

Modelagem do Projeto Integrado

Psicotec



Cesar Medeiros Simionato -	19001440
Felipe Olmedo Vidolin Gonçalves -	17000902
Gian Henrique Benedito -	19000432
Giovanny Augusto de Oliveira -	19001444
Paulo Henrique Lisboa -	19000981

Modelo de entidade-relacionamento

Um **Administrador** possui zero ou vários **Pacientes**, que pode enviar zero ou várias **Mensagens**.

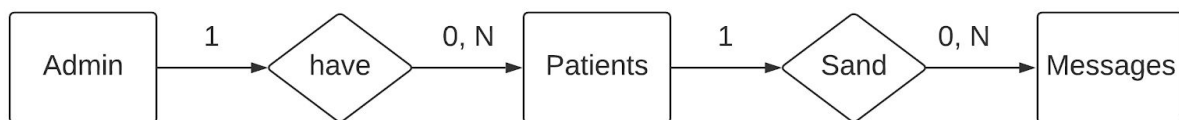
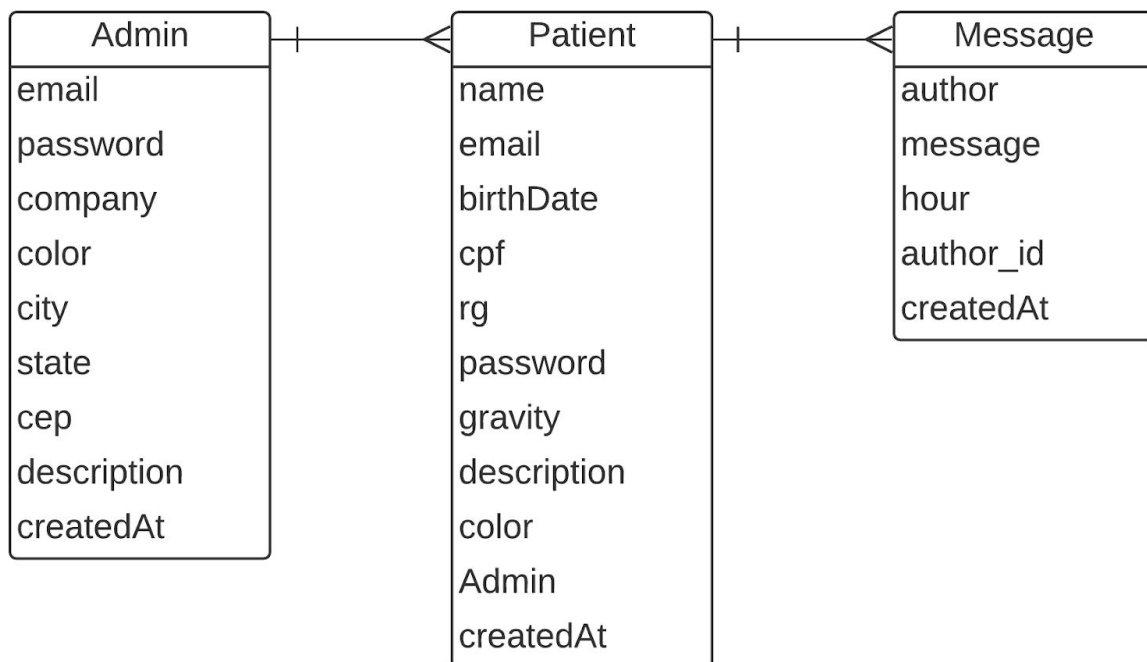


Diagrama de entidade-relacionamento



Nota: O campo `_id` é criado automaticamente em toda “tabela” do mongo, por isso não está presente no diagrama.

Schema Admin

```
email: {  
  type: String,  
  unique: true,  
  required: true,  
  lowercase: true  
},  
password: {  
  type: String,  
  required: true,  
  select: false  
},  
company: {  
  type: String,  
},  
color: {  
  type: String,  
},  
state: {  
  type: String,  
},
```

```
cep: {
  type: String,
},
description: {
  type: String,
},
createdAt: {
  type: Date,
  default: Date.now
}

// Antes de salvar transformamos a senha em um hash
AdminSchema.pre('save', async function(next) {
  const hash = await bcryptjs.hash(this.password, 10);
  this.password = hash;
  next();
})
```

Schema Patient

```
name: {  
  type: String  
},  
age: {  
  type: Number  
},  
cpf: {  
  type: String,  
  unique: true  
},  
rg: {  
  type: String  
},  
email: {  
  type: String,  
  unique: true,  
  required: true,  
  lowercase: true  
},
```

```
password: {
  type: String,
  required: true,
  select: false
},
gravity: {
  type: String,
  default: "green"
},
description: {
  type: String
},
color: {
  type: String,
  default: "blue"
},
admin: {
  type: mongoose.Schema.Types.ObjectId,
  ref: 'Admin',
  require: true
},
```

```
createdAt: {
  type: Date,
  default: Date.now
}
// Antes de salvar transformamos a senha em um hash
PatientsSchema.pre('save', async function(next) {
  const hash = await bcryptjs.hash(this.password, 10);
  this.password = hash;
  next();
})
```

Schema Message

```
author: {  
  type: String,  
  require: true  
},  
message: {  
  type: Object,  
  require: true,  
},  
hour: {  
  type: String,  
  require: true  
},  
author_id: {  
  type: String,  
  require: true  
},  
createdAt: {  
  type: Date,  
  default: Date.now  
}
```


Rotas auth

Criar administrador: Essa é uma rota não presente no nosso front-end, porém precisamos dela para criar nosso administrador, com a evolução do projeto teremos nosso próprio dashboard para fazer o gerenciamento dos Admins, mas por hora criamos de forma manual acessando a rota **/auth/register**.

