



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA



Relatório Final

Projeto de Base de Dados

Desenvolvido por:

Miguel Curto Castela - uc2022212972 - micas.castela@gmail.com

Francisco Lapa Silva - uc2022213583 - francisco.lapamsilva@gmail.com

Débora Alves - uc2022213498 - debora.alves.004@gmail.com

Licenciatura em engenharia informática - 2º ano - 2023/2024

Índice

Conteúdo

Índice	1
Gestão de tarefas.....	2
Manual de Instalação	3
Programas utilizados	3
Como instalar e usar a aplicação.....	3
Manual de utilizador	7
Comandos Postman desenvolvidos	7
Default Values	7
Funcionalidades	9
ADD PATIENT	9
.....	10
ADD DOCTOR	10
ADD NURSE	10
.....	11
ADD ASSISTANT	11
LOGIN.....	12
ADD APPOINTMENT	12
SEE APPOINTMENT	13
ADD SURGERY	13
.....	14
ADD NEW PRESCRIPTION	15
SEE PRESCRIPTIONS	15
PAYMENT BILL	15
MONTHLY REPORT	16
DAILY SUMMARY	17
LIST TOP 3	17
Diagramas finais	19
Alterações efetuadas	20
Overview do projeto desenvolvido.....	20

Gestão de tarefas

Encontra-se aqui as tarefas efetuadas por cada elemento do grupo:

- Desenvolvimento de código:

Débora: Appointments_and_Surgeries.py , Prescriptions.py e trigger_Bill.sql;

Francisco: Constraints, hospital_stats, registers, drop every table, locks e debug;

Miguel: API, registers.py, Requests no Postman, debug, criacao_de_tabelas.sql, logins.py , trigger_Bill.sql e locks.

Todo o código desenvolvido foi testado pelos elementos do grupo, principalmente pelo Miguel e pelo Francisco que se encarregaram dos testes e revisões de código, tal como a implementação de locks e atributos específicos para cada *request*.

- Escrita do relatório final e slides:

Todo o material desenvolvido nesta parte também foi revisto e acordado por todos os elementos do grupo, mas foi desenvolvido pela Débora.

O processo de teste e revisão representa um dos maiores investimento de tempo no projeto, seguido do desenvolvimento de código e por fim a escrita do relatório, desenvolvimento dos slides para o pitch e ajustes finais.

O tempo gasto para o desenvolvimento do projeto foi à volta de 1 semana e meia.

Manual de Instalação

Programas utilizados

Para implementação da base de dados foi utilizado o **postgreSQL** que usa a linguagem *sql* para gerir bases de dados. Como interface visual para testar se os dados são inseridos corretamente nas tabelas foi usado o **PSQL** e o **PGADMIN**. Neste manual, serão apresentadas as opções de configuração tanto utilizando o **PGADMIN** e o **PSQL**, respetivamente. Adicionalmente, para desenvolvimento dos endpoints, gerenciamento de erros e rest API foi utilizada a linguagem **python**. Por fim, para interação com a aplicação API desenvolvida é utilizado o programa *postman*. Este manda pedidos html para os endpoints definidos na aplicação.

Como instalar e usar a aplicação

Primeiramente, é necessário “darmos *Setup*” à nossa base de dados. Para este efeito é preciso instalar o postgresql e o programa pgAdmin (ou como alternativa usar o psql), mencionado acima, e criar uma nova base de dados:

- Criar o utilizador para conexão com a base de dados no perfil postgres. O utilizador no programa desenvolvido chama-se **aulaspl** e a sua password é **aulaspl**. Com estes dados é possível o rest-api ligar-se diretamente à base de dados, por meio dos pacotes python:

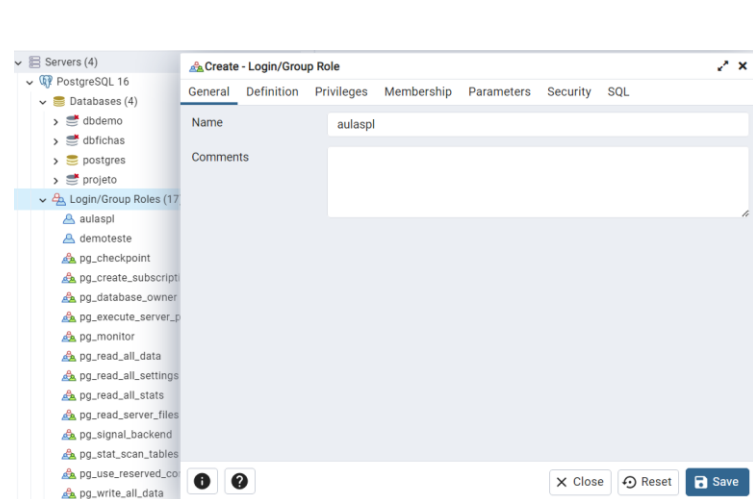


Fig 1: criar o utilizador “aulaspl”

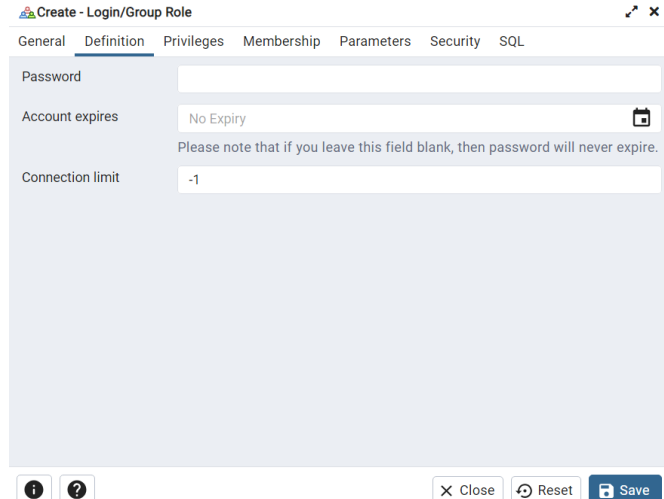


Fig 2: definir a password para o utilizador

Para o **PSQL** utilizamos os seguintes comandos no terminal:

```
psql -h localhost -p 5432 -d postgres -U postgres
```

Fig 3: aceder ao “postgres” com o utilizador “postgres”

```
postgres=# create user aulaspl password 'aulaspl';
```

Fig 4: criar o utilizador “aulaspl” com a password “aulaspl”

- Ainda no utilizador postgres criamos a base de dados, esta tem o nome “projeto”:

