Licenciatura em Engenharia Informática e de Computadores

Court&Go

Plataforma de Gestão de Jogos e Campos de Padel e Ténis

Projeto e Seminário 2024/2025

João Mendonça, nº 48180, email: a48180@alunos.isel.pt

Tiago Silva, nº 48252, email: a48252@alunos.isel.pt

Orientadores:

Paulo Pereira, email: paulo.pereira@isel.pt

Luís Falcão, email: luis.falcao@isel.pt

Julho de 2025

Resumo

O Court&Go é uma aplicação web para marcação de campos de ténis e padel, que permite aos utilizadores consultar disponibilidade, efetuar reservas e ver histórico de reservas. A plataforma inclui ainda funcionalidades para gestão de campos, localizações e perfis de proprietários.

A aplicação foi desenvolvida com uma arquitetura serverless, utilizando serviços da AWS como RDS para base de dados e Lambda para execução da lógica de negócio. A abordagem serverless permite tirar partido de uma arquitetura altamente escalável sem necessidade de gerir servidores diretamente, o que se traduziu num desenvolvimento mais ágil e alinhado com práticas modernas. A infraestrutura foi gerida com recurso a ferramentas de Infrastructure as Code, assegurando consistência no processo de deployment.

Durante o desenvolvimento, foram enfrentados desafios como a definição do modelo relacional de dados, a configuração dos serviços AWS e a integração de diferentes componentes da infraestrutura. No âmbito do desenvolvimento em Kotlin Multiplatform [1] também se verificou alguma falta de bibliotecas compatíveis simultaneamente para Android e para iOS.

Esta aplicação representa uma solução funcional para o setor desportivo, que cobre o ciclo completo de reserva de campos, desde a criação até à confirmação dos participantes. Capaz de ser integrada por clubes, academias ou campos independentes que pretendam otimizar a gestão de reservas e melhorar a experiência dos seus utilizadores.

Palavras-chave: ténis, padel, Android, iOS, multiplatform, kotlin, reservas, aplicação móvel, AWS.

Abstract

Court&Go is a web application for booking tennis and padel courts, which allows users to check availability, make reservations, and view the reservation history. The platform also includes features for managing courts, locations, and owner profiles.

The application was developed with a serverless architecture, using AWS services such as RDS for the database and Lambda for executing business logic. The serverless approach allows for a highly scalable architecture without the need to directly manage servers, resulting in more agile development aligned with modern practices. The infrastructure was managed using Infrastructure as Code tools, ensuring consistency in the deployment process.

During development, challenges were faced such as defining the relational data model, configuring AWS services, and integrating different infrastructure components. In the context of development on the Kotlin Multiplatform [1], there was also a lack of libraries that were simultaneously compatible with Android and iOS.

This application represents a functional solution for the sports sector, which covers the complete cycle of court booking, from creation to confirmation of participants. Capable of being integrated by clubs, gyms, or independent courts that want to optimize reservation management and improve the experience of their users.

Keywords: tennis, padel, Android, iOS, multiplatform, kotlin, reservation, mobile application, AWS services.

Índice

Lista de Figuras

Capítulo 1

Introdução

Os sistemas de marcação e gestão de campos desportivos têm ganho relevância nos últimos anos, em resposta à crescente digitalização de serviços e à procura por soluções que promovam a eficiência e acessibilidade na organização de atividades recreativas.

Embora já existam várias plataformas digitais para marcação de campos, a sua evolução técnica e integração com tecnologias modernas é um fenómeno recente, impulsionado pela crescente necessidade de gerir de forma mais eficiente a disponibilidade de infraestruturas desportivas, especialmente no contexto do ténis e do padel.

Com o aumento da procura por estes desportos, torna-se essencial dispor de ferramentas que centralizem a informação de clubes e campos, permitam a reserva em tempo real, e integrem funcionalidades como gestão de perfis, histórico de reservas e pesquisa de campos a nível nacional. Neste contexto, o desenvolvimento de aplicações móveis com suporte multiplataforma torna-se uma solução eficaz para colmatar estas necessidades.

A aplicação Court&Go foi concebida com o objetivo de oferecer uma plataforma completa para a marcação de campos de ténis e padel, direcionada tanto para utilizadores individuais como para entidades gestoras. A arquitetura serverless adotada, baseada em serviços da AWS como Lambda e API Gateway V2, permite uma execução escalável e flexível da lógica de negócio, sem dependência da gestão direta de servidores. Esta abordagem é reforçada pela utilização de práticas modernas como Infrastructure as Code, assegurando reprodutibilidade e consistência no processo de deployment.

Durante o desenvolvimento da aplicação, surgiram desafios significativos relacionados com a definição do modelo relacional de dados, a configuração da infraestrutura cloud e a integração de componentes multiplataforma, nomeadamente no contexto do Kotlin Multiplatform, onde foi identificada alguma escassez de bibliotecas compatíveis com Android e iOS de forma simultânea.

O Court&Go representa, assim, uma solução funcional e extensível para o setor desportivo, com capacidade de integração por clubes, academias ou campos independentes. Através da automatização do processo de reservas e da disponibilização de ferramentas de gestão centralizadas, a plataforma contribui para a melhoria da experiência do utilizador e para a otimização da operação por parte das entidades gestoras. A natureza modular e escalável do sistema permite ainda a sua evolução contínua, assegurando adaptabilidade a novas exigências ou integrações futuras.

1.1 Objetivos

A principal finalidade deste projeto consiste no desenvolvimento de uma aplicação multiplataforma que permita a marcação e gestão de campos de ténis e padel, proporcionando uma experiência simples e eficiente tanto para os utilizadores como para os proprietários dos campos. Pretende-se disponibilizar funcionalidades que possibilitem a consulta da

disponibilidade de campos em tempo real, a realização de reservas, acesso ao histórico de reservas e perfil com dados dedicados.

Adicionalmente, é disponibilizada uma área dedicada à gestão por parte dos proprietários, onde estes podem inserir os dados dos seus campos para os mesmos estarem disponíveis para reservas.

Com este projeto, pretende-se não só responder às necessidades dos utilizadores e clubes desportivos, como também explorar práticas modernas de desenvolvimento multiplataforma e de serviços da AWS.

1.2 Estrutura do Documento

Este documento consiste em nove capítulos, com a finalidade explicar em detalhe a aplicação Court&Go.

No Capítulo 2 apresenta a análise do problema, onde o mesmo é descrito e são abordados alguns conceitos essenciais e apresentadas algumas funcionalidades fulcrais para resolução do problema, é exposta também a motivação. Desta forma é demonstrada uma visão mais clara do objetivo do projeto e da motivação.

No Capítulo 3 descreve-se a solução implementada de forma detalhada através da abordagem dos requisitos obrigatórios e opcionais, a arquitetura do sistema e as tecnologias utilizadas, tanto para Frontend como para Backend.

No Capítulo 4, está explicado de uma forma geral como funciona cada uma das funcionalidades do projeto.

O Capítulo 5 aborda em detalhe a implementação da API (backend) como funciona o paradigma Serverless e os serviços aws utilizados no funcionamento da API.

O Capítulo 6 evidencia os detalhes da implementação e organização do site web para donos de clubes da plataforma Court&Go e a jornada de navegação de um utilizador.

