FILA ZERO

- **Harã Heique**
- **Jennifer Amaral**
- **Lucas Gomes**
- **Luiz Henrique**

Introdução

Os motivos da escolha do sistema proposto são da população não ter a necessidade de ir até o posto sem ter a certeza da obtenção do medicamento, o que consequentemente evitaria filas enormes para a solicitação.

Modelagem do sistema

Diagrama de classes - atendimento população

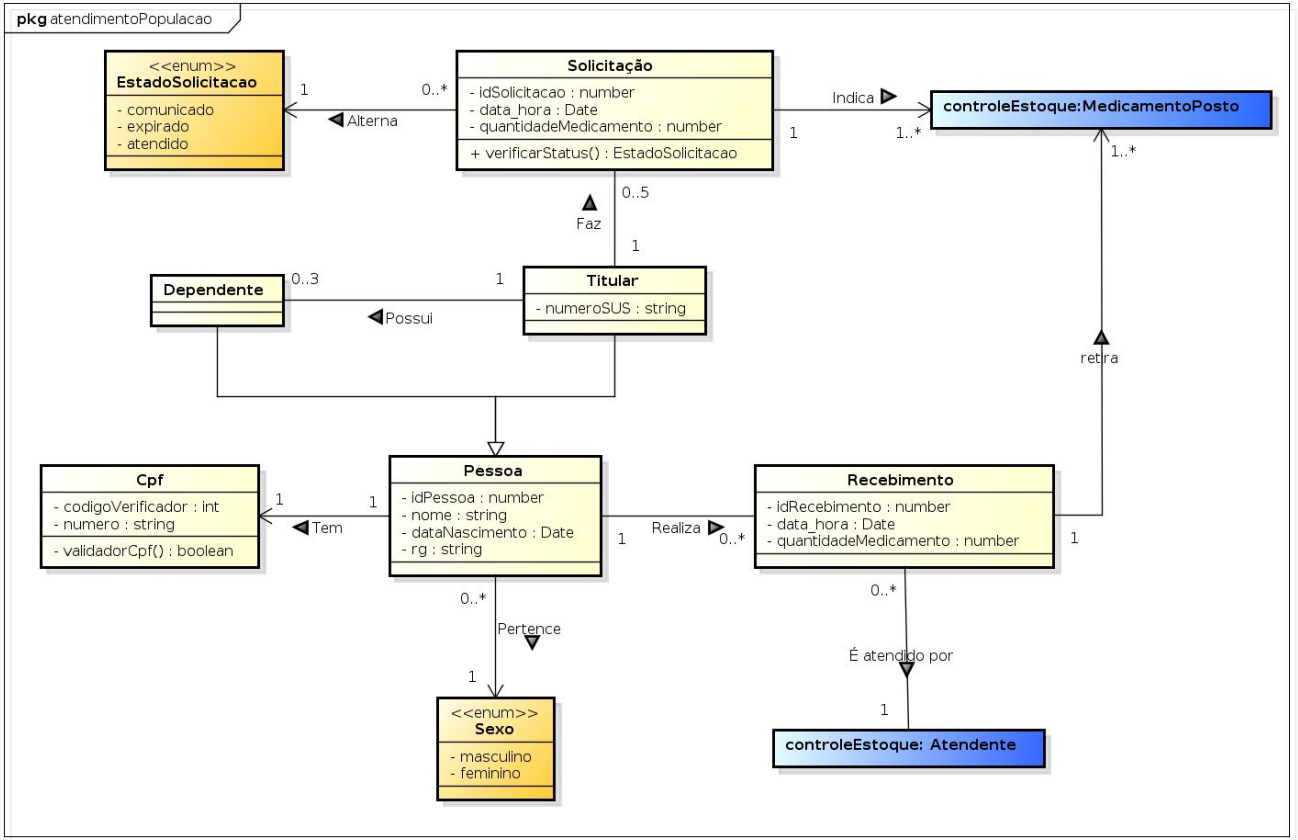
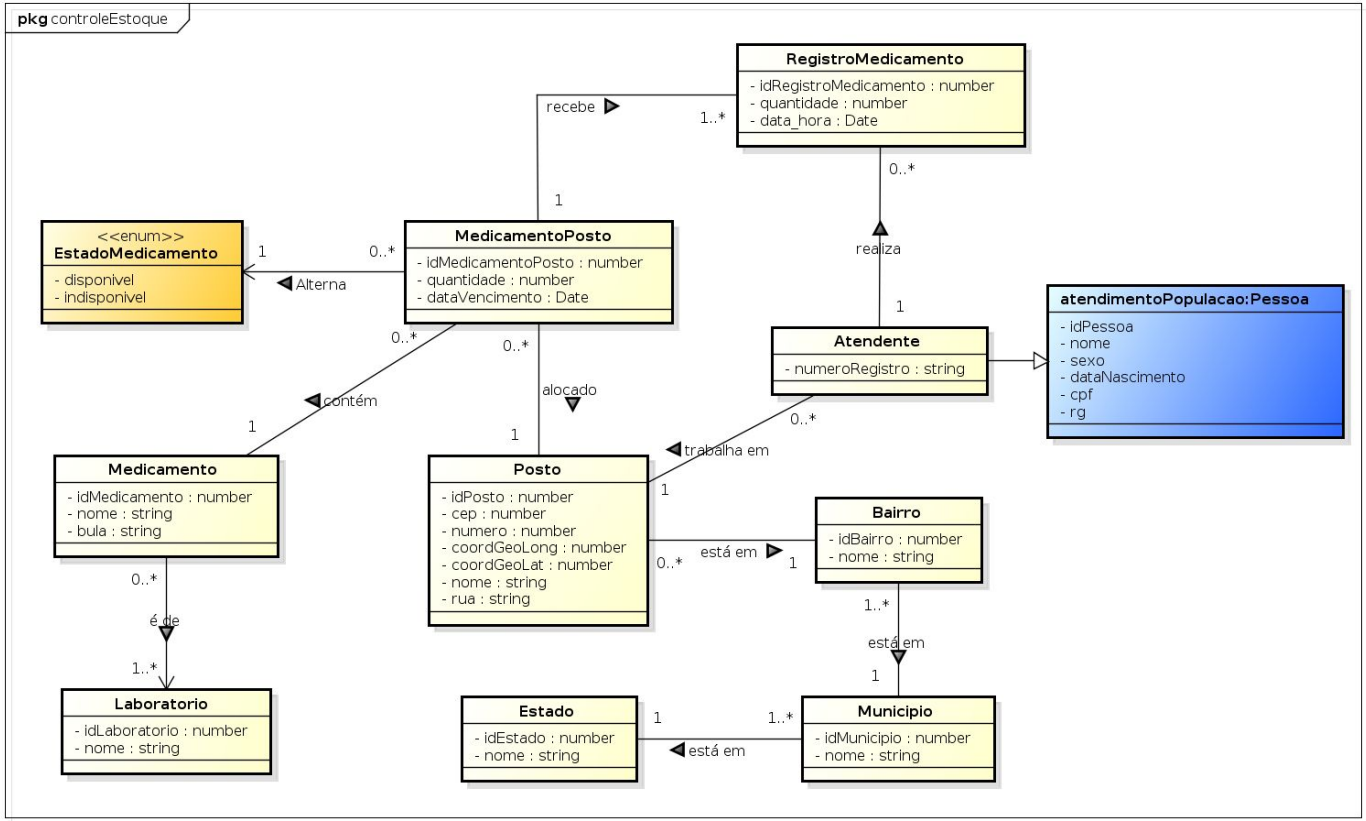
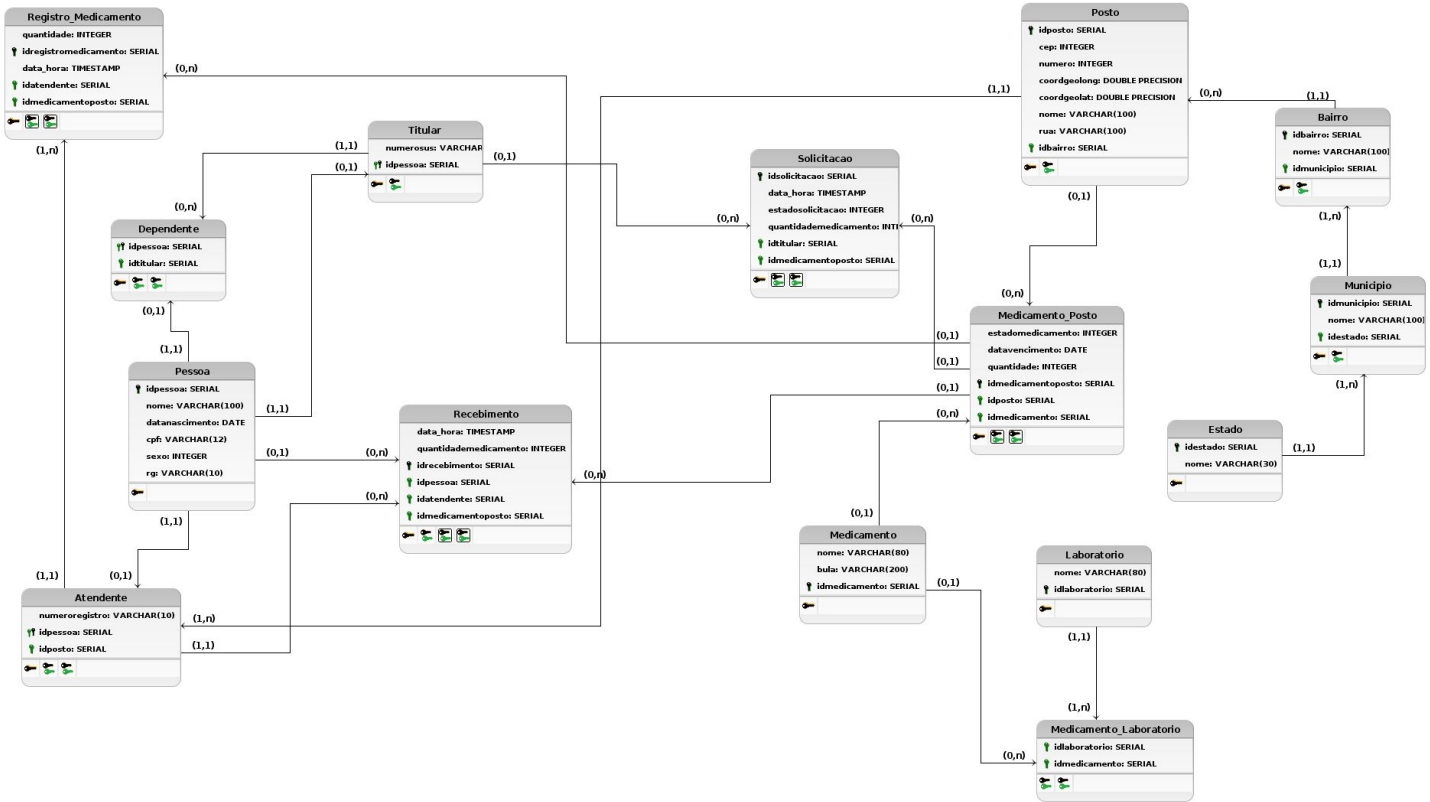