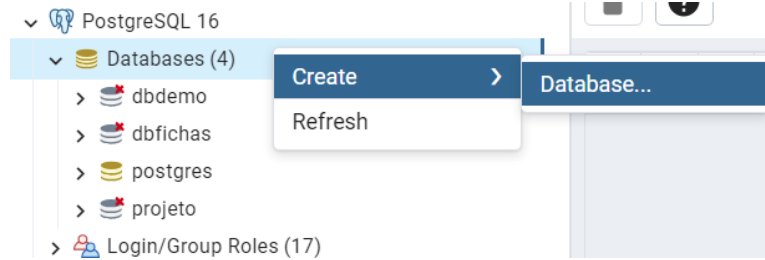


Fig 5: seleccionar a opção Create>Database

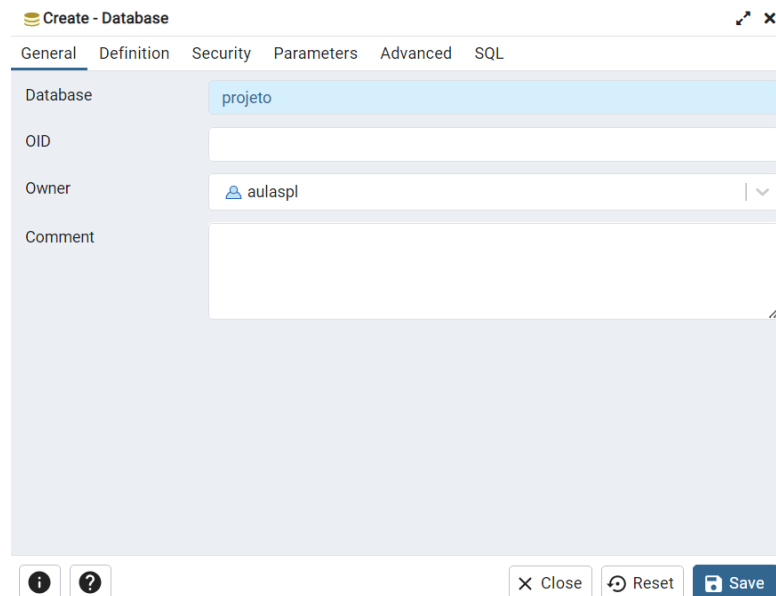


Fig 6: criar a base de dados “projeto” e declarar o nosso utilizador como dono

Utilizando **PSQL**:

```
postgres=# create database projeto;
```

Fig 7: criar a base de dados “projeto”

```
psql -h localhost -p 5432 -d projeto -U postgres
```

Fig 8: aceder a projeto com o utilizador “postgres”

```
projeto=# grant all on schema public to aulaspl;
```

Fig 9: conceder permissões ao novo utilizador criado nessa base de dados

- Depois é necessário criar o servidor que vai ser *host* da base de dados criada, permitindo apenas que o nosso utilizado tenha acesso (ao utilizar as credenciais já mencionadas). Para esta fase utilizamos apenas o **PGADMIN** para facilitar o processo de instalação:

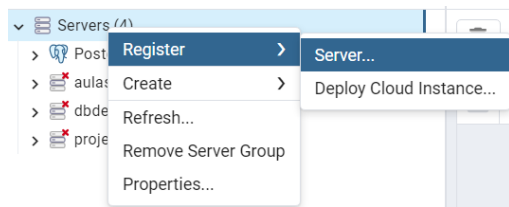


Fig 10: seleccionar a opção Register>Server

Fig 12: definir o *host address* e o utilizador a conectar

Fig 11: escolher o nome do servidor

Após a criação do servidor e da base de dados, foi necessário ter a ferramenta **pgAdmin** ou o **PSQL** aberto e *logged-in* no utilizador criado, dentro da nossa base de dados. Em simultâneo, corremos o nosso código rest-API (**hospital_rest_api.py**), este liga-se à base de dados e usa todos os ficheiros e triggers usados para os endpoints desejados. A nossa função que permite a conexão com a base de dados encontra-se no ficheiro **funções_globais.py**:

```
def db_connection():
    db = psycopg2.connect(
        user='aulaspl',
        password='aulaspl',
        host='127.0.0.1',
        port='5432',
        database='projeto'
    )

    return db
```

Fig 13: função “db_connection” usada para conectar à base de dados

Por fim, a interação com a base de dados é realizada através da ferramenta Postman, basta importar o *postman collection*, que também acompanha o projeto e contém os *requests* que podem ser realizados

na nossa base de dados. Uma explicação mais detalhada do modo de funcionamento dos comandos pode ser encontrada a seguir, no nosso manual de utilizador.

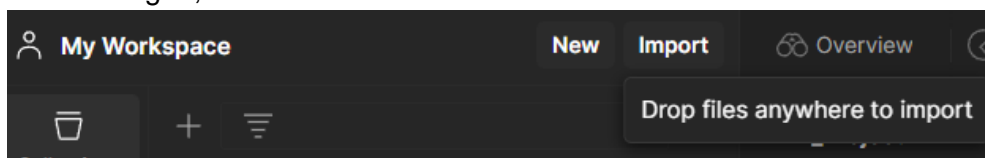


Fig 14: seleccionar a “Import” na página inicial do postman

Manual de utilizador

Comandos Postman desenvolvidos

Para interação com a base de dados, foram implementados os seguintes pedidos:

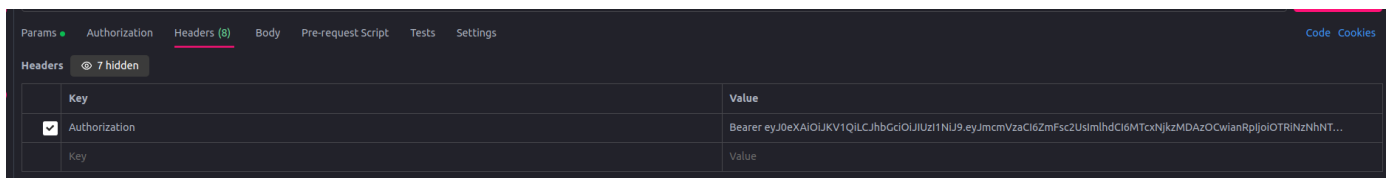
- ADD PATIENT
- LOGIN
- ADD APPOINTMENT
- SEE APPOINTMENT
- ADD SURGERY NEW HOSPITALIZATION
- ADD SURGERY EXISTING HOSPITALIZATION
- ADD DOCTOR
- ADD ASSISTANT
- ADD NURSE
- ADD NEW PRESCRIPTION
- SEE PRESCRIPTIONS
- PAYMENT BILL
- MONTHLY REPORT
- LIST TOP 3
- DAILY SUMMARY

Para utilizar estas funcionalidades é necessário executar o programa como explicado anteriormente, e no postman, no endpoint correto dar SEND aos dados que queremos enviar ou receber. Cada pedido implementado é uma funcionalidade da nossa aplicação.

Para utilizar algumas funcionalidades, é necessário realizar autenticação. Isto pode ser feito através do URL 'http://[...]/dbproj/utilizador', e um body json com os atributos "mail" e "password_", com as credenciais corretas de um utilizador.

Se a autenticação funcionar, o servidor envia um package json, onde o atributo "response" contém os atributos "role" e "id", indicando as características do utilizador, e "access_token", com o jwt gerado para servir como autenticação para as sessões deste utilizador antes de expirar em 30 minutos. Este "access_token" deve ser copiado e colado numa header de key "Authorization" com o value de "Bearer " seguido do token.

Deste modo, o utilizador poderá aceder às funcionalidades restritas ao seu role e id, com apenas um login.