No Capítulo 7, aborda-se os detalhes da implementação da aplicação móvel, a jornada de navegação do utilizador ao longo dos ecrãs da aplicação, interação com a API e processo de Deploy.

O Capítulo 8, foca-se nos tipos de testes realizados na aplicação ao longo do projeto.

Por último, o Capítulo 9 descreve os desafios enfrentados ao longo do desenvolvimento do projeto, e também analisa o restante trabalho a ser feito no futuro.

Capítulo 2

Análise do Problema

Este capítulo realiza uma análise do contexto que motiva o desenvolvimento da aplicação Court&Go, abordando os conceitos fundamentais que sustentam o domínio do problema, bem como as principais funcionalidades necessárias para responder às necessidades identificadas.

Abordam-se conceitos fundamentais e funcionalidades, estes tópicos fornecem o enquadramento necessário para compreender as decisões adotadas ao longo do desenvolvimento.

2.1 Conceitos Fundamentais

Os conceitos fundamentais da aplicação Court&Go centram-se na gestão de reservas de campos de padel e ténis, no perfil do jogador e na administração de infraestruturas por parte dos proprietários.

Reservas de Campos: A aplicação permite aos utilizadores consultar a disponibilidade de campos de ténis e padel, e efetuar reservas com base na data, localização e tipo de campo pretendido. O processo é simplificado, permitindo ao utilizador selecionar rapidamente um horário e confirmar a reserva.

Jogadores e Participações: Os jogadores são entidades centrais da aplicação. O utilizador pode criar e cancelar reservas, os seus dados desportivos e preferências são armazenados, incluindo informações sobre o seu estilo de jogo, estatísticas e histórico de resultados.

Gestão de Infraestruturas: Os proprietários de campos podem registar os seus espaços, definir preços por court e gerir os horários disponíveis. Isto garante que as infraestruturas são facilmente acessíveis por parte dos utilizadores e que os proprietários mantêm controlo total sobre as suas condições de aluguer.

Localização e Organização: A aplicação organiza os campos por localização, cidade e país, permitindo aos utilizadores uma navegação intuitiva. Esta estrutura facilita a procura de campos em zonas específicas e ajuda na expansão do serviço para diferentes regiões.

2.2 Funcionalidades Implementadas

Na temática **autenticação**, o utilizador pode:

- **Criar conta:** O processo de registo é disponibilizado no ecrã inicial da aplicação. O utilizador deve preencher os seus dados e confirmar através do botão de registo.
- **Entrar com uma conta já existente:** Também no ecrã inicial, o utilizador pode autenticar-se inserindo as suas credenciais no formulário de login.

- **Iniciar Sessão/Registar-se com conta Google:** O utilizador tem também possibilidade de entrar na aplicação com a sua conta google e tornar o processo mais rápido.
- **Terminar sessão:** Estando autenticado, o utilizador pode encerrar a sessão a partir do menu do seu perfil, clicando no botão de logout.

Relativamente à **reserva de campos**, um utilizador pode:

- **Pesquisar campos disponíveis:** Através de filtros como desporto (ténis ou padel), localização e horário, é possível procurar campos livres para reserva, também pode pesquisar pelo próprio nome do clube.
- **Efetuar uma reserva:** Ao selecionar um campo e um horário disponível, o utilizador pode proceder com a reserva, indicando a duração e confirmando o preço estimado.
- **Confirmação e cancelamento de uma reserva:** O criador da reserva pode aceder à mesma e confirmar ou cancelar, desde que se verifiquem as regras definidas (por exemplo, antecedência mínima).

Quanto à **gestão dos campos e clubes**, os proprietários podem, através de uma plataforma auxiliar:

- **Registar clubes e campos associados:** Proprietários registados têm acesso à gestão dos seus clubes e podem adicionar campos, especificando tipo, piso e capacidade.
- **Definir horários semanais:** Cada campo pode ter um horário de funcionamento semanal com dias e horas de abertura e fecho.
- **Criar exceções em dias específicos:** É possível adicionar horários especiais, para datas como feriados, com indicação se o campo estará disponível ou não e se sim qual será o horário desse dia.

No **perfil do jogador**, o utilizador pode:

- **Atualizar dados pessoais:** O utilizador pode alterar o seu nome, contacto, país, peso, altura, estilo de jogo, entre outros.
- **Gestão das notificações:** O utilizador pode definir o tipo de notificações que pretende receber por parte da aplicação.

Capítulo 3

Descrição da Solução

3.1 Requisitos Obrigatórios

Os requisitos obrigatórios definem o que o sistema deve suportar e como se deve comportar.

Os requisitos obrigatórios identificados para a aplicação Court&Go são os seguintes:

- **Reserva de campos**: Pesquisa de campos e reserva dos mesmos, com seleção de horários e preços e possibilidade cancelamento.
- **Perfil de Jogador**: Cada utilizador tem o seu perfil dedicado, onde pode consultar e alterar os seus dados e preencher dados extra.
- **Histórico de Reservas**: Consulta de histórico de reservas futuras e passadas e os seus detalhes.
- **Integração com Aplicações de Calendário**: Sincronização automática das reservas com Google Calendar, Outlook e Apple Calendar.
- **Sistema de notificações por push, email ou SMS**: Lembretes de reservas por push, email ou SMS.

3.2 Requisitos Opcionais

Os requisitos opcionais identificados para a aplicação Court&Go são os seguintes:

- **Ranking de Jogadores**: Sistema de pontuação baseado nos resultados dos utilizadores da aplicação.
- **Sistema de Medalhas Virtuais**: Sistema de conquistas de acordo com participação, desempenho dos jogadores e outras metas de progresso.
- **Gestão de Torneios**: Possibilidade de criar torneios de pade e ténis.
- **Histórico de torneios**: Caso haja implementação de torneios, possibilidade de visualizar torneios passados e outras estatísticas dos mesmos.

3.3 Arquitetura do Sistema

O sistema Court&Go implementa uma **arquitetura distribuída e multiplataforma** baseada em padrões modernos de desenvolvimento. A solução é composta por três componentes principais que comunicam através de APIs REST, seguindo princípios de **separação de responsabilidades** e **modularidade**.

O projeto adota uma arquitetura cliente-servidor com dois frontends que consomem um backend serverless centralizado:

- **Backend Serverless (AWS)**: API REST com funções Lambda;
- **Frontend Web Admin (React)**: Interface para gestão de clubes;
- **Frontend Mobile (Kotlin Multiplatform)**: Aplicação para utilizadores finais;

O backend da aplicação CourtAndGo está implementado utilizando uma arquitetura serverless baseada na cloud AWS, garantindo escalabilidade automática, alta disponibilidade e custos otimizados. A implementação utiliza o framework SST (Serverless Stack) para orquestração e deployment da infraestrutura.

A comunicação com o frontend é realizada através do AWS API Gateway V2, que fornece um endpoint único centralizado para todas as operações e roteamento automático para as funções Lambda correspondentes. O backend é composto por funções Lambda organizadas por domínio de negócio, seguindo uma arquitetura de microserviços. As funções Lambda por sua vez executam as queries que vão por fim comunicar com a base de dados e enviar ou receber dados dependendo da operação. Na figura 1 está representada a estrutura em camadas do Backend baseado numa arquitetura Serverless.

