Schema({
  professor: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Professor",
    required: true,
    validate: {
      validator: async function (id: string) {

```

```

        const professor = await Professor.findById(id); // verifica se id existe na
coleção professores
        return !!professor; // true se o usuário existir
    },
    message: 'O ID do professor fornecido não existe',
},
disciplina: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Disciplina",
    required: true,
    validate: {
        validator: async function (id: string) {
            const disciplina = await Disciplina.findById(id); // verifica se id existe na
coleção disciplinas
            return !!disciplina; // true se o usuário existir
        },
        message: 'O ID da disciplina fornecido não existe',
    },
},
});

// mongoose.model compila o modelo
const Professor_has_Disciplina = mongoose.model("Professor_has_Disciplina",
Professor_has_DisciplinaSchema);

export { Professor, Disciplina, Professor_has_Disciplina };

```

13. Arquivo **models/validCPF.ts**: validação do CPF

Arquivo: src/models/validaCPF.ts

```

export function isValidCPF(value: string): boolean {
    if (typeof value !== 'string') {
        return false;
    }

    value = value.replace(/[^\d]+/g, '');

    if (value.length !== 11 || !value.match(/(\d)\1{10}/)) {
        return false;
    }

    const values = value.split('').map(el => +el);
    const rest = (count: number) => (values.slice(0, count - 12).reduce((soma, el, index) =>
(soma + el * (count - index)), 0) * 10) % 11 % 10;

    return rest(10) === values[9] && rest(11) === values[10];
}

```

14. Arquivo **controllers/UserController.ts**: CRUD (Create, Read, Update e Delete) na coleção Professores.

Arquivo: src/controllers/ProfessorController.ts

```
import { Request, Response } from "express";
import { Professor } from "../models";

class ProfessorController {

    public async create(req: Request, res: Response): Promise<any> {
        const { nome, email, cpf } = req.body;
        try {
            //a instância de um modelo é chamada de documento
            const document = new Professor({ nome, email, cpf });
            // ao salvar serão aplicadas as validações do esquema
            const resp = await document.save();
            return res.json(resp);
        } catch (error: any) {
            if (error.code === 11000 || error.code === 11001) {
                // código 11000 e 11001 indica violação de restrição única (índice duplicado)
                return res.json({ message: "CPF ou e-Mail já em uso" });
            } else if (error && error.errors["nome"]) {
                return res.json({ message: error.errors["nome"].message });
            } else if (error && error.errors["email"]) {
                return res.json({ message: error.errors["email"].message });
            } else if (error && error.errors["cpf"]) {
                return res.json({ message: error.errors["cpf"].message });
            }
            return res.json({ message: error.message });
        }
    }

    public async list(_: Request, res: Response): Promise<any> {
        try {
            const objects = await Professor.find().sort({ mail: "asc" });
            return res.json(objects);
        } catch (error: any) {
            return res.json({ message: error.message });
        }
    }

    public async delete(req: Request, res: Response): Promise<any> {
        const { id: _id } = req.body; // _id do registro a ser excluído
        try {
            const object = await Professor.findByIdAndDelete(_id);
            if (object) {
                return res.json({ message: "Professor excluído com sucesso" });
            } else {
                return res.json({ message: "Professor inexistente" });
            }
        } catch (error: any) {
            return res.json({ message: error.message });
        }
    }

    public async update(req: Request, res: Response): Promise<any> {
        const { id, nome, email, cpf } = req.body;
        try {
            // busca o usuário existente na coleção antes de fazer o update
            const document = await Professor.findById(id);
            if (!document) {
                return res.json({ message: "Professor inexistente" });
            }
        }
    }
}
```

```

        // atualiza os campos
        document.nome = nome;
        document.email = email;
        document.cpf = cpf;
        // ao salvar serão aplicadas as validações do esquema
        const resp = await document.save();
        return res.json(resp);
    } catch (error: any) {
        if (error.code === 11000 || error.code === 11001) {
            // código 11000 e 11001 indica violação de restrição única (índice duplicado)
            return res.json({ message: "CPF ou e-Mail já em uso" });
        } else if (error && error.errors["nome"]) {
            return res.json({ message: error.errors["nome"].message });
        } else if (error && error.errors["email"]) {
            return res.json({ message: error.errors["email"].message });
        } else if (error && error.errors["cpf"]) {
            return res.json({ message: error.errors["cpf"].message });
        }
        return res.json({ message: error.message });
    }
}

export default new ProfessorController();

```

15. Arquivo **controllers/DisciplinaController.ts**: CRUD (Create, Read, Update e Delete) na coleção disciplinas.

Arquivo: src/controllers/DisciplinaController.ts