Exemplo de uso de token dado no login

Default Values

Antes de passar à explicação da utilização dos endpoints, é necessário ter em atenção que certas tabelas servem apenas para armazenar valores já pré-definidos, facilitando o acesso à informação e criação de

elementos de outras tabelas com que se relacionam. De seguida, temos uma tabela com a informação utilizada como *default values* para teste do código:

Tabela	Valor
<i>medication</i> (<i>medicação, id</i>)	paracetamol, 1
	ibuprofeno, 2
	aspirina, 3
	dorflex, 4
<i>sideeffect</i>	dor de cabeça
	dor de barriga
	dor de costas
	ataque cardíaco
<i>specialization</i>	cardiologia
	neurologia
	ortopedia
<i>effect_property</i> (<i>ocorrencias, severidade, side_effect_id, medication_id</i>)	1,1,1,1
	2,2,2,2
	3,3,3,3
	4,4,4,4
<i>role</i>	anesthesiologist
	overseer
	backup

<i>type_</i> (tipo de appointment)	oftalmologia
	cardiologia
	neurologia
	ortopedia
<i>category</i>	junior
	senior
	leader

Funcionalidades

Segue-se uma explicação detalhada de como utilizar cada pedido e as suas funcionalidades.

Primeiramente, apresentamos os comandos de registo na base de dados de pacientes, médicos, enfermeiras e assistentes. Estes pedidos não têm restrições de utilização e simplesmente são usados para inserir utilizadores com diferentes *roles* na base de dados. Todos os pedidos de registo adicionam um utilizador à tabela “**utilizador**”.

ADD PATIENT

Este pedido adiciona um novo utilizador à tabela “**utilizador**” e, de seguida, insere na tabela “**patient**” um novo paciente utilizando a foreign key **utilizador_id**, que é comum ao **id** da tabela utilizador. Este **id** é auto incremento. Os campos a inserir no json enviado pelo request do postman são:

- **birthday**: tem de ser do tipo data (mesmo formato) e uma data válida, inferior à atual.
- **id_user_cc**: corresponde ao cartão de cidadão e tem de conter 9 dígitos.
- **name_**: corresponde ao nome do utilizador.
- **nationality**: nacionalidade do utilizador.
- **password_**: password do utilizador, pode conter caracteres especiais e dígitos.
- **mail**: tem de ter um “@” e é único por utilizador.

```
1 {
2   "birthday" : "2004-10-03",
3   "id_user_cc" : 123456789,
4   "username" : "Patient",
5   "nationality" : "portugues",
6   "password" : "CAOGATO",
7   "email" : "micas.castl@...
8 }
```

dy Cookies Headers (5) Test Results

Pretty Raw Preview JSON

```
1 {
2   "id": 1,
3   "message": "Patient registered successfully",
4   "status": 200
5 }
```

Fig 15: Request do register/patient.

ADD DOCTOR

Este pedido insere o novo utilizador na tabela “**utilizador**”, e, após isto, é retornado o **id** atribuído e adicionado o utilizador na tabela **employee** e **doctor**, mais uma vez utilizando em ambas a foreign key **utilizador_id**, que corresponde ao id em **utilizador**. Cada doutor tem um campo de especialização definido. Os atributos a inserir (para além dos pertencentes à tabela **utilizador**) são:

- **medical_license_id**: representa a licença médica, e é única por doutor.
- **university**: nome da universidade onde estudou.
- **specialization_expertise**: campo de experiência, tem de ser um dos campos já existentes na nossa tabela de especializações.

```
1 {
2   "birthday" : "2004-10-03",
3   "id_user_cc" : 123456789,
4   "username" : "Doutor",
5   "nationality" : "portugues",
6   "password" : "CAOGATO",
7   "email" : "micas.casl@",
8   "medical_license_id" : 123456789,
9   "university" : "Some University",
10  "specialization_expertise" : "cardiologia"
11 }
```

dy Cookies Headers (5) Test Results

Pretty Raw Preview JSON

```
1 {
2   "id": 2,
3   "message": "Doctor registered successfully",
4   "status": 200
5 }
```

Fig 16: Request do register/doctor

ADD NURSE

Cada enfermeira pertence a uma categoria já pré-estabelecida, assim como acontece com o médico e a sua especialização, ou seja, ao adicionar categorias aos enfermeiros estes já têm de existir na nossa tabela de categorias. Após inserir o utilizador na tabela “**utilizador**”, é retornado o **id** atribuído e adiciona o

utilizador na tabela **employee** e **nurse**, utilizando para isto a foreign key **utilizador_id**. O campo adicional aos da tabela utilizador é:

- **category_category_name**: foreign key que atribui a um enfermeiro o seu estatuto. Representa a hierarquia entre enfermeiros.

```
1 {
2   "birthday": "2004-10-03",
3   "id_user_cc": 123456789,
4   "username": "Nurse",
5   "nationality": "portugues",
6   "password": "CAOGATO",
7   "email": "micas.@",
8   "category_category_name": "junior"
9 }
```

dy Cookies Headers (5) Test Results

Pretty Raw Preview JSON ↕

```
1 {
2   "id": 4,
3   "message": "Nurse registered successfully",
4   "status": 200
5 }
```

Fig 17: Request do register/nurse

ADD ASSISTANT

Tal como os restantes funcionários, após inserir o utilizador na tabela “**utilizador**”, é retornado o **id** atribuído e adiciona o utilizador na tabela **employee** e **assistant** utilizando a foreign key **employee_utilizador_id**. Para este pedido não há campos adicionais aos da tabela **utilizador**.

```
1 {
2   "birthday": "2004-10-03",
3   "id_user_cc": 123456739,
4   "username": "Assistant",
5   "nationality": "portugues",
6   "password": "CAOGATO",
7   "email": "micas.c@"
8 }
```

dy Cookies Headers (5) Test Results

Pretty Raw Preview JSON ↕

```
1 {
2   "id": 3,
3   "message": "Assistant registered successfully",
4   "status": 200
5 }
```

Fig 18: Request do register/assistant

Seguidamente, temos o comando responsável por fazer a autenticação do utilizador à plataforma.

LOGIN

Este comando pode ser utilizado por qualquer utilizador já registado, previamente, na base de dados. Caso o login seja bem sucedido é retornado um token que permite ao usuário aceder a funcionalidades da nossa aplicação restringidas pelo seu *role* (*nurse*, *doctor*, *assistant* ou *patient*). Os campos a inserir são:

- mail: email utilizado para o registo.
- password_: palavra-passe inserida no registo.