Diagrama de classes - controle de estoque

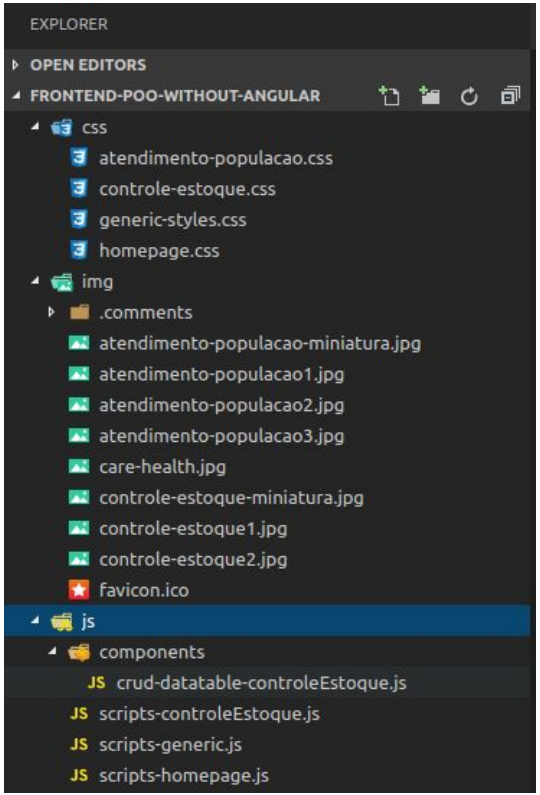


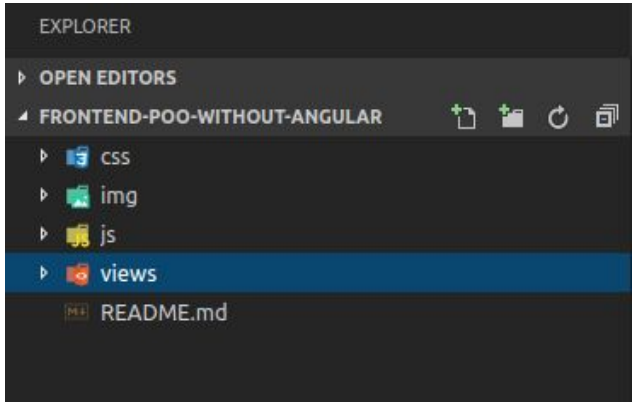
Modelo lógico do banco de dados

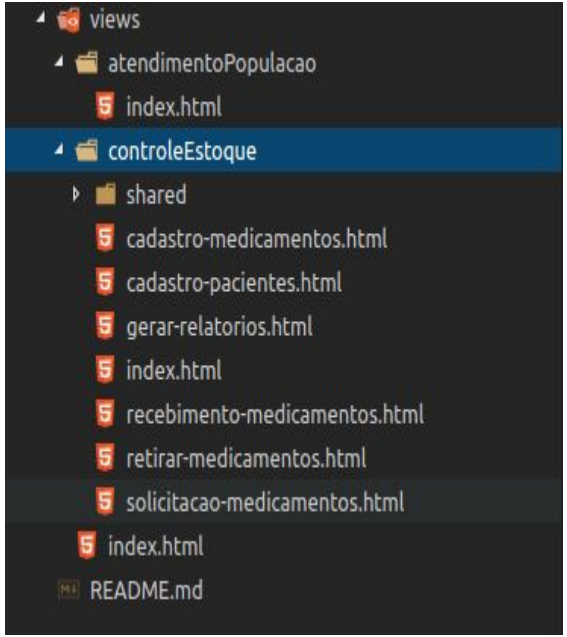


FRONTEND

