

IPCA

Escola Superior Tecnologia



Projeto de Desenvolvimento de Software

Joel Jonassi, nº19698

Joel Phillippe Melo Figueiras, nº20809

Nuno Miguel Carvalho Araújo, nº 20078

Tiago João Coelho Azevedo, nº21153

Vitor Hugo Sá Machado, nº21158

Professor Nuno Feixa Rodrigues

Professor Óscar Ribeiro

2021/2022

Índice

1	Introdução	1
1.1	Contextualização	1
1.2	Motivação e Objetivos	1
1.3	Âmbito do projeto	1
2	Requisitos funcionais - Diagramas UML	2
2.1	Diagrama de casos de uso	2
2.2	Diagrama de componentes	4
2.3	Diagrama de Visão geral de interação	5
2.4	Diagrama de Classes UML	7
2.5	Diagrama de Entidade relação	8
3	Requisitos Não Funcionais	9
3.1	Escalabilidade	9
3.2	Performance	9
3.3	Usabilidade	9
3.4	Disponibilidade	10
3.5	Manutenção	10
3.6	Segurança	11
3.7	Cultura	11
3.8	Portabilidade e compatibilidade	11
4	Mockups	12
5	Product Backlog	16
6	Divisão das sprints	17
6.1	Análise de requisitos e modelação	17
6.2	vAlfa	18
6.3	vBeta	19
6.4	vRTW	19
7	Versão Alfa	20
7.1	Teste Unitário - Artigo	20
7.2	Teste Unitário - Código	22
7.3	Teste Unitário - Estado	23
7.4	Teste Unitário - Fase Processual	24

7.5	Teste Unitário - Prazo Codigo	27
7.6	Teste Unitário - Prazo	28
7.7	Teste Unitário - Processo	30
7.8	Teste Unitário - Tema	32
7.9	Teste Unitário - Tipo Prazo	34
7.10	Teste Unitário - Tipo Processo	35
7.11	Teste Unitário - Utilizador	36
7.12	Bugs Conhecidos	38

1 Introdução

1.1 Contextualização

No âmbito da unidade curricular de Projeto de Desenvolvimento de Software, foi solicitado a elaboração de um projeto, com o tema à nossa escolha.

Em grupo, optamos por desenvolver um portal de apoio jurídico.

1.2 Motivação e Objetivos

O projeto tem como objetivo apresentar os princípios e os valores do desenvolvimento ágil de projetos de software e incentivar os membros da equipa a utilizar as técnicas e ferramentas mais adequadas, à luz destes princípios, ao longo de todo o processo de desenvolvimento do sistema de software.

É pretendido que o grupo consiga planear, gerir e executar todas as atividades que constam no processo de desenvolvimento de um sistema de software.

1.3 Âmbito do projeto

Como mencionado anteriormente, decidimos elaborar um portal de apoio jurídico, onde todos os juristas Portugueses poderão armazenar os processos, relembrar, atualizar, modificar, e planear datas referentes a cada processo que os mesmos terão em mão.

A ideia para este projeto surgiu após um stakeholder da área lamentar a falta um site/aplicação que pudesse satisfazer as suas necessidades concretas.

Como nenhum membro da equipa tem conhecimento na área, tivemos de proceder a alguma pesquisa e fazer o levantamento de requisitos com a ajuda de um stakeholder que solicitou o desenvolvimento do portal.

Solicitamos, então, uma descrição concreta de todas as componentes funcionais pretendidas, bem como uma síntese geral do assunto a ser implementado, de forma a conseguirmos delinear todas as fases de desenvolvimento do sistema de software.

2 Requisitos funcionais - Diagramas UML

2.1 Diagrama de casos de uso

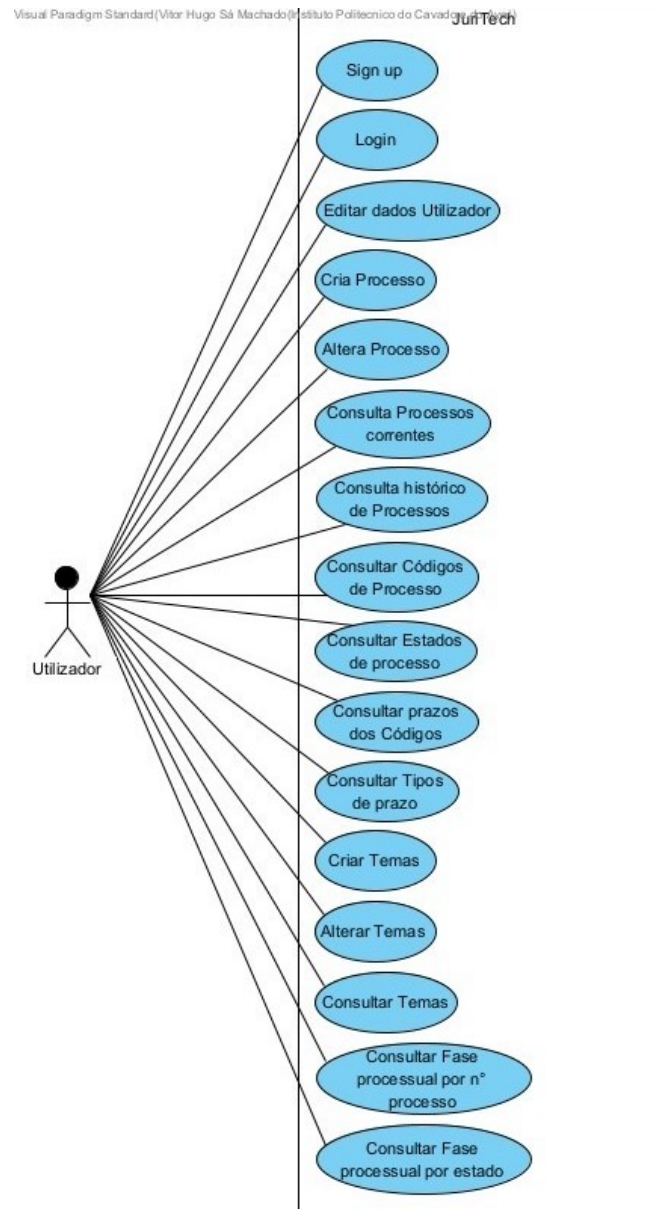


Figura 1: Diagrama de caso de usos(1)

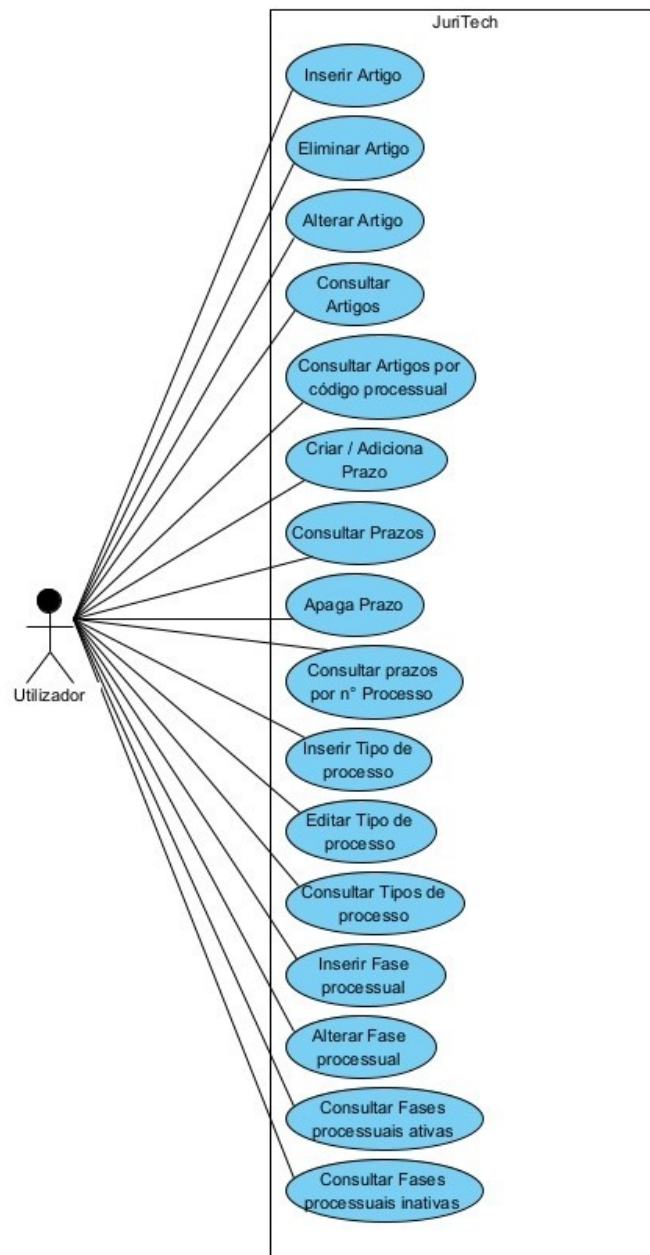


Figura 2: Diagrama de caso de usos(2)