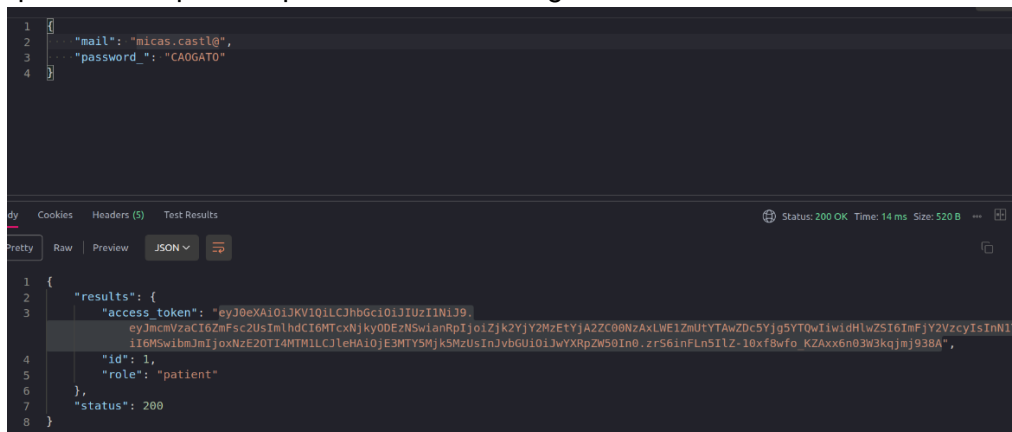


Fig 19: Request do /user - authentication request

De seguida, temos as funcionalidades restritas, ou seja, requerem o *login* e token do utilizador que as queira utilizar (isto verifica se este tem permissão para tal), dentre as quais: marcar consultas, cirurgias, adicionar prescrições, efetuar o pagamento de uma conta, gerar relatórios ou até mesmo ver o top 3 de pacientes do hospital.

ADD APPOINTMENT

Este pedido pode ser utilizado apenas pelo paciente que pretende marcar a consulta. Para as consultas temos que ter obrigatoriamente um doutor, mas ter nurses associadas à consulta é opcional. Caso não queiramos associar nenhum enfermeiro, basta não inserir nada nesse campo. No entanto, caso queiramos basta inserir o *id* da *nurse* juntamente com o seu *role*. O campo "type_type_" representa o tipo de consulta que, assim como nas especializações, tem de ser um **type_** existente na tabela **type_** senão é considerado inválido (.

Adicionalmente, um doutor e enfermeira podem ter no máximo 5 consultas por dia e apenas é possível marcar uma consulta com médicos e enfermeiros disponíveis na data inserida.

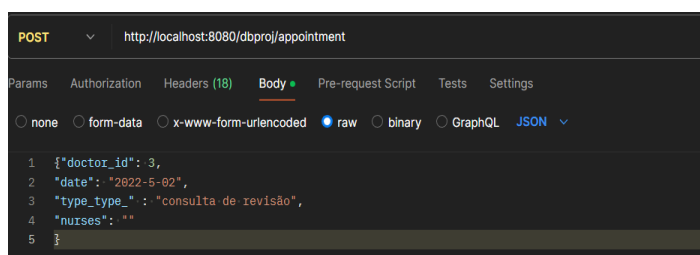


Fig 20: Request do /appointment sem nurses

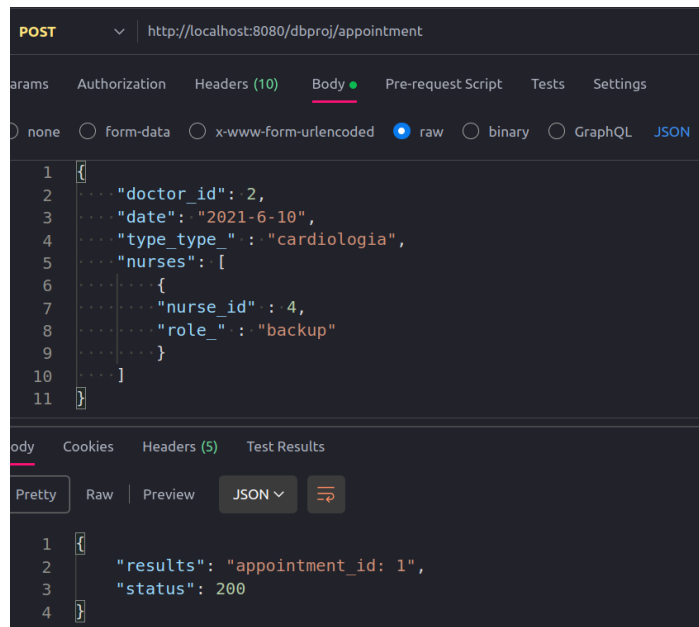


Fig 21: Request do /appointment com nurses

SEE APPOINTMENT

Este comando permite consultar todas as consultas marcadas para um paciente à escolha. Para o utilizar basta inserir no último campo do url o **id** do paciente que pretende consultar. Apenas assistentes e o paciente podem utilizar este comando.

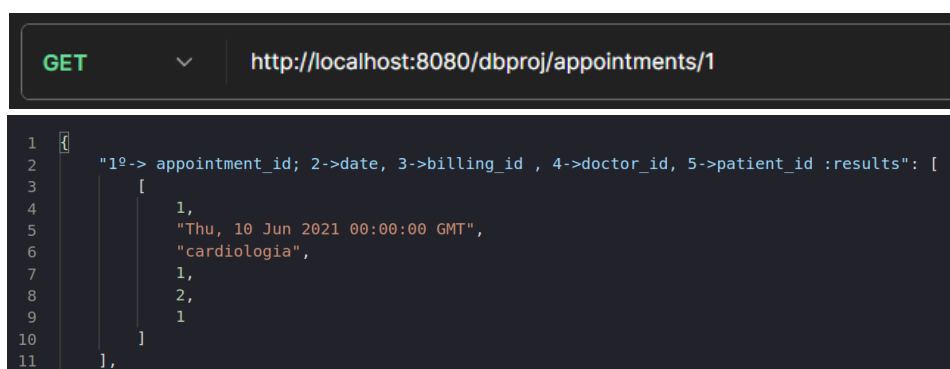
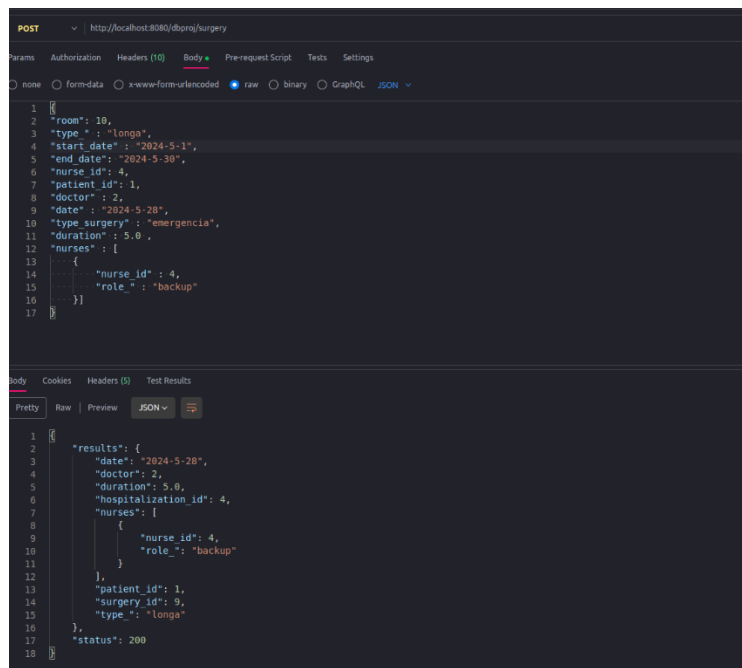