Estrutura do código



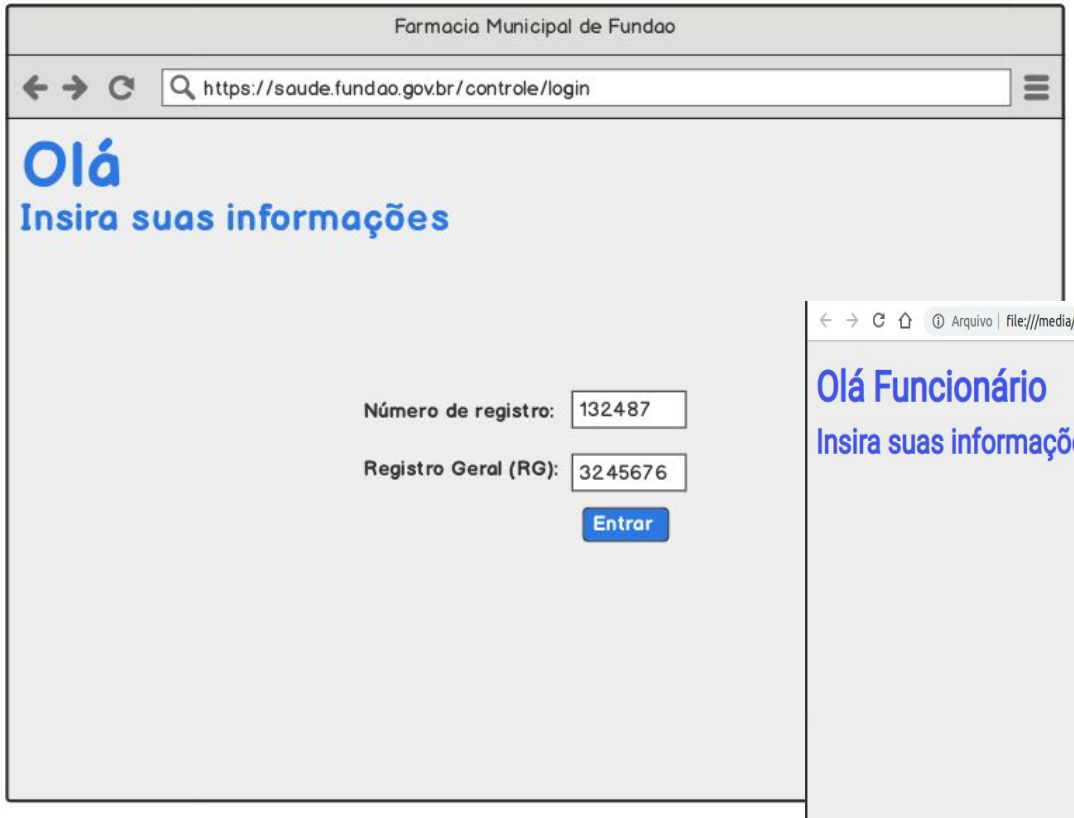


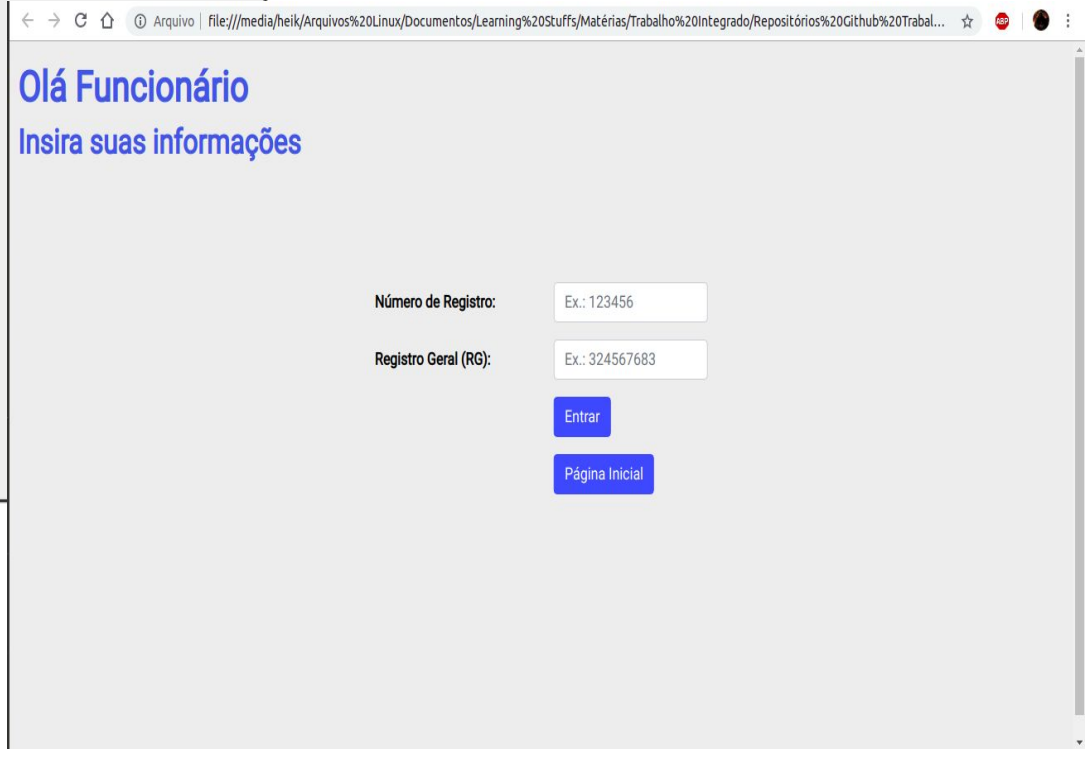


ESCOLHA O SEU DESTINO

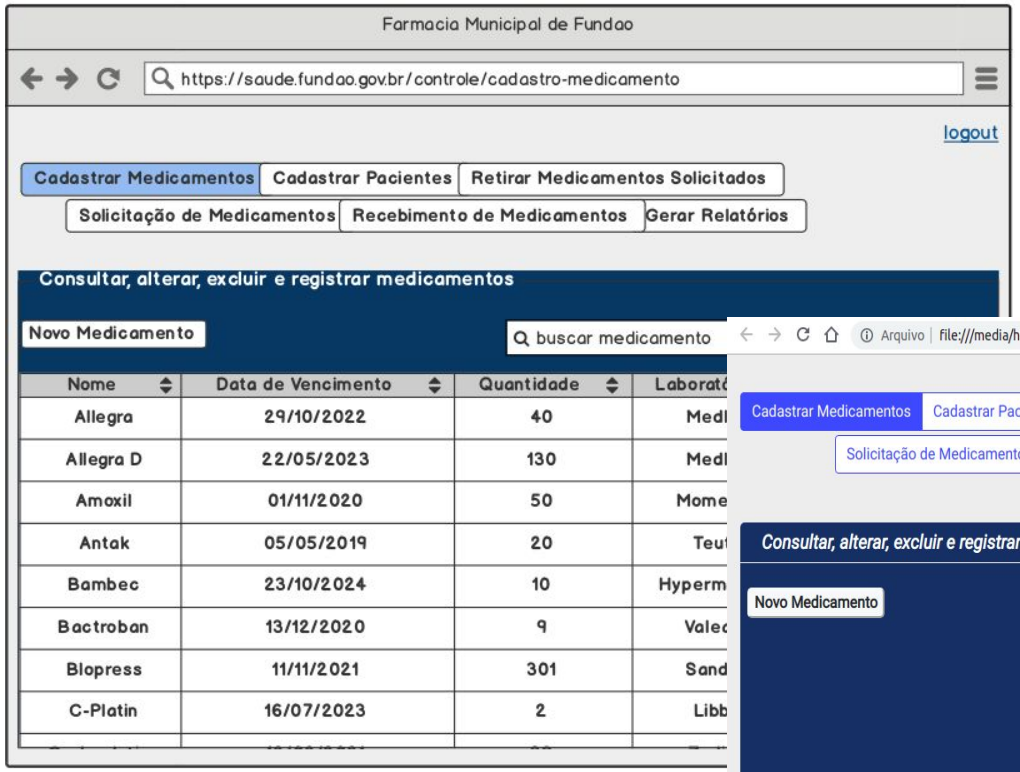
HOMEPAGE: REDIRECIONAMENTO PARA OS SUBSISTEMAS