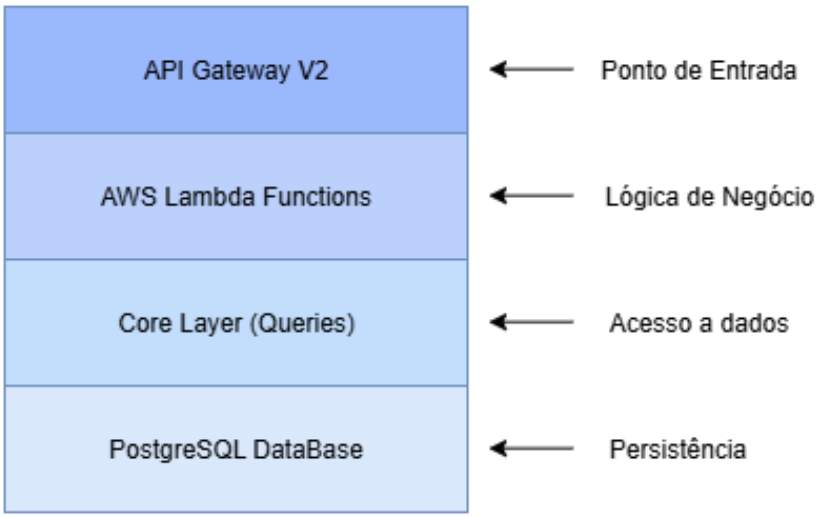

Figura 1: Estrutura em camadas da Arquitetura Backend

O Frontend Web Admin segue um padrão arquitetural simples, componentes de React funcionado como parte visual da aplicação, Context a camada onde se guarda o estado global (autenticação e dados globais), camada API, onde são feitos os pedidos HTTP à API de backend, pages onde estão as paginas completas da aplicação e ficheiro principal contém todas as rotas do site web.

O Frontend Mobile, para jogadores, está desenvolvido numa arquitetura baseada três em camadas. Como podemos observar na figura 2, a **Camada de Domínio** que contém

todas as entidades de negócio e define as regras fundamentais do sistema. A **Camada de Dados** que utiliza repositórios para abstrair as fontes dos dados, implementando interfaces como AuthRepository, ClubRepository entre outras, que podem ser concretizadas através de serviços HTTP reais ou implementações Mock para desenvolvimento e testes. Por fim, a **Camada de Apresentação** contém os ViewModels que gerem os estados de cada um dos ecrãs da aplicação comunicando com a UI declarativa construída em Jetpack Compose.

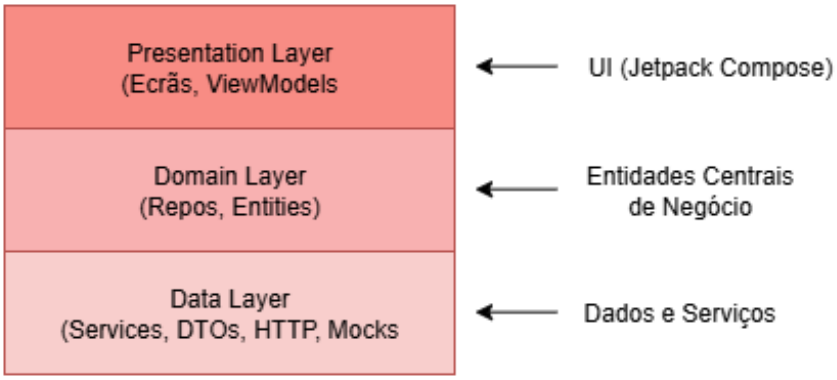

Figura 2: Estrutura em camadas da Arquitetura Frontend Mobile

3.4 Modelo de Dados

O modelo de dados tem como entidades centrais **Jogador** e **Clube**. A entidade Jogador está composta pelas informações gerais sobre o utilizador como email, peso, altura, género, entre outras, sendo a chave primária o id do jogador. A entidade Clube está ligada aos donos dos clubes e aos courts que estão ligados às reservas, ou seja, entidade central do processo de marcação da reserva.

Como especificidades da base dados é de realçar a distinção entre os tipos de horários, existe uma divisão entre Horário Semanal e Horário Especial.

A tabela **Horário Semanal** espera receber sete entradas para cada um dos dias da semana com o respetivo horário. Os atributos são código do court associado, dia da semana, hora de abertura e hora de fecho.

A tabela **Horário Especial** está idealizada para receber as datas mais especificas como feriados ou dias em únicos em que o clube estará fechado, é esperado o dono do clube introduzir cada uma dessas datas manualmente. Esta entidade tem como atributos o court associado, a data especifica (dd-mm-aaaa), a hora de abertura, hora de fecho e um atributo chamado working (do tipo Boolean) caso o dono pretenda especificar que o clube apenas vai estar fechado nesse dia.

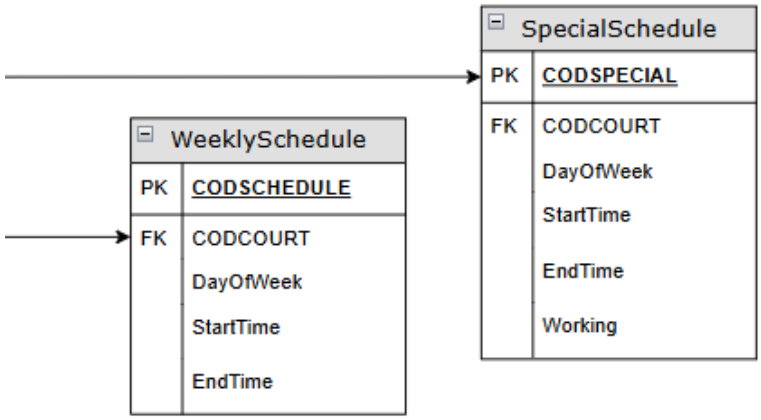

Figura 3: Tabelas das Entidades Horário Semanal e Horário Especial

O modelo de dados completo encontra-se no Anexo 1.

3.5 Tecnologias Utilizadas

A implementação da plataforma divide-se em duas grandes componentes: *Backend* e *Frontend*. Cada uma recorre a tecnologias específicas, nesta subsecção são abordadas as várias tecnologias utilizadas para a implementação de ambas as partes. São também enunciadas as várias razões que levaram à escolha das mesmas.

3.5.1 *Backend*

A componente de "servidor"da aplicação Court&Go está desenvolvida com recurso a uma arquitetura totalmente **serverless**, utilizando os serviços cloud da **Amazon Web Services (AWS)** [7]. A lógica de negócio está implementada inteiramente em **TypeScript**, com as funcionalidades distribuídas por **funções Lambda**, permitindo uma

escalabilidade automática, gestão simplificada de recursos e faturação baseada apenas no consumo efetivo.