Fig 22: Request do /appointment/utilizador_id - utilizado para consultar marcações de consultas

ADD SURGERY

Este *request* permite marcar uma cirurgia e pode ser utilizado apenas por assistentes. Há dois modos de marcar uma cirurgia: é possível marcar sem haver uma hospitalização indicada, nesse caso uma hospitalização é criada para esse propósito, ou podemos criar uma cirurgia que será associada a uma hospitalização já existente. Para isso basta colocar o id da hospitalização que queremos associar no

endpoint, não havendo necessidade de preencher os campos associados à hospitalização no pedido. Adicionalmente temos um enfermeiro responsável por cada hospitalização e um doutor, juntamente, com um certo número opcional de enfermeiros por cirurgia (para isto é necessário indicar o id e role do enfermeiro). Contrariamente a appointments, o **type_** de uma cirurgia não está pré-definido. Apenas é possível marcar cirurgias com médicos e enfermeiros que estejam disponíveis naquela data, e cada enfermeiro e doutor pode apenas ter uma cirurgia por dia.

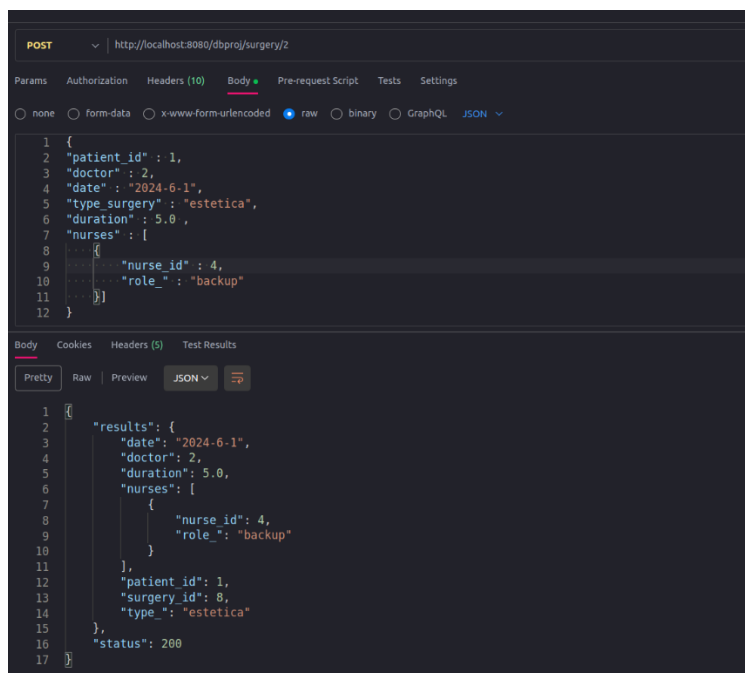


```
POST http://localhost:8080/dbproj/surgery

{
  "room": 10,
  "type": "longa",
  "start_date": "2024-5-1",
  "end_date": "2024-5-30",
  "nurse_id": 4,
  "patient_id": 1,
  "doctor": 2,
  "date": "2024-5-28",
  "type_surgery": "emergencia",
  "duration": 5.0,
  "nurses": [
    {
      "nurse_id": 4,
      "role": "backup"
    }
  ]
}
```

```
{
  "results": {
    "date": "2024-5-28",
    "doctor": 2,
    "duration": 5.0,
    "hospitalization_id": 4,
    "nurses": [
      {
        "nurse_id": 4,
        "role": "backup"
      }
    ],
    "patient_id": 1,
    "surgery_id": 6,
    "type": "longa"
  },
  "status": 200
}
```

Fig 23: Request do /surgery com nova hospitalização



```
POST http://localhost:8080/dbproj/surgery/2

{
  "patient_id": 1,
  "doctor": 2,
  "date": "2024-6-1",
  "type_surgery": "estetica",
  "duration": 5.0,
  "nurses": [
    {
      "nurse_id": 4,
      "role": "backup"
    }
  ]
}
```

```
{
  "results": {
    "date": "2024-6-1",
    "doctor": 2,
    "duration": 5.0,
    "nurses": [
      {
        "nurse_id": 4,
        "role": "backup"
      }
    ],
    "patient_id": 1,
    "surgery_id": 8,
    "type": "estetica"
  },
  "status": 200
}
```

Fig 24: Request do /surgery/hospitalization_id (hospitalização existente)

ADD NEW PRESCRIPTION

Este *endpoint* pode ser acessado apenas por doutores e permite atribuir uma prescrição a um paciente de um ou mais medicamentos existentes na base de dados do hospital. Basta inserir a data de validade da mesma e os respectivos medicamentos associados. Ademais, uma receita médica pode estar associada tanto a um appointment como a uma hospitalização, sendo necessário indicar o id destes no **event_id**.

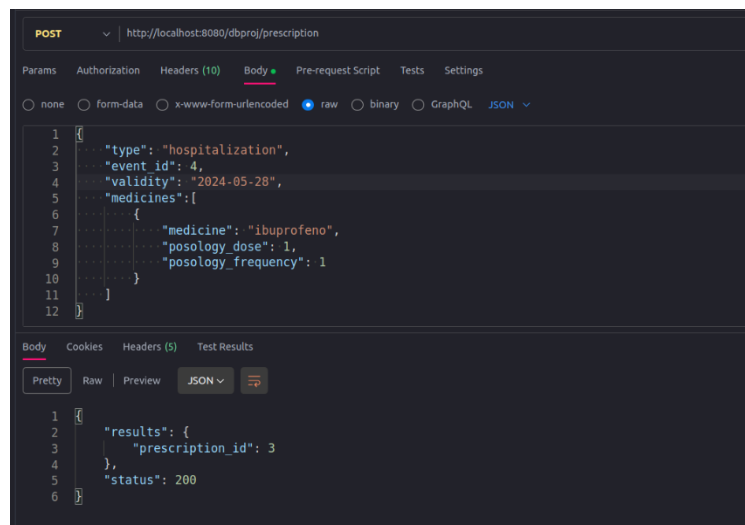


Fig 25: Request do /prescription

SEE PRESCRIPTIONS

Neste endpoint é possível consultar as prescrições de um determinado paciente. Apenas pode ser utilizado por funcionários e pelo próprio paciente.



Fig 26: Request do /prescriptions/hospitalization_id

PAYMENT BILL

Este endpoint permite ao paciente pagar as contas que deve ao hospital. É criada uma nova fatura sempre que um utilizador marca um appointment ou lhe é associada uma cirurgia, isto é feito a partir de um trigger (presente em trigger_bill.sql). Uma conta pode ser paga de uma vez ou em vários pagamentos. Para utilizar

basta inserir no url do request o *bill_id* que pretende pagar, juntamente com o montante e método de pagamento. Este endpoint está restrito ao paciente a que a bill está associada.

