

**Instituto Politécnico do Cávado e do Ave**  
**Relatório de Projeto de Desenvolvimento de Software**  
**BookingItHere**

Licenciatura em Engenharia em Sistemas Informáticos

João Ponte, nº 17694

João Carvalho, nº 12747

Pedro Martins, nº 23527

Luís Anjo, nº 23528

Diogo Silva, nº 23893

Barcelos, Portugal  
2022/2023

## Índice

Introdução.....	5
Contexto.....	6
Motivação para a resolução dos problemas .....	7
Descrição dos Problemas .....	8
Regras de Negócio .....	9
Diagrama de Casos de Uso.....	10
Histórias dos Utilizadores .....	13
Modelação Processos de Negócio BPMN: .....	20
Diagrama Classes .....	23
Diagrama ER.....	24
Diagramas Sequência.....	25
Gestão do Projeto .....	37
Criação da API .....	38
<b>Escolha Base de Dados:</b> .....	38
<b>Testar Rotas:</b> .....	39
<b>REDIS:</b> .....	40
<b>JWT (TOKEN)</b> .....	40
<b>Testes Unitários / Integração</b> .....	41
Conclusão.....	57

## Índice Figuras

Figura 1- Diagrama Casos de Uso.....	10
Figura 2- BPMN Alugar Alojamento .....	20
Figura 3- BPMN Apaga Alojamento .....	21
Figura 4- BPMN Serviços Adicionais.....	21
Figura 5- BPMN Promoção Alojamento .....	22
Figura 6- BPMN Apaga Reserva .....	22
Figura 7- Diagrama Classes .....	23
Figura 8- Diagrama ER.....	24
Figura 9- Diagrama Sequência Aluga Alojamento.....	25
Figura 10- Diagrama Sequência Apaga Alojamento.....	25
Figura 11- Diagrama Sequência Serviços Adicionais.....	27
Figura 12- Diagrama Sequência Promove Anúncio.....	27
Figura 13- Diagrama Sequência Cancela Reserva .....	28
Figura 14- Página Principal Mock.....	29
Figura 15 - Ecrã de login.....	30
Figura 16 - Ecrã de registo .....	31
Figura 17 - Criar alojamento .....	32
Figura 18 - Detalhe Anúncio.....	33
Figura 19 - Criar Serviços Extra .....	33
Figura 20 - Detalhes da reserva efetuada .....	34
Figura 21 - Lista de alojamentos .....	35
Figura 22 - Lista de Reservas .....	36
Figura 23- Azure DevOps.....	37
<b>Figura 24- Teste Postman.....</b>	<b>39</b>
Figura 25- Teste Swagger .....	39
Figura 26- Rotas Swagger API .....	39
Figura 27- Sign Generate Token.....	40
Figura 28- Tabela User com Token.....	40
Figura 29- Teste Cria Utilizador.....	41
Figura 30- Teste Login .....	42
Figura 31- Teste Cria Casa .....	42
Figura 32- Teste Cria Anúncio .....	43
Figura 33 - Confirmação Teste Anúncio .....	43
Figura 34 - Seeds do Anúncio.....	44
Figura 35 - Modelo Feedback.....	45
Figura 36 - Modelo Reserva .....	45
Figura 37 - Mensagens Sucesso Casas .....	46
Figura 38 - Mensagens Erro Casas .....	46
Figura 39 - Rota POST Casas.....	47
Figura 40 - Rota PUT Casas.....	48
Figura 41 - Função Criar Utilizador .....	49
Figura 42 - Função RN Cancelar Reserva.....	50

Figura 43 - Função RN Apagar Alojamento .....	52
Figura 44 - Função RN Anúncios .....	53
Figura 45 - Função RN Serviços .....	54
Figura 46 - Função RN Serviços Casa.....	54
Figura 47- Home do Website .....	55
Figura 48- Reserva House Website .....	56

## Introdução

No âmbito da unidade curricular Projeto de Desenvolvimento de Software, desenvolvemos o presente relatório para descrever as funcionalidades que serão implementadas na nossa plataforma.

Representamos as funcionalidades através de vários diagramas (Diagrama BPMN, Diagrama de Casos de uso, Diagrama de Classes, Diagrama Entidade Relação e Diagrama de Sequência).

Descreveremos o processo de criação da API, incluindo a configuração do ambiente de desenvolvimento, a escolha das tecnologias utilizadas e a implementação das funcionalidades previstas.

## Contexto

A plataforma web será capaz de gerir informações sobre alojamentos, incluindo dados como o tipo de alojamento, localização, preço, disponibilidade, e outros detalhes importantes tais como os serviços adicionais. Além disso, terá uma utilização facilitada para os proprietários e para os clientes. Também permitirá aos anunciantes a promoção dos alojamentos.

## Motivação para a resolução dos problemas

As motivações para o desenvolvimento de um website booking são:

Atrair Clientes: com o desenvolvimento do nosso website pretendemos fornecer aos nossos clientes, um excelente serviço de gestão reservas de alojamentos e divulgação dos mesmos. Com isto pretendemos tornar-nos uma referência no âmbito de portais para reservas e divulgação de alojamentos.

Recompensas financeiras: a nossa plataforma terá uma taxa fixa para qualquer reserva efetuada, podendo lucrar com qualquer reserva realizada na plataforma. Também haverá um sistema de anúncios onde o anunciante poderá promover o seu alojamento, dependendo do valor definido.

## Descrição dos Problemas

O projeto visa resolver o problema da falta de disponibilização de websites para partilha de anúncios de alojamento, de modo a tornar mais fácil a procura e oferta de alojamentos para reservar. O projeto propõe o desenvolvimento de uma plataforma web que permita aos proprietários de alojamentos disponibilizarem os seus espaços a um determinado valor, por sua vez aos clientes terem acesso a uma lista de alojamentos disponíveis para o local onde pretendem deslocar-se e fazer a reserva do alojamento escolhido.



## Regras de Negócio

- O utilizador poderá cancelar a sua reserva até alguns dias antes da entrada no alojamento sem qualquer prejuízo para o mesmo, o dinheiro será reembolsado na totalidade. Passando esta data definida pelo anunciante, o utilizador ainda terá a hipótese de cancelar a sua reserva, mas apenas será reembolsado em 50% do valor pago.
- O anunciante poderá cancelar as reservas do seu alojamento a qualquer momento desde que não exista nenhuma reserva fora do âmbito de cancelamento gratuito. Caso o cancelamento afete reservas já agendadas e confirmadas o anunciante terá que em prejuízo próprio reembolsar o valor total pago pelos clientes.
- Se houver provas de danos ou uma utilização indevida por parte do utilizador, o anunciante definirá um valor de caução a ser pago pelo cliente de acordo com as regras definidas pelo mesmo.
- O anunciante poderá promover qualquer um dos seus alojamentos desde que o mesmo seja aprovado por um administrador, e o valor resultante da promoção do anúncio deverá ser pago no ato da promoção.

## Diagrama de Casos de Uso

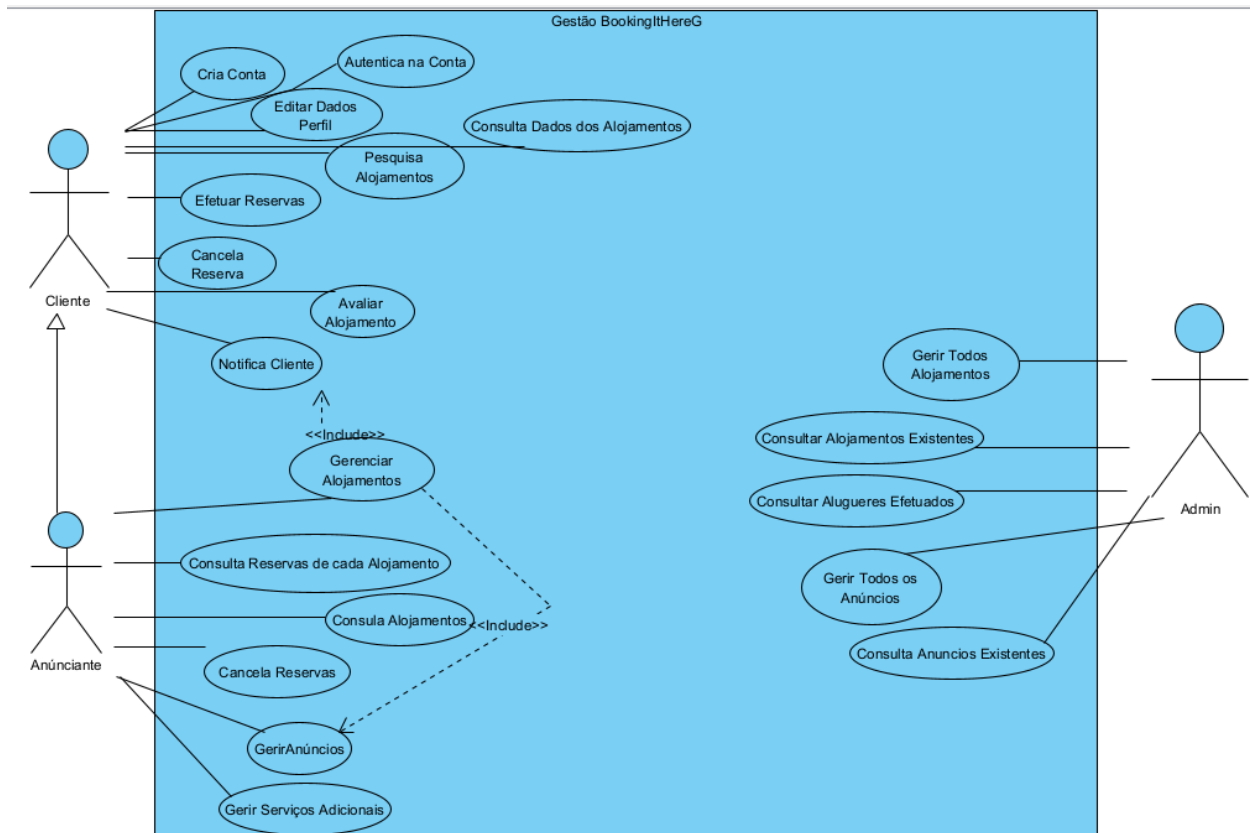


Figura 1- Diagrama Casos de Uso

## Requisitos Funcionais / Casos de Uso

### UC 1 - Casos de Uso Cliente (LOCATÁRIO / HOSPEDE)

**UC 1.1- Criar Conta** – A plataforma permite ao utilizador a criação da sua própria conta.

**UC 1.2- Editar Dados Perfil** - Após a criação da conta, o utilizador poderá alterar os dados do seu perfil.

**UC 1.3- Pesquisar Alojamentos** – O cliente poderá pesquisar alojamentos conforme os seus desejos de pesquisa.

**UC 1.4- Efetuar Reservas** - Após a escolha do alojamento, o cliente partirá para a reserva do alojamento colocando os dados necessários para a mesma.