A infraestrutura está definida com recurso à framework **SST (Serverless Stack) v3**, que fornece uma camada de abstração sobre o **AWS CDK (Cloud Development Kit)**. Esta framework permite gerir a infraestrutura como código (IaC), facilitando a criação, manutenção e atualização de recursos AWS, como:

- **AWS Lambda** – para a execução das funções backend que tratam da lógica da aplicação (autenticação, reservas, campos, utilizadores, etc.).
- **Amazon API Gateway V2** – para expor as funções Lambda através de endpoints HTTP acessíveis pela aplicação móvel e pelo website dos proprietários.
- **Amazon RDS (PostgreSQL)** – para hospedagem da base de dados.
- **Amazon Cognito** – para a gestão da autenticação e do ciclo de vida dos utilizadores.
- **AWS IAM** – para a gestão de permissões e políticas de segurança entre os serviços.
- **Amazon VPC** – serviço AWS que permite criar uma rede virtual privada dentro da infraestrutura da Amazon para correr a base de dados.
- **AWS SDK (Software Development Kit)** – biblioteca oficial que permite interação com os serviços AWS através de funções Lambda em Node.js.

A adoção deste stack tecnológico resulta num sistema modular e escalável, com custos operacionais reduzidos e um processo de desenvolvimento e *deploy* ágil, beneficiando da forte integração com os serviços da AWS e do apoio proporcionado por uma documentação extensa e bem estruturada.

3.5.2 *Frontend* da Aplicação Móvel

No desenvolvimento da aplicação móvel recorre-se a tecnologias em que o foco seja suporte para multiplataforma, de modo a abranger o máximo de dispositivos possíveis. A linguagem principal utilizada é o *Kotlin Multiplatform* [1], que possibilita a partilha de lógica comum entre as plataformas Android e iOS, promovendo a reutilização de código e uma maior eficiência no desenvolvimento.

No frontend , adota-se o *Compose Multiplatform*, um framework declarativo para interfaces gráficas que permite a criação de interfaces reativas com base no paradigma *Jetpack Compose*. Por opção a interface visual segue as diretrizes do *Material Design 3*, de modo a tornar a experiência de utilização alinhada com os padrões modernos.

A autenticação com conta google é iniciada no cliente utilizando os componentes da *Google Sign-In* fornecidos pela biblioteca *KMPAuth* [8]. Após a autenticação com sucesso, o token de identidade é transmitido ao *Amazon Cognito* (serviço da AWS), que valida o utilizador e gera os tokens de acesso utilizados na comunicação com o backend, o mesmo procedimento é seguido para o register/login tradicional.

3.5.3 *Frontend* do Site para Donos de Clubes

O site web para donos de clubes, está desenvolvido de uma bastante simples, é utilizado de TypeScript e React, para a navegação entre páginas React Router e também foram usados maioritariamente componentes do Material-UI(MUI).

3.5.4 *Base de Dados*

A base de dados está desenvolvida em PostgreSQL.

Capítulo 4

Funcionalidades

4.1 Autenticação

4.1.1 Registo Normal

A funcionalidade de registo permite a criação de contas de utilizador na plataforma Court&Go, possibilitando o acesso a funcionalidades da aplicação como marcação de campos, visualização de reservas, entre outras.

O sistema de autenticação está implementado com base em AWS Cognito, um serviço gerido de identidade e autenticação fornecido pela Amazon Web Services. Esta escolha permitiu garantir uma integração segura, escalável e compatível com práticas modernas de autenticação, reduzindo a complexidade de gestão de credenciais sensíveis no lado da aplicação.

Durante o processo de registo, os utilizadores introduzem dados básicos como: nome, email, contacto telefónico, código telefónico do país e palavra-passe. Estes dados são enviados para a User Pool do Cognito. A AWS valida, guarda os dados com segurança e gere todo o processo de autenticação.

4.1.2 Registo/Login com Google

A plataforma oferece a funcionalidade de registo e login através de uma conta Google, facilitando o acesso do utilizador ao sistema sem necessidade de criar manualmente um novo conjunto de credenciais. Esta funcionalidade está implementada recorrendo ao mecanismo de (login federado) disponibilizado pelo AWS Cognito.

Ao selecionar a opção "Registo/Login com Google", o utilizador é redirecionado para o sistema de autenticação da Google, onde autoriza a aplicação a aceder a informações básicas da sua conta, como nome e email. Após a autorização, o AWS Cognito recebe um token de identidade da Google e realiza automaticamente a integração com a sua User Pool.

O Cognito trata de forma transparente a distinção entre registo e login:

- Se for a primeira vez que o utilizador acede à plataforma com a conta Google, o Cognito considera que se trata de um registo e cria automaticamente um novo utilizador na User Pool com os dados recebidos da Google.
- Se o utilizador já tiver usado a conta Google anteriormente, o Cognito reconhece-o e realiza apenas o processo de login, permitindo o acesso sem duplicação de contas.

Este processo visa simplificar a experiência do utilizador visto que dispensa a criação manual de conta com email e palavra-passe.

A utilização do Cognito como ponto central de autenticação garante que, mesmo utilizando múltiplos provedores (como Google ou Apple), o sistema mantém uma gestão uniforme de utilizadores, facilitando a escalabilidade e manutenção da aplicação.

4.1.3 Login

O login permite que os utilizadores autenticados acedam à plataforma, com as credenciais habituais email e palavra-passe. O login está implementado com base no AWS Cognito, recorrendo à sua User Pool para autenticar os utilizadores com credenciais previamente registadas (email e palavra-passe) ou como referido acima através de provedores externos, como a Google. Esta abordagem permite combinar login tradicional com login federado.

4.1.4 Logout

O logout permite ao utilizador terminar a sua sessão na plataforma, garantindo que os seus dados e permissões de acesso deixam de estar ativos no dispositivo atual.

4.2 Clubes

4.2.1 Pesquisa de clube

A pesquisa de clubes permite aos utilizadores explorarem os vários clubes inseridos na plataforma pelos administradores dos mesmos. Com o objetivo de facilitar a pesquisa, esta está divida em duas abas, ténis e padel. Esta funcionalidade está integrada numa barra de pesquisa que integra quatros filtros simultaneamente, permitindo uma pesquisa por diversos parâmetros sendo eles: nome do clube, distrito, concelho e código postal, dinamizando e acelerando o processo de pesquisa para o utilizador.

Após pesquisa são apresentados os clubes que correspondem aos filtros, os clubes são apresentados com diversas informações como: fotografia do clube, nome, preview de horários disponíveis, preço estimado e localização, resumindo um cartão de visita.

4.2.2 Detalhes do clube

Após a seleção de um clube na lista de pesquisa, a plataforma apresenta uma página dedicada com os detalhes completos do clube, permitindo ao utilizador analisar melhor as opções disponíveis antes de efetuar uma reserva.

Esta funcionalidade oferece uma visão detalhada das principais informações relacionadas com o clube, a página está organizada com a foto do clube, o nome, as modalidades do clube, o tipo de piso dos courts, a capacidade de jogadores que cada court pode suportar, a quantidade de campos do clube e a localização do mesmo.

Esta interface foi desenhada de forma a destacar a informação mais relevante de cada clube, isto permite ao utilizador avaliar mais facilmente se o clube escolhido tem o que procura.

4.3 Reservas

4.3.1 Horários Disponíveis

Uma das funcionalidades centrais desta plataforma é a apresentação dos horários disponíveis para reserva, permitindo aos utilizadores consultar, de forma intuitiva e em tempo real, os períodos livres para marcar um campo do clube escolhido.