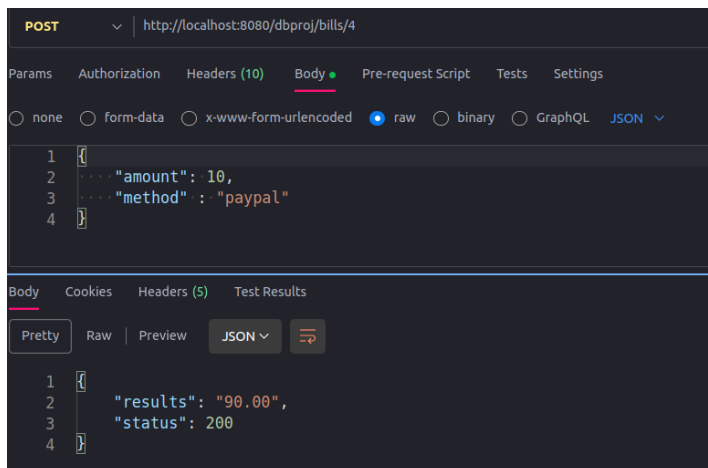


Fig 27: Request do /bills/bill_id

MONTHLY REPORT

Apenas assistentes podem utilizar este *endpoint*. Ao ser utilizado permite gerar um relatório que mostra informações sobre os doutores com o maior número de cirurgias por mês nos últimos 12 meses.

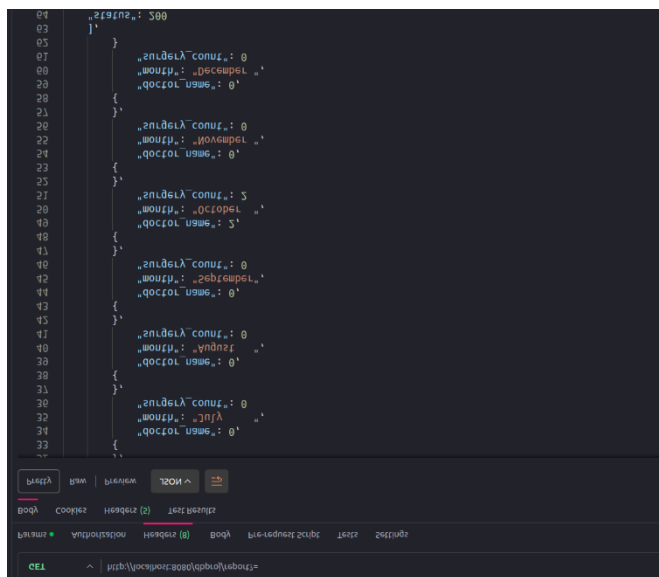
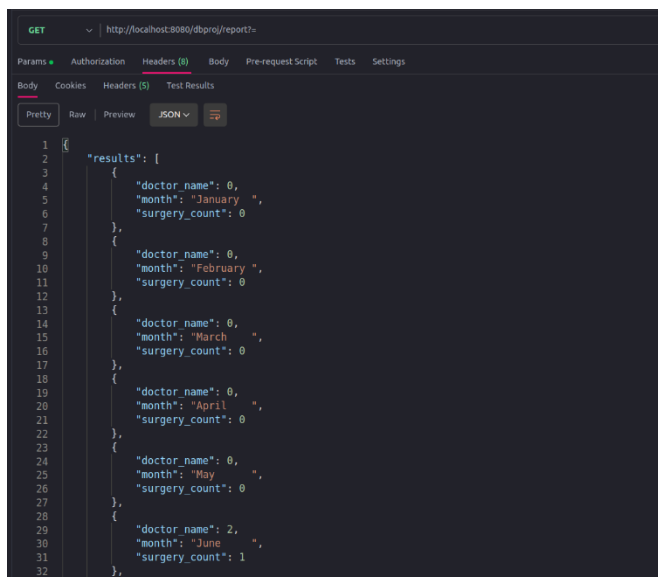


Fig 27: Request do /report

DAILY SUMMARY

Permite gerar um sumário de um dia específico no hospital, apresentando o total de cirurgias, consultas e pagamentos feitos nesse dia. Apenas pode ser acessado por assistentes.

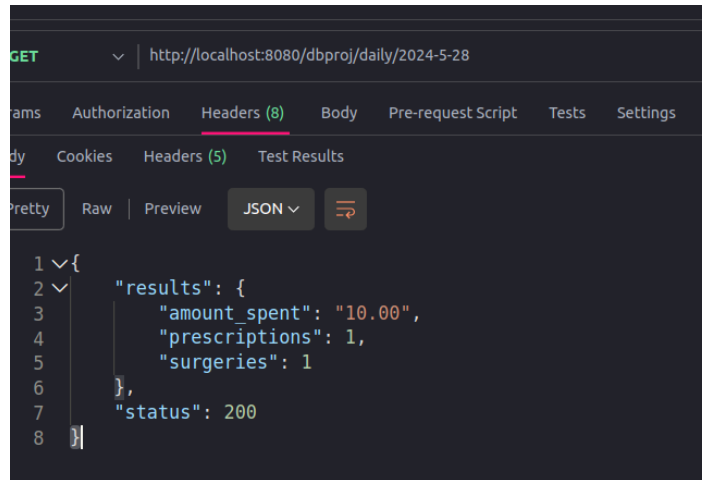


Fig 28: Request do /daily/{ano-mês-dia}

LIST TOP 3

Por fim, este endpoint permite-nos ver os 3 pacientes que gastaram mais dinheiro no mês no hospital, apresentando por cada um os detalhes dos procedimentos efetuados.

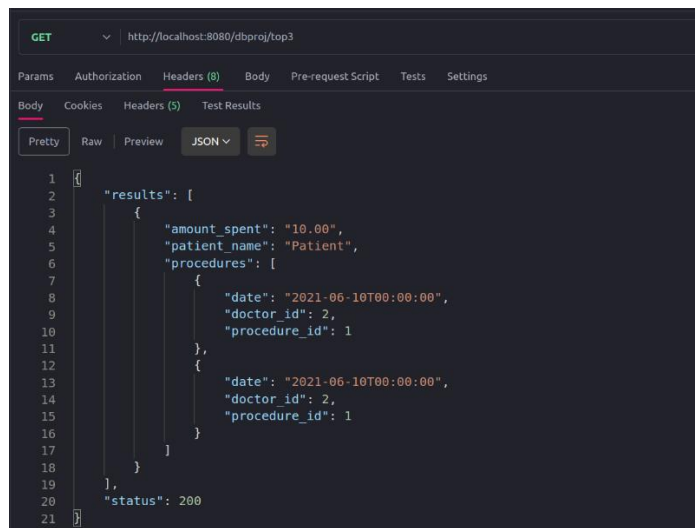


Fig 29: Request do /top3

Diagramas finais

Aqui serão apresentados os diagramas finais, que comparado à primeira meta, contêm evoluções e mudanças explicadas no ponto seguinte.