**UC 1.5- Cancelar Reservas** – Caso haja um imprevisto, o cliente poderá cancelar a sua reserva.

**UC 1.6- Consultar Reservas** – O cliente terá acesso aos alojamentos que tem reservados e os alojamentos que usufruiu anteriormente.

**UC 1.7- Consulta Dados Alojamento** – Antes de efetuar a reserva, o cliente deverá consultar os dados do alojamento para verificar se o alojamento será o desejado.

**UC 1.8- Atualiza Estado Reserva** – A reserva terá os seguintes estados: concluída, reservada, em uso e cancelada. Os estados serão alterados conforme as ações relativas à reserva.

**UC 1.9- Avalia Alojamento** – A plataforma disponibiliza uma ferramenta de avaliação onde os utilizadores podem avaliar o alojamento que reservaram.

**UC 1.10- Notifica Cliente** – A plataforma envia uma notificação ao utilizador sempre que haja alguma alteração no estado seja da reserva ou do alojamento do proprietário.

### UC 2 - Casos de Uso Anunciante

**UC 2.1- Adiciona Alojamento** – O anunciante poderá adicionar um alojamento na plataforma e assim disponibilizar o mesmo para os clientes reservarem.

**UC 2.2- Apaga Alojamento** – O anunciante poderá apagar o alojamento caso deseje.

**UC 2.3- Consulta Reservas de Cada Alojamento** – o anunciante conseguirá visualizar as todas as reservas referentes a cada alojamento.

**UC 2.4- Consulta Alojamentos** – O anunciante poderá consultar todos os alojamentos que inseriu na plataforma.

**UC 2.5- Cancela Reservas** – Caso o anunciante pretenda poderá cancelar as reservas nos seus alojamentos.

**UC 2.6 - Gerir Anúncios** - O anunciante poderá adicionar, remover e editar os anúncios de cada alojamento.

**UC 2.7 - Gerir Serviços Adicionais** – Cada anunciante terá os seus serviços adicionais. O mesmo poderá adicionar ou remover serviços caso deseje.

## UC 3 – Casos de Uso Administradores

**UC 3.1 – Gerir Todos Alojamentos** – O administrador poderá gerir os alojamentos que todos os anunciantes disponibilizem na plataforma.

**UC 3.2 – Consulta Alojamentos Existentes** – O administrador poderá consultar todos os alojamentos que os anunciantes insiram na plataforma.

**UC 3.3 – Consulta Alugueres Efetuados** - O administrador poderá consultar todos os alugueres efetuados pelos clientes.

**UC 3.4 – Valida Todos Anúncios** - Antes da inserção dos anúncios na plataforma, os mesmos terão de passar por uma validação.

**UC 3.5 – Consulta Anúncios Existentes** – O administrador deverá consultar os anúncios disponíveis na plataforma de modo a evitar alterações indevidas ou mal avaliadas.

### Requisitos não funcionais:

- Plataforma web - A nossa plataforma deverá ser compatível com as versões atuais dos browsers
- Segurança – A nossa plataforma está feita de forma aos dados do utilizador aquando do login estarem salvaguardados, usamos uma ferramenta para encriptar a password e colocar a mesma na base de dados.

## Histórias dos Utilizadores

### Histórias de Cliente (UC1):

Como cliente quero conseguir criar uma conta na plataforma.

Como cliente quero procurar os melhores alojamentos conforme os meus interesses.

Como Cliente pretendo alugar um alojamento depois de o escolher.

Como Cliente pretendo aceder a uma página que liste todas as minhas reservas.

Como cliente, é do meu interesse, avaliar o alojamento que aluguei.

### Utilizador:

Como um utilizador registado, quero poder editar os meus dados de perfil, para poder manter as minhas informações atualizadas.

Como um utilizador registado, quero poder recuperar a minha password, no caso de a ter esquecido, para poder aceder à minha conta novamente.

Como um utilizador da plataforma, quero poder cancelar uma reserva já criada, para poder alterar as minhas datas de viagem ou escolher outro alojamento.

**Histórias de Anunciante (UC2):**

Como o anunciante de um alojamento, quero poder registar o meu alojamento na plataforma, para que os utilizadores possam pesquisar e reservar o meu alojamento.

Como o anunciante de alojamento, quero poder ver todos os meus alojamentos registados na plataforma, para poder geri-los facilmente.

Como o anunciante de alojamento, quero poder editar as informações de um alojamento já registado, para poder mantê-lo atualizado.

Como o anunciante de alojamento, quero poder apagar um alojamento registado, para que os utilizadores não possam mais pesquisar e reservar o alojamento.

Como o anunciante de alojamento, quero poder confirmar as reservas criadas pelos utilizadores, para poder garantir a disponibilidade do alojamento.

Como o anunciante de alojamento, quero poder promover o meu alojamento na plataforma, para atrair mais utilizadores e reservas.

Como o anunciante de alojamento, quero poder adicionar serviços extra ao meu alojamento, para poder oferecer uma experiência mais completa aos clientes.

**Histórias de Administrador (UC3):**

Como administrador pretendo proceder à verificação dos alojamentos que serão colocados na plataforma pelos anunciantes.

Como administrador pretendo proceder à verificação dos anúncios dos alojamentos.

## Tasks / Work Items

Caso de Uso	Criar Conta
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de registo de utilizador;</li><li>• Definir os campos necessários para registo do utilizador (ex. nome, sobrenome, e-mail, password);</li><li>• Validar dados do registo (ex. verificar se o e-mail já existe);</li></ul>	

Caso de Uso	Editar Dados Perfil
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de edição dos dados do utilizador;</li><li>• Adicionar uma validação dos dados atualizados.</li></ul>	

Caso de Uso	Pesquisar Alojamentos
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de pesquisa de alojamentos;</li><li>• Definir os filtros de pesquisa (ex. localização, preço, tipo de alojamento);</li><li>• Exibir resultados da pesquisa.</li></ul>	

Caso de Uso	Efetuar Reserva
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de reserva de alojamento;</li><li>• Permitir que o utilizador escolha as datas de check-in e check-out;</li><li>• Validar disponibilidade do alojamento para as datas escolhidas;</li><li>• Calcular valor da reserva;</li><li>• Permitir que o anunciante valide a reserva.</li></ul>	

Caso de Uso	Cancelar Reserva
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de cancelamento de reserva;</li><li>• Permitir que o utilizador selecione a reserva a ser cancelada;</li><li>• Confirmar o cancelamento da reserva.</li></ul>	

Caso de Uso	Consultar Reserva
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de consulta de reservas;</li><li>• Exibir todas as reservas do utilizador;</li></ul>	



Caso de Uso	Consultar Dados Alojamento
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de visualização de dados do alojamento;</li><li>• Exibir informações sobre o alojamento (ex. descrição, fotos, preço, disponibilidade).</li></ul>	

Caso de Uso	Atualiza Estado Reserva
Implementação:	
<ul style="list-style-type: none"><li>• Permitir que o anunciante do alojamento atualize o status de uma reserva (ex. confirmada, cancelada, pendente);</li><li>• Enviar notificação para o cliente sobre a atualização.</li></ul>	

Caso de Uso	Avaliar Alojamento
Implementação:	
<ul style="list-style-type: none"><li>• Permitir que o cliente avalie um alojamento após a estadia;</li><li>• Salvar avaliação e exibir na página do alojamento.</li></ul>	

Caso de Uso	Notificar Cliente
Implementação:	
<ul style="list-style-type: none"><li>• Enviar notificações para o cliente sobre as reservas (ex. confirmação de reserva, atualização de status);</li></ul>	

Caso de Uso	Adicionar Alojamento
Implementação:	
<ul style="list-style-type: none"><li>• Permitir que os proprietários de alojamentos adicionem novos alojamentos ao sistema;</li><li>• Definir campos necessários para registo do alojamento (ex. localização, preço, tipo de habitação);</li><li>• Validar dados do registo;</li><li>• Possibilidade de associar serviços ao alojamento.</li></ul>	

Caso de Uso	Apagar Alojamento
Implementação:	
<ul style="list-style-type: none"><li>• Permitir que os proprietários de alojamentos apaguem os seus alojamentos do sistema;</li><li>• Confirmar a exclusão do alojamento.</li></ul>	

Caso de Uso	Consultar Reservas de Cada Alojamento
Implementação:	
<ul style="list-style-type: none"><li>• Permitir que os proprietários de alojamentos visualizem todas as reservas para um determinado alojamento;</li></ul>	

- Exibir informações sobre as reservas (ex. data de reserva, status, valor).

