OBJETIVO E ESCOPO:

Os sistemas de atendimento ambulatoriais em sua maioria costumam estar superlotados, além de não serem flexíveis. Ao chegar no hospital os pacientes têm de enfrentar diversos passos repetitivos até serem de fato atendidos, como por exemplo preencher um formulário, passar na triagem para relatar os sintomas e somente então aguardar para serem atendidos por um médico

As pessoas costumam estar bastante habituadas com esses processos de atendimento. Mas no geral eles costumam ser afetados pela quantia de pacientes, urgência de sua consulta, e uma baixa quantia de profissionais para atende-los

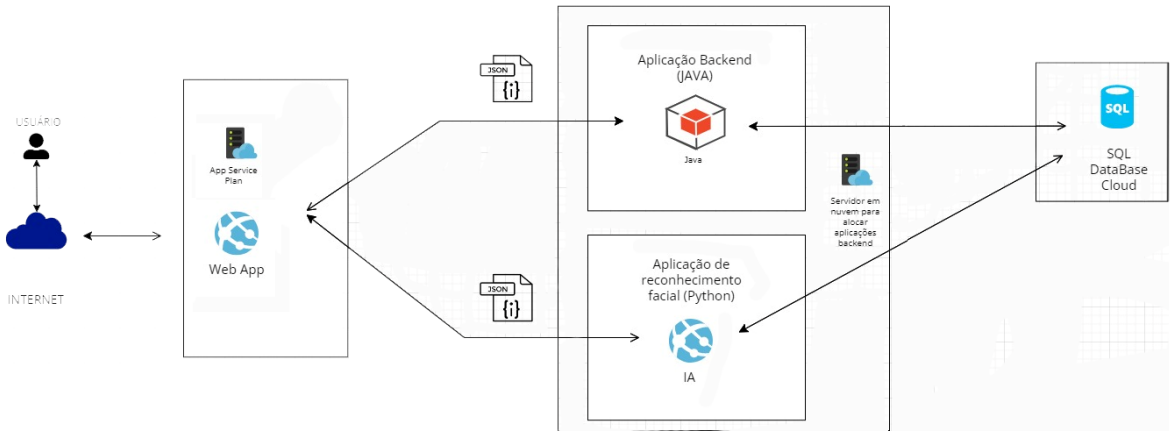
Um estudo do Banco Mundial mostrou que o uso de inteligência artificial para análise de prontuários de pacientes pode economizar até 22 bilhões, evitando repetições desnecessárias de exames e tratamentos, além de otimizar processos internos.

Com isso nosso público-alvo são hospitais, unidades básicas de saúde (UBS), clínicas médicas e outros ambulatórios que precisam implementar melhores sistemas de gestão e atendimento do paciente.

O intuito do aplicativo é facilitar o processo de cadastro dos pacientes, ao abrir o aplicativo o mesmo irá se cadastrar e abrir uma ficha, na qual sera perguntado sobre seus sintomas e outros dados necessários para agilizar o processo de chegada. Com a ficha criada os dados serão encaminhados diretamente aos médicos, e então o paciente deverá aguardar até que seja chamado para consulta.

Além do aplicativo mostrar para seu usuário qual o hospital mais proxímo no momento e com menos tempo de espera, fazendo um balanço para que o paciente possa ser atendido o mais rápido possível e diminuindo a burocracia para chegar finamente ao seu atendiemento.

ARQUITETURA DA SOLUÇÃO:



- Aplicação em Cloud armazenado em um WebApp pela AZURE.
- Banco de dados ORACLE em Cloud.
- Github Actions configurado para todas mudanças feitas na branch principal sejam automaticamente realizado o deploy da aplicação.