2.2 Diagrama de componentes

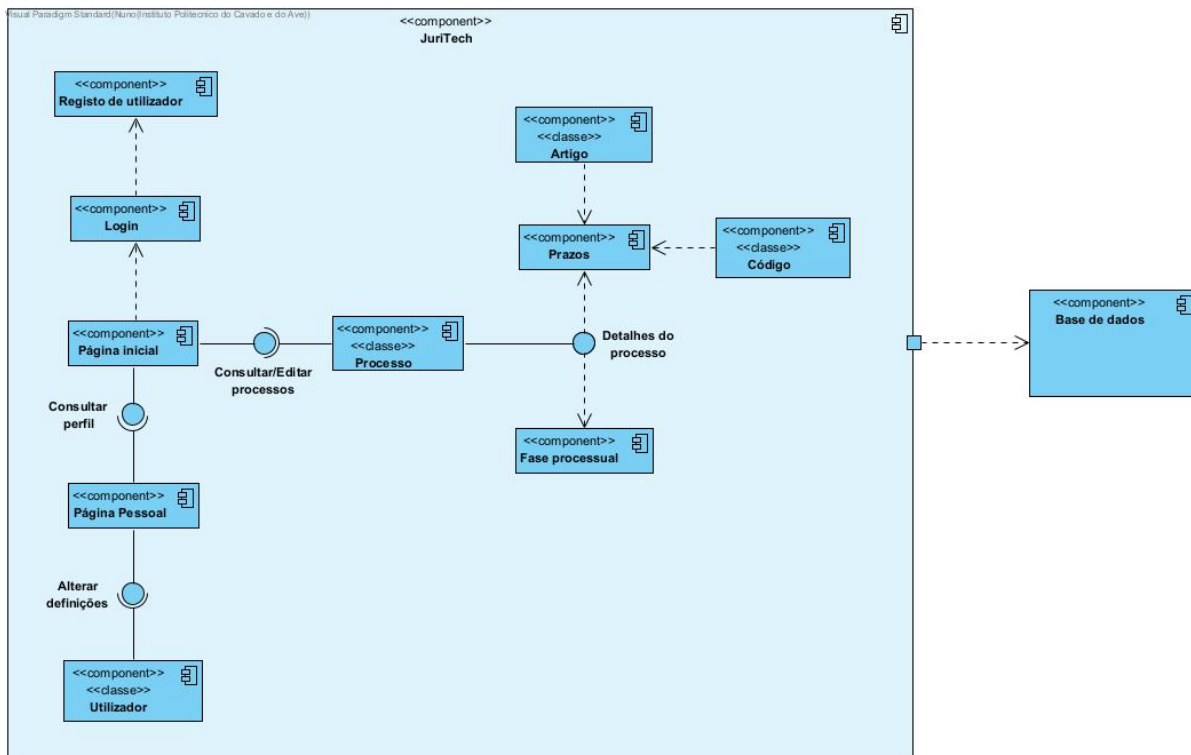


Figura 3: Diagrama de componentes

2.3 Diagrama de Visão geral de interação

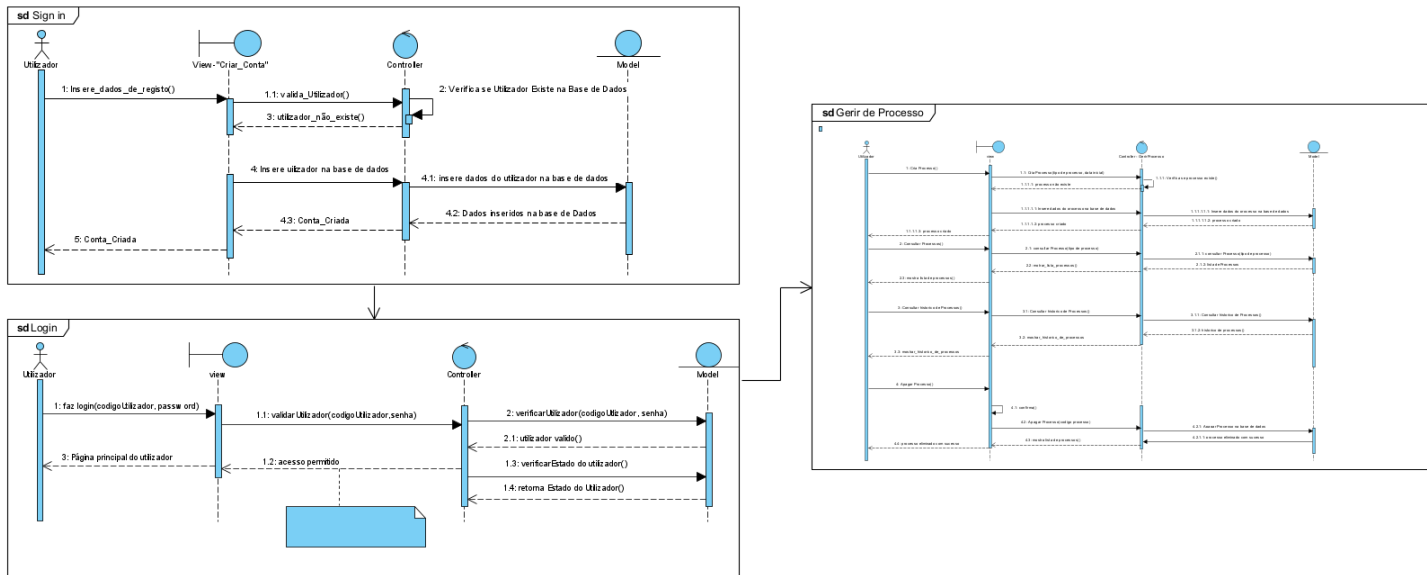


Figura 4: Diagrama de visão geral de interação

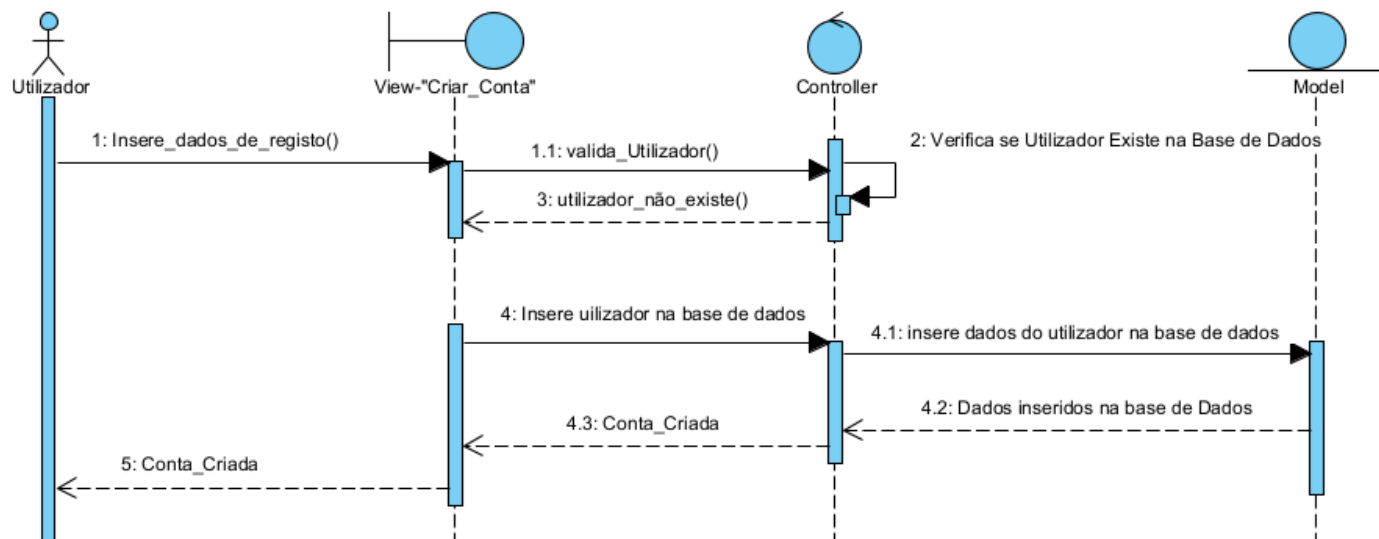


Figura 5: Diagrama de visão geral de interação "Sign Up"

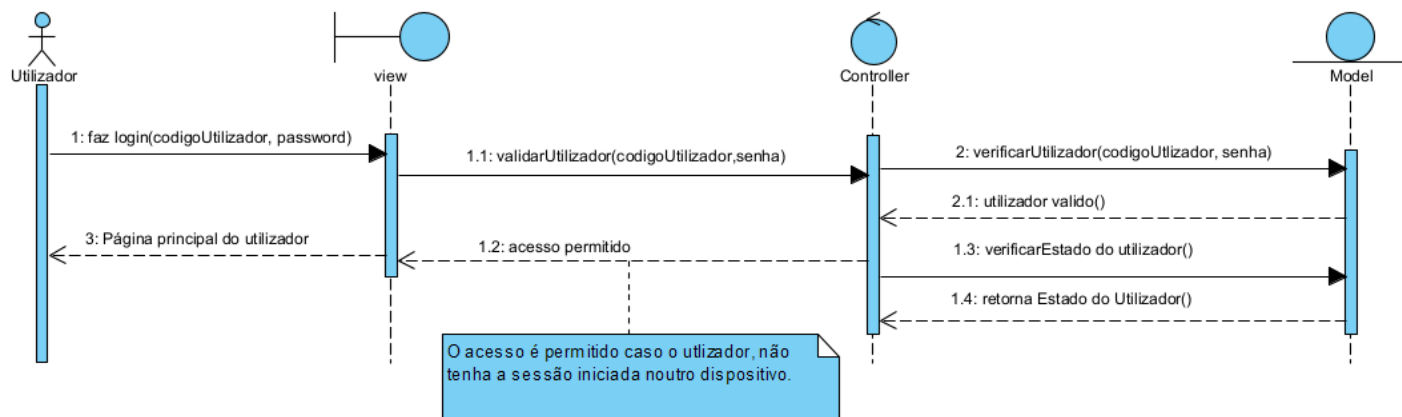


Figura 6: Diagrama de visão geral de interação "Login"

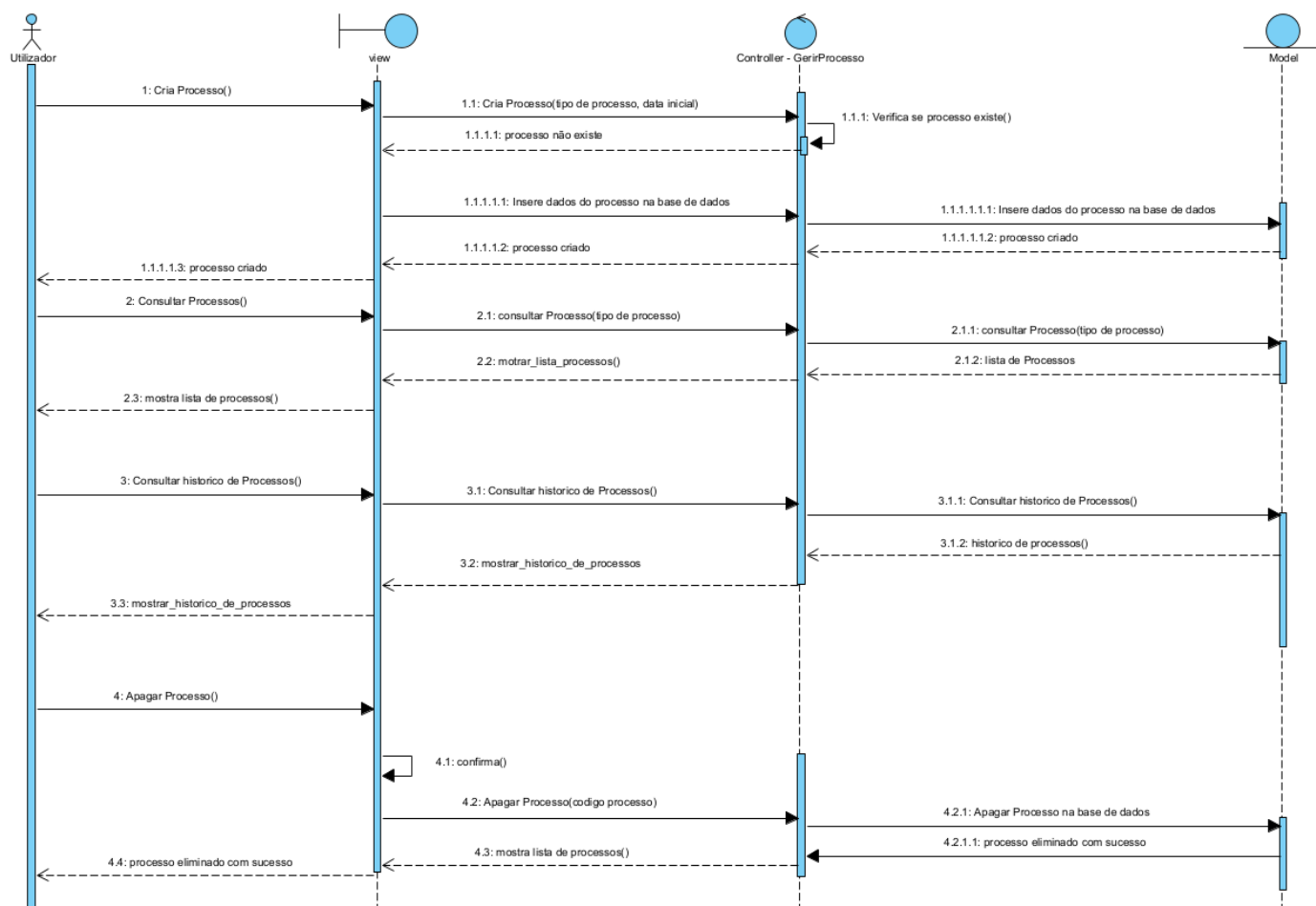


Figura 7: Diagrama de visão geral de interação "Gerir Processo"