Caso de Uso	Consultar Alojamentos
Implementação:	
<ul style="list-style-type: none"> <li>• Criar página de consulta de alojamentos;</li> <li>• Exibir todos os alojamentos pertencentes ao anunciante;</li> </ul>	

Caso de Uso	Cancelar Reservas
Implementação:	
<ul style="list-style-type: none"> <li>• Criar página de cancelamento de reserva;</li> <li>• Permitir que o utilizador selecione a reserva a ser cancelada;</li> <li>• Confirmar o cancelamento da reserva.</li> </ul>	

Caso de Uso	Gerenciar Anúncios
Implementação:	
<ul style="list-style-type: none"> <li>• Permitir que os anunciantes adicionem, editem e excluam anúncios;</li> <li>• Exibir informações sobre os anúncios (ex. visualizações, cliques, reservas).</li> </ul>	

Caso de Uso	Gerenciar Serviços Adicionais
Implementação:	
<ul style="list-style-type: none"> <li>• Permitir que proprietários de alojamentos adicionem serviços adicionais aos seus alojamentos (ex. limpeza, pequeno-almoço);</li> <li>• Definir preços para cada serviço adicional.</li> </ul>	

Caso de Uso	Gerir Todos os Alojamentos
Implementação:	
<ul style="list-style-type: none"> <li>• Permitir que os administradores adicionem, editem e excluam alojamentos;</li> <li>• Exibir informações sobre os alojamentos (ex. proprietário, localização, número de reservas).</li> </ul>	

Caso de Uso	Consultar Alojamentos Existentes
Implementação:	
<ul style="list-style-type: none"> <li>• Criar página de consulta de alojamentos;</li> <li>• Exibir todos os alojamentos registados no sistema;</li> <li>• Permitir que o utilizador filtre os alojamentos por tipo, preço, localização.</li> </ul>	

Caso de Uso	Consultar Reservas Efetuados
Implementação:	
<ul style="list-style-type: none"><li>• Permitir que administradores do sistema visualizem todas as reservas feitas no sistema;</li><li>• Exibir informações sobre as reservas (ex. data de reserva, alojamento, cliente).</li></ul>	

Caso de Uso	Consultar Anúncios Existentes
Implementação:	
<ul style="list-style-type: none"><li>• Criar página de consulta de anúncios;</li><li>• Exibir todos os anúncios registados no sistema;</li><li>• Permitir que o utilizador filtre os anúncios por tipo, preço, localização.</li></ul>	

Caso de Uso	Gerir Todos Anúncios
Implementação:	
<ul style="list-style-type: none"><li>• Permitir que os administradores adicionem, editem e excluam anúncios;</li><li>• Exibir informações sobre os anúncios (ex. proprietário, visualizações, cliques, reservas).</li></ul>	

## Modelação Processos de Negócio BPMN:

## PN1 - Alugar Alojamento

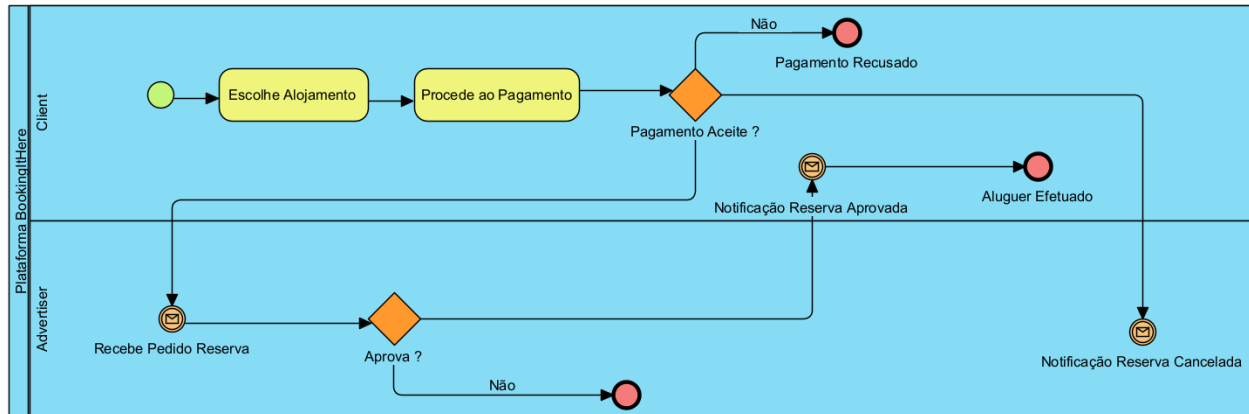


Figura 2- BPMN Alugar Alojamento

O utilizador irá pesquisar por um alojamento na nossa plataforma, irá escolher o pretendido, proceder ao pagamento e se este for aceite, irá proceder para a aprovação do anunciante, caso seja aprovado, o cliente receberá uma notificação e terá a sua reserva efetuada.

## PN2 – Apagar Alojamento

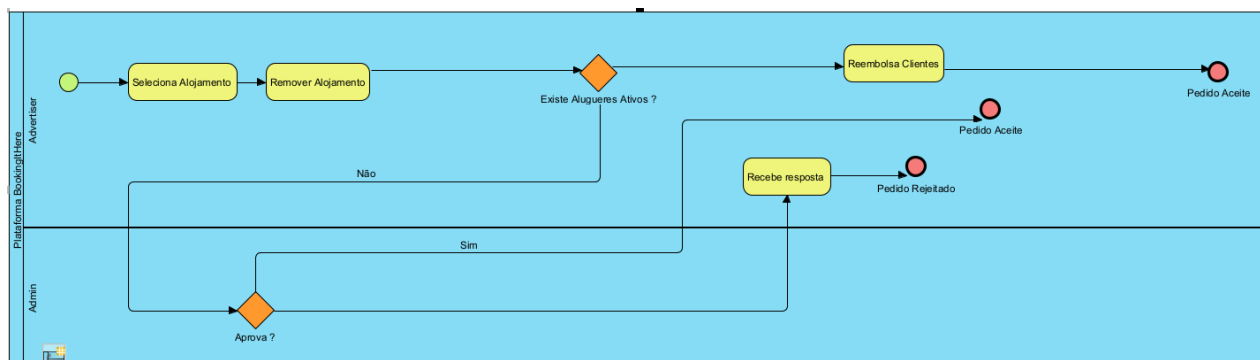


Figura 3- BPMN Apaga Alojamento

O anunciante começa o processo de negócio por escolher o alojamento que pretende apagar, de seguida é verificado pelo sistema se esse alojamento contém reservas ou não, caso não tenha, o mesmo será obrigado a descrever o motivo podendo ou não ser aceite posteriormente pelo administrador. Caso tenha reservas ativas o anunciante também deverá descrever o motivo e terá de reembolsar todos os clientes que reservaram. Depois de todas estas etapas o alojamento será apagado

## PN3 - Serviços Adicionais

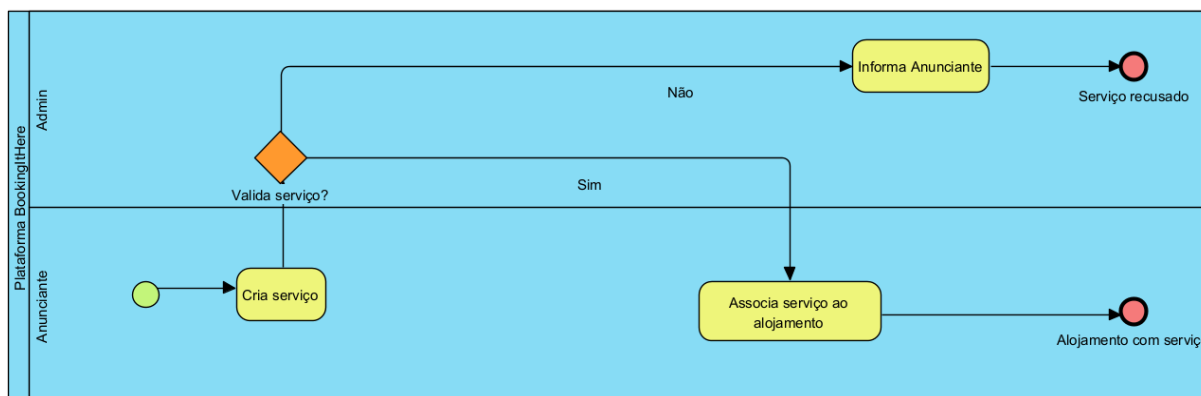


Figura 4- BPMN Serviços Adicionais

O anunciante deverá criar todos os serviços que poderá ter e posteriormente irá associar os mesmos a cada alojamento.

## PN4 - Promoção Alojamento

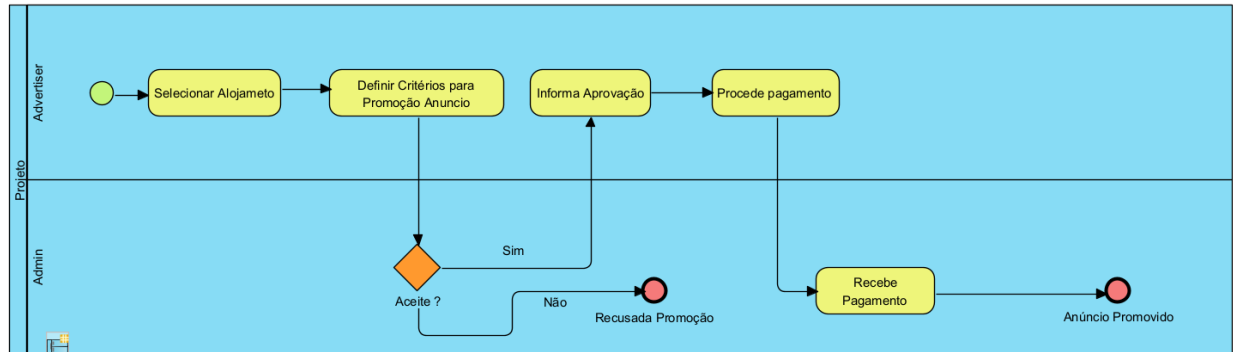


Figura 5- BPMN Promoção Alojamento

O anunciante poderá seleccionar um alojamento para o promover, irá seleccionar quanto pretende pagar por clique, definindo um orçamento mensal, conforme a promoção do seu anúncio esta será validada pelo administrador e irá aparecer ao utilizador.

## PN5 – Apaga Reserva

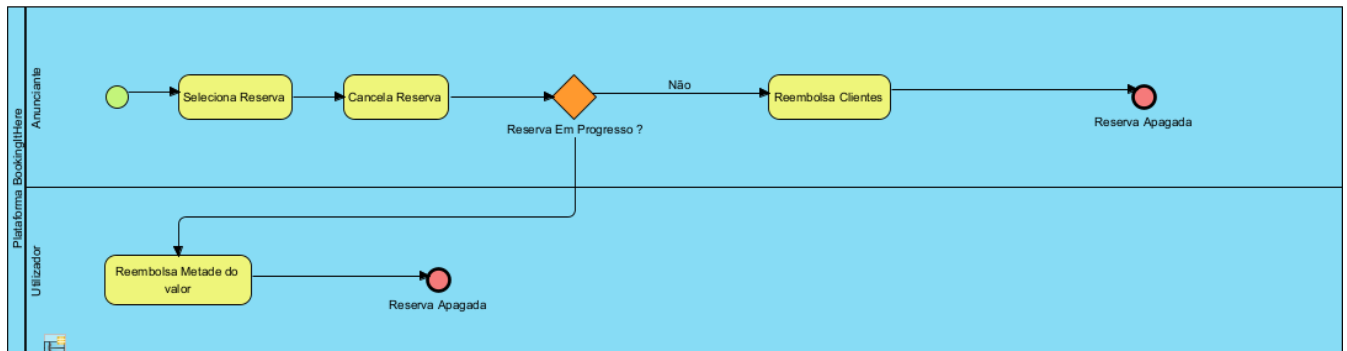


Figura 6- BPMN Apaga Reserva

O anunciante poderá cancelar uma reserva agendada ou em progresso, se estiver agendada reembolsa o cliente, se a mesma estiver em progresso devolve metade do valor.