```

import { Request, Response } from "express";
import { Disciplina } from "../models";

class DisciplinaController {

    public async create(req: Request, res: Response): Promise<any> {
        const { descricao } = req.body;
        try {
            //a instância de um modelo é chamada de documento
            const document = new Disciplina({ descricao });
            // ao salvar serão aplicadas as validações do esquema
            const resp = await document.save();
            return res.json(resp);
        } catch (error: any) {
            if (error && error.errors["descricao"]) {
                return res.json({ message: error.errors["descricao"].message });
            }
            return res.json({ message: error.message });
        }
    }

    public async list(_: Request, res: Response): Promise<any> {
        try {
            const objects = await Disciplina.find().sort({ mail: "asc" });
            return res.json(objects);
        } catch (error: any) {
            return res.json({ message: error.message });
        }
    }

    public async delete(req: Request, res: Response): Promise<any> {

```

```

const { id: _id } = req.body; // _id do registro a ser excluído
try {
  const object = await Disciplina.findByIdAndDelete(_id);
  if (object) {
    return res.json({ message: "Disciplina excluída com sucesso" });
  } else {
    return res.json({ message: "Disciplina inexistente" });
  }
} catch (error: any) {
  return res.json({ message: error.message });
}

public async update(req: Request, res: Response): Promise<any> {
  const { id, descricao } = req.body;
  try {
    // busca o documento existente na coleção antes de fazer o update
    const document = await Disciplina.findById(id);
    if (!document) {
      return res.json({ message: "Disciplina inexistente" });
    }
    // atualiza os campos
    document.descricao = descricao;
    // ao salvar serão aplicadas as validações do esquema
    const resp = await document.save();
    return res.json(resp);
  } catch (error: any) {
    if (error && error.errors["descricao"]) {
      return res.json({ message: error.errors["descricao"].message });
    }
    return res.json({ message: error.message });
  }
}
export default new DisciplinaController();

```

16. Arquivo **controllers/Professor_has_DisciplinaController.ts**: CRUD (Create, Read, Update e Delete) na coleção professor_has_disciplinas.

Arquivo: src/controllers/Professor_has_DisciplinaController.ts

```

import { Request, Response } from "express";
import { Professor_has_Disciplina } from "../models";

class Professor_has_DisciplinaController {

  // create

  public async create(req: Request, res: Response): Promise<any> {
    const { professor, disciplina } = req.body;
    try {
      const document = new Professor_has_Disciplina({ professor, disciplina });
      // ao salvar serão aplicadas as validações do esquema
      const response = await document.save();
      return res.json(response);
    } catch (error: any) {
      if (error && error.errors["professor"]) {
        return res.json({ message: error.errors["professor"].message });
      }
    }
  }
}

```

```

        } else if (error && error.errors["disciplina"]) {
            return res.json({ message: error.errors["disciplina"].message });
        }
    return res.json({ message: error });
}

// list

public async list(_: Request, res: Response): Promise<any> {
    try {
        const objects = await Professor_has_Disciplina.find()
            .populate("professor")
            .populate("disciplina")
            .select("professor disciplina")
            .sort({ nome: "asc" });
        return res.json(objects);
    } catch (error: any) {
        return res.json({ message: error.message });
    }
}

// delete

public async delete(req: Request, res: Response): Promise<any> {
    const { id: _id } = req.body; // _id do registro a ser excluído
    try {
        const object = await Professor_has_Disciplina.findByIdAndDelete(_id);
        if (object) {
            return res.json({ message: "Registro excluído com sucesso" });
        } else {
            return res.json({ message: "Registro inexistente" });
        }
    } catch (error: any) {
        return res.json({ message: error.message });
    }
}

// update

public async update(req: Request, res: Response): Promise<any> {
    const { id, professor, disciplina } = req.body;
    try {
        // busca o registro existente na coleção antes de fazer o update
        const document = await Professor_has_Disciplina.findById(id);
        if (!document) {
            return res.json({ message: "Registro inexistente!" });
        }
        // atualiza os campos
        document.professor = professor;
        document.disciplina = disciplina;
        // ao salvar serão aplicadas as validações do esquema
        const response = await document.save();
        return res.json(response);
    } catch (error: any) {
        if (error && error.errors["professor"]) {
            return res.json({ message: error.errors["professor"].message });
        } else if (error && error.errors["disciplina"]) {
            return res.json({ message: error.errors["disciplina"].message });
        }
        return res.json({ message: error });
    }
}

```