O sistema de horários foi concebido para mostrar todas as horas possíveis para reserva num determinado dia no clube escolhido. Neste ecrã existe também possibilidade de filtrar os horários para mostrar as horas não disponíveis.

4.3.2 Escolha de detalhes da Reserva

Após escolha de um horário disponível, vem o ecrã seguinte, de escolha dos detalhes da reserva, onde se personaliza os parâmetros da marcação antes de ser terminada. Esta etapa garante que a reserva é feita de acordo com as preferências do jogador.

Nesta funcionalidade o utilizador tem a possibilidade de escolher o court onde quer jogar dentro dos disponíveis no clube e escolhe a duração da reserva. Tem informação do horário que termina a reserva e do preço estimado. Existe atualizações automáticas conforme as escolhas que vão sendo feitas por parte do utilizador.

Após o término deste passa a estar associada ao utilizador e entra na lista de reservas futuras do mesmo.

4.3.3 Recibo de Reserva

O recibo de reserva é um resumo da reserva após a finalização da mesma, mostra todas as informações relevantes da reserva que acaba de ser feita, como horários de início e fim da reserva. E oferece naquele momento a possibilidade ao utilizador de adicionar a reserva ao seu calendário Google ou Apple.

4.3.4 Reservas Futuras

A plataforma permite aos utilizadores autenticados acederem à lista de reservas futuras que tenham efetuado, e possibilidade de visualizar os detalhes de cada reserva. Este ecrã ainda tem um aviso visual, 24 horas antes da hora da reserva, para que o utilizador confirme a sua reserva.

4.3.5 Últimas Reservas

À semelhança da funcionalidade anterior esta também apresentar uma lista, mas de reservas passadas para permitir ao utilizador saber os clubes que frequentou e possivelmente reservar de novo, ou até consultar os detalhes das reservas anteriores.

4.3.6 Confirmação ou Cancelamento de Reservas

Esta funcionalidade permite ao utilizador confirmar ou cancelar as suas reservas diretamente através da aplicação. Esta funcionalidade é essencial para garantir a fiabilidade da gestão de campos e assegurar que os recursos são utilizados de forma eficiente.

Após a submissão de uma reserva, o sistema atribui um estado pendente à reserva. O jogador recebe um aviso 24 horas antes do início da sua partida para relembrar para confirmar a mesma e tem até 1 hora antes do início da partida para a poder cancelar.

A confirmação tem como objetivo: reforçar o compromisso do utilizador com a reserva e prevenir faltas sem aviso.

O cancelamento é tratado como uma mudança de estado da reserva e permite atualizar a disponibilidade do clube no horário correspondente de forma automática após cancelamento.

4.4 Perfil de Jogador

4.4.1 Edição de Perfil

No perfil de Jogador, é onde o utilizador pode encontrar todas as suas informações e pode também editar as mesmas. O utilizador pode alterar o seu nome, data de nascimento, peso e altura, entre outras.

4.5 Notificações

A plataforma Court&Go emite vários tipos de notificações sendo elas, notificações de push e email. As notificações são enviadas automaticamente 24 horas antes do horário da reserva, servindo como lembrete para o utilizador confirmar a sua reserva. As notificações por email podem ser ativadas ou desativadas no ecrã implementado com essa finalidade. As notificações via push, tem de ser desativadas nas definições do telemóvel em questão.

4.6 Funcionalidades de Donos de Clubes

Os donos dos clubes têm acesso a uma página web onde devem criar uma conta, e identificar-se. Após isso podem registar o seu clube e courts e os respetivos horários.

4.6.1 Registo e Login

O sistema de autenticação implementa um processo de registo e autenticação baseado em AWS Cognito. São requisitados o email, nome, contacto telefónico e palavra-passe. A submissão deste formulário contem os dados necessários para executar o comando SignUpCommand do AWS Cognito e criar o utilizador na User Pool.

O processo de login executa o comando InitiateAuthCommand com email e password preenchidos pelo utilizador. O estado de autenticação é mantido através do AuthContext que utiliza React Context API para partilha global do estado, persistindo o ID do proprietário.

4.6.2 Logout

A funcionalidade de logout executa a limpeza completa do estado de autenticação. Remove o id do utilizador e redireciona o utilizador para a página de login.

4.6.3 Criar Clubes e Adicionar Courts

A criação de clubes está implementada em duas etapas. Primeiro, é criada uma localização requerendo ao utilizador os dados de morada, concelho, distrito e código postal. Posteriormente, o clube é criado associando-o à localização previamente criada e ao proprietário autenticado, também com as devidas informações sobre o clube, preenchidas pelo dono.

A adição de courts, apesar de opcional nesta fase, permite a criação múltipla de courts numa única operação. O formulário captura informações como nome, tipo de desporto, tipo de superfície, capacidade e preço por hora. Cada court é criado individualmente através de chamadas sequenciais.

4.6.4 Lista de Clubes

Esta funcionalidade apresenta a lista dos clubes possuídas pelo dono autenticado na plataforma, esta interface apresenta cada clube numa lista estruturada com botões de ação que permitem navegação direta para funcionalidades específicas como edição de clube, gestão de courts e configuração de horários.

4.6.5 Editar Clubes

A edição de clubes permite modificação tanto das informações do clube quanto da sua localização. O processo de atualização executa duas operações: primeiro atualiza a localização e posteriormente atualiza os dados do clube. O formulário pré-popula todos os campos com os valores atuais.

4.6.6 Editar Courts

A funcionalidade de edição de courts permite modificação das características individuais de cada court. O sistema carrega os courts existentes e apresenta formulários individuais para cada court. Podem ser alterados dados como nome, tipo, superfície, capacidade e preço por hora.

4.6.7 Adicionar Horários Semanais e Especiais

O componente de horários semanais gere configurações para cada dia da semana, permitindo definição de horas de abertura e fecho ou marcar dias como fechados.

O componente de horários especiais permite definir exceções para datas específicas, sobrepondo o horário normal. Este sistema suporta tanto horários alternativos quanto marcação de dias completamente fechados.

Capítulo 5

Implementação da API (Backend)

A API do Court&Go é implementada utilizando a framework SST v3 (Serverless Stack), permitindo a definição da infraestrutura como código e o deploy automático de recursos serverless na AWS. A arquitetura segue o modelo backend desacoplado, com rotas RESTful expostas via API Gateway e funções Lambda, conectadas a uma base de dados relacional PostgreSQL.

A escolha da AWS como plataforma cloud baseia-se na sua maturidade tecnológica, cobertura global e ecossistema robusto de serviços especializados para aplicações web modernas.

5.1 SST (Serverless Stack) v3

O SST (Serverless Stack) v3 é uma framework moderna de infraestrutura como código (Infrastructure as Code - IaC) especificamente projetada para desenvolvimento de aplicações serverless na AWS. Esta framework revoluciona a forma como os desenvolvedores constroem, testam e implementam aplicações cloud-native, oferecendo uma experiência de desenvolvimento otimizada com type safety completo e eliminando a complexidade tradicional associada à gestão de infraestrutura cloud[10].