## Diagrama Classes

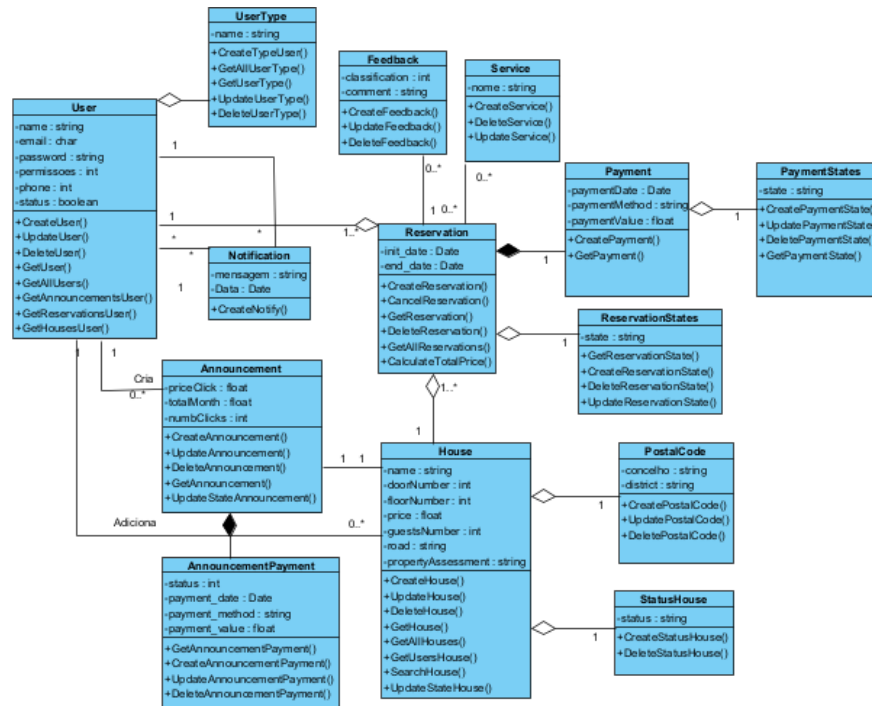


Figura 7- Diagrama Classes

Na nossa plataforma, o utilizador poderá efetuar reservas de vários alojamentos que estarão listados por parte de outros utilizadores, denominados de anunciantes.

A reserva irá ter o alojamento, este alojamento irá ter uma agregação.