TABELA DOS ENDPOINTS:

SINTOMAS:

PATH: "/api/sintoma",
MÉTODO: GET
DESCRIÇÃO: Lista todos os sintomas cadastrados.
PAGINADO: SIM
TAMANHO: 10

PATH: "/api/sintoma/{id}",
MÉTODO: GET
DESCRIÇÃO: Lista apenas o sintoma referente ao ID passado.
PARAMETROS: ID do sintoma

PATH: "/api/sintoma",
MÉTODO: POST
DESCRIÇÃO: Cadastra um novo sintoma.
BODY:
{
"sintoma": String
}

PATH: "/api/sintoma/{id}",
MÉTODO: DELETE
DESCRIÇÃO: Deleta um sintoma pelo ID passado.
PARAMETROS: ID do sintoma

PATH: "/api/sintoma/{id}",
MÉTODO: PUT
DESCRIÇÃO: Atualiza o sintoma já cadastrado pelo parametro passado.
PARAMETROS: ID do sintoma
BODY:
{
"sintoma": String
}

SINTOMAS-ATENDIMENTO:

PATH: "/api/sintomasatendimento",
MÉTODO: GET
DESCRIÇÃO: Lista todos os sintomas-atendimentos cadastrados.
PAGINADO: SIM
TAMANHO: 10

PATH: "/api/sintomasatendimento/{id}",
MÉTODO: GET
DESCRIÇÃO: Lista apenas o sintomas-atendimentos referente ao ID passado.
PARAMETROS: ID do sintomas-atendimentos

PATH: "/api/sintomasatendimento",
MÉTODO: POST
DESCRIÇÃO: Cadastra um novo sintomas-atendimentos.
BODY:
{
sintoma: Sintoma,
atendimento: Atendimento
}

PATH: "/api/sintomasatendimento/{id}",
MÉTODO: DELETE
DESCRIÇÃO: Deleta um sintomas-atendimentos pelo ID passado.
PARAMETROS: ID do sintomas-atendimentos

PATH: "/api/sintomasatendimento/{id}",
MÉTODO: PUT
DESCRIÇÃO: Atualiza o sintomas-atendimentos já cadastrado pelo parametro passado.
PARAMETROS: ID do sintomas-atendimentos
BODY:
{
sintoma: Sintoma,
atendimento: Atendimento
}

PACIENTE:

PATH: "/api/paciente",
MÉTODO: GET
DESCRIÇÃO: Lista todos os pacientes cadastrados.
PAGINADO: SIM
TAMANHO: 10

PATH: "/api/paciente/{id}",
MÉTODO: GET
DESCRIÇÃO: Lista apenas o paciente referente ao ID passado.
PARAMETROS: ID do paciente

PATH: "/api/paciente",
MÉTODO: POST
DESCRIÇÃO: Cadastra um novo paciente.
BODY:

```
{
nome: String,
cpf: string,
dataNascimento: date,
dataCadastro: Date
}
```

PATH: "/api/paciente/{id}",
MÉTODO: DELETE
DESCRIÇÃO: Deleta um paciente pelo ID passado.
PARAMETROS: ID do paciente

PATH: "/api/paciente/{id}",
MÉTODO: PUT
DESCRIÇÃO: Atualiza o paciente já cadastrado pelo parametro passado.
PARAMETROS: ID do paciente
BODY:

```
{
nome: String,
cpf: string,
dataNascimento: date,
dataCadastro: Date
}
```

MÉDICO:

PATH: "/api/medico",
MÉTODO: GET
DESCRIÇÃO: Lista todos os médicos cadastrados.
PAGINADO: SIM
TAMANHO: 10

PATH: "/api/medico/{id}",
MÉTODO: GET
DESCRIÇÃO: Lista apenas o médico referente ao ID passado.
PARAMETROS: ID do médico

PATH: "/api/medico",
MÉTODO: POST
DESCRIÇÃO: Cadastra um novo médico.
BODY:

```
{
nome: String,
crm: String,
dataAdmissao: Date,
dataSaida: date
}
```

PATH: "/api/medico/{id}",
MÉTODO: DELETE
DESCRIÇÃO: Deleta um médico pelo ID passado.
PARAMETROS: ID do médico

PATH: "/api/medico/{id}",
MÉTODO: PUT
DESCRIÇÃO: Atualiza o médico já cadastrado pelo parametro passado.
PARAMETROS: ID do médico
BODY:

```
{
nome: String,
crm: String,
dataAdmissao: Date,
dataSaida: date
}
```

ENDEREÇO:

PATH: “/api/endereco”,
MÉTODO: GET
DESCRIÇÃO: Lista todos os endereços cadastrados.
PAGINADO: SIM
TAMANHO: 10