O SST v3 representa uma reescrita completa da framework, abandonando a arquitetura baseada em AWS CDK das versões anteriores em favor de uma engine própria construída sobre Pulumi. Esta mudança fundamental permite maior performance, flexibilidade e uma experiência de desenvolvimento superior, especialmente no que toca ao desenvolvimento local integrado com recursos cloud reais. A framework implementa uma abordagem declarativa onde toda a infraestrutura é definida através de código TypeScript escrito no ficheiro central `sst.config.ts`, que atua como "blueprint"da aplicação, definindo desde recursos de base de dados até APIs e deployments de frontend de forma unificada.

A funcionalidade mais inovadora do SST v3 é o Live Lambda Development, que transforma radicalmente a experiência de desenvolvimento serverless. Tradicionalmente, o desenvolvimento de funções Lambda exigia ciclos lentos de deploy e teste, criando fricção significativa no processo de desenvolvimento. O SST v3 resolve este problema permitindo desenvolvimento local com ligação direta às funções Lambda em execução na AWS. Durante o comando `sst dev`, as funções executam na cloud mas o código fonte permanece local, criando um híbrido único que combina o melhor dos dois mundos. As alterações ao código são refletidas instantaneamente através de hot-reload, permitindo debugging nativo com breakpoints e logs em tempo real no terminal local, eliminando a necessidade de redeploys constantes e proporcionando acesso direto aos recursos AWS reais.

O SST v3 implementa type safety completo através de TypeScript, gerando automaticamente tipos para todos os recursos AWS criados. Esta funcionalidade elimina uma categoria inteira de erros comuns em infraestrutura como código, onde referências incorretas entre recursos podem causar falhas em tempo de execução. No projeto Court&Go, isto significa que referências como `database.host` ou `api.url` são validadas em tempo de compilação, garantindo consistência arquitetural. O sistema de resource linking complementa esta type safety permitindo que funções Lambda acedam automaticamente a outros recursos AWS sem configuração manual. Por exemplo, a linha `link: [database]` no API Gateway injeta automaticamente todas as configurações de conectividade da base de dados nas funções Lambda correspondentes, incluindo host, porta, credenciais e permissões IAM necessárias.

A framework fornece constructs de alto nível que encapsulam melhores práticas para casos de uso comuns. O construct `sst.aws.Postgres` não apenas cria uma instância PostgreSQL, mas também configura automaticamente VPC, security groups, subnets e otimizações de conectividade para ambientes serverless. Similarmente, `sst.aws.ApiGatewayV2` estabelece não só o API Gateway, mas também todo o roteamento declarativo e linking automático com recursos dependentes. Esta abstração permite que desenvolvedores foquem na lógica de negócio em vez de detalhes de configuração de infraestrutura, mantendo transparência total e permitindo acesso aos recursos AWS subjacentes quando necessário.

Ao contrário das versões anteriores que utilizavam AWS CDK, o SST v3 é construído sobre o Pulumi, uma decisão arquitetural que proporciona benefícios significativos em performance e flexibilidade. O Pulumi oferece uma engine otimizada para deployments mais rápidos e gestão de estado mais eficiente que CloudFormation, resultando em tempos de deploy reduzidos e melhor experiência de desenvolvimento.

O SST v3 implementa gestão inteligente de ambientes através do conceito de stages, permitindo configurações diferentes para desenvolvimento e produção. No projeto Court&Go, a configuração utiliza `removal: "retain"` para ambientes de produção, garantindo que recursos críticos não sejam removidos acidentalmente, enquanto ambientes de desenvolvimento podem ser limpos automaticamente. O workflow de desenvolvimento segue um padrão otimizado onde `sst dev` cria um ambiente de desenvolvimento completo na AWS, `sst build` valida toda a configuração, `sst deploy` implementa a aplicação com otimizações de produção, e `sst console` fornece uma interface web para monitorização e debugging.

No contexto específico do Court&Go, o SST v3 transforma fundamentalmente a produtividade de desenvolvimento. O Live Lambda Development permite iterações rápidas na implementação de funcionalidades como gestão de reservas, autenticação de utilizadores e processamento de dados de clubes e campos, eliminando os ciclos lentos típicos do desenvolvimento serverless tradicional. A infraestrutura consistente garantida pela definição declarativa assegura que ambientes de desenvolvimento, staging e produção sejam idênticos, reduzindo significativamente bugs relacionados com diferenças ambientais. O deployment simplificado através de um único comando acelera releases e reduz a possibilidade de erros humanos durante o processo de implementação. A escolha do SST v3 para o projeto Court&Go baseia-se na sua capacidade única de abstrair a complexidade da AWS mantendo controlo total sobre a infraestrutura.

5.2 *Serverless*

Serverless é um paradigma de computação em cloud onde o provedor (AWS) gere automaticamente a infraestrutura de servidores, permitindo que os desenvolvedores foquem exclusivamente na lógica de negócio. Apesar do nome "serverless", os servidores ainda

existem, mas são completamente abstraídos e geridos pelo provedor cloud.

Na prática, o backend tem várias funções lambda como microserviços, ou seja, cada funcionalidade é implementada como uma função lambda independente. Cada função lambda é executada apenas quando é invocada, escala automaticamente e isola as falhas. As funções são desencadeadas através dos pedidos HTTP (via API Gateway). No backend o ponto de entrada único é a API Gateway V2 em que este tem a configuração de todas as rotas HTTP no SST, recebe também todas as requisições HTTP dos clientes, roteia automaticamente para a função lambda correspondente, gere autenticação e transforma as requisições HTTP em eventos Lambda.

A figura abaixo explica visualmente como funciona o fluxo de execução Serverless, que se inicia quando um cliente faz um pedido HTTP, o API Gateway recebe esse pedido, valida-o e transforma-o num evento Lambda, a respetiva função lambda é instanciada e AWS cria um container temporário caso seja necessário, a função é executada processando a lógica de negócio. Por fim, retorna-se então a resposta via API Gateway ao cliente e após um período de inatividade o container é destruído.

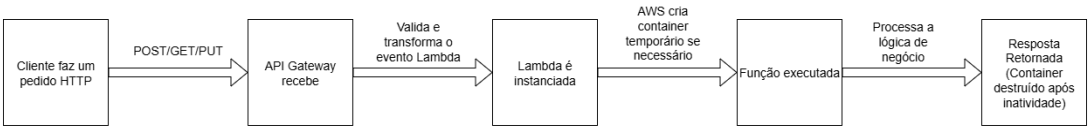

Figura 4: Fluxo de Execução Serverless

As principais vantagens de utilizar uma abordagem Serverless são:

- Auto escalamento infinito (por exemplo: num cenário de tráfego baixo, 10 requests por minuto existem 1 a 2 lambdas ativas, mas se num pico de tráfego em que haja 10.000 pedidos por minutos haverão mais de 1000 lambdas automaticamente);
- Infraestrutura com bom custo por utilização;
- Baixa manutenção, não é necessário configurar, atualizar ou monitorar servidores;
- Desenvolvimento Mais Rápido, foco na lógica de negócio, sem preocupações com infraestrutura;
- Alta Disponibilidade garantida pelo fornecedor da cloud, sem configurações extra.

O uso de serverless permite maior escalabilidade, menor custo operacional e um desenvolvimento focado na lógica de negócio, sem a complexidade de gestão de servidores.