## Diagrama ER

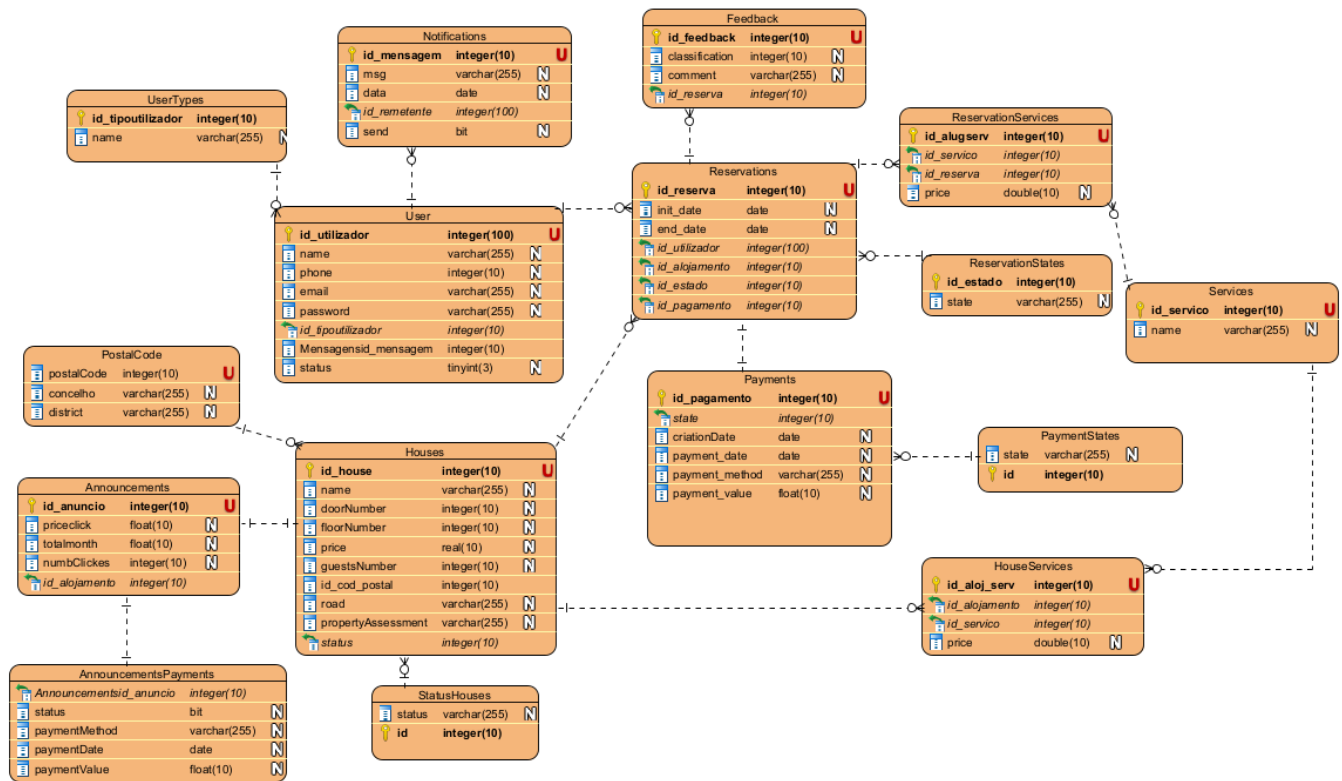


Figura 8- Diagrama ER



## Diagramas Sequência

### PN1 - Alugar Alojamento

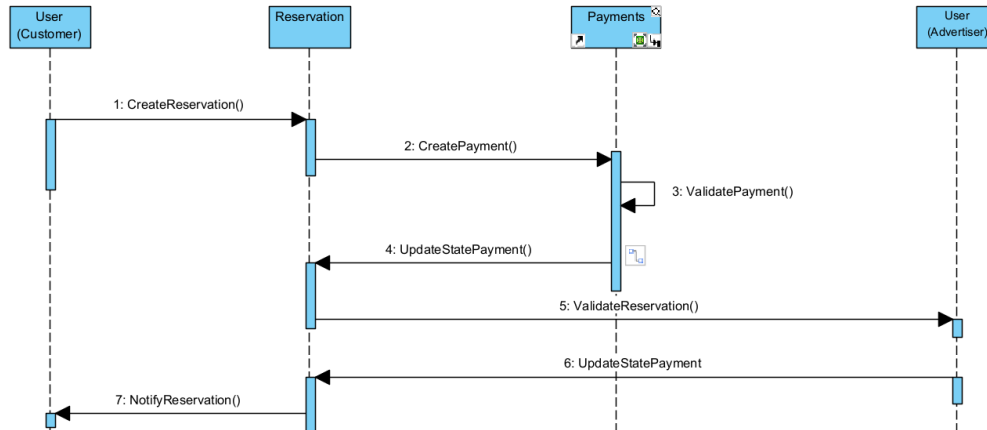


Figura 9- Diagrama Sequência Aluga Alojamento

### PN2 – Apagar Alojamento

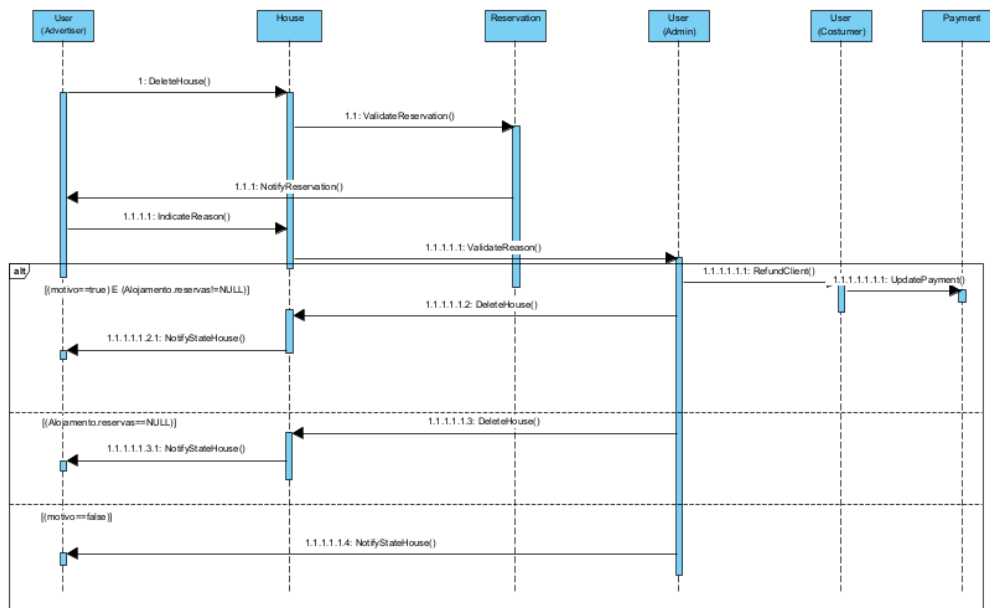


Figura 10- Diagrama Sequência Apaga Alojamento



### PN3 - Serviços Adicionais

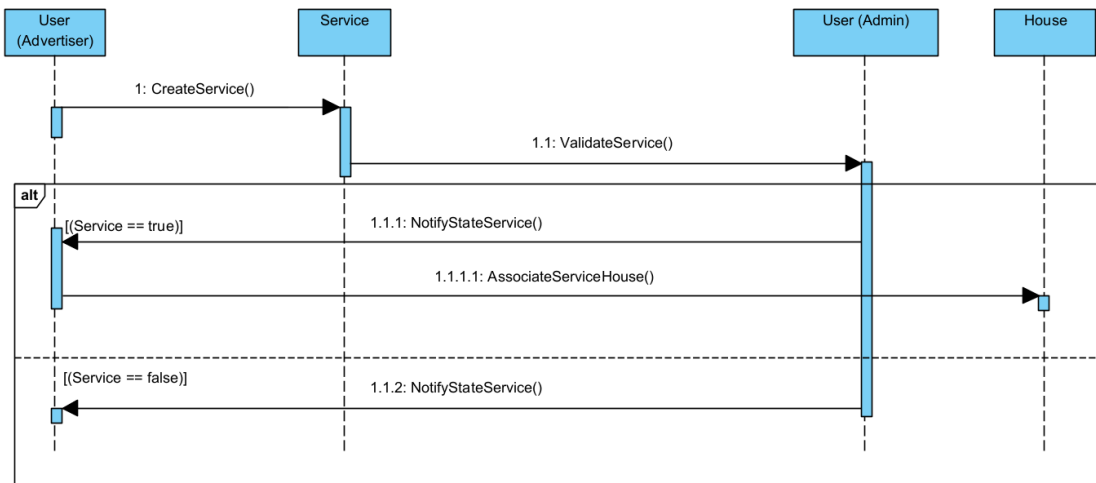


Figura 11- Diagrama Sequência Serviços Adicionais

### PN4 - Promoção Alojamento

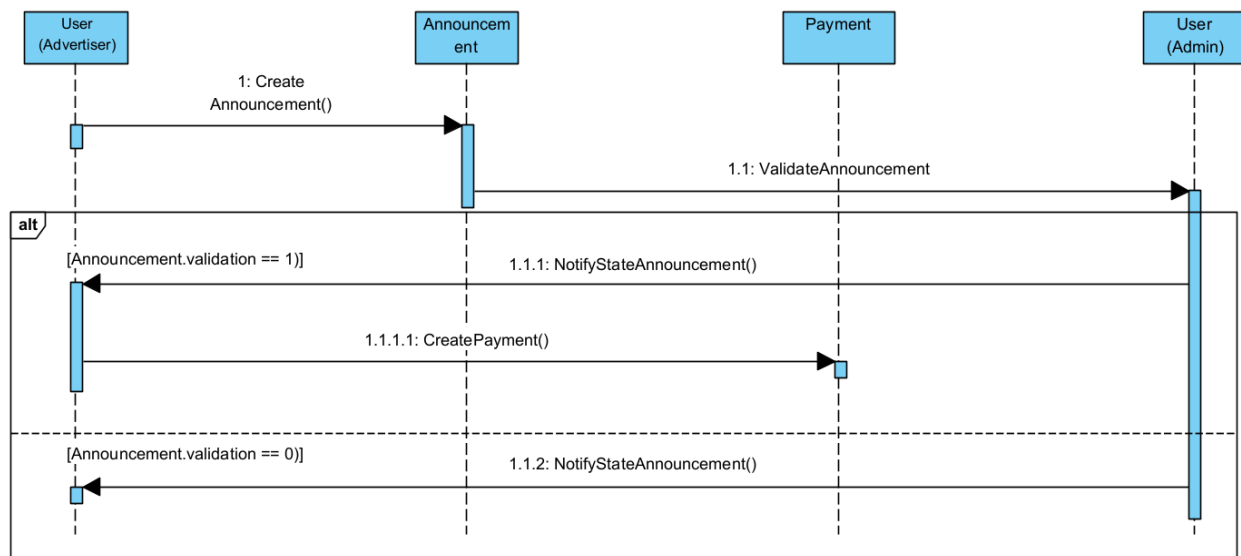


Figura 12- Diagrama Sequência Promove Anúncio

## PN5 - Apaga Reserva

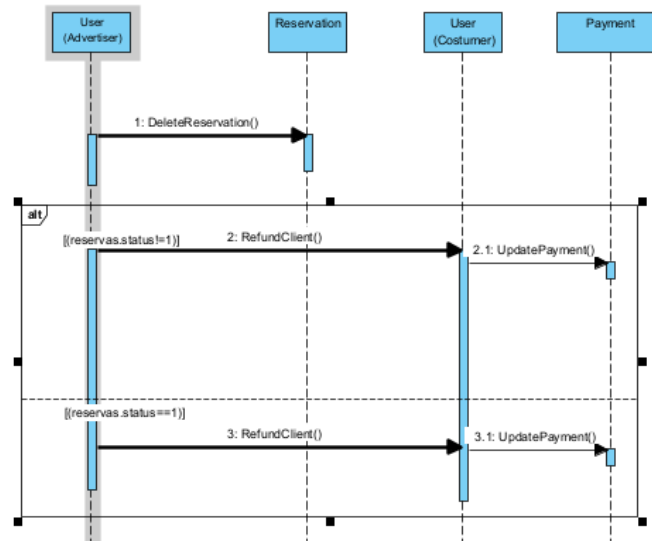


Figura 13- Diagrama Sequência Cancela Reserva

## Mockups

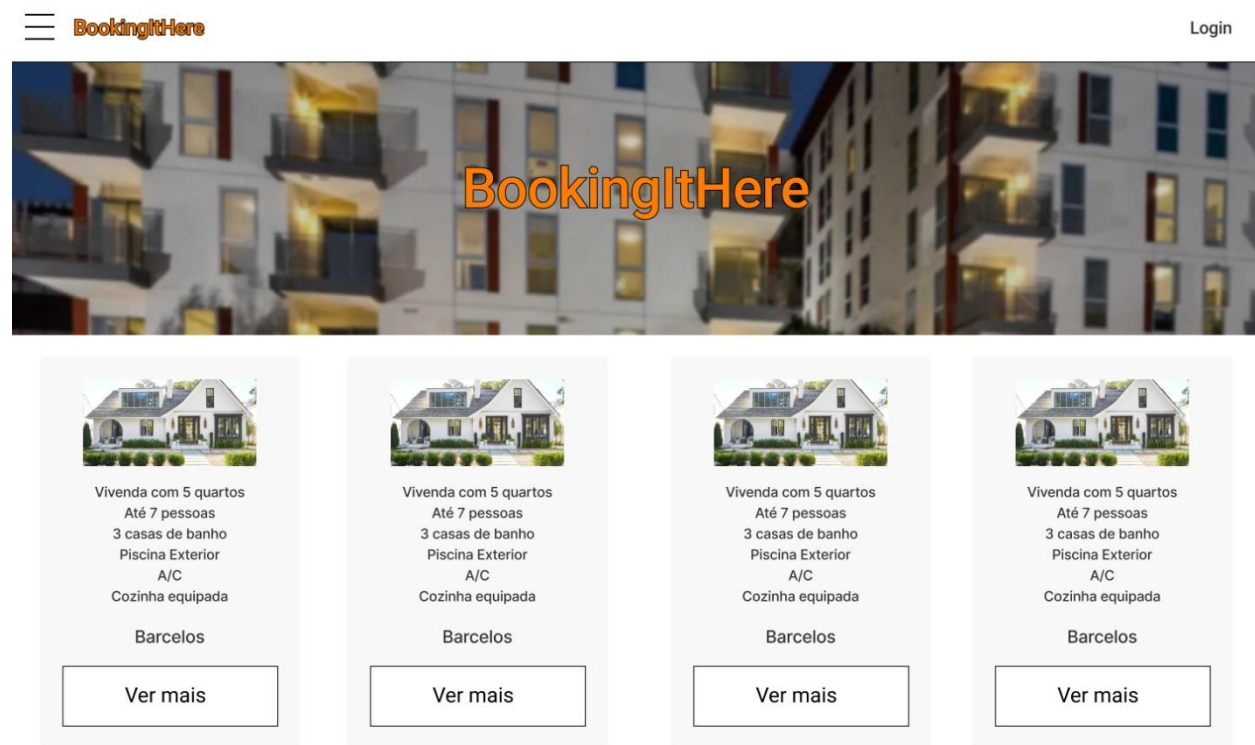



Figura 14- Página Principal Mock



**LOGIN**

**LOGIN**

[ESQUECEU-SE DA PALAVRA PASSE? CLIQUE AQUI](#)

**REGISTAR**




Figura 15 - Ecrã de login



**REGISTO**

REGISTAR

*Figura 16 - Ecrã de registo*

Morada
Número de Quartos 
Número de Casas de Banho 
Tem A/C? 
Observações



(CLIQUE PARA ADICIONAR IMAGENS DO ALOJAMENTO)

CRIAR ALOJAMENTO

Figura 17 - Criar alojamento



**O que o Espaço Oferece**

Casa de férias com 5 quartos Até 7 pessoas 3 casas de banho Piscina Exterior A/C Cozinha equipada.

Localiza-se a 5 minutos do centro da cidade de Barcelos e a 20 minutos da praia

**50€ noite**

Check-In

20/03/2023

Check-Out

25/03/2023

Hospedes: 1 Hospede

**Reservar**

50€ x 5 noites

250€



35 comentários

Figura 18 - Detalhe Anúncio

**CRIAR SERVIÇO**

Título Serviço

Descrição

Valor

**CRIAR SERVIÇO**

Figura 19 - Criar Serviços Extra



### O que o Espaço Oferece

Casa de férias com 5 quartos Até 7 pessoas 3 casas de banho Piscina Exterior A/C Cozinha equipada.

Localiza-se a 5 minutos do centro da cidade de Barcelos e a 20 minutos da praia

Check-In	Check-Out
20/03/2023	25/03/2023
Hospedes: 1 Hospede	
250€	

### Serviços Contratados:

- Limpeza
- Catering

CANCELAR RESERVA

Figura 20 - Detalhes da reserva efetuada

## ALOJAMENTOS













	Alojamento 1	 
	Alojamento 2	 
	Alojamento 3	 
	Alojamento 4	 

Figura 21 - Lista de alojamentos



## RESERVAS









	Reserva 1	20/03/2023 - 25/03/2025	
	Reserva 2	20/03/2023 - 25/03/2025	
	Reserva 3	20/03/2023 - 25/03/2025	
	Reserva 4	20/03/2023 - 25/03/2025	

Figura 22 - Lista de Reservas

## Gestão do Projeto

Ao longo do projeto fomos gerindo o que cada um ia fazendo pelo moodle, mas usamos o azure devops como ferramenta principal para a gestão de tarefas. Atribuímos as tarefas que cada um ia fazendo e fomos criando branches associadas a cada task para cada membro do grupo trabalhar individualmente. Depois dávamos como concluídas as tarefas e fechávamos as branches.