2.4 Diagrama de Classes UML

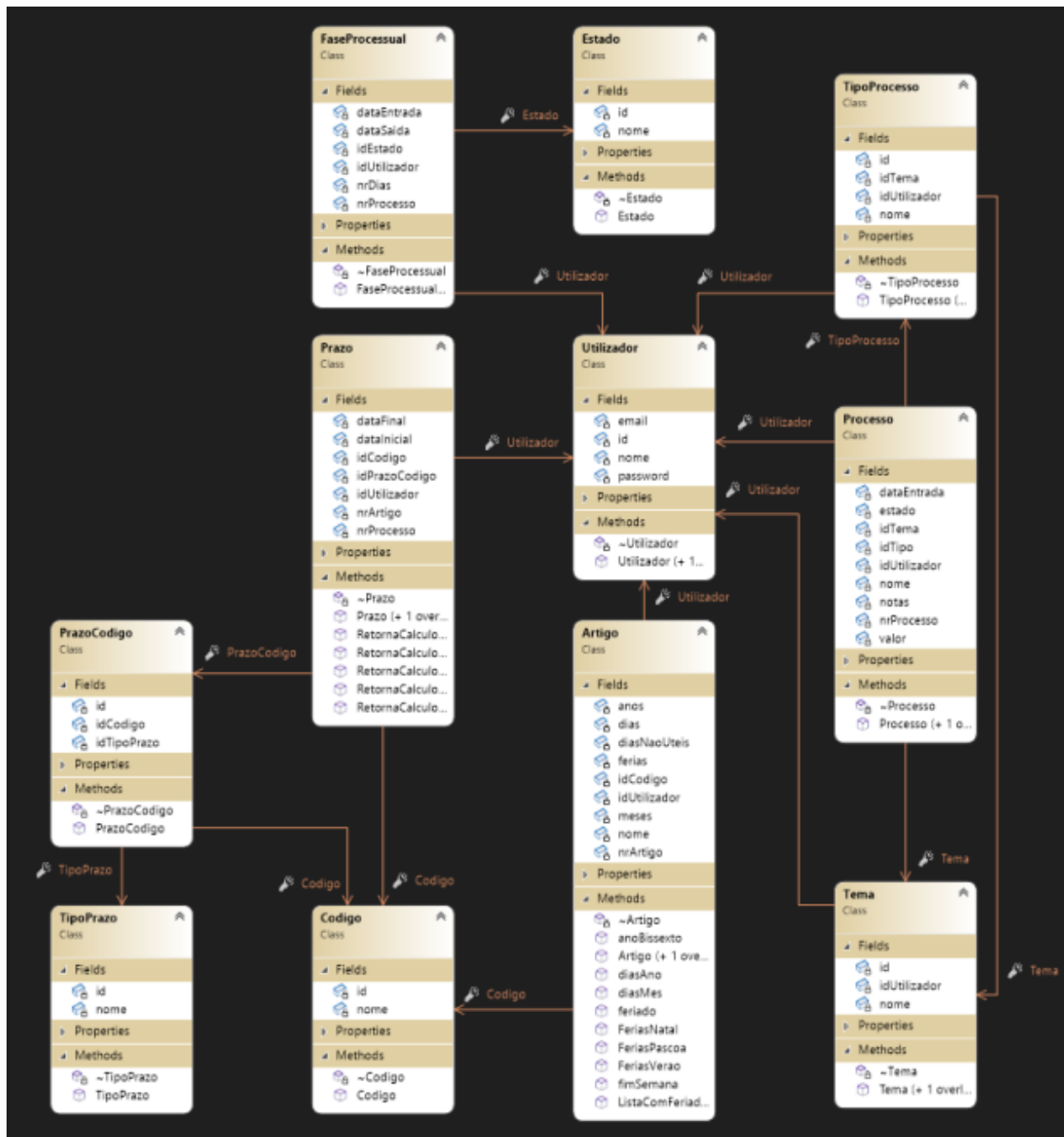


Figura 8: Diagrama de classes UML

2.5 Diagrama de Entidade relação

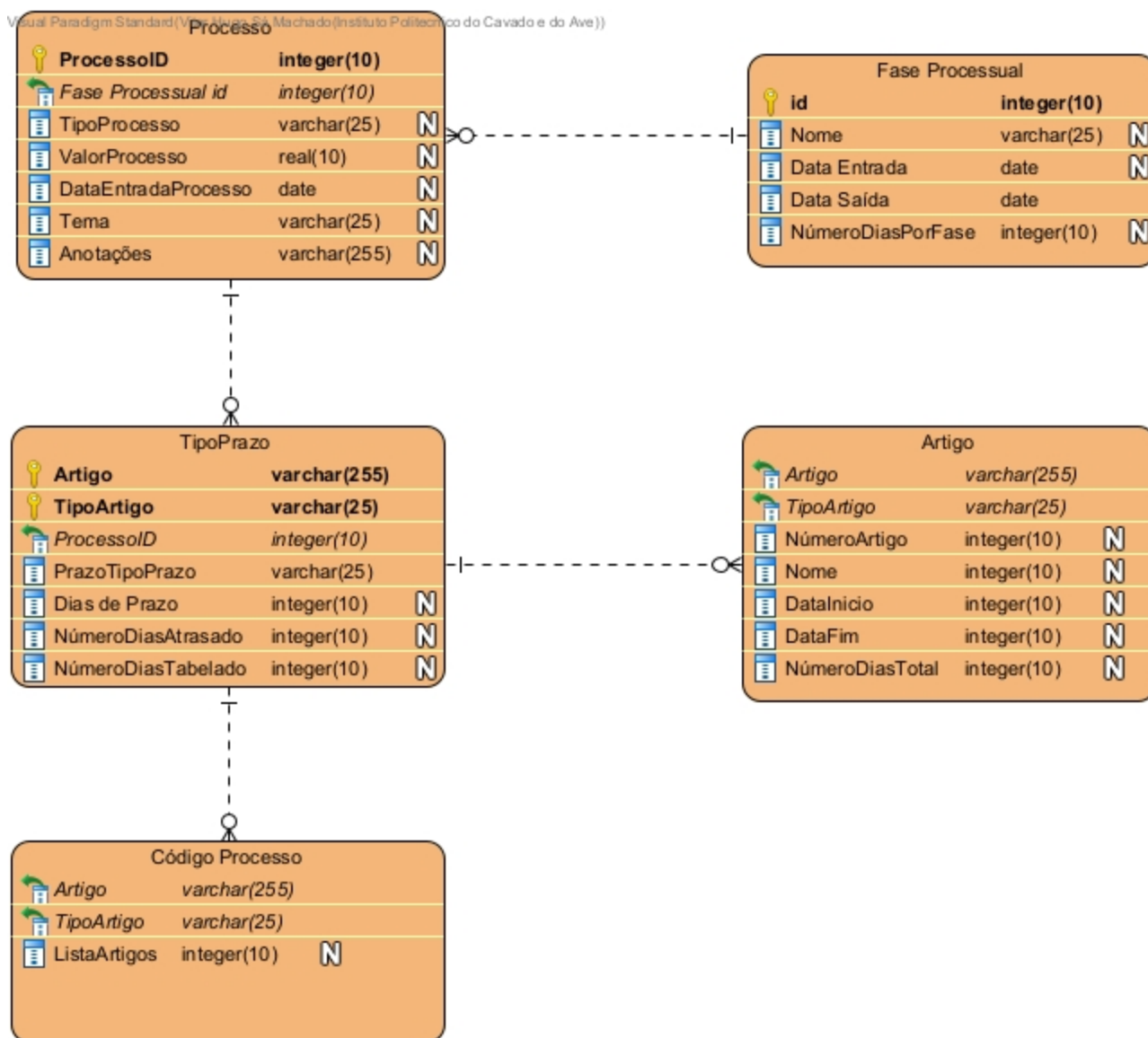


Figura 9: Diagrama entidade relação - diagrama da futura base dados

3 Requisitos Não Funcionais

3.1 Escalabilidade

- A aplicação suporta 10 000 requisições ao servidor, para tal deverá ser escalável verticalmente.

3.2 Performance

- O utilizador não deve esperar mais de 3 segundos, em média, para visualizar novo conteúdo após um clique do rato, a parte gráfica não será o foco principal.
- Necessita de uma quantidade maior de espaço em disco para processar o grande volume de dados. Para atender a essa necessidade, o armazenamento disponível deve ser de 1TB de espaço em disco.

3.3 Usabilidade

- A plataforma seguirá os padrões de software já conhecidos, tornando-o fácil de utilizar e de aprender:
- Facilidade de aprender: O tempo e o esforço exigido para poder usar o sistema é de 2 dias de treino.
- Facilidade de uso: Velocidade de execução de tarefas pela adição de atalhos, redução de erros com a implementação de perguntas para determinadas operações no sistema.
- Serão Adicionadas teclas de atalho dinâmicas das funcionalidades mais utilizadas de acordo com as requisições do utilizador.