```
}
```

```
export default new Professor_has_DisciplinaController();
```

17. Arquivos: **professores.ts**, **disciplinas.ts** e **professor_has_disciplinas.ts**: definição das as rotas.

Arquivo: src/routes/professores.ts

```
import { Router } from "express";
import controller from "../controllers/ProfessorController";

const routes = Router();

routes.post('/', controller.create);
routes.get('/', controller.list);
routes.delete('/', controller.delete);
routes.put('/', controller.update);

export default routes;
```

Arquivo: src/routes/disciplinas.ts

```
import { Router } from "express";
import controller from "../controllers/DisciplinaController";

const routes = Router();

routes.post('/', controller.create);
routes.get('/', controller.list);
routes.delete('/', controller.delete);
routes.put('/', controller.update);

export default routes;
```

Arquivo: src/routes/professor_has_disciplinas.ts

```
import { Router } from "express";
import controller from "../controllers/Professor_has_DisciplinaController";

const routes = Router();

routes.post('/', controller.create);
routes.get('/', controller.list);
routes.delete('/', controller.delete);
routes.put('/', controller.update);

export default routes;
```

18. Arquivo **routes/index.ts**: Rotas.

Arquivo: src/routes/index.ts

```
import { Router, Request, Response } from "express";
import professor from './professores';
import disciplina from './disciplinas';
import professor_has_disciplina from './professor_has_disciplinas';

const routes = Router();

routes.use("/professor", professor);
routes.use("/disciplina", disciplina);
routes.use("/professor_has_disciplina", professor_has_disciplina);

//aceita qualquer método HTTP ou URL
routes.use(_: any, res: any) => res.json({ error: "Requisição desconhecida" }));