ID	↑	Title	Assigned To	State
77		Calcular valor da reserva (nº de noites e serviços adicionais)	Luís Anjo	Done
84		Consultar Alojamentos do Anunciante	Pedro Martins	Done
85		Consultar Reservas de cada Alojamento	Diogo Silva	Done
87		Consultar Reservas efetuadas (todas as reservas)	Luís Anjo	Done
90		Correção de Bugs	Luís Anjo	Done
97		CRUD Reservas	Luís Anjo	Done
98		Migrate DB mongo to Mysql	João Ponte	Done
99		Criar relações entre o user e o usertype	João Ponte	Done
100				

Figura 23- Azure DevOps

## Criação da API

Para criar a API inicialmente começamos por criar um ambiente Docker, dessa forma garantimos que cada membro do grupo possa trabalhar nas mesmas versões das tecnologias adotadas.

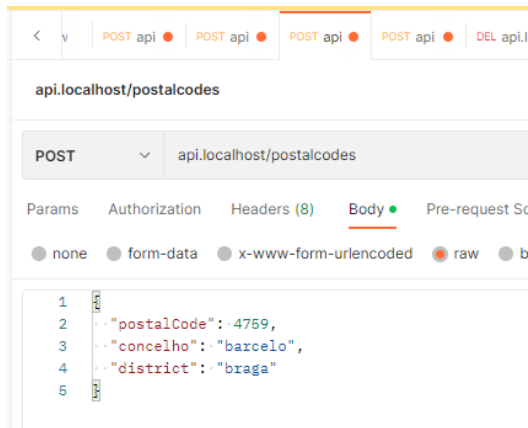
Após configurar o ambiente Docker, optamos por utilizar o Node.js como plataforma de desenvolvimento para a API. Foram definidos os endpoints da API (as rotas), que são as URLs para aceder aos recursos e executar operações específicas. Utilizamos frameworks e bibliotecas populares do ecossistema Node.js, como Express.js, para facilitar a criação dos endpoints e a gestão das rotas.

### Escolha Base de Dados:

Apesar de numa primeira fase termos tentado explorar uma solução com NoSQL (não relacional), em conversa com o professor, chegámos à conclusão que não era a melhor escolha e que deveríamos alterar para MySQL (relacional), e para isso usamos uma ORM chamada Sequelize, que também nos facilitou o trabalho devido a ter métodos próprios ao invés de escrever queries SQL complexas manualmente.

## Testar Rotas:

Inicialmente fomos usando o postman para testar as rotas. Depois acabamos por usar o swagger para testar as rotas que serviu de documentação para a nossa API.



**Figura 24- Teste Postman**



**Figura 25- Teste Swagger**

## Rotas Existentes das Houses



**Figura 26- Rotas Swagger API**

## REDIS:

Usamos também ferramentas tais como o redis para guardar em cache os dados, para ser mais rápido ao aceder. Também usamos o Bcrypt que é uma ferramenta que faz hashing das passwords guardadas na base de dados, para proteger as passwords.

## JWT (TOKEN)

Usamos o JWT para a autenticação, sempre que fazemos login é gerado um token e guardado na tabela do user durante a sessão.

```
const token = jwt.sign(payload, process.env.JWT_SECRET, {  
  expiresIn: process.env.JWT_EXPIRES_IN  
});
```

Figura 27- Sign Generate Token

Update na tabela do user quando faz login com o novo token gerado.

id	name	email	password	phone	token
1					
192	Pattie Hickle	Pattie22@bookingithere.com	\$2b\$10\$imTxiSQEIR0BRG2k3Z1FeO5SCtHQ6kPooxCT1FRMY7...	989789806	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTkyL...

Figura 28- Tabela User com Token



## Testes Unitários / Integração

Depois para fazermos os testes usamos o Jest, fizemos testes unitários para as funções que criamos e testes de integração de forma a cobrir a API parcialmente. Verificamos ao fim, ao fazer os testes de integração que testam as rotas, se registou na base de dados os dados correspondentes.

Para gerar dados válidos para os testes de integração e unitários usamos a biblioteca Faker. Esta biblioteca permitirá a criação de dados fictícios.

### Teste integração (Anúncio):

Para testar a criação do anúncio tivemos de fazer a sequência até o utilizador conseguir criar um anúncio. Para isso este teste terá de criar um utilizador, fazer login com os dados do utilizador criado anteriormente, registar uma casa e por fim criar um anúncio para a casa.

#### Passo 1 (Criação de um user anunciante):

```
62 describe('Anunciante', () => {
63   /**
64    * Create
65    */
66   it('create Anunciante', async () => {
67     try {
68       response = await request(app)
69         .post('/users')
70         .send({
71           email: email,
72           password: password,
73           confirmPassword: password,
74           name: name,
75           phone: phone
76         });
77       // userId = response.body.user.id;
78     } catch (err) {
79       console.log(err);
80     }
81     expect(response.status).toBe(201);
82   });
83 }
```

Figura 29- Teste Cria Utilizador

## Passo 2 (Login)

```

84      /**
85       * Login
86       */
87      it('Login Anunciante', async () => {
88        try {
89          response = await request(app)
90            .post('/auth/login')
91            .send({
92              email: email,
93              password: password
94            });
95
96          token = response.body.token;
97          user = response.body.user;
98
99        } catch (err) {
100          console.error(err);
101        }
102        expect(response.status).toBe(200);
103      });

```

Figura 30- Teste Login

## Passo 3 (Criação de uma casa)

```

97      /**
98       * Registrar uma casa
99       */
100     it('Registrar uma casa', async () => {
101       try {
102         response = await request(app)
103           .post('/houses')
104           .set({
105             Authorization: `Bearer ${token}`
106           })
107           .send({
108             name: houseName,
109             doorNumber: houseDoorNumber,
110             floorNumber: houseFloorNumber,
111             price: parseFloat(housePrice),
112             guestsNumber: houseGuestsNumber,
113             postalCode: housePostalCode,
114             road: houseRoad,
115             propertyAssessment: housePropertyAssessment,
116             concelho: houseConcelho,
117             district: houseDistrict,
118             services: [
119               {
120                 // id: 0,
121                 name: houseServiceName,
122                 price: houseServicePrice
123               }
124             ]
125           });
126
127         houseId = response.body.data.id;
128
129       } catch (err) {
130         console.error(err);
131       }
132       expect(response.status).toBe(201);
133     });

```

Figura 31- Teste Cria Casa

#### Passo 4 (Criar anúncio)

```

135 /**
136  * Registrar um anuncio
137  */
138
139 it('Registrar um anuncio', async () => {
140   try {
141
142     response = await request(app)
143       .post('/announcements')
144       .set({
145         Authorization: `Bearer ${token}`
146       })
147       .send({
148         priceClick: priceClick,
149         numbClicks: numbClicks,
150         house_id: houseId,
151         state: state,
152         end_date: '2023-10-28'
153       });
154
155     announcementId = response.body.data.id
156
157     const createdAnnouncement = await Announcement.findByIdPk(announcementId);
158     expect(createdAnnouncement).toBeDefined();
159     expect(createdAnnouncement.priceClick).toEqual(priceClick);
160     expect(createdAnnouncement.numbClicks).toEqual(numbClicks);
161     expect(createdAnnouncement.house_id).toEqual(houseId);
162     expect(createdAnnouncement.state).toEqual(state);
163     expect(createdAnnouncement.end_date.toISOString().slice(0, 10)).toEqual('2023-10-28');
164
165     const checkPayment = await AnnouncementPayment.findByIdPk(announcementId);
166     expect(checkPayment.announcement).toEqual(announcementId); // Verifica se o pagamento foi criado após a criação do anúncio
167
168   } catch (err) {
169     console.error(err);
170   }
171   expect(response.status).toBe(201);
172
173 });
174

```

Figura 32- Teste Cria Anúncio

De forma a verificar se os dados inseridos correspondem aos dados passados por parâmetro usamos os *expect statements*. Estes servirão para validar o comportamento esperado do sistema em teste.

No teste do anúncio, verificamos se os dados inseridos correspondem aos dados passados por parâmetro e verificamos também se o pagamento foi criado após a criação do anúncio.

Por fim, apenas verificamos se o estado da resposta HTTP retorna o código 201, ou seja, se a solicitação foi bem-sucedida.

Após a compilação do comando de testes, o teste foi bem-sucedido:

```

PASS  __tests__/integration-tests/announcement.test.js (9.77 s)

```

Figura 33 - Confirmação Teste Anúncio

## Seeds

Fizemos um seed e usamos o faker para gerar dados conforme o tipo de dados para a nossa api, esse ficheiro deve ser gerado na primeira vez que são geradas as tabelas.

```
267 // Announcements Seeds
268 async function generateAnnouncements(num) {
269   let i = 0;
270   for (i = 0; i < num; i++) {
271     const priceClick = faker.datatype.number({min: 0.1, max: 5});
272     const numbClicks = faker.datatype.number({min: 1, max: 1000});
273     const state = faker.datatype.number({min: 0, max: 1});
274     const end_date = faker.date.future();
275     const house_id = faker.datatype.number({min: 1, max: 100});
276     const status = faker.datatype.number({min: 1, max: 2});
277     const creationDate = faker.date.past();
278     const paymentDate = faker.date.future();
279     const paymentValue = faker.datatype.number({min: 30, max: 200});
280
281     let findHouse = await House.findOne({where:{id: house_id}});
282     if(findHouse){
283
284       let announcement = await Announcement.create({
285         priceClick: priceClick,
286         numbClicks: numbClicks,
287         house_id: house_id,
288         state: state,
289         end_date: end_date,
290       });
291
292       await AnnouncementPayment.create({
293         announcement: announcement.id,
294         status: status,
295         creationDate: creationDate,
296         paymentDate: paymentDate,
297         paymentMethod: "MBway",
298         paymentValue: paymentValue,
299       });
300     }
301   }
302   return 'Announcements=' + (i === num ? 'done' : 'fail');
303 }
304
```

Figura 34 - Seeds do Anúncio

Este seed irá fazer a verificação do id da casa gerado, caso passe pela validação irá criar um anúncio e o pagamento do anúncio correspondente.