Pegamos todos os dados através da requisição.

Pegamos o email isoladamente:

```
const { email } = req.body;
```

Verificamos se existe esse email no sistema, se sim retornamos um erro:

```
if(await Admin.findOne({ email }))  
    return res.status(400).send({ erro: 'O email já  
existe no sistema' })
```

Nota: findOne seria um **SELECT** email **FROM** Admin **WHERE** email = req.body.email;

Criamos o registro na “tabela”:

```
const admin = await Admin.create(req.body);
```

Nota: Admin.create seria **INSERT INTO** Admin(todos os campos) **VALUES**(req.body);

Autenticar administrador: Rota que verifica se o administrador tem ou não acesso a aplicação, rota **/auth/authenticate**.

Pegamos o email e password da requisição:

```
const {email, password} = req.body;
```

Verificamos e pegamos o administrador com o email recebido:

```
const admin = await Admin.findOne({ email  
}).select('+password');
```

Nota: findOne seria um **SELECT** email, password **FROM** Admin **WHERE** email = req.body.email **AND** password = req.body.password;

Nota: A senha do administrador é oculta em uma requisição, por isso usamos o .select para conseguir pegá-la.

Verifica se o email não está no sistema:

```
if(!admin)  
    return res.status(400).send({ erro: "Usuario nao encontrado" })
```

Verifica se a senha não corresponder com a do banco:

```
if(!await bcrypt.compare(password, admin.password))  
    return res.status(400).send({erro: "Senha invalida"})
```

Nota: bcrypt.compare é o responsável por criar o hash da senha digitada e comparar com o da senha salva no banco.

Autenticar paciente: Pacientes criados têm acesso ao chat, portanto temos que autenticá-los, rota `/auth/login` :

Pegamos o email e password da requisição:

```
const {email, password} = req.body;
```

Verificamos e pegamos o paciente com o email recebido:

```
const patients = await Patients.findOne({ email  
}).select('+password');
```

Nota: findOne seria um `SELECT` email, password `FROM` Patients `WHERE` email = req.body.email `AND` password = req.body.password;

Nota: A senha do paciente é oculta em uma requisição, por isso usamos o `.select` para conseguir pegá-la.


Verifica se o email não esta no sistema:

```
if(!patients)  
    return res.status(400).send({ erro: "Usuario nao  
encontrado" })
```

Verifica se a senha não corresponder com a do banco:

```
if(!await bcrypt.compare(password, patients.password))  
    return res.status(400).send({ erro: "Senha invalida"  
})
```

Nota: `bcrypt.compare` é o responsavel por criar o hash da senha digitada e comparar com o da senha salva no banco.



Visualizar administrador: Para mostrar para o administrador os dados do mesmo temos a rota **/auth/admin/:adminId** :

Pegamos o id do administrador através do parâmetro da URL:

```
const admin = await  
Admin.findById(req.params.adminId);
```

Nota: findById seria o mesmo que **SELECT * FROM Admin WHERE _id = req.params.adminId;**

Rotas dashboard

Visualizar pacientes: rota `/dashboard/patients` :

```
const patients = await
Patients.find().populate('admin');
```

Nota: find seria o mesmo que `SELECT * FROM` patients;

Nota: populate é usado para trazer os dados de quem criou o paciente.

Visualizar paciente: Buscar um paciente em específico, rota `/dashboard/patients/:patientsId` :

Pegamo paciente através do parâmetro que vem da URL:

```
const patient = await
Patients.findById(req.params.patientsId).populate('admin');
```

Nota: findById seria o mesmo que `SELECT * FROM Patients WHERE _id = req.params.patientsId`;

Nota: populate é usado para trazer os dados de quem criou o paciente.

Criar paciente: , rota `/dashboard/patients/` :

Pegamos os dados do paciente e do administrador através da requisição.

Criamos o paciente:

```
const patient = await Patients.create({ ...req.body,
admin: req.adminId });
```

Nota: create seria o mesmo que **INSERT INTO** Patients(todos os campos) **VALUES** (req.body);

Atualizar paciente: Atualiza um paciente em específico, rota **/dashboard/patients/:patientsId** :

Atualizamos o paciente através do parâmetro que vem da URL:

```
const patient = await
Patients.findByIdAndUpdate(req.params.patientsId,
req.body, {
  new:true
});
```

Nota: findByIdAndUpdate seria o mesmo que **UPDATE** Patients **SET** todosOsCampos = req.body **WHERE** _id = req.params.patientsId;

Apagar paciente: Apaga um paciente em específico, rota **/dashboard/patients/:patientsId** :

Apaga o paciente através do parâmetro que vem da URL:

```
const patient = await
Patients.findByIdAndRemove(req.params.patientsId);
```

Nota: findByIdAndRemove seria o mesmo que **DELETE FROM** Patients **WHERE** _id = req.params.patientsId;

Rotas messages

Ainda não temos uma boa gestão das mensagens, isso virá em breve, por enquanto apenas criamos.

Criando mensagem: Pegamos os dados através de um evento do web Socket, no caso o sendMessage:

```
socket.on('sendMessage', data => {  
  const message = Message.create(data);  
  socket.broadcast.emit('receivedMessage', data);  
})
```

Nota: create seria o mesmo que `INSERT INTO Message VALUES(data);`