5.3 Serviços de AWS

A API (backend) do projeto CourtAndGo está implementada na infraestrutura cloud da Amazon Web Services (AWS), aproveitando um conjunto abrangente de serviços geridos que proporcionam escalabilidade, disponibilidade e segurança empresarial.

A API utiliza os seguintes serviços:

- **AWS Lambda** - Constitui o núcleo computacional da arquitetura serverless, fornecendo a capacidade de execução de código sem necessidade de provisionar ou gerir servidores. Cada endpoint da API corresponde a uma função Lambda independente, organizadas por domínio de negócio (UserService, ClubService, ReservationService, etc.). As funções executam em runtime Node.js com auto-scaling

automático de zero a milhares de execuções concorrentes, cobrando apenas pelos milissegundos de computação efetivamente utilizados. Esta abordagem garante isolamento de falhas entre funcionalidades, deployment independente de cada serviço e otimização de custos através do modelo pay-per-execution.

- **Amazon API Gateway V2** - Atua como proxy reverso e ponto de entrada único para todas as requisições HTTP da aplicação. Este serviço gere automaticamente o roteamento de requests para as funções Lambda correspondentes, implementa CORS para suporte de pedidos de outros domínios, e fornece capacidades de throttling e rate limiting.

- **Amazon Cognito** - Implementa o sistema de gestão de identidades e autenticação da aplicação, suportando múltiplos fluxos de autenticação incluindo email/password e Autenticação com Google. O serviço gere automaticamente o ciclo de vida completo de utilizadores: registo, verificação de utilizador, login, logout e renovação de tokens. A implementação utiliza JWT tokens (Access, ID, Refresh). O Cognito integra-se nativamente com as funções Lambda através do AWS SDK, permitindo operações como SignUp, AdminConfirmSignUp e InitiateAuth. Esta centralização da autenticação garante consistência de segurança em toda a aplicação e com as melhores práticas de gestão de identidades.

- **Amazon RDS (PostgreSQL)** - Fornece uma instância PostgreSQL totalmente gerida para persistência de dados relacionais da aplicação. O serviço elimina tarefas administrativas como patching, backup, monitoring e scaling, mantendo alta disponibilidade. A implementação utiliza connection pooling via biblioteca node-postgres para otimizar a reutilização de conexões entre invocações Lambda. O RDS está isolado numa VPC privada, garantindo que a base de dados seja inacessível diretamente da internet e apenas acessível através das funções Lambda autorizadas.

- **Amazon VPC** - Cria um ambiente de rede isolado e seguro para recursos sensíveis, particularmente a instância RDS PostgreSQL. A VPC implementa uma arquitetura de rede com subnets públicas e privadas, onde a base de dados reside exclusivamente em subnets privadas inacessíveis da internet. Esta configuração garante que dados sensíveis permaneçam protegidos por múltiplas camadas de isolamento de rede, satisfazendo requisitos de segurança empresarial e compliance.

- **AWS IAM** - O AWS Identity and Access Management (IAM) estabelece o modelo de segurança e controlo de acesso entre todos os serviços AWS utilizados. Através do SST framework, são criadas automaticamente roles e políticas que seguem o princípio de menor privilégio, garantindo que cada serviço acede apenas aos recursos estritamente necessários. As funções Lambda recebem roles específicas para acesso ao CloudWatch Logs, RDS, e Cognito, enquanto o API Gateway possui permissões para invocar as funções correspondentes. O IAM elimina a necessidade de credenciais hardcoded ou gestão manual de permissões, implementando um modelo de segurança onde cada interação entre serviços é explicitamente autorizada.

- **Amazon CloudFront** - Implementa uma Content Delivery Network (CDN) global para distribuição eficiente da aplicação web *frontend-admin* destinada aos proprietários de clubes. A integração com o SST permite deployment automático da aplicação React, fazendo build da mesma e criando automaticamente distribuições otimizadas e configurando domínios personalizados quando necessário.

- **AWS SDK** - O AWS SDK (Software Development Kit) constitui a biblioteca que permite interação programática com os serviços AWS a partir das funções Lambda Node.js. Na API Backend, o SDK é utilizado especificamente através do pacote *@aws-sdk/client-cognito-identity-provider* para gestão completa do ciclo de vida de autenticação de utilizadores. O SDK v3 implementa uma arquitetura modular onde cada serviço AWS é distribuído como um pacote independente, reduzindo significativamente o tamanho do bundle final e melhorando performance das funções Lambda através de cold start otimizado.

5.4 Configuração do Servidor

A configuração do "servidor"API backend é implementada através do Serverless Stack v3 (SST) framework, utilizando uma abordagem declarativa de Infrastructure as Code (IaC). O ficheiro sst.config.ts, presente no Anexo 2, define toda a infraestrutura AWS necessária de forma programática, garantindo deployments reproduzíveis e compatibilidade com a versão da infraestrutura. Sendo este ficheiro o ponto crucial da correta ativação do servidor.

Capítulo 6

Implementação de Site Web para Donos de Clubes

O site web para donos de clubes, está desenvolvido em React com TypeScript, especificamente concebido para os proprietários de clubes gerirem as suas instalações e horários das mesmas.

Para iniciar a sua jornada na aplicação o dono de clube, deve registar-se ou fazer login com os dados habituais, e em seguida pode então iniciar o seu processo de criação de clubes e manutenção de informações à cerca dos mesmos.

Na figura 5, encontra-se representado o diagrama da jornada de um utilizador dono de clubes ao longo desta aplicação.

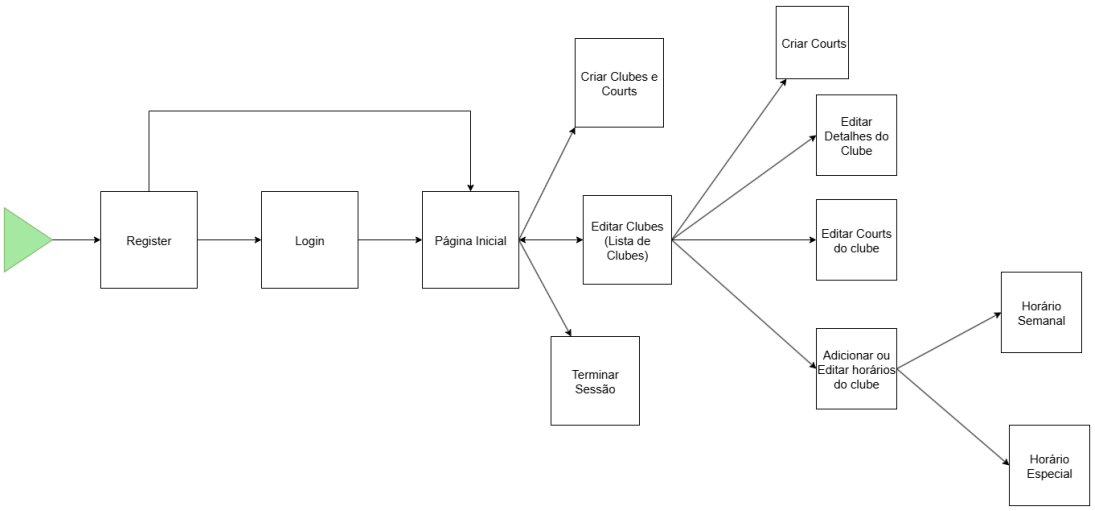

Figura 5: Diagrama de navegação do site Web de Donos de clubes na plataforma Court&Go

6.1 Organização

A aplicação organiza-se numa hierarquia clara de componentes funcionais, divididos entre **páginas** (componentes de alto nível representando ecrãs completos), **componentes reutilizáveis** (formulários especializados e elementos de interface) e uma diretoria chamada **api**, reservada para os pedidos HTTP ao backend serverless.