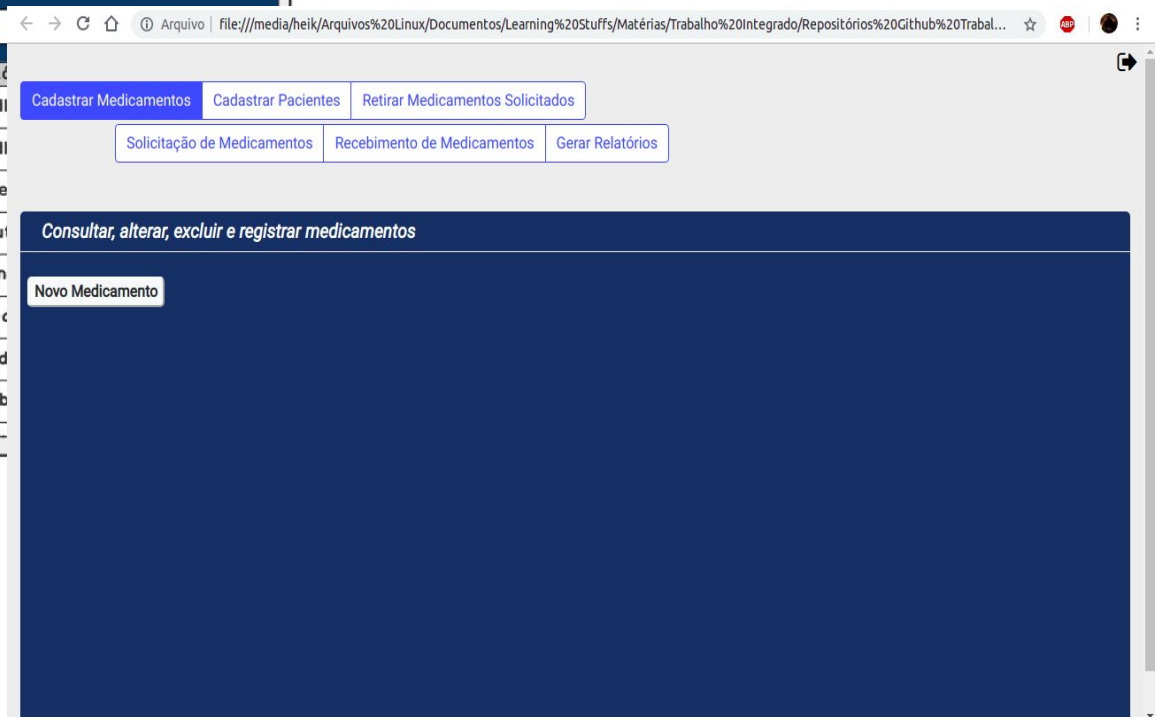
Protótipo vs aplicação



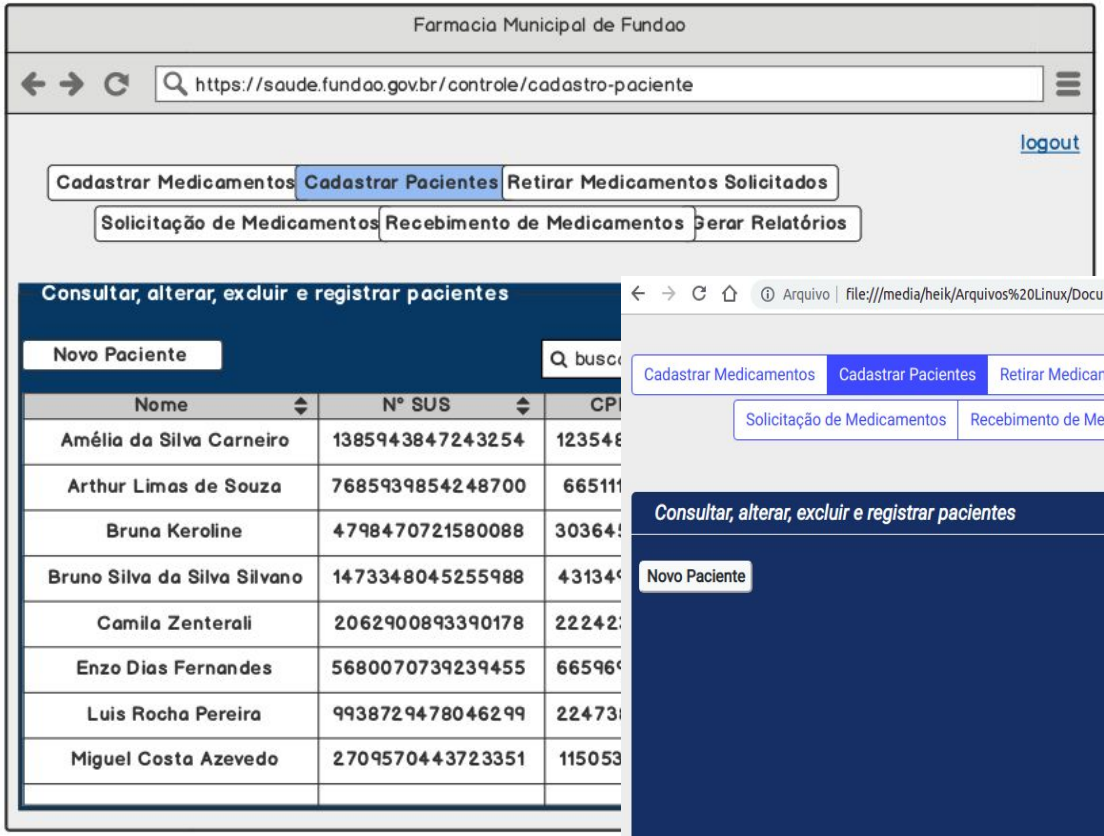


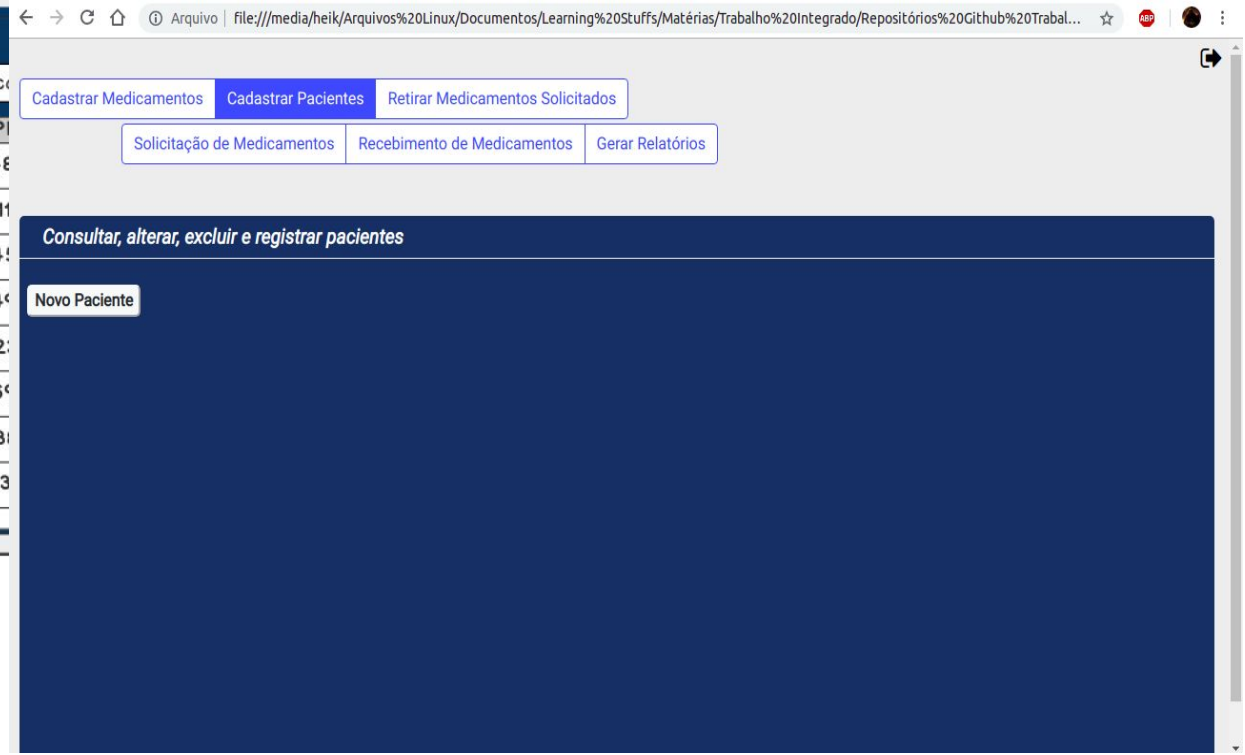
Protótipo vs aplicação



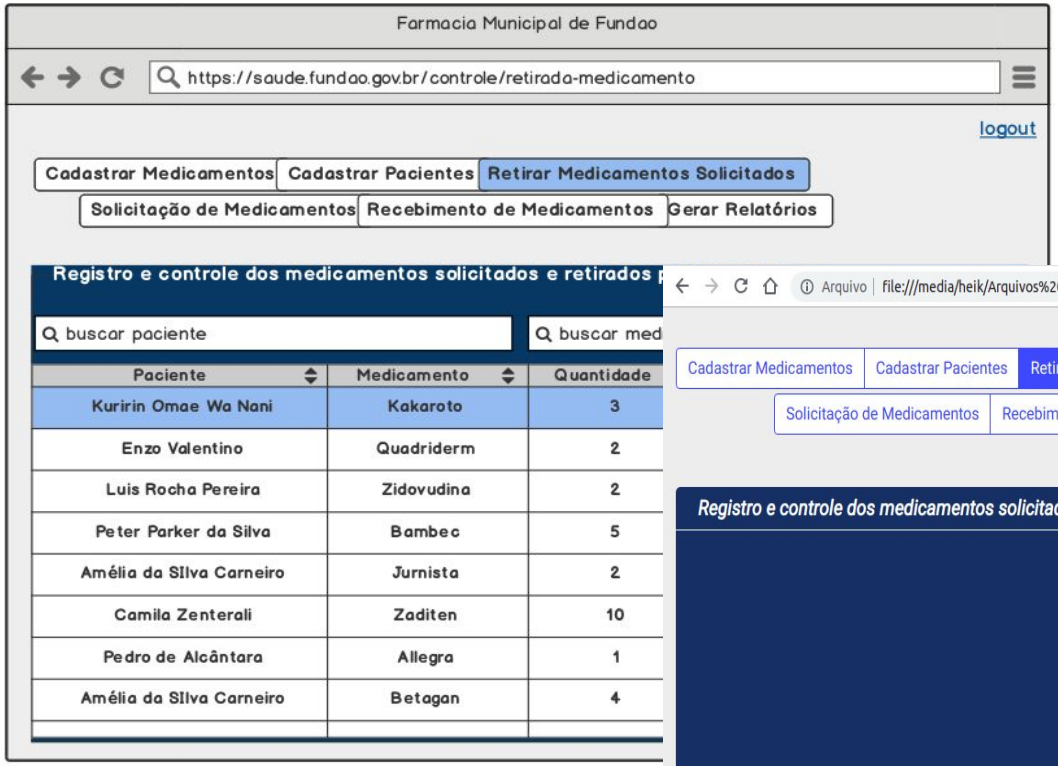


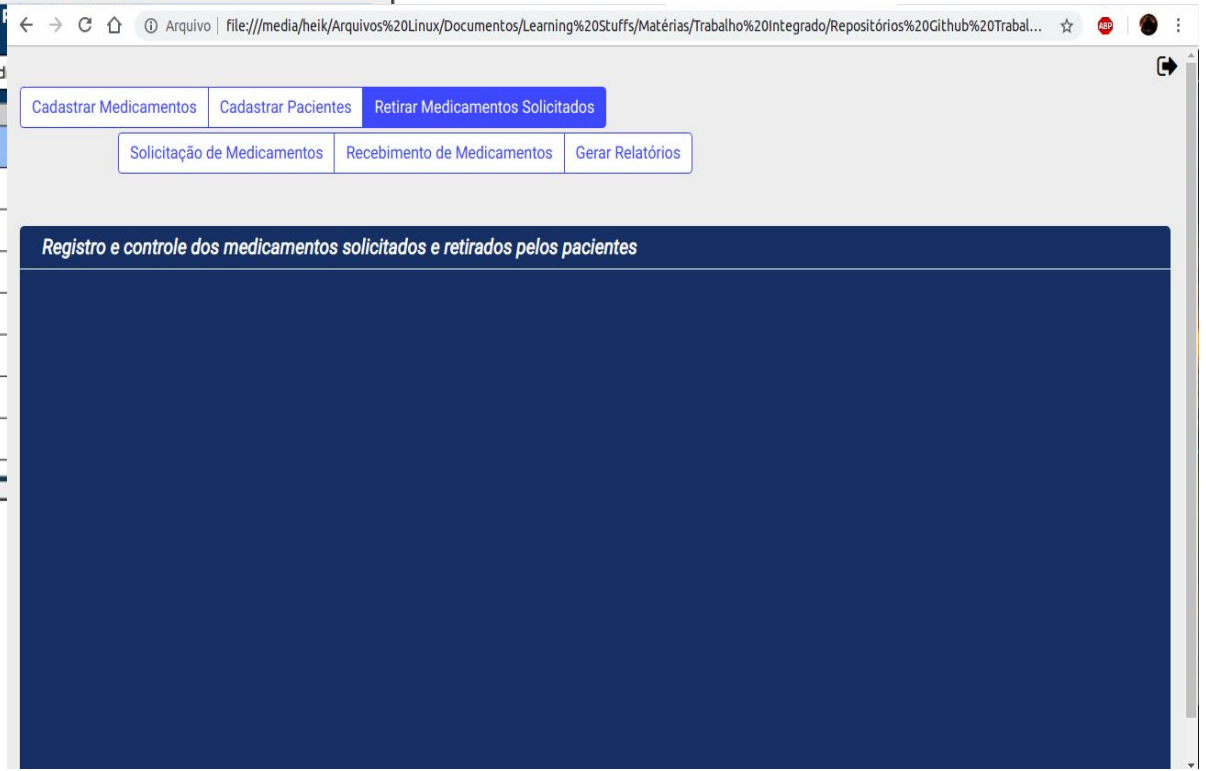
Protótipo vs aplicação





Protótipo vs aplicação





O QUE ESTÁ SENDO E SERÁ UTILIZADO?

- Pilares do front-end:
  - HTML;
  - CSS;
  - JS;
- Bibliotecas, frameworks e plugins:
  - Jquery;
  - Bootstrap;
  - DataTables JS (plugin for jquery);
  - Highcharts;
  - FontAwesome web Application;
  - Afins...

Backend

Estrutura do código

- src
  - **Controller**
    - atendente-controller.ts
  - **Service**
    - atendente-service.ts
  - **Model**
    - atendente-entity.ts
  - **Database**
    - database.provider.ts

Entidade - Atendente

```
@Entity()      You, 2 months ago • criando tabelas do BD
export class Atendente extends BaseEntity {