## Back-End

### Modelos

Usando o Sequelize como ORM definimos modelos para as funcionalidades que pretendemos implementar na nossa API. Os Models foram usados para definir as estruturas das tabelas bem como as relações entre elas. Com estes models podemos realizar várias operações na base de dados através da interação com objetos de JavaScript em vez de escrever manualmente queries de SQL. Para demonstrar os exemplos de models utilizados no nosso código temos o exemplo:

```

57 class Reservation extends Model {
58
59   static init(sequelize) {
60     super.init({
61       id: {
62         type: DataTypes.INTEGER,
63         primaryKey: true,
64         autoIncrement: true,
65       },
66       init_date: {
67         type: DataTypes.DATE,
68         required: true
69       },
70       end_date: {
71         type: DataTypes.DATE,
72         required: true
73       },
74       guestsNumber: {
75         type: DataTypes.INTEGER,
76         required: true
77       },
78       user_id: {
79         type: DataTypes.INTEGER,
80         allowNull: true,
81         required: true
82       },
83       house_id: {
84         type: DataTypes.INTEGER,
85         required: true,
86         allowNull: true
87       },
88       state_id: {
89         type: DataTypes.INTEGER,
90         allowNull: true,
91         required: true
92       }
93     }, {sequelize})
94   }
95
96   static associate(models) {
97     this.belongsTo(models.User, {foreignKey: 'user_id'});
98     this.belongsTo(models.ReservationState, {foreignKey: 'state_id'});
99     this.belongsTo(models.House, {foreignKey: 'house_id'});
100    this.belongsToMany(models.Service, {through: 'ReservationServices'});
101    this.hasOne(models.Feedback, {foreignKey: 'reservation'});
102    this.hasOne(models.Payment, {foreignKey: 'reservation_id'});
103  }
104 }
105

```

Figura 36 - Modelo Reserva

```

31 class Feedback extends Model {
32
33   static init(sequelize){
34     return super.init({
35       id:{
36         type: DataTypes.INTEGER,
37         primaryKey: true,
38         autoIncrement: true,
39       },
40       reservation:{
41         type: DataTypes.INTEGER,
42         required: true
43       },
44       comment:{
45         type: DataTypes.STRING,
46         required: true
47       },
48       classification: {
49         type: DataTypes.INTEGER,
50         required: true
51       }
52     }, { sequelize })
53   }
54
55   static associate(models) {
56     this.belongsTo(models.Reservation, { foreignKey: 'reservation'});
57   }
58
59 }
60
61 module.exports = Feedback;

```

Figura 35 - Modelo Feedback

No modelo da reserva iremos definir os atributos e o tipo de cada atributo. No final de cada modelo fazemos as relações que são necessários para o funcionamento da aplicação. Neste caso temos uma relação de um para muitos, ou seja, uma reserva terá vários feedbacks, mas um feedback irá pertencer a apenas uma reserva.

## Mensagens status code

Criamos mensagens para cada controller, para controlar os erros. Cada erro terá o seu status code. Dividimos a secção das mensagens em “success” e “error”:

```
success: {
  houseApproval: {
    http: 201,
    code: "houseInApproval",
    type: "success"
  },
  houseUpdated: {
    http: 200,
    code: "houseUpdated",
    type: "success"
  },
  houseDeleted: {
    http: 200,
    code: "houseDeleted",
    type: "success"
  }
},
```

Figura 37 - Mensagens Sucesso Casas

```
error: {
  fieldMissing: {
    http: 401,
    code: "FieldMissing",
    type: "error"
  },
  houseAlreadyRegistered: {
    http: 401,
    code: "HouseAlreadyRegistered",
    type: "error"
  },
  houseDontExist: {
    http: 400,
    code: "houseDontExist",
    type: "error"
  },
  houseStatusDontExist: {
    http: 400,
    code: "houseStatusDontExist",
    type: "error"
  }
}
```

Figura 38 - Mensagens Erro Casas

## Middlewares

Usamos middlewares de forma a processar e controlar requisições HTTP.

De forma a organizar e filtrar o acesso às nossas operações, definimos middlewares seja para o admin, anunciante e utilizador. Definimos o middleware “authMiddleware”, este middleware irá obrigar a autenticação do utilizador para a execução de qualquer processo que contenha este middleware.

De seguida, definimos o middleware “advertiserMiddleware”, este servirá também para controlar o acesso às rotas, apenas os anunciantes, ou seja, os utilizadores que têm alojamento disponibilizados no website, é que vão ter acesso a estas rotas.

Por fim, definimos o middleware “adminMiddleware”. Esta implementação garante que somente utilizadores com permissões de administrador possam aceder às rotas protegidas pelo middleware do administrador.

## Rotas

Usamos rotas para cada controller, atribuímos conforme a operação que pretende ser realizada a rota na api.

Cada rota terá os seus middlewares de forma a filtrar os acessos das rotas.

Exemplo de algumas rotas das operações do controlador das casas:

Rota POST '/houses':

```
54 router.post('/', [
55   check('name').isString(),
56   check('doorNumber').isInt(),
57   check('floorNumber').isLength({min: 1, max: 12}),
58   check('price').isFloat(),
59   check('guestsNumber').isInt(),
60   check('postalCode').isInt(),
61   check('road').isString(),
62   check('propertyAssessment').isString(),
63   check('concelho').isString(),
64   check('district').isString()
65 ], authMiddleware, (req, res) => {
66   const errors = validationResult(req);
67   if (!errors.isEmpty()) {
68     return res.status(406).json({errors: errors.array()});
69   }
70   return houseController.create(req, res);
71 });
```

Figura 39 - Rota POST Casas

- Middlewares utilizados: authMiddleware
- Descrição: Esta rota servirá para a criação da casa do utilizador. Para o utilizador criar uma casa terá de estar registado no website de forma a conseguir identificar o dono da casa que será criada.

Rota PUT '/houses/:id':

```
115 router.put('/:id',[
116     check('name').optional().isString(),
117     check('doorNumber').optional().isInt(),
118     check('floorNumber').optional().isLength({min: 1, max: 12}),
119     check('price').optional().isFloat(),
120     check('guestsNumber').optional().isInt(),
121     check('postalCode').optional().isInt(),
122     check('road').optional().isString(),
123     check('propertyAssessment').optional().isString(),
124     check('concelho').optional().isString(),
125     check('district').optional().isString()
126 ],[
127     authMiddleware,
128     advertiserMiddleware
129 ], (req, res) => {
130     const errors = validationResult(req);
131     if (!errors.isEmpty()) {
132         return res.status(406).json({errors: errors.array()});
133     }
134     return houseController.update(req, res);
135 });
```

Figura 40 - Rota PUT Casas

- Middlewares utilizados: authMiddleware, advertiserMiddleware
- Descrição: Esta rota permitirá ao anunciante alterar dados da sua casa. O tipo de utilizador irá ser alterado para anunciante após a criação da sua casa.



## Controllers

Os controllers serão os responsáveis por receber as requisições HTTP vindas das rotas e determinar qual ação deve ser executada. Estes vão processar os dados recebidos, interagir com os models e retornar respostas apropriadas.

Dentro dos controladores, temos métodos para cada tipo de requisição HTTP, como GET, POST, PUT e DELETE.

Os controllers irão fornecer uma camada intermediária entre as rotas e os modelos, permitindo que as regras de negócio sejam implementadas de forma eficiente e modular.

Exemplo de um POST utilizadores:

```
259 *
260 * @author João Ponte
261 * @param req
262 * @param res
263 * @returns {Promise<*>}
264 */
265 exports.create = async (req, res) => {
266   try {
267     if (!req.body.email || !req.body.password || !req.body.name) {
268       return res.status(userMessage.error.fieldMissing.http).send(userMessage.error.fieldMissing);
269     }
270
271     if (req.body.password !== req.body.confirmPassword) {
272       userMessage.error.fieldMissing.errors = [{
273         param: 'password',
274         msg: 'Password and Confirm Password does not match.'
275       }];
276       return res.status(userMessage.error.fieldMissing.http).send(userMessage.error.fieldMissing);
277     }
278
279     let user;
280     try {
281       user = await User.create({
282         email: req.body.email,
283         password: bcrypt.hashSync(req.body.password, parseInt(process.env.BCRYPT_SALTROUNDS)),
284         name: req.body.name,
285         phone: req.body.phone,
286         user_type_id: req.body.user_type_id ?? 1
287       });
288     } catch (e) {
289       console.log(e);
290       return res.status(userMessage.error.emailAlreadyRegistered.http).send(userMessage.error.emailAlreadyRegistered);
291     }
292
293     userMessage.success.userCreated.data = {
294       id: user.id
295     };
296     return res.status(userMessage.success.userCreated.http).send(userMessage.success.userCreated);
297   } catch (err) {
298     console.error('Error during registration', err);
299
300     return res.status(500).send({error: 'Internal server error'});
301   }
302 };
```

Figura 41 - Função Criar Utilizador

Esta função valida se os dados passados não estão vazios, caso algum parâmetro esteja vazio, o utilizador receberá uma mensagem de erro do tipo `fieldMissing` que foi declarado anteriormente nas mensagens.

Esta irá confirmar também se a confirmação da password equivale à password e caso não corresponda o utilizador receberá um erro avisando que a mesma não corresponde.

Por fim, caso passe por todas as verificações irá criar um utilizador com os dados que o utilizador inseriu encriptando a password do mesmo com a ferramenta `Bcrypt` de forma a proteger os dados do utilizador.

## Código das regras de negócio

### Cancelar reserva

```
405 exports.softDelete = async (req, res) => {
406   const {id} = req.params;
407   try {
408     const result = await Reservation.findByPk(id);
409     if (!result) {
410       return res.status(reservationMessage.error.reservationDontExist.http).send(reservationMessage.error.reservationDontExist);
411     }
412     const cancelState = await ReservationState.findOne({ where: { state: "Cancelada" } });
413     if(result.state_id==1){
414       if(result && Date.now()>result.init_date && Date.now()<result.end_date){
415         await result.update({state_id:cancelState.id});
416         const cancelPrice = await Payment.findOne({
417           where: { reservation_id: id },
418         });
419         if (cancelPrice) {
420           await cancelPrice.update({
421             state_id: 3,
422             paymentValue: cancelPrice.paymentValue * 0.5
423           });
424         }
425       } else if (result) {
426         await result.update({state_id:cancelState.id});
427         const cancelPrice = await Payment.findOne({
428           where: { reservation_id: id },
429         });
430         if (cancelPrice) {
431           await cancelPrice.update({
432             state_id: 3
433           });
434         }
435       }
436     } else{
437       return res.status(reservationMessage.error.reservationDontExist.http).send(reservationMessage.error.reservationDontExist);
438     }
439     return res.status(reservationMessage.success.reservationUpdated.http).send(reservationMessage.success.reservationUpdated);
440   } catch (err) {
441     console.error('Error during process', err);
442     return res.status(500).send({error: 'Internal server error'});
443   }
444 };
```

Figura 42 - Função RN Cancelar Reserva

Esta função começa por receber o id da reserva que o utilizador deseja cancelar e verifica se a mesma existe. Caso exista e esta esteja em estado 1, ou seja, pendente, irá verificar se o dia atual está entre a data inicial da reserva e a data final, se o utilizador cancelar a reserva entre as datas início e fim a reserva passará a estar cancelada, o pagamento ficará cancelado e apenas metade do valor do pagamento é devolvido. Caso este cancele antecipadamente a reserva passará a ficar cancelada e o pagamento fica também cancelado com o mesmo valor que o utilizador gastou na reserva.