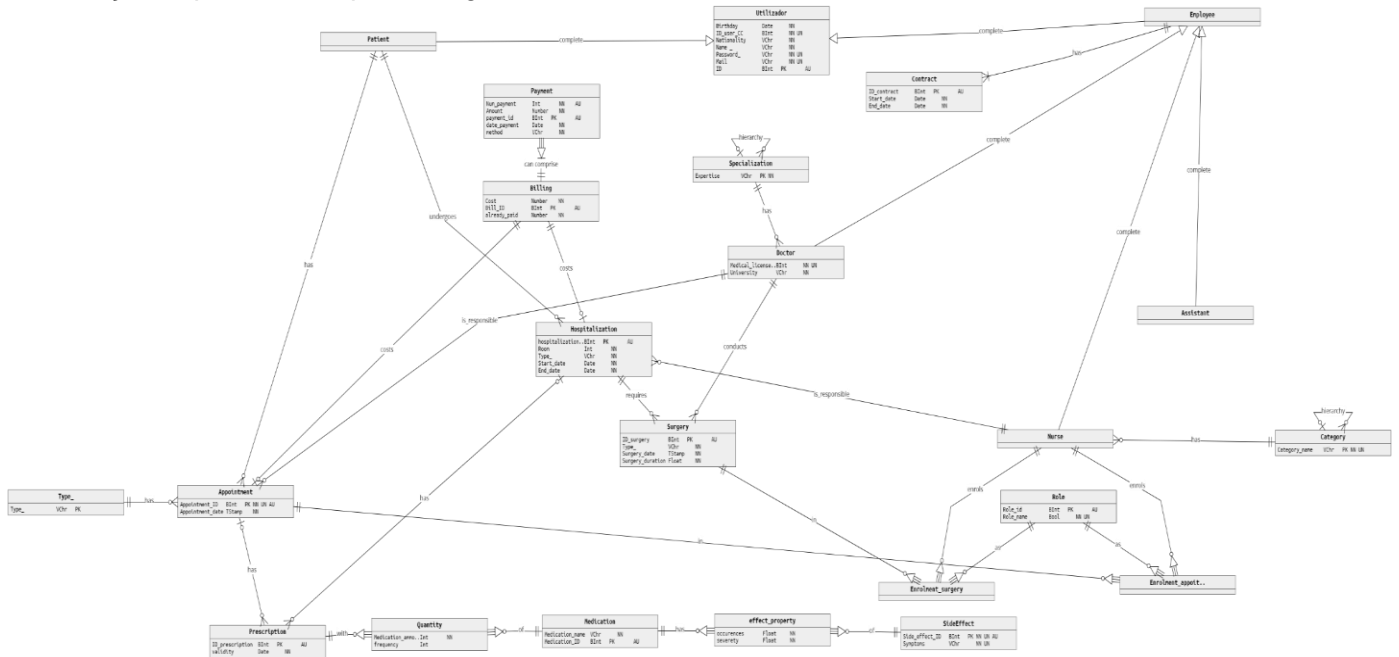


Fig 30: diagrama conceitual final

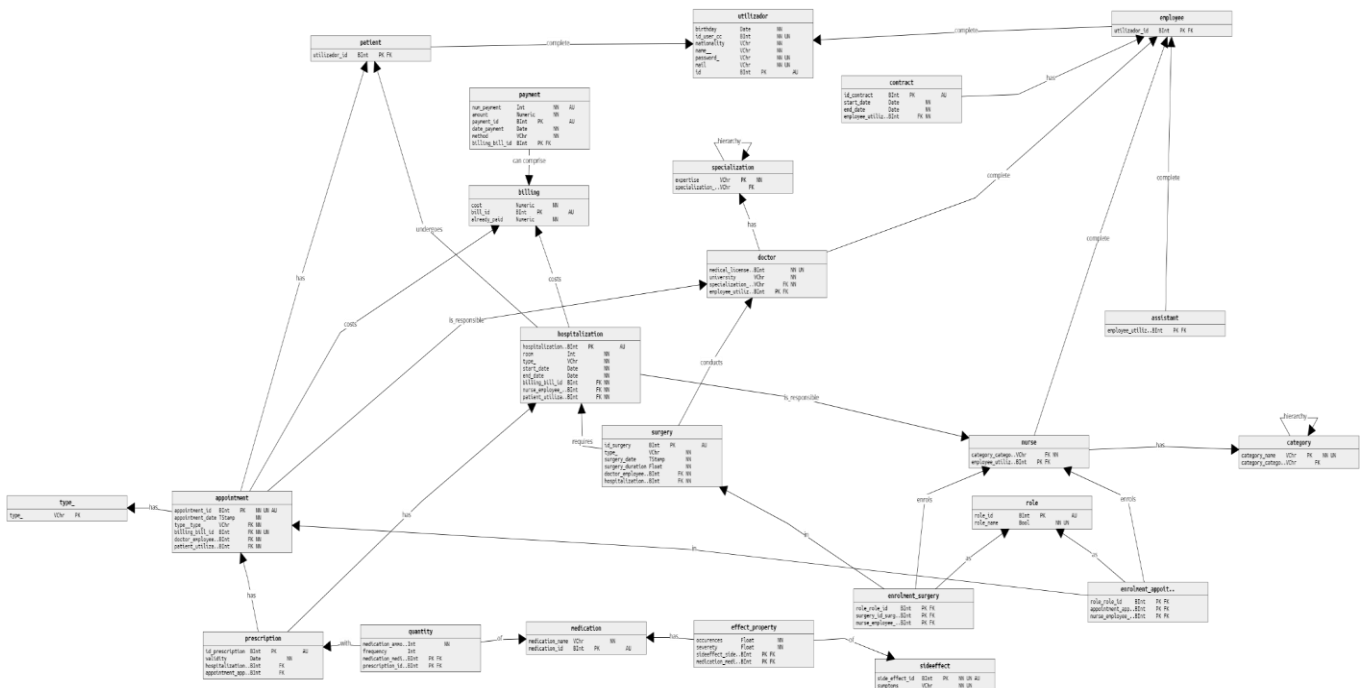


Fig 31: diagrama físico final

Alterações efetuadas

Desde a primeira meta, foram implementadas algumas alterações no conceito da nossa base de dados. A maior mudança foi atribuir ao utilizador um **id** do tipo BIGSERIAL, em vez de utilizar o BI do mesmo como primary key, de modo a termos um identificador único por cada usuário na base de dados que fosse conveniente para o processo de desenvolvimento da aplicação. Foi também decidido tornar **mail** um atributo único, não só por questões lógicas, mas também para a verificação do login. Ademais, foram retiradas informações desnecessárias de **assistant** e **employee**, tal como adicionados atributos necessários à tabela de prescrições para uma melhor implementação. Por fim, foram corrigidos tipos e restrições em alguns atributos que não estavam corretos, assim como nomes para evitar utilizar palavras protegidas. De uma maneira geral, a base da nossa ideia inicial já cumpria em grande parte, os requerimentos do projeto, facilitando a implementação do mesmo.