  @PrimaryColumn()
  idpessoa: number;

  @Column({ nullable: false, length: 10, unique: true })
  numeroregistro: string;

  //#####################################################################
  //############################ RELAÇÕES ################################
  //#####################################################################

  @OneToMany(type => Recebimento, recebimento => recebimento.atendente)
  recebimento: Recebimento[];

  @ManyToOne(type => Posto, posto => posto.atendente, {cascade: true, onDelete: "CASCADE"})
  @JoinColumn({ name: 'idposto' })
  posto: Posto;

  @OneToMany(
    type => RegistroMedicamento,
    registroMedicamento => registroMedicamento.atendente,
  )
  registroMedicamento: RegistroMedicamento[];

  @ManyToOne(type => Pessoa, pessoa => pessoa.atendente, {
    eager: true, cascade: true, onDelete: "CASCADE"
  })
  @JoinColumn({ name: 'idpessoa' })
  pessoa: Pessoa;
```

Controller - Atendente

```
@Get('/atendente')
async readAll(@Res() res) {
  try {
    let atendente: Atendente[] = await this.atendenteService.readAll();
    if (atendente != undefined) {
      res.status(HttpStatus.OK).send(atendente);
    } else {
      res
        .status(HttpStatus.NOT_FOUND)
        .send('Nenhum atendente encontrado na busca');
    }
  } catch (err) {
    res.status(HttpStatus.BAD_GATEWAY).send(err.message);
  }
}

@Post('/atendente/create')
async Create(@Res() res, @Body() body) {
  try {
    let atendente = await this.atendenteService.Create(body);
    if (atendente != undefined) {
      res.status(HttpStatus.OK).send(atendente);
    } else {
      res
        .status(HttpStatus.NOT_FOUND)
        .send('Erro ao salvar o atendente');      You, a few seconds ago
    }
  } catch (err) {
    res.status(HttpStatus.BAD_GATEWAY).send(err.message);
  }
}
```