PATH: “/api/endereco/{id}”,
MÉTODO: GET
DESCRIÇÃO: Lista apenas o endereço referente ao ID passado.
PARAMETROS: ID do endereço

PATH: “/api/endereco”,
MÉTODO: POST
DESCRIÇÃO: Cadastra um novo endereço.
BODY:

```
{
logradouro: String,
numero: Int,
complemente: String,
cep: String,
bairro: String,
municipio: String,
siglaEstado: String
}
```

PATH: “/api/endereco/{id}”,
MÉTODO: DELETE
DESCRIÇÃO: Deleta um endereço pelo ID passado.
PARAMETROS: ID do endereço

PATH: “/api/endereco/{id}”,
MÉTODO: PUT
DESCRIÇÃO: Atualiza o endereco já cadastrado pelo parametro passado.
PARAMETROS: ID do endereço
BODY:

```
{
logradouro: String,
numero: Int,
complemente: String,
cep: String,
bairro: String,
municipio: String,
siglaEstado: String
}
```

CONTATO:

PATH: “/api/contatos”,
MÉTODO: GET
DESCRIÇÃO: Lista todos os contatos cadastrados.
PAGINADO: SIM
TAMANHO: 10

PATH: “/api/contatos/{id}”,
MÉTODO: GET
DESCRIÇÃO: Lista apenas o contato referente ao ID passado.
PARAMETROS: ID do contato

PATH: “/api/contatos”,
MÉTODO: POST
DESCRIÇÃO: Cadastra um novo contato.
BODY:

```
{
ddd: String,
telefone: String,
email: String
}
```

PATH: “/api/contatos/{id}”,
MÉTODO: DELETE
DESCRIÇÃO: Deleta um contato pelo ID passado.
PARAMETROS: ID do contato

PATH: “/api/contatos/{id}”,
MÉTODO: PUT
DESCRIÇÃO: Atualiza o contato já cadastrado pelo parametro passado.
PARAMETROS: ID do contato
BODY:

```
{
ddd: String,
telefone: String,
email: String
}
```

ATENDIMENTO:

PATH: "/api/atendimento",
MÉTODO: GET
DESCRIÇÃO: Lista todos os atendimentos cadastrados.
PAGINADO: SIM
TAMANHO: 10

PATH: "/api/atendimento/{id}",
MÉTODO: GET
DESCRIÇÃO: Lista apenas o atendimento referente ao ID passado.
PARAMETROS: ID do atendimento

PATH: "/api/atendimento",
MÉTODO: POST
DESCRIÇÃO: Cadastra um novo atendimento.
BODY:

```
{
data: Date,
classificacoRisco: ClassificaoRisco
}
```

PATH: "/api/atendimento/{id}",
MÉTODO: DELETE
DESCRIÇÃO: Deleta um atendimento pelo ID passado.
PARAMETROS: ID do atendimento

PATH: "/api/atendimento/{id}",
MÉTODO: PUT
DESCRIÇÃO: Atualiza o atendimento já cadastrado pelo parametro passado.
PARAMETROS: ID do atendimento
BODY:

```
{
data: Date,
classificacoRisco: ClassificaoRisco
}
```

FUNCIONALIDADES NO BACKEND:

O backend funciona no lado do servidor da aplicação e é responsável por tudo o que acontece atrás da tela, desde o login até o final do processamento do arquivo do usuário/paciente. Ele é responsável por desenvolver as instruções do aplicativo para que o usuário possa acessar a interface e atender às solicitações feitas.

Configuramos o Spring Security para poder realizar a autenticação dos nossos usuários e dar as permissões necessárias para os respectivos, assim permitindo que eles possam fazer somente o que cabe a eles, caso seja um paciente por exemplo, poderá realizar uma cadastro ou listar caso já exista.

Nossa aplicação tem como suas principais funcionalidades cadastrar os pacientes/médicos e relaciona-los entre eles, registrando em um atendimento. Portanto é necessário que o usuário nos envie algumas outras informações como os sintomas que sente, dados de cadastro obrigatórios como endereço para que o backend possa processar esses dados e gerar os formulários corretamente.

Todas essas informações são recebidas através da nossa API que tem suas funcionalidades de CADASTRO, CONSULTA, ALTERAÇÃO e EXCLUSÃO como foi descrito na tabela de endpoint acima.