## Apagar alojamento

```
391 exports.hardDelete = async (req, res) => {
392   const id = parseInt(req.params.id);
393
394   try {
395     const house = await House.findByPk(id);
396     if (!house) {
397       return res.status(houseMessage.error.houseDontExist.http).send(houseMessage.error.houseDontExist);
398     }
399
400     const activeReservations = await Reservation.findAll({
401       where: {
402         house_id: house.id,
403         state_id: {
404           [Op.or]: [1, 2]
405         }
406       }
407     });
408
409     if (activeReservations.length > 0) {
410       for (const activeReservation of activeReservations) {
411         const updateReservation = await activeReservation.update({
412           state_id: 5,
413         });
414
415         const payment = await Payment.findOne({
416           where: {
417             reservation_id: activeReservation.id
418           }
419         });
420
421         if (payment) {
422           const editPayment = await payment.update({
423             userId: activeReservation.userId,
424             paymentValue: payment.amount,
425             creationDate: new Date(),
426             paymentDate: new Date(),
427             paymentMethod: payment.paymentMethod,
428             reservation_id: activeReservation.id,
429             state_id: 3,
430           });
431         }
432       }
433
434       await house.destroy();
435
436       return res.status(houseMessage.success.houseDeleted.http).send(houseMessage.success.houseDeleted);
437     } catch (err) {
438       console.error('Error during process', err);
439
440       return res.status(500).send({error: 'Internal server error'});
441     }
442   }
443 }
444 };
```

Figura 43 - Função RN Apagar Alojamento

A função começa por procurar pela casa com o id que recebe por parâmetro. De seguida, declaramos uma variável que vai procurar por todas as reservas que têm o house\_id do parâmetro e que estão com o id 1 e 2 (pendente e em uso) para saber quais as reservas ativas na casa que o anunciante está prestes a apagar. Caso a casa tenha reservas ativas irá entrar no *for*. Dentro do ciclo *for*, o estado da reserva irá passar para 5 (Cancelada) e o pagamento associado passará também ao estado 3 (Reembolsado).

Anúncios (Promoção do alojamento)

```

71 exports.create = async (req, res) => {
72   const {
73     house_id,
74     priceClick,
75     numbClicks,
76     end_date
77   } = req.body;
78
79   try {
80     if (!house_id || !numbClicks || !end_date) {
81       return res.status(announcementMessage.error.fieldMissing.http).send(announcementMessage.error.fieldMissing);
82     }
83
84     let findHouse = await Announcement.findOne({where: { house_id: house_id }});
85     let findH = await House.findOne({where: {id: house_id}});
86     let announcementPayment;
87
88     if(!findH){
89       return res.status(houseMessage.error.houseDontExist.http).send(houseMessage.error.houseDontExist);
90     }
91     if (findHouse) {
92       return res.status(announcementMessage.error.announcementAlreadyRegistered.http).send(announcementMessage.error.announcementAlreadyRegistered);
93     }
94
95     let announcement = await Announcement.create({
96       house_id,
97       priceClick,
98       numbClicks,
99       state: 0,
100       end_date
101     });
102
103     const valor = await Announcement.calculateTotalValue(priceClick, numbClicks, end_date);
104     announcementPayment = await AnnouncementPayment.create({
105       announcement : announcement.id,
106       status : 0,
107       creationDate : new Date(),
108       paymentDate : new Date(),
109       paymentMethod : "Mbway",
110       paymentValue : valor
111     });
112
113     announcementMessage.success.announcementApproval.data = {
114       id: announcement.id
115     };
116
117     return res.status(announcementMessage.success.announcementApproval.http).send(announcementMessage.success.announcementApproval);
118
119   } catch (err) {
120     console.error('Erro during creation', err);
121
122     return res.status(500).send({error: 'Internal server error'});
123   }
124 };

```

Figura 44 - Função RN Anúncios

Esta função começa por fazer 2 pesquisas. Pesquisa pelo anúncio que está associado à casa caso já exista e pesquisa pelo id da casa que o utilizador referiu. Caso não encontre a casa, o mesmo receberá um erro a informá-lo que a casa não existe. Caso exista a casa poderá não passar na verificação da variável "findHouse" esta verificação irá validar se a casa já foi anunciada. Por fim, se passar por todas as verificações, o anúncio será criado com a casa correspondente e também será criado um pagamento para o anúncio.

## Serviços

```
67 exports.create = async (req, res) => {
68   const {name, state} = req.body;
69
70
71   try {
72     if (!name){
73       return res.status(serviceMessage.error.fieldMissing.http).send(serviceMessage.error.fieldMissing);
74     }
75
76     let service = await Service.findOne({where: { name: name }});
77     if (service) {
78       return res.status(serviceMessage.error.serviceAlreadyRegistered.http).send(serviceMessage.error.serviceAlreadyRegistered);
79     }
80
81     service = await Service.create({name, state: 0});
82
83     if(service === false){
84       return res.status(serviceMessage.error.fieldMissing.http).send(serviceMessage.error.fieldMissing);
85     }
86
87     return res.status(serviceMessage.success.serviceApproval.http).send(serviceMessage.success.serviceApproval);
88   } catch (err) {
89     console.error('Erro during creation', err);
90
91     return res.status(500).send({error: 'Internal server error'});
92   }
93 };
```

Figura 45 - Função RN Serviços

Esta função verifica se o nome do serviço que o anunciante está a tentar criar já existe. Caso o nome do serviço já exista o utilizador receberá um aviso relativamente ao erro. Caso o serviço ainda não exista, o mesmo será criado com o estado 0 pois precisará de ser aprovado por administradores.

```
if (services) {
  await Promise.all(services.map(async service => {
    let findService = await Service.findOne({where: {name: service.name}});

    if (!findService) {
      findService = await Service.create({name: service.name});
    }

    if(findService){
      const houseService = await HouseServices.create({
        house_id: house.id,
        service_id: findService.id,
        price: service.price,
      });
    }
  })))
}
```

Figura 46 - Função RN Serviços Casa

Esta condição *if* está localizada no create do houseController, ou seja, caso o anunciante pretenda adicionar serviços na casa, estes terão de passar por algumas validações. Começa por verificar se o serviço existe procurando pelo nome do serviço na tabela dos serviços. Caso não encontre o serviço, este será criado.

Se o serviço for criado, será adicionado um registo na tabela HouseServices, esta tabela será a tabela intermediária entre a House e Services, de forma a conseguir guardar vários serviços numa só casa.

## Front-End

### Implementação

Para o desenvolvimento do Front-End do nosso projeto usamos React.js. Começamos no App.js por definir as rotas do front e as suas páginas correspondentes. Depois dentro de cada página chamamos as rotas da api para processar a informação conforme necessário.

Por fim também colocamos online a nossa API com o front end. Usamos uma vps para ter o nosso site em execução.

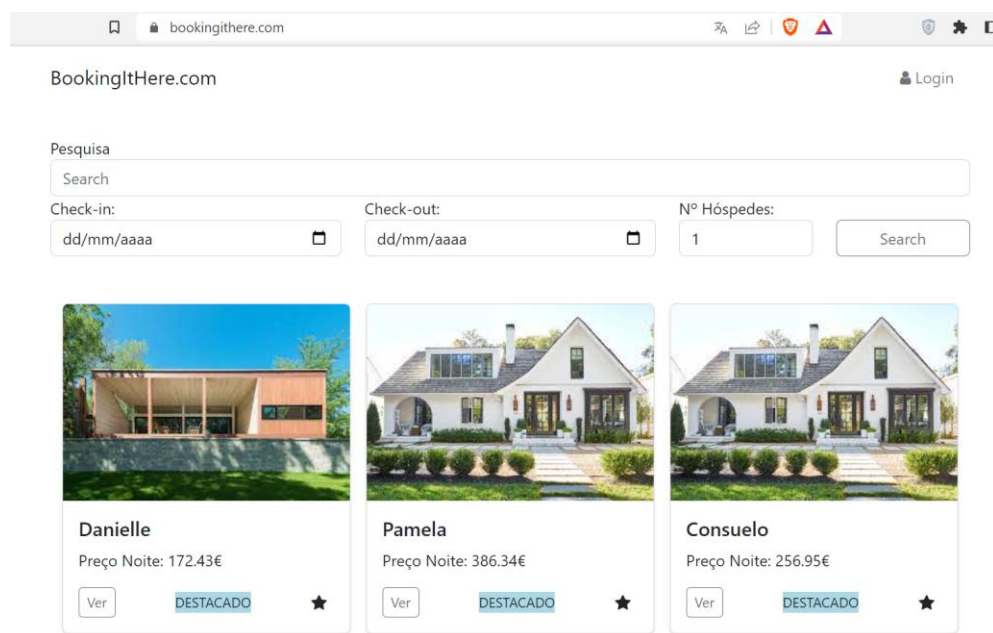



Figura 47- Home do Website

🔍🏠 bookingithere.com/houses/30

🔍🏠🔗🛡️🚩



## Serviços Adicionais

- Limpeza | 33€

## Efetuar Reserva

Check-in:

dd/mm/aaaa📅

Check-out:

dd/mm/aaaa📅

Nº Hóspedes:

Serviços:

Sem serviços  
Limpeza

Reservar

## Descrição da Casa

Preço Noite: 873.94€

Nº Max Pessoas: 8

Andar: 10

Proprietário: Iustus Caccin

Código Postal: 5572

Rua: Gerlach Unions

NumPorta: 145

Figura 48- Reserva House Website



## Conclusão

Neste trabalho foi possível colocar em prática os conhecimentos obtidos em várias Unidades Curriculares lecionadas anteriormente. Utilizamos conhecimentos adquiridos para nos ajudar a planejar e gerir o projeto através de Sprints e de um project backlog e através de vários diagramas como os diagramas de classes, diagramas entidade relação e diagramas de sequência. Com a análise das sprints foi também possível uma melhor estruturação e análise do projeto.

Também desenvolvemos as nossas capacidades relativamente ao desenvolvimento de código num contexto de backend da aplicação com a construção de uma API e no contexto de Frontend da aplicação construindo uma interface web.