Overview do projeto desenvolvido

Ficheiros que acompanham o projeto e funções presentes nos mesmos:

Ficheiros Python	
hospital_rest_api.py	
main	estrutura principal do código, onde temos os <i>endpoints</i> , onde é feita a criação de tabelas e execução de código
funcoes_globais.py	
db_connection	estabelece a conexão com o servidor da base de dados
check_missingfields	verifica se um certo atributo está em falta num certo grupo de dados
appointments_and_surgeries.py	
check_surgery_date	verifica se a data de cirurgia está dentro do intervalo de hospitalização
doctor_check_appt	ver se um doutor tem consultas numa determinada data
doctor_check_surgery	ver se o doctor tem cirurgias numa determinada data
check_doctor_availability	determinar se um doutor está disponível através das funções “doctor_check_appt” e “doctor_check_surgery”
check_nurse_availability	verifica se uma nurse está disponível numa determinada data utilizando as funções “check_nurse_availability_surgery” e

	"check_nurse_availability_appointment".
check_nurse_availability_surgery	verifica se a nurse tem cirurgias numa determinada data
check_nurse_availability_appointment	verifica se a nurse tem consultas numa determinada data
check_nurse_fields	verifica se foram inseridos os campos necessários para adicionar uma nurse às tabelas enrolment_appointment ou enrolment_surgery
schedule_appointment	adiciona uma consulta à tabela appointment
see_appt_patient	devolve todas as consultas de um determinado paciente
check_hospitalization_id	verifica se um hospitalization_id dado já existe
schedule_surgery_existing_hosp	adiciona à tabela surgery uma cirurgia para associar a uma hospitalização já existente
schedule_surgery_new_hosp	adiciona à tabela surgery uma cirurgia sem nenhuma hospitalização previamente criada
logins.py	
authenticate_user	permite efetuar um "login" na base de dados, devolvendo um token de autenticação
print_jwt_info	dá print da informação contida no token
registers.py	
add_patient	adiciona um paciente á tabela patient
add_assistant	adiciona um paciente á tabela assistant
add_nurse	adiciona um paciente á tabela nurse
add_doctor	adiciona um paciente á tabela doctor
prescriptions.py	
get_prescriptions	devolve as prescrições de um utilizador dado
add_prescription	adiciona uma prescrição à tabela prescription

payment.py	
execute_payment	simula o pagamento de uma conta, inserindo o pagamento dessa conta na tabela payment . Caso o montante total dos pagamentos associados cubra o valor da bill , ela é dada como “paid”
hospital_stats.py	
list_top3	devolve o TOP 3 de pacientes que gastaram mais dinheiro no hospital
daily_report	devolve todos os procedimentos efetuados numa determinada data
monthly_report	devolve os médicos com mais cirurgias efetuadas nos últimos 12 meses
Ficheiros sql	
criacao_tabelas.sql	contém o código sql para criação das tabelas, insere os valores <i>default</i> e executa as <i>constraints</i>
trigger_bill.sql	contém o código sql que executa os triggers relacionados com a tabela de contas
drop_every_table.sql	contém o código sql que elimina todas as tabelas e constraints da base de dados

Foram implementadas todas as funcionalidades que eram exigidas pelo enunciado de projeto. Do ponto de vista de sql foram implementadas, para além das *constraints* necessárias para a lógica da nossa base de dados, *triggers* para realizar a atualização correta de atributos na tabela de **billing**. Ou seja, sempre que um *appointment* ou *hospitalization* são adicionados às suas respectivas tabelas, é necessário atualizar a conta do paciente em questão. No entanto, caso a base de dados seja acedida por mais do que um utilizador, foram adicionados *locks* de maneira a evitar os seguintes problemas de concorrências nas seguintes transações:

- **Registo de utilizador** - *exclusive lock* na tabela **utilizador** tal como nas restantes tabelas adicionais, dependendo do tipo de registo, para evitar que sejam inseridos utilizadores com dados repetidos caso esta função seja usada em simultâneo.
- **Agendamento de consultas/cirurgias** - *exclusive lock* nas tabelas **appointment**, **surgery** e **hospitalization**, apenas no caso de ser necessário criar uma nova na marcação de uma cirurgia. Deste modo, evitamos informação repetida ou agendamentos com doutores/enfermeiros que não estejam realmente disponíveis, no caso de marcações simultâneas.

- **Adicionar prescrição** - *exclusive lock* para evitar que, na criação de receitas simultâneas, haja corrupção de informação.
- **Top 3** - *exclusive lock* à tabela **payment** para durante a geração do TOP 3 pacientes ser possível a sua consulta, mas evitando que outras transações influenciem os dados.
- **Relatório Mensal** – *exclusive lock* na tabela **surgery** para que durante a geração do relatório mensal, que apresenta os médicos com mais cirurgias nos últimos 12 meses, não haja corrupção de informação por parte de transações simultâneas.

Além disso, sempre que não conseguimos efetuar uma transação ou a mesma dá erro é utilizado o comando *rollback* para desfazer todas as alterações efetuadas, evitando informação corrompida nas nossas tabelas.

Adicionalmente, usámos isolamento nas tabelas acedidas nas transações dos comandos GET, isto foi feito para evitar **dirty reads**.

Quanto a melhorias na base de dados, decidimos adicionar o campo **already_paid** nas faturas. Isto faz com que, apesar de ser redundante, não seja necessário percorrer todos os pagamentos para verificar quanto é que já foi pago em cada fatura.

Foi também decidido termos uma tabela de utilizadores de onde cada *nurse*, *doctor*, *assistant* e *patient* descendem, permitindo-nos armazenar todas as informações comuns aos mesmos num único sítio. Desta forma, a implementação das funções e distinções de papéis foram facilitadas visto que bastava verificar a foreign key **utilizador_id**. Adicionalmente, cada funcionário à medida que é registado, é colocado na tabela **employee** para uma melhor distinção. Foram tomadas as decisões de implementar a hierarquia tanto das especializações como das categorias de cada enfermeiro, através de uma ligação das tabelas **specialization** e **category** para si mesmas. Também, foi decidido ter tabelas com atributos default que permitisse simplesmente inserir atributos que de facto o hospital oferecesse, dentre as quais temos: **role**, **type_**, **medication** e **sideeffect**. Deste modo, podemos simplesmente associar informação certa a novos elementos das tabelas com que estas se relacionam. É importante realçar, que a tabela **type_** com tipos já definidos de consultas é utilizada apenas para o tipo de **appointments**, sendo que no caso de **hospitalizations** a assistente é livre de meter um tipo que não está já pré-definido. As tabelas **enrolment_appointment** e **enrolment_surgery**, teorizadas na meta 1, foram usadas para podermos associar diferentes enfermeiras com roles diferentes a cada tipo de evento (para além da responsável, no caso das hospitalizações com cirurgias). Este método de relacionar informação é utilizado também nas **prescriptions** para facilitar a passagem de uma receita médica (com a tabela **amount**, que liga os diferentes atributos de uma prescription presentes em tabelas diferentes), há medicamentos já definidos que se podem adicionar tanto a um appointment como uma surgery de certo paciente.

Concluindo, a nossa implementação permite aos pacientes marcarem consultas e pagarem as suas contas, aos médicos passar prescrições, aos enfermeiros consultá-las e aos assistentes gerirem procedimentos, sendo uma solução sólida e organizada, que permite uma gestão de informação de um sistema hospitalar.