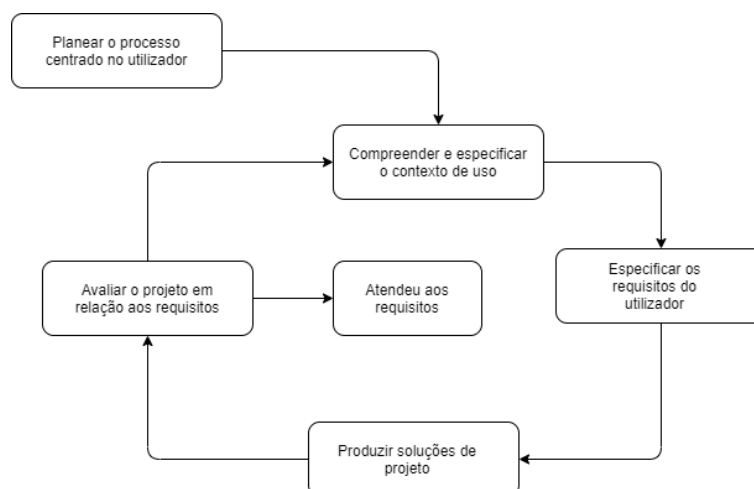


Figura 10: Processo do projeto centrado no utilizador

3.4 Disponibilidade

- Pretende-se um sistema disponível em qualquer dispositivo que tenha acesso à internet e um browser instalado, sendo os mais importantes os telemóveis, computadores e tablets.

Tendo em conta a necessidade da utilização do portal por parte dos usuários, as atualizações necessárias serão feitas em horas de baixa adesão. O site estará inacessível por um período máximo de duas horas, de forma a não comprometer o seu normal funcionamento.

- Os dados indispensáveis serão alocados também num servidor externo, assim sendo, se o servidor interno estiver inacessível, o utilizador poderá aceder, na mesma, a determinados dados.

3.5 Manutenção

- Todos os anos, pelo menos uma vez, o sistema irá sofrer uma intervenção manual, para atualizar a calendarização do ano seguinte;
- Como temos um objetivo inicial simples e concreto, haverá um esforço por parte da equipa de desenvolvedores para adicionar novas funcionalidades e ferramentas que possam vir a ser úteis ao longo do tempo. Esta implementação terá em conta o feedback e as necessidades dos utilizadores;
- A interface do portal será atualizada, conforme a disponibilidade, para evitar que o seu uso frequente se torne monótono e enfadonho, proporcionando assim uma experiência mais agradável para o utilizador.

3.6 Segurança

- Sendo uma área de trabalho muito confidencial, o utilizador ao iniciar sessão terá de ter ou um dispositivo móvel ou um email associado à conta para receber um código de segurança (autenticação de dois fatores) de modo a prevenir tentativas de acessos alheios.
- Após a inatividade de um utilizador por mais de 60 minutos a sessão será encerrada automaticamente obrigando-o a efetuar o início de sessão quando retomar.
- Sendo uma aplicação WEB este não vai dar autorização ao navegador que guarde os dados (Username e Palavra-Passe) por exemplo numa conta google. Assim no caso de roubo ou perda de algum dispositivo este não terá acesso à aplicação.
- A aplicação deverá cumprir com as políticas de privacidade de dados.

3.7 Cultura

- Tendo em conta que a maioria dos países têm processos distintos na área judicial e o projeto se adequando aos processos portugueses implica a impossibilidade de “vender” o produto para fora de Portugal. Ou seja, para cada país iria necessitar de esquematizações diferentes para podermos “vender” o produto.

3.8 Portabilidade e compatibilidade

- O produto como é uma aplicação WEB poderá ser acedido por qualquer dispositivo em qualquer lugar sem necessitar dos dispositivos pessoais, e sem qualquer tipo de instalação.
- Inicialmente a aplicação WEB vai ser suportado para computador e depois também para telemóvel. Sendo uma aplicação WEB não teremos uma carga extra de adaptar tanto para ANDROID como para IOS. Ou seja, a aplicação irá ser compatível com qualquer plataforma.