export default routes;
```

19. Testando Cadastro de Professores:

Comando: curl -X POST http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"nome\": \"Henrique Louro\", \"email\": \"henrique.louro@fatec.sp.gov.br\", \"cpf\": \"07494812857\"}"

Resposta: {"nome": "Henrique Louro", "email": "henrique.louro@fatec.sp.gov.br", "cpf": "07494812857", "_id": "682f6384f4bd0fb518a18a28", "__v": 0}

Comando: curl -X POST http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"nome\": \"Carlos Silva\", \"email\": \"carlos.silva@fatec.sp.gov.br\", \"cpf\": \"63479695051\"}"

Resposta: {"nome": "Carlos Silva", "email": "carlos.silva@fatec.sp.gov.br", "cpf": "63479695051", "_id": "682f6623f4bd0fb518a18a2c", "__v": 0}

Comando: curl -X POST http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"nome\": \"Odete Roitman\", \"email\": \"odete.roitman@fatec.sp.gov.br\", \"cpf\": \"32082128016\"}"

Resposta: {"nome": "Odete Roitman", "email": "odete.roitman@fatec.sp.gov.br", "cpf": "32082128016", "_id": "682f6b14f4bd0fb518a18a37", "__v": 0}

20. Testando Unique no Cadastro de Professores:

Comando: curl -X POST http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"nome\": \"Henrique Louro\", \"email\": \"henrique.louro@fatec.sp.gov.br\", \"cpf\": \"07494812857\"}"

Resposta: {"message": "CPF ou e-Mail já em uso"}

21. Testando validação do CPF:

Comando: curl -X POST http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"nome\": \"Henrique Louro\", \"email\": \"henrique.louro@fatec.sp.gov.br\", \"cpf\": \"12345678910\"}"

Resposta: {"message": "12345678910 não é um CPF válido"}

22. Testando validação do e-Mail:

Comando: curl -X POST http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"nome\": \"Henrique Louro\", \"email\": \"henrique.louro@fatec\", \"cpf\": \"12345678910\"}"

Resposta: {"message": "henrique.louro@fatec não é um formato de e-mail válido"}

23. Listando Professores:

Comando: curl -X GET http://localhost:3000/professor
Resposta: [{"_id": "682f6384f4bd0fb518a18a28", "nome": "Henrique Louro", "email": "henrique.louro@fatec.sp.gov.br", "cpf": "07494812857", "__v": 0}, {"_id": "682f6623f4bd0fb518a18a2c", "nome": "Carlos Silva", "email": "carlos.silva@fatec.sp.gov.br", "cpf": "63479695051", "__v": 0}, {"_id": "682f6b14f4bd0fb518a18a37", "nome": "Odete Roitman", "email": "odete.roitman@fatec.sp.gov.br", "cpf": "32082128016", "__v": 0}]

24. Update Professor:

Comando: curl -X PUT http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"id\": \"682f669ff4bd0fb518a18a2e\", \"nome\": \"Odetinha Roitman\", \"email\": \"odetinha.roitman@fatec.sp.gov.br\", \"cpf\": \"32082128016\"}"
Resposta: {"_id": "682f6b14f4bd0fb518a18a37", "nome": "Odetinha Roitman", "email": "odetinha.roitman@fatec.sp.gov.br", "cpf": "32082128016", "__v": 0}

25. Deletando Professor:

Comando: curl -X DELETE http://localhost:3000/professor -H "Content-Type: application/json" -d "{\"id\": \"682f6b14f4bd0fb518a18a37\"}"
Resposta: {"message": "Professor excluído com sucesso"}

26. Listando Professores novamente:

Comando: curl -X GET http://localhost:3000/professor
Resposta: [{"_id": "682f6384f4bd0fb518a18a28", "nome": "Henrique Louro", "email": "henrique.louro@fatec.sp.gov.br", "cpf": "07494812857", "__v": 0}, {"_id": "682f6623f4bd0fb518a18a2c", "nome": "Carlos Silva", "email": "carlos.silva@fatec.sp.gov.br", "cpf": "63479695051", "__v": 0}]

27. Testando Cadastro de Disciplinas:

Comando: curl -X POST http://localhost:3000/disciplina -H "Content-Type: application/json" -d "{\"descricao\": \"Técnicas de Programação II\"}"
Resposta: {"descricao": "Técnicas de Programação II", "_id": "682f6d0cf4bd0fb518a18a3c", "__v": 0}
Comando: curl -X POST http://localhost:3000/disciplina -H "Content-Type: application/json" -d "{\"descricao\": \"Lógica de Programação\"}"
Resposta: {"descricao": "Lógica de Programação", "_id": "682f6dbdf4bd0fb518a18a3e", "__v": 0}

28. Listando as Disciplinas:

Comando: curl -X GET http://localhost:3000/disciplina
Resposta: [{"_id": "682f6d0cf4bd0fb518a18a3c", "descricao": "Técnicas de Programação II", "__v": 0}, {"_id": "682f6dbdf4bd0fb518a18a3e", "descricao": "Lógica de Programação", "__v": 0}]

29. Associando Professores às Disciplinas:

Comando: curl -X POST http://localhost:3000/professor_has_disciplina -H "Content-Type: application/json" -d "{\"professor\": \"682f6384f4bd0fb518a18a28\", \"disciplina\": \"682f6d0cf4bd0fb518a18a3c\"}"
Resposta: {"professor": "682f6384f4bd0fb518a18a28", "disciplina": "682f6d0cf4bd0fb518a18a3c", "_id": "682f78b5f4bd0fb518a18a43", "__v": 0}

Comando: curl -X POST http://localhost:3000/professor_has_disciplina -H "Content-Type: application/json" -d "{\"professor\": \"682f6623f4bd0fb518a18a2c\", \"disciplina\": \"682f6dbdf4bd0fb518a18a3e\"}"
Resposta: {"professor": "682f6623f4bd0fb518a18a2c", "disciplina": "682f6dbdf4bd0fb518a18a3e", "_id": "682f7963f4bd0fb518a18a47", "__v": 0}

30. Listando Professores e Disciplinas

Comando: curl -X GET http://localhost:3000/professor_has_disciplina

Resposta:

```
[{"_id": "682f78b5f4bd0fb518a18a43", "professor": {"_id": "682f6384f4bd0fb518a18a28", "nome": "Henrique Louro", "email": "henrique.louro@fatec.sp.gov.br", "cpf": "07494812857", "__v": 0}, "disciplina": {"_id": "682f6d0cf4bd0fb518a18a3c", "descricao": "Técnicas de Programação II", "__v": 0}}, {"_id": "682f7963f4bd0fb518a18a47", "professor": {"_id": "682f6623f4bd0fb518a18a2c", "nome": "Carlos Silva", "email": "carlos.silva@fatec.sp.gov.br", "cpf": "63479695051", "__v": 0}, "disciplina": {"_id": "682f6dbdf4bd0fb518a18a3e", "descricao": "Lógica de Programação", "__v": 0}]
```

Observação: Os códigos estão disponíveis no Git: <https://github.com/hdblouro/escola>