Service - Atendente

```
async readAll(): Promise<Atendente[]> {
  return Atendente.find();
}

async readOne(id: number): Promise<Atendente> {
  return Atendente.findOne({ idpessoa: id });
}

async Create(body: any): Promise<Atendente> {
  let atendente = new Atendente();
  try {
    atendente.numeroregistro = body.numeroregistro;
    atendente.idpessoa = body.idpessoa;
    atendente.posto = body.idposto;
    return await Atendente.save(atendente);
  } catch (err) {
    throw new Error(
      `Erro ao salvar atedente\n Erro: ${err.name}\n Mensagem: ${
        err.message
      }\n Os parametros estao certos?`,
    );
  }
}
```

Conexão com o banco de dados

```
export const databaseProviders = [
  {
    provide: 'DbConnectionToken',
    useFactory: async () => await createConnection({
      type: 'postgres',
      host: 'elmer.db.elephantsql.com',
      port: 5432,
      username: 'aiexkamd',
      password: 'QpfBdkd4AT2chCRjdjGbPPoSXdctwU9y',
      database: 'aiexkamd',
      entities: [
          __dirname + '/../**/*.entity{.ts,.js}',
      ],
      synchronize: true,
    }),
  },
];      You, 24 days ago • fazendo crud
```

Padrões de projeto

```typescript
@Entity()
export class Atendente extends BaseEntity {
  @PrimaryColumn()
  idpessoa: number;
```

```typescript
  /**
   * Gets current entity's Repository.
   */
  static getRepository<T extends BaseEntity>(this: ObjectType<T>): Repository<T>;
  /**
   * Returns object that is managed by this repository.
   * If this repository manages entity from schema,
   * then it returns a name of that schema instead.
   */
```

```typescript
/**
 * Finds entities that match given options.
 */
static find<T extends BaseEntity>(this: ObjectType<T>, options?: FindManyOptions<T>): Promise<T[]>;
```

```typescript
@Injectable()
export class AtendenteService {

  async buscaTodosAtendentes(): Promise<Atendente[]> {
    return await Atendente.find();
  }
```

Padrão Repository

Padrão método Fábrica

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  await app.listen(parseInt(process.env.PORT) || 3001);
}
bootstrap();
```

```
export declare class NestFactoryStatic {
    private readonly logger;
    /**
     * Creates an instance of the NestApplication
     * @returns {Promise}
     */
    create(module: any): Promise<INestApplication & INestExpressApplication>;
```

```
@Module({
  providers: [...modelProvider, ...modelService],
  controllers: [...modelController],
})
export class PooModule {}
```

Padrão injeção de dependência

```
@Injectable()
export class AtendenteService {
      You, 23 days ago • criando tabelas do BD
  async buscaTodosAtendentes(): Promise<Atendente[]> {
    return await Atendente.find();
  }
```

```
/**
 * Defines the injectable class. This class can inject dependencies through constructor.
 * Those dependencies have to belong to the same module.
 */
export declare function Injectable(): ClassDecorator;
```

```
declare type ClassDecorator = <TFunction extends Function>(target: TFunction) => TFunction | void;
```

```
        * @param args A list of arguments the function accepts.
        */
    new(...args: string[]): Function;
    (...args: string[]): Function;
    readonly prototype: Function;
}

declare const Function: FunctionConstructor;
```

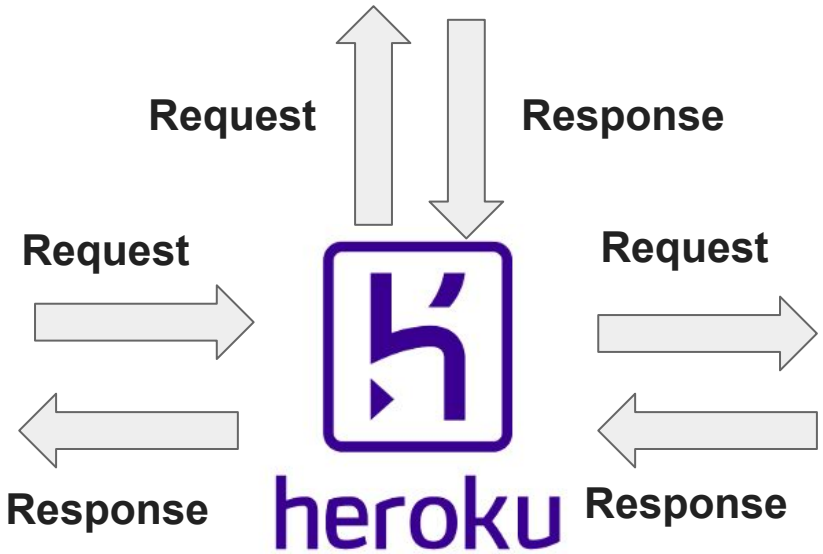
Implantação

Implantação

A API se encontra atualmente no heroku.

O Heroku é uma plataforma de cloud que oferece "Platform as a Service", ou seja, ele permite que você hospede suas aplicações em um ambiente facilmente escalável e com suporte a várias tecnologias. Ele tem um plano free, que é indicado para testes, e opções pagas com mais funcionalidades e suporte.

Arquitetura



ElephantSQL

GitHub

Resposta do heroku

https://poo2.herokuapp.com/atendente


```
[
  {
    "idpessoa": 963,
    "numeroregistro": "4",
    "pessoa": {
      "idpessoa": 963,
      "nome": "Diva Pinheiro Carneiro",
      "datanascimento": "1988-06-28T00:00:00.000Z",
      "cpf": "963",
      "sexo": 0,
      "rg": "963"
    }
  },
  {
    "idpessoa": 66,
    "numeroregistro": "5",
    "pessoa": {
      "idpessoa": 66,
      "nome": "Elisa Cerejeira",
      "datanascimento": "1971-01-08T00:00:00.000Z",
      "cpf": "66",
      "sexo": 0,
      "rg": "66"
    }
  },
```

Github

- https://github.com/lukasg18/Topicos-Trabalho-BD2
- https://github.com/lukasg18/poo2-backend
- https://github.com/HaraHeique/frontend-POO-without-angular