4 Mockups

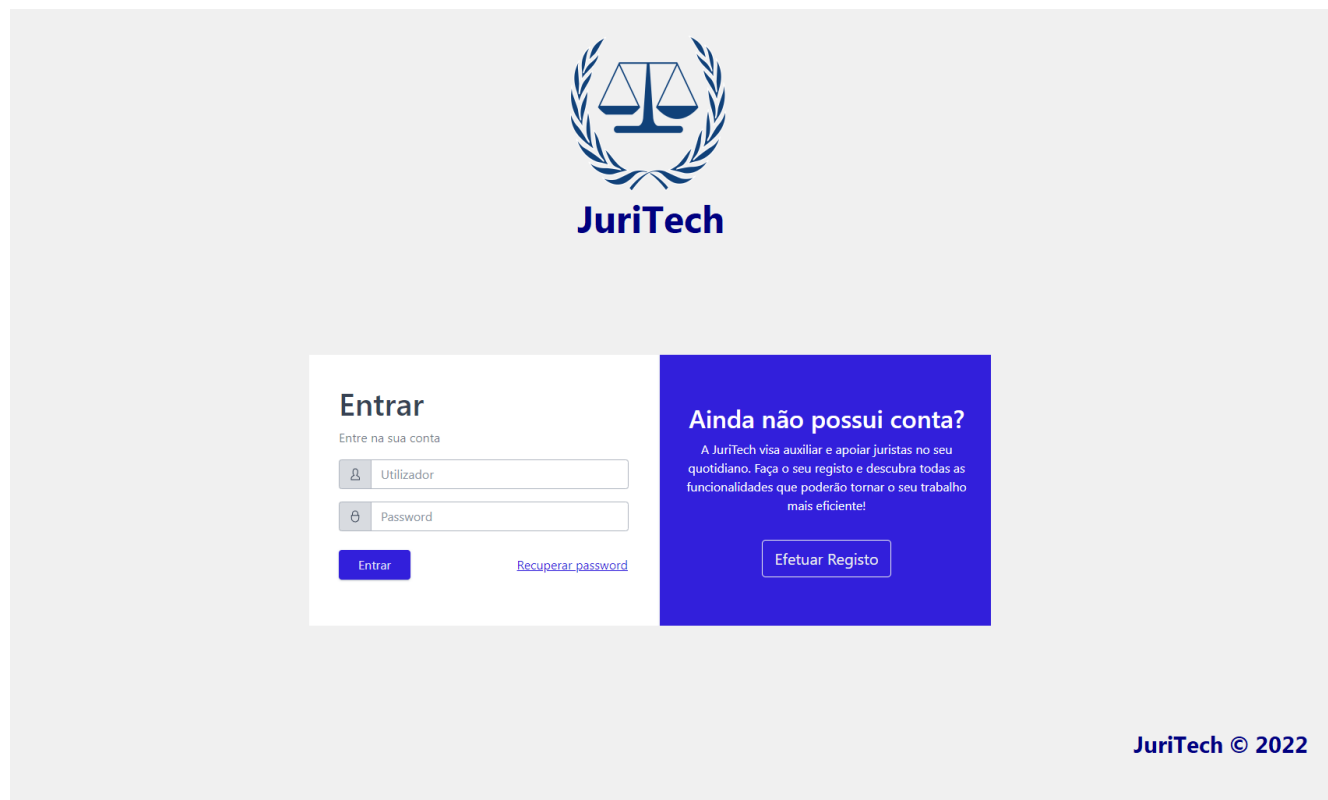


Figura 11: Mockup login

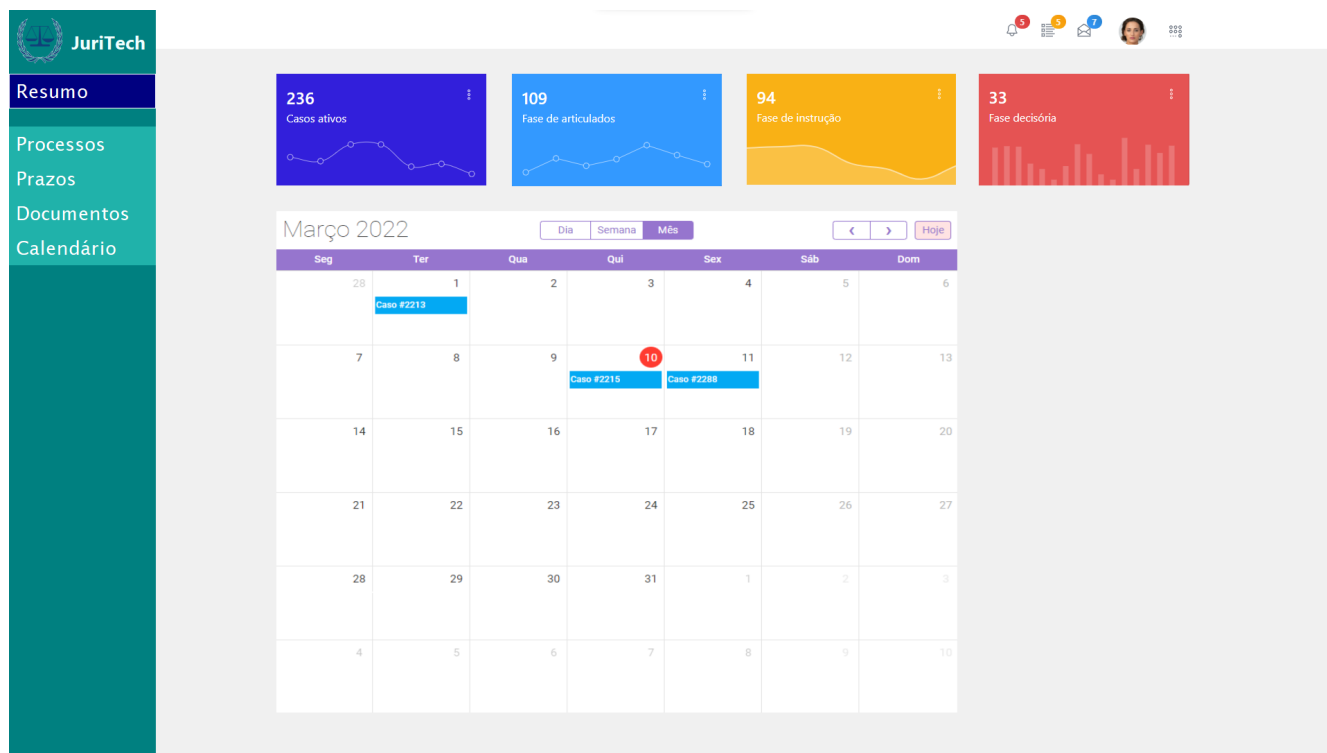


Figura 12: Mockup consulta calendário

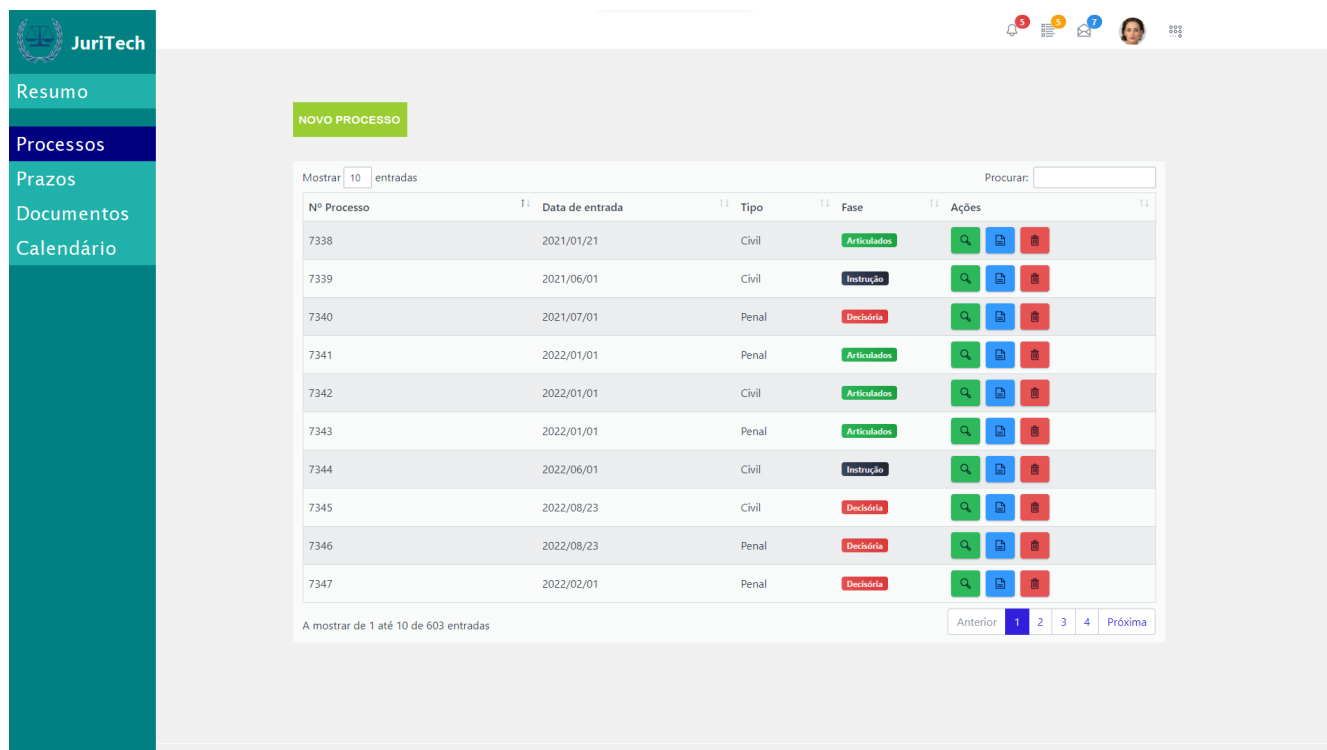


Figura 13: Mockup consulta processos

JuriTech

Resumo
Processos
Prazos
Documentos
Calendário

5

5

7

NOVO PROCESSO

Nº Processo

Tipo

Valor

Data de Entrada

Fase Processual

Tema

Anotações

CANCELAR

GRAVAR

Figura 14: Mockup inserir processo

JuriTech

Resumo
Processos
Prazos
Documentos
Calendário

5

5

7

NOVO PRAZO

Mostrar 10 entradas


Procurar:

Nº de Processo	Dias de Prazo	Artigo	Tipo de Prazo	Ações
165548	6	2	Procedimental	
165549	8	4	Caducidade	
165550	4	1	Processual	
165551	9	9	Procedimental	
165552	2	11	Procedimental	
165553	7	10	Procedimental	
165554	7	3	Caducidade	
165555	2	5	Prescrição	
165556	9	2	Prescrição	
165557	11	8	Prescrição	

Mostrando de 1 a 10 de 302 entradas

Anterior
1
2
3
4
Próximo

Figura 15: Mockup consultar prazos

 JuriTech






Resumo

Processos

Prazos

Documentos

Calendário



NOVO PRAZO

Nº de Processo

Tipo de Prazo

Artigo

Dias de Prazo

Nº Dias Atrasado

Nº Dias Tabelado

CANCELAR

GRAVAR

Figura 16: Mockup inserir prazos

5 Product Backlog

Neste Projeto o product backlog é feito com base nas users stories.

PB Codigo	PB Nome	Prioridade
PB001	Como utilizador devo ser capaz registar-me no sistema	Must
PB002	Como utilizador quero criar processos	Must
PB003	Como utilizador devo ser capaz de organizar os processos em função da fase processual, com a contagem do tempo de duração em cada uma dessas fases (termos básicos: fase de articulados, fase de instrução, fase decisória)	Must
PB004	Como utilizador devo ser capaz de organizar os processos em função da forma processual, e da área temática	Must
PB005	Como utilizador pretendo gerir processos	Should
PB006	Como utilizador pretendo gerir os prazos dos processos existentes	Should
PB007	Como utilizador pretendo recuperar a senha em caso de lapso	Should
PB008	Como utilizador pretendo receber alertas caso o prazo de um processo esteja prestes a caducar	Should
PB009	Como utilizador saber em que fase o processo se encontra, há quanto tempo o processo se encontra nessa fase e há quanto tempo o processo está/esteve no tribunal	Should
PB010	Como utilizador quero ver o historico de processos	Should
PB011	Como utilizador pretendo receber email de confirmação para algumas operações	Could
PB012	Como utilizador quero ver gráficos com desvios de prazo	Could
PB013	Como ulizador quero poder ver os meus processos por relevância, prazo ou tipo	Could
PB014	Como utilizador pretendo visualizar as tarefas num calendário para fazer a gestão dos casos	Could
PB015	Como utilizador pretendo ver uma aba de resumos	Could

6 Divisão das sprints

6.1 Análise de requisitos e modelação

ID	P.	Status	Task	E.H	Who	RT.H
AM-1	2	Done	Diagrama casos de uso	2	Vitor Machado	4
AM-2	3	Done	Diagramas de interação	3	Joel Jonassi	2
AM-3	1	Done	Diagrama de classes UML	5	Joel Figueiras	8
AM-4	4	Done	Diagrama de sequência	3	Nuno Araújo	5
AM-5	7	Done	Mockups	6	Tiago Azevedo	5
AM-6	5	Done	Diagrama de entidades	2	Vitor / Joel	3
AM-7	6	Done	Diagrama de componentes	4	Nuno Araújo	4
AM-8	8	Done	Levantamento e especificações dos RNF	3	Todos	3

Table 1: Especificação Sprint 1 - Start: 22/02/2022 — End: 14/03/2022

ID	P.	Status	Task	E.H	Who	RT.H
AM-9	1	Done	Backlog completo	3	Todos	3
AM-10	2	Done	Planeamento inicial das sprints	3	Todos	2

Table 2: Especificação Sprint 2 - Start: 14/03/2022 — End: 16/03/2022

6.2 vAlfa

ID	P.	Status	Task	E.H	Who	RT.H
ALFA-00	1	Done	Implementação da Base de dados	15	Jonassi/Vitor/Joel	22
ALFA-01	2	Done	Criação das classes em CSharp - Prazo	15	Joel/Nuno	20
ALFA-02	2	Done	Criação das classes em CSharp - Processo	15	Joel/Nuno	17
ALFA-03	3	Done	Implementação das DLLs	3	Tiago/Jonassi	2
ALFA-04	3	Done	Preenchimento da BD com dados de teste	2	Tiago/Vitor	4

Table 3: vAlfa Sprint 1 - Start: 17/03/2022 — End: 30/03/2022

ID	P.	Status	Task	E.H	Who	RT.H
ALFA-05	2	Done	Iniciação da implementação de Controllers	25	Joel/Nuno/Vitor	26
ALFA-06	3	Done	Iniciação da implementação de front-end	25	Jonassi/Tiago	22
ALFA-07	2	Done	Iniciação da implementação dos Testes Unitários	25	Joel/Nuno/Vitor	25
ALFA-08	1	Done	Aprofundamento da inserção de dados na BD (Dados realistas)	10	Jonassi/Tiago/Vitor	7
ALFA-09	1	Done	Implementação das características das classes (Models)	10	Joel/Nuno	15
ALFA-10	1	Done	Correções imprevistas de tarefas anteriores	5	Todos	8

Table 4: vAlfa Sprint 2 - Start: 31/03/2022 — End: 20/04/2022

ID	P.	Status	Task	E.H	Who	RT.H
ALFA-11	1	Done	Conclusão da implementação de Controllers	25	Joel/Nuno/Vitor	22
ALFA-12	3	Done	Conclusão da implementação de front-end	25	Jonassi/Tiago	22
ALFA-13	1	Done	Conclusão da implementação dos Testes Unitários	25	Todos	30
ALFA-14	2	Delayed	E-mail de alerta de confirmação de caducidade de um prazo	15	Jonassi/Tiago	1
ALFA-15	1	Done	Correções imprevistas de tarefas anteriores	5	Todos	10
ALFA-16	3	Done	Relatório	5	Joel/Nuno/Vitor	5

Table 5: vAlfa Sprint 3 - Start: 21/04/2022 — End: 04/05/2022

6.3 vBeta

ID	P.	Status	Task	E.H	Who	RT.H
BETA-00	1	To-Do	Correção de Bugs Detetados nas sprints vAlfa	15	Joel/Nuno/Vitor	
BETA-01	1	To-Do	Melhorias funcionais e não funcionais	10	Jonassi/Tiago	
BETA-02	2	To-Do	Atualização dos testes unitários	5	Jonassi/Tiago	
BETA-03	1	To-Do	Correções imprevistas da sprint vAlfa	10	Todos	
BETA-04	3	To-Do	Relatório	3	Joel/Nuno/Vitor	

Table 6: Sprint 5 - Start: 16/04/2022 — End: 01/05/2022

6.4 vRTW

ID	P.	Status	Task	E.H	Who	RT.H
RTW-00		To-Do	Correção de Todos os Bugs conhecidos e desconhecidos	15	Joel/Nuno/Vitor	
RTW-01		To-Do	Atualização Final dos testes unitários	5	Jonassi/Tiago	
RTW-02		To-Do	Preparação do material promocional	10	Jonassi/Tiago	
RTW-03		To-Do	Preparação de toda a documentação final	7	Todos	
RTW-04		To-Do	Relatório	3	Joel/Nuno/Vitor	

Table 7: Sprint 6 - Start: 02/05/2022 — End: 07/06/2022

7 Versão Alfa

7.1 Teste Unitário - Artigo

Na legislação, em determinado artigo está definido o tempo de espera para o código processual em questão, bem como a contagem de férias e dias não uteis. Desta forma, optamos por construir o nosso artigo com os seguintes atributos:

- IdCodigo: ID de um determinado código
- NrArtigo: Número do artigo a que correspondem os dados inseridos
- IdUtilizador: ID do utilizador a que corresponde determinado artigo
- Nome: Nome atribuído ao artigo pelo utilizador
- Dias: Número de dias correspondente ao artigo
- Meses: Número de meses correspondente ao artigo
- Anos: Número de anos correspondente ao artigo
- Ferias: True caso o artigo contabilize as férias, false caso contrário
- DiasNaoUteis: True caso o artigo contabilize os dias não uteis, falso caso contrário

```
public class Artigo
{
    #region Attributes

    int idCodigo, nrArtigo, idUtilizador, dias, meses, anos;
    string nome;
    bool ferias, diasNaoUteis;
    #endregion
}
```

Figura 17: Artigo Model

Em relação aos casos de uso, definimos que um utilizador seria capaz de inserir, alterar, eliminar artigos, bem como conseguir listar todos os artigos inseridos, ou, os artigos inseridos por determinado código processual.

Procedemos então aos testes unitários para cada um destes casos de uso.

Na inserção, criamos uma instância de um artigo com dados de teste e tentamos inseri-lo num determinado utilizador. O resultado obtido foi de acordo com o esperado.

Na alteração de um artigo, pegamos nos atributos do artigo de teste criado anteriormente e procedemos à alteração de atributos. Obtivemos novamente sucesso no teste criado.

Em relação à remoção de um artigo, tentamos eliminar o artigo de teste e o resultado foi, também, positivo.

Na listagem de artigos, primeiro tentamos obter os artigos por número de utilizador e, por fim, tentamos a consulta de artigos por utilizador e por código processual. Ambos os testes correram como esperado.

```
#region Inserir artigo
// Obter o status code 200 (Artigo inserido)
[Theory]
[InlineData("/api/Artigo/")]
0 references | Nuno Miguel Carvalho Araújo, 18 hours ago | 1 author, 1 change
public async Task AdicionarArtigo(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Artigo artigo = new() {
        IdCodigo = 1,
        NrArtigo = 11,
        IdUtilizador = 4,
        Nome = "Princípio da colaboração com os particulares",
        Dias = 1,
        Meses = 2,
        Anos = 1,
        Ferias = false,
        DiasNaoUteis = true };
    JsonContent content = JsonContent.Create(artigo);
    url += $"{artigo.IdCodigo}/{artigo.NrArtigo}/{artigo.IdUtilizador}";

    // Act
    var response = await client.PostAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 18: Inserir Artigo

```
#region Alterar artigo
// Obter um status code 204 (Artigo alterado)
[Theory]
[InlineData("/api/Artigo?idCod=1&nrArtigo=11&utilizador=4")]
0 references | Nuno Miguel Carvalho Araújo, 18 hours ago | 1 author, 1 change
public async Task AlterarArtigo(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Artigo artigo = new() {
        IdCodigo = 1,
        NrArtigo = 11,
        IdUtilizador = 4,
        Nome = "Ateração para o teste",
        Dias = 1,
        Meses = 2,
        Anos = 1,
        Ferias = false,
        DiasNaoUteis = true };
    JsonContent content = JsonContent.Create(artigo);

    // Act
    var response = await client.PutAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 19: Alterar Artigo

```
#region Eliminar artigo
// Obter um status code 204 (Artigo apagado)
[Theory]
[InlineData("/api/Artigo/1/11/4")]
0 references | Nuno Miguel Carvalho Araújo, 18 hours ago | 1 author, 3 changes
public async Task EliminarArtigo(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.DeleteAsync(url);

    // Assert
    Assert.Equal(HttpStatusCode.NoContent, response.StatusCode);
}
#endregion
```

Figura 20: Eliminar Artigo

```
#region Consultar artigos (by UserID)
// Obter um status code 200 (artigos consultados)
[Theory]
[InlineData("/api/Artigo/1")]
0 references | Nuno Miguel Carvalho Araújo, 2 days ago | 1 author, 1 change
public async Task ObterArtigo(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 21: Consultar Artigo (by UserID)

```
#region Consultar artigos por código processual (by UserID)
// Obter um status code 200 (artigos consultados)
[Theory]
[InlineData("/api/Artigo/1/2")]
0 references | Nuno Miguel Carvalho Araújo, 2 days ago | 1 author, 1 change
public async Task ObterArtigoByCode(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 22: Consultar Artigos por Código Processual (by UserID)

✓ ArtigoTestes (5)	2,7 s
✓ AdicionarArtigo(url: "/api/Artigo/")	184 ms
✓ AlterarArtigo(url: "/api/Artigo?idCod=1&nrA...")	1,1 s
✓ EliminarArtigo(url: "/api/Artigo/1/11/4")	1,2 s
✓ ObterArtigo(url: "/api/Artigo/1")	49 ms
✓ ObterArtigoByCode(url: "/api/Artigo/1/2")	31 ms

Figura 23: Artigo Testes

7.2 Teste Unitário - Código

Pretendíamos obter uma listagem genérica dos códigos processuais, desta forma, criamos um objeto um objeto para satisfazer essas necessidades. Assim sendo, o código tem os seguintes atributos:

- Id: ID para identificação do código noutros objetos
- Nome: Descriminação jurídica de um código (procedimento administrativo, processo cível, processo penal e regime ilícito de mera ordenação social)

```
public classCodigo
{
    #region Attributes
    int id;
    string nome;
    #endregion
}
```

Figura 24: Código Model

Como estamos a falar de uma classe com uma listagem genérica, apenas permitimos a consulta dos códigos processuais inseridos em base de dados, assim sendo, testamos a obtenção desta lista. O teste correu conforme as expectativas.

```

#region Consultar Codigos
// Obter um status code 200 (artigos consultados)
[Theory]
[InlineData("/api/Codigo")]
0 references | Nuno Miguel Carvalho Araújo, 2 days ago | 1 author, 1 change
public async Task ObterCodigos(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 25: Consultar Codigos

▲	✓	CodigoTestes (1)	1 s
	✓	ObterCodigos(url: "/api/Codigo")	1 s

Figura 26: Código Testes

7.3 Teste Unitário - Estado

Assim como o Código, o estado armazena uma listagem genérica de estados de uma fase processual. Sendo assim, criamos o estado com os seguintes atributos:

- Id: ID para a identificação do estado noutros objetos
- Nome: Descriminação de um estado (Fase articulados, fase de instrução e fase decisória)

```

public class Estado
{
    #region Attributes
    int id;
    string nome;
    #endregion
}

```

Figura 27: Estado Model

Falando, novamente, de uma listagem genérica, apenas testamos a obtenção da lista dos estados existentes. O teste foi de encontro às expectativas.

```

#region Consultar Estados
// Obter um status code 200 (artigos consultados)
[Theory]
[InlineData("/api/Estado")]
0 references | Nuno Miguel Carvalho Araújo, 2 days ago | 1 author, 1 change
public async Task ObterEstados(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 28: Consultar Estados

▲ ✓ EstadoTestes (1)	997 ms
✓ ObterEstados(url: "/api/Estado")	997 ms

Figura 29: Código Testes

7.4 Teste Unitário - Fase Processual

Uma fase processual regista o tempo que o respetivo processo esteve em determinada fase. Existem três fases: fase articulados, fase de instrução e fase decisória. Cada processo só pode passar uma vez por cada uma destas fases e só pode mudar para a fase seguinte se a fase anterior já estiver terminada. Não existe ordem definida na passagem de um processo pelas diferentes fases.

Para conseguir garantir o bom funcionamento da fase processual, criamos a estrutura com os seguintes atributos:

- IdUtilizador: ID para identificação do utilizador a que pertence a fase. (nota: o utilizador já deverá ter registado o processo para lhe atribuir uma fase)
- IdEstado: ID que identifica qual a fase em que o processo se encontra
- NrProcesso: Número do processo a que a fase processual está atribuída
- DataEntrada: Data de entrada do processo em determinada fase processual
- DataSaida: Data de saída do processo em determinada fase processual
- NrDias: Número de dias em que o processo esteve em determinada fase processual

```
public class FaseProcessual
{
    #region Attributes
    int idEstado, idUtilizador;
    string nrProcesso;
    int? nrDias;
    string dataEntrada;
    string? dataSaida;
    #endregion
}
```

Figura 30: Fase Processual Model

Um utilizador pode criar uma fase processual, assim como alterar a sua data de saída. É também capaz de obter várias listagens de fases processuais, sendo elas, as fases processuais por número de processo, por fase, assim como as ativas e inativas.

Começamos então por testar a criação de uma fase processual e, em seguida, testamos a alteração da data final de uma fase processual. Ambos os testes de gestão de fases processuais tiveram sucesso.

Por fim, testamos os quatro tipos de consulta de fases diferentes. Todos os testes obtiveram, também, resultado positivo.

```

#region Criar Fase Processual (by UserID)
// Obter o status code 200 (fase inserida)
[Theory]
[InlineData("/api/FaseProcessual")]
public async Task AdicionarFaseProcessual(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    FaseProcessual faseProcessual = new()
    {
        IdUtilizador = 1,
        IdEstado = 2,
        NrProcesso = "986/22.8T8ACB",
        DataEntrada = "02-03-2022",
        DataSaida = null,
        NrDias = null,
    };
    JsonContent content = JsonContent.Create(faseProcessual);
    url += "/1/986%2F22.8T8ACB/2/2022-03-02";

    // Act
    var response = await client.PostAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 31: Criar Fase Processual (by UserID)

```

#region Editar Fase Processual
// Testar a edição de uma Fase Processual
[Theory]
[InlineData("/api/FaseProcessual")]
public async Task EditarFaseProcessual(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    FaseProcessual faseProcessual = new()
    {
        IdUtilizador = 1,
        IdEstado = 2,
        NrProcesso = "986/22.8T8ACB",
        DataEntrada = "2022-03-02",
        DataSaida = "2022-04-02",
        NrDias = null,
    };
    JsonContent content = JsonContent.Create(faseProcessual);
    url += "/1/986%2F22.8T8ACB/2/2022-03-02";

    // Act
    var response = await client.PutAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 32: Editar Fase Processual (by UserID)

```

#region Consultar Fases processuais ativas (by UserID)
// Obter um status code 200 (fases processuais consultadas)
[Theory]
[InlineData("/api/FaseProcessual/1?AtivasNaoAtivas=true")]
public async Task ObterFaseProcessualByAtivas(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 33: Consultar Fases Processuais Ativas (by UserID)

```

#region Consultar Fases processuais inativas (by UserID)
// Obter um status code 200 (fases processuais consultadas)
[Theory]
[InlineData("/api/FaseProcessual/1?AtivasNaoAtivas=false")]
public async Task ObterFaseProcessualByInativas(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 34: Consultar Fases Processuais Inativas (by UserID)

```

#region Consultar Fase Processual por IdEstado (by UserID)
// Obter um status code 200 (fases processuais consultadas)
[Theory]
[InlineData("/api/FaseProcessual/1?Todas=true&id=1")]
public async Task ObterFaseProcessualByIdEstado(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 35: Consultar Fase Processual por IdEstado (by UserID)

```

#region Consultar Fase Processual por n° processo (by UserID)
// Obter um status code 200 (fases processuais consultadas)
[Theory]
[InlineData("/api/FaseProcessual/1/986%2F22.4T8ACB")]
public async Task ObterFaseProcessualByNrProcesso(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 36: Consultar Fase Processual por N° Processo (by UserID)

✓ FaseProcessualTestes (6)	7,3 s
✓ AdicionarFaseProcessual(url: "/api/FaseProcessual")	1,2 s
✓ EditarFaseProcessual(url: "/api/FaseProcessual")	1,3 s
✓ ObterFaseProcessualByAtivas(url: "/api/FaseProcessual/1?AtivasNaoAtivas=true")	1,3 s
✓ ObterFaseProcessualByIdEstado(url: "/api/FaseProcessual/1?Todas=true&id=1")	1,2 s
✓ ObterFaseProcessualByInativas(url: "/api/FaseProcessual/1?AtivasNaoAtivas=false")	1,1 s
✓ ObterFaseProcessualByNrProcesso(url: "/api/FaseProcessual/1/986%2F22.4T8ACB")	1,2 s

Figura 37: Fase Processual Testes

7.5 Teste Unitário - Prazo Codigo

Para fazer a ligação entre os prazos e os códigos processuais, criamos o objeto Prazocodigo. Este objeto é constituído pelos seguintes atributos:

- Id - ID para identificação de um prazo código noutros objetos
- IdTipoPrazo - ID para identificação do tipo de prazo
- IdCodigo - ID para identificar o respetivo código processual existentes. O teste foi de encontro às expectativas.

```
public class PrazoCodigo
{
    #region Attributes
    int id, idTipoPrazo, idCodigo;
    #endregion
}
```

Figura 38: Prazo Código Model

O objetivo é armazenar os dados seguindo a seguinte estrutura genérica:

- Prazo procedimental do código procedimento administrativo
- Prazo de caducidade co código do processo civil
- Prazo de caducidade do código do processo penal
- Prazo processual do código do processo civil
- Prazo processual do código do processo penal
- Prazo de prescrição do código do processo civil
- Prazo de prescrição do código do processo penal
- Prazo de prescrição do regime ilicito de mera ordenação social

Tendo em conta que apenas é necessária a listagem genérica, procedemos ao teste da sua obtenção. Este teste correu como o planeado.

```
#region Consultar Prazo Codigos
// Obter um status code 200 (artigos consultados)
[Theory]
[InlineData("/api/PrazoCodigo")]
0 references | Nuno Miguel Carvalho Araújo, 2 days ago | 1 author, 1 change
public async Task ObterCodigos(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 39: Consultar Prazo Códigos

▲	✔	PrazoCodigoTestes (1)	1 s
	✔	ObterCodigos(url: "/api/PrazoCodigo")	1 s

Figura 40: Prazo Código Testes

7.6 Teste Unitário - Prazo

Um processo tem sempre um prazo associado. Este prazo é calculado conforme o artigo que lhe for determinado. Cada artigo tem características singulares para o cálculo do prazo de um processo. Este cálculo é efetuado no prazo, e para isso, criamos uma estrutura com os seguintes atributos:

- IdPrazoCodigo: ID para identificação do tipo de prazo do processo
- IdUtilizador: ID para identificação do utilizador a que pertence o prazo
- NrProcesso: Número de processo a que o prazo está associado
- DataInicial: Data inicial definida para o prazo
- DataFinal: Data final prevista para término do prazo, tendo em conta as características do artigo atribuído
- NrArtigo: Identificação do artigo e respetivas características para cálculo do prazo (dias, meses, anos, dias úteis, férias, etc.)
- IdCódigo: ID para identificação do tipo de código processual a que pertence o prazo

```

public class Prazo
{
    #region Attributes
    int idPrazoCodigo, idUtilizador, idCodigo, nrArtigo;
    string nrProcesso;
    string dataInicial, dataFinal;

    #endregion
}

```

Figura 41: Prazo Model

Um utilizador pode criar ou apagar um prazo, bem como consultar os prazos adicionados.

Testamos a criação de um prazo pelo utilizador e o teste foi bem-sucedido.

Em seguida testamos a eliminação do prazo anteriormente criado. Este teste correu conforme o esperado.

Por fim, testamos a consulta de todos os prazos de determinado utilizador, bem como os prazos por número de processo. Ambos os testes foram bem-sucedidos.

```

#region Criar prazo (by UserID)
// Obter o status code 200 (Prazo inserido)
[Theory]
[InlineData("/api/Prazo")]
// 0 referências | Nuno Miguel Carvalho Araújo, há 2 horas | 1 autor, 1 alteração
public async Task AdicionarPrazo(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Prazo prazo = new()
    {
        idPrazoCodigo = 3,
        idUtilizador = 4,
        nrProcesso = "762/22.4T8ACB",
        dataInicial = "2022-05-01",
        dataFinal = "2023-10-02",
        nrArtigo = 7,
        idCodigo = 3,
    };
    JsonConvert content = JsonConvert.SerializeObject(prazo);
    url += "?idCodigo=3&idUtilizador=4&nrArtigo=7&nrProcesso=762/22.4T8ACB&tipoPrazo=2&dataInicial=2022-05-01";
    // Act
    var response = await client.PostAsync(url, content);
    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 42: Criar Prazo (by UserID)

```

#region Consultar prazos (by UserID)
// Obter um status code 200 (Prazos consultados)
[Theory]
[InlineData("/api/Prazo/4")]
// 0 referências | Nuno Miguel Carvalho Araújo, há 2 horas | 1 autor, 1 alteração
public async Task ObterPrazo(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 44: Consultar Prazo (by UserID)

```

#region Eliminar prazo
// Obter um status code 204 (Prazo apagado)
[Theory]
[InlineData("/api/Prazo/3/4/762%2F22.4T8ACB")]
// 0 referências | Nuno Miguel Carvalho Araújo, há 2 horas | 1 autor, 1 alteração
public async Task Eliminarprazo(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.DeleteAsync(url);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 43: Eliminar Prazo (by UserID)

```

#region Consultar prazos por n° processo (by UserID)
// Obter um status code 200 (Prazos consultados)
[Theory]
[InlineData("/api/Prazo/4/762%2F22.4T8ACB")]
// 0 referências | Nuno Miguel Carvalho Araújo, há 2 horas | 1 autor, 1 alteração
public async Task ObterPrazoByProcesso(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 45: Consultar Prazo por N°Processo (by UserID)

▲ ✓ PrazoTestes (4)	4,7 s
✓ AdicionarPrazo(url: "/api/Prazo")	1,3 s
✓ Eliminarprazo(url: "/api/Prazo/3/4/762%2F22.4T8ACB")	1,2 s
✓ ObterPrazo(url: "/api/Prazo/4")	1,1 s
✓ ObterPrazoByProcesso(url: "/api/Prazo/4/762%2F22.4T8ACB")	1,1 s

Figura 46: Prazo Testes

7.7 Teste Unitário - Processo

O processo é um dos temas chave neste projeto. É necessário existir um processo para que lhe sejam atribuídos os mais variados campos pretendidos. Assim sendo, criamos o processo com os seguintes atributos:

- NrProcesso: Número de determinado processo
- IdTipo: ID para identificação do tipo de processo
- IdTema: ID para identificação do tema do processo
- IdUtilizador: ID para identificação do utilizador a que o processo corresponde
- Valor: Valor monetário envolvido no processo
- Nome: Nome atribuído pelo utilizador ao processo
- Notas: Comentários que o utilizador pretenda acrescentar ao processo
- DataEntrada: Data de entrada do processo
- Estado: Estado do processo. True se o processo estiver ativo, false caso contrário.

```
public class Processo
{
    #region Attributes
    int idTipo, idTema, idUtilizador;
    string nrProcesso, nome, notas;
    decimal valor;
    DateTime dataEntrada;
    bool estado;
    #endregion
}
```

Figura 47: Processo Model

Para cumprir com os objetivos delineados, começamos por testar a criação de um processo. Para isso, criamos uma instância de teste de um processo e tentamos fazer a sua adição. Este teste correu conforme o esperado.

Em seguida, testamos a edição de um processo existente. Para isto, pegamos na instância de teste criada anteriormente e alteramos alguns campos. Mais uma vez, o teste ocorreu conforme as expectativas.

Por fim, testamos a consulta de processos por utilizador. Esta consulta funciona sem problemas.

```
#region Criar Processo (by UserID)
// Obter o status code 200 (Processo inserido)
[Theory]
[InlineData("/api/Processo/")]
0 references | Nuno Miguel Carvalho Araújo, 18 hours ago | 1 author, 1 change
public async Task AdicionarProcesso(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Processo processo = new() {
        NrProcesso = "986/22.8T8ACB",
        IdTipo = 1,
        IdTema = 1,
        IdUtilizador = 1,
        Valor = 5028.40M,
        Nome = "Processo teste",
        Notas = "Processo teste",
        DataEntrada = new DateTime(),
        Estado = true };

    processo.DataEntrada.ToString("2022-04-20T00:00:00");
    processo.DataEntrada.ToString("d");

    JsonConvert content = JsonConvert.Create(processo);
    url += $"{{processo.IdUtilizador}}/986%2F22.8T8ACB";

    // Act
    var response = await client.PostAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 48: Criar Processo (by UserID)

```
#region Alterar Processo (by UserID)
// Obter o status code 204 (Processo Alterado)
[Theory]
[InlineData("/api/Processo/")]
0 references | Nuno Miguel Carvalho Araújo, 18 hours ago | 1 author, 2 changes
public async Task AlterarProcesso(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Processo processo = new()
    {
        NrProcesso = "986/22.8T8ACB",
        IdTipo = 1,
        IdTema = 1,
        IdUtilizador = 1,
        Valor = 5028.40M,
        Nome = "Processo teste alterado",
        Notas = "Processo teste alterado",
        DataEntrada = new DateTime(),
        Estado = true
    };

    processo.DataEntrada.ToString("2022-04-20T00:00:00");
    processo.DataEntrada.ToString("d");

    JsonConvert content = JsonConvert.Create(processo);
    url += $"{{processo.IdUtilizador}}/986%2F22.8T8ACB";

    // Act
    var response = await client.PutAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 49: Alterar Processo (by UserID)

```
#region Consultar Processos (by UserID)
// Obter um status code 200 (processos consultados)
[Theory]
[InlineData("/api/Processo/1")]
0 references | Nuno Miguel Carvalho Araújo, 16 minutes ago | 1 author, 3 changes
public async Task ObterProcesso(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Processo processo = new();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 50: Consultar Processo (by UserID)

▲ ✓ ProcessoTestes (3)	3,2 s
✓ AdicionarProcesso(url: "/api/Processo/")	1 s
✓ AlterarProcesso(url: "/api/Processo/")	1,2 s
✓ ObterProcesso(url: "/api/Processo/1")	1,1 s

Figura 51: Processo Testes

7.8 Teste Unitário - Tema

Cada processo tem o seu tipo e, dentro desse tipo, está atribuído um tema. Assim sendo, criamos o Tema com os seguintes atributos:

- Id: ID para identificação do tema
- IdUtilizador: ID para identificação do utilizador a que pertence o tema
- Nome: Nome atribuído ao tema, pelo utilizador

```
public class Tema
{
    #region Attributes
    int id, idUtilizador;
    string nome;
    #endregion
}
```

Figura 52: Tema Model

Em relação aos casos de uso, determinamos que cada utilizador poderia inserir um tema, alterar um tema ou consultar todos os temas por ele inseridos.

Criamos, então, uma instância teste de um tema para proceder à sua inserção. O teste foi executado com sucesso.

Em seguida, modificamos a instância criada anteriormente para testar a sua alteração. Este teste também teve sucesso.

Por fim, testamos a consulta de temas inseridos por utilizador, obtendo novamente resultado positivo.

```

#region Inserir tema
// Obter o status code 200 (Artigo inserido)
[Theory]
[InlineData("/api/Tema?IdUtilizador=2")]
0 references | Nuno Miguel Carvalho Araújo, 19 hours ago | 1 author, 1 change
public async Task AdicionarTema(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Tema tema = new()
    {
        Id = 0,
        IdUtilizador = 2,
        Nome = "Tema para teste2",
    };
    JsonContent content = JsonContent.Create(tema);

    // Act
    var response = await client.PostAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 53: Inserir Tema

```

#region Alterar Tema
// Obter um status code 204 (Artigo alterado)
[Theory]
[InlineData("/api/Tema/121/1")]
0 references | Nuno Miguel Carvalho Araújo, 19 hours ago | 1 author, 1 change
public async Task AlterarTema(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Tema tema = new()
    {
        Id = 0,
        IdUtilizador = 1,
        Nome = "Tema para teste alterado",
    };
    JsonContent content = JsonContent.Create(tema);

    // Act
    var response = await client.PutAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 54: Alterar Tema

```

#region Consultar temas existentes (by UserID)
// Obter um status code 200 (temas consultados)
[Theory]
[InlineData("/api/Tema/1")]
0 references | Nuno Miguel Carvalho Araújo, 19 hours ago | 1 author, 1 change
public async Task ObterTema(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 55: Consultar Temas (by UserID)

▲ ✓ TemaTestes (3)	3,4 s
✓ AdicionarTema(url: "/api/Tema?IdUtilizador=...")	1,1 s
✓ AlterarTema(url: "/api/Tema/121/1")	1,1 s
✓ ObterTema(url: "/api/Tema/1")	1,2 s

Figura 56: Tema Testes

7.9 Teste Unitário - Tipo Prazo

Assim como o Código, o tipo de prazo armazena uma listagem genérica de tipos de prazo de um processo. Sendo assim, criamos o tipo de prazo com os seguintes atributos:

- Id: ID para a identificação do tipo de prazo noutros objetos
- Nome: Discriminação de um tipo de prazo (Prazo procedimental, prazo de caducidade, prazo processual e prazo de prescrição)

```
public class TipoPrazo
{
    #region Attributes
    int id;
    string nome;
    #endregion
}
```

Figura 57: Tipo Prazo Model

Sendo uma listagem genérica, apenas testamos a obtenção da lista dos tipos de prazo existentes. O teste foi de encontro às expectativas.

```
#region Consultar Tipos de prazo
// Obter um status code 200 (artigos consultados)
[Theory]
[InlineData("/api/TipoPrazo")]
0 references | Nuno Miguel Carvalho Araújo, 19 hours ago | 1 author, 1 change
public async Task ObterTipoPrazo(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 58: Consultar Tipos de Prazo

▲	✔ TipoPrazoTestes (1)	1 s
	✔ ObterTipoPrazo(url: "/api/TipoPrazo")	1 s

Figura 59: Tipo Prazo Testes

7.10 Teste Unitário - Tipo Processo

Como referido anteriormente, cada processo tem o seu tipo e, a cada tipo, está um tema associado. O tipo de processo deve ser inserido a gosto do utilizador, mas terá, sempre um tema associado. Desta forma, criamos o tipo de processo com os seguintes atributos:

- Id: ID para identificação do tipo de processo noutros objetos
- IdUtilizador: ID para identificação do utilizador a que pertence o tipo de prazo
- Nome: Nome atribuído ao tipo de processo, pelo utilizador
- IdTema: ID do tema associado ao tipo de processo

```
public class TipoProcesso
{
    #region Attributes
    int id, idTema, idUtilizador;
    string nome;
    #endregion
}
```

Figura 60: Tipo Processo Model

Para corresponder com os objetivos delineados, é necessário um utilizador poder criar um tipo de processo assim como alterá-lo.

Para testar a criação de um tipo de processo, criamos uma instância de um tipo de processo e procedemos à implementação. O resultado do teste foi satisfatório.

Por fim, testamos a alteração de campos da instância criada anteriormente. Obtemos, também, um resultado positivo neste teste.

```

#region Criar tipo de processo (by UserID)
// Obter o status code 200 (TipoProcesso inserido)
[Theory]
[InlineData("/api/TipoProcesso?IdUtilizador=4")]
0 references | Nuno Miguel Carvalho Araújo, 19 hours ago | 1 author, 1 change
public async Task AdicionarTipoProcesso(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    TipoProcesso tipoProcesso = new()
    {
        Id = 0,
        IdUtilizador = 4,
        Nome = "Tipo processo para teste",
        IdTema = 120,
    };
    JsonContent content = JsonContent.Create(tipoProcesso);

    // Act
    var response = await client.PostAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 61: Criar Tipo de Processo (by UserID)

```

#region Editar tipo de processo (by UserID)
// Obter um status code 204 (Artigo alterado)
[Theory]
[InlineData("/api/TipoProcesso/121/4")]
0 references | Nuno Miguel Carvalho Araújo, 19 hours ago | 1 author, 1 change
public async Task AlterarTipoProcesso(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    TipoProcesso tipoProcesso = new()
    {
        Id = 0,
        IdUtilizador = 1,
        Nome = "Tema para teste alterado",
        IdTema = 120,
    };
    JsonContent content = JsonContent.Create(tipoProcesso);

    // Act
    var response = await client.PutAsync(url, content);

    // Assert
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion

```

Figura 62: Editar Tipo de Processo (by UserID)

▲ ✓ TipoProcessoTestes (2)	1,4 s
✓ AdicionarTipoProcesso(url: "/api/TipoProcesso?IdUtilizador=4")	176 ms
✓ AlterarTipoProcesso(url: "/api/TipoProcesso/121/4")	1,3 s

Figura 63: Tipo Processo Testes

7.11 Teste Unitário - Utilizador

Na nossa perspetiva, um utilizador é um cliente do nosso serviço. Um cliente é constituído pelos seguintes atributos:

- Id: ID para identificação do utilizador noutros objetos
- Nome: Nome do utilizador
- Email: Email do utilizador, necessário para acesso à plataforma e para a receção de notificações à cerca dos prazos
- Password: Senha de acesso ao portal

```

public class Utilizador
{
    #region Attributes
    int id;
    string nome;
    string email;
    string password;
    #endregion
}

```

Figura 64: Utilizador Model

Neste projeto, o utilizador é o principal interveniente. Ele tem permissões para criar conta com um email que ainda não esteja associado, bem como alterar os dados da sua conta.

Criamos então um utilizador de exemplo para testar o seu sign up. O resultado do teste foi positivo.

Em seguida, alteramos dados do utilizador criado anteriormente e testamos. O resultado obtido foi, novamente, satisfatório.

```
#region Sign Up
// Testar acesso a uma api existente. O objetivo é obter status code 200 (acesso com sucesso)
[Theory]
[InlineData("/api/Utilizador")]
<references|Nuno Miguel Carvalho Araújo, 1 hour ago | 1 author, 2 changes
public async Task CriarConta(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Utilizador utilizador = new() { Id = 0, Nome = "Dados testes", Email = "dados testes@juritech.pt", Password = "testes123" };
    JsonContent content = JsonContent.Create(utilizador);

    // Act
    var response = await client.PostAsync(url, content);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 65: Sign Up

```
#region Login
[Theory]
[InlineData("/api/Utilizador/joaquim%40juritech.pt/joaquim123")]
<references|Nuno Miguel Carvalho Araújo, 1 hour ago | 1 author, 2 changes
public async Task Login(string url)
{
    // Arrange
    var client = _factory.CreateClient();

    // Act
    var response = await client.GetAsync(url);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 66: Login

```
#region Editar dados de utilizador
// Testar a edição de um utilizador, por ID. O objetivo é obter status code 200 (ocorrência com sucesso)
[Theory]
[InlineData("/api/Utilizador/5")]
<references|Nuno Miguel Carvalho Araújo, 1 hour ago | 1 author, 2 changes
public async Task EditarDados(string url)
{
    // Arrange
    var client = _factory.CreateClient();
    Utilizador utilizador = new()
    {
        Id = 0, Nome = "Dados testes - Joaquina Fernandes", Email = "joaquina@juritech.pt", Password = "joaquina123"
    };
    JsonContent content = JsonContent.Create(utilizador);

    // Act
    var response = await client.PutAsync(url, content);

    // Assert
    response.EnsureSuccessStatusCode();
    Assert.Equal(HttpStatusCode.OK, response.StatusCode);
}
#endregion
```

Figura 67: Editar Dados de Utilizador

UtilizadorTestes (3)	2,5 s
✓ CriarConta(url: "/api/Utilizador")	177 ms
✓ EditarDados(url: "/api/Utilizador/5")	1,2 s
✓ Login(url: "/api/Utilizador/joaquim%40juritech.pt/joaquim123")	1,2 s

Figura 68: Utilizador Testes

7.12 Bugs Conhecidos

Neste momento, as notificações para o utilizador ainda não estão implementadas. Temos consciência que as notificações via email são uma das funcionalidades mais importantes no projeto, no entanto, ainda não tivemos a possibilidade de as implementar. Esperamos resolver este problema o mais rapidamente possível.

Após a implementação das notificações, pretendemos possibilitar também ao utilizador a recuperação da sua password através do email. Supomos que ambas as funcionalidades tenham parecenças, por isso, pretendemos implementar as notificações e a recuperação de senhas simultaneamente.

É possível que existam também alguns bugs e falhas de verificações de campos. Durante a execução dos testes unitários, fomos encontrando alguns e procedemos à sua resolução. No entanto, estamos preparados para encontrar mais conforme vamos experimentando o projeto.