6.2 *Build e Deployment*

A aplicação utiliza Create React App para build com otimizações automáticas (code splitting, minificação), sendo o deploy via SST para Amazon CloudFront garantindo distribuição global. O processo integra-se no pipeline de Infrastructure as Code, sincronizando versões entre frontend e backend durante releases.

Capítulo 7

Implementação da Aplicação Móvel

A aplicação móvel é constituída por quatro conjuntos principais de ecrãs sendo estes: autenticação, pesquisa de clube, reservas e perfil de jogador.

O utilizador começa a sua jornada, no ecrã de registo, podendo ir para o ecrã de Login caso já tenha conta criada. Após o processo de autenticação todas as funcionalidades da aplicação ficam disponíveis no Home Screen (página inicial).

O diagrama de navegação apresentado na figura abaixo, representa em detalhe a jornada do utilizador nos diversos ecrãs da aplicação.

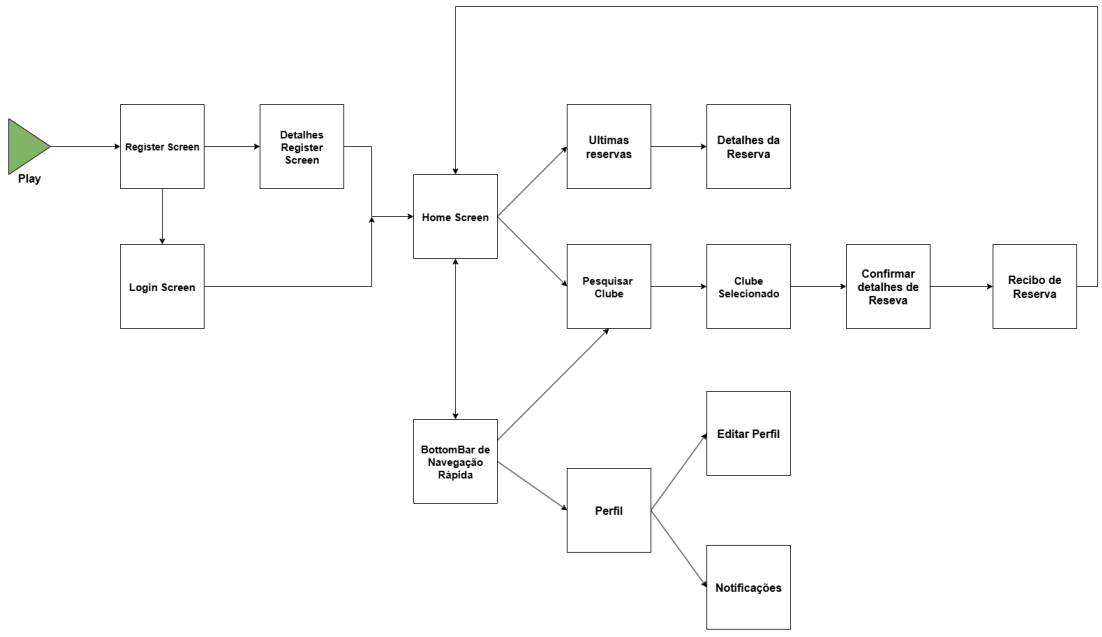

Figura 6: Diagrama de navegação da aplicação móvel Court&Go

7.1 Organização

A implementação da aplicação móvel Court&Go está desenvolvida com recurso a *Kotlin Multiplatform*, permitindo a partilha de lógica de negócio entre plataformas. A organização do projeto está pensada de forma modular, com uma separação clara entre a interface gráfica (UI), a lógica de apresentação (ViewModels), os serviços de comunicação

com o backend e os modelos de dados. O projeto encontra-se dividido nas seguintes diretorias principais:

- **commonMain**: Diretoria onde se encontra a maior parte da lógica de negócio partilhada, incluindo os modelos de dados, serviços, *ViewModels*, estados de UI e lógica de reserva, autenticação e pesquisa de clube. Esta camada é reutilizada por todas as plataformas suportadas.
- **androidMain**: Contém o código específico da plataforma Android. Inclui, por exemplo, a implementação de interações com o sistema operativo Android, como o acesso ao calendário, permissões nativas, recursos do sistema e emissão de notificações. Esta camada é onde se cria o HttpClient para Android, que irá comunicar com o backend.
- **iosMain**: Reservado para implementações específicas para iOS, tem implementação para emissão e pedido de permissão de notificações em iOS. Aqui à semelhança da diretoria anterior também se cria o HttpClient mas para clientes iOS.
- **commonTest**: Inclui os testes automáticos para o código partilhado em `commonMain`, testes unitários e de fluxo de dados.
- **androidTest**: Contém os testes instrumentados de interface específicos para Android, implementados com Jetpack Compose UI Test, testes de componente visual e de navegação.

7.2 Contexto de autenticação

A autenticação na aplicação móvel é implementada através de uma arquitetura híbrida. Para autenticação por email/password, a aplicação comunica com endpoints da API que internamente utilizam AWS Cognito para validação de credenciais e geração de tokens. A aplicação móvel não integra diretamente com o AWS Cognito, delegando a gestão de utilizadores e autenticação ao backend serverless. Os tokens de acesso gerados pelo Cognito são retornados pela API e geridos localmente através do `TokenHolder`.

Para implementar a autenticação com Google na aplicação Court&Go, foi utilizada a biblioteca `KMPAuth` [8], uma solução multiplataforma desenhada especificamente para aplicações desenvolvidas com *Kotlin Multiplatform* [1].

A `KMPAuth` [8] fornece componentes pré-construídos compatíveis com Jetpack Compose, para autenticação integrada com provedores como Google, Apple ou GitHub com uma única base de código. Esta abordagem permite reutilizar a lógica de autenticação entre plataformas (Android e iOS), simplificando significativamente o processo de desenvolvimento e manutenção da aplicação.

A figura 7 ilustra o fluxo de autenticação num computo geral. O processo inicia-se quando a aplicação móvel envia credenciais de utilizador para o Amazon API Gateway.

O API Gateway roteia automaticamente a requisição de autenticação para o Amazon Cognito, serviço gerido de gestão de identidades que valida as credenciais fornecidas. Após validação bem-sucedida, o Cognito aciona a função Lambda correspondente, que processa a lógica de negócio e interage com a base de dados para operações relacionadas com o utilizador.

O fluxo completa-se com o retorno de uma resposta da API contendo tokens JWT (Access Token, ID Token, Refresh Token) que são enviados de volta à aplicação móvel. Estes tokens garantem autenticação segura em requisições subsequentes, implementando um modelo stateless onde cada requisição é independente e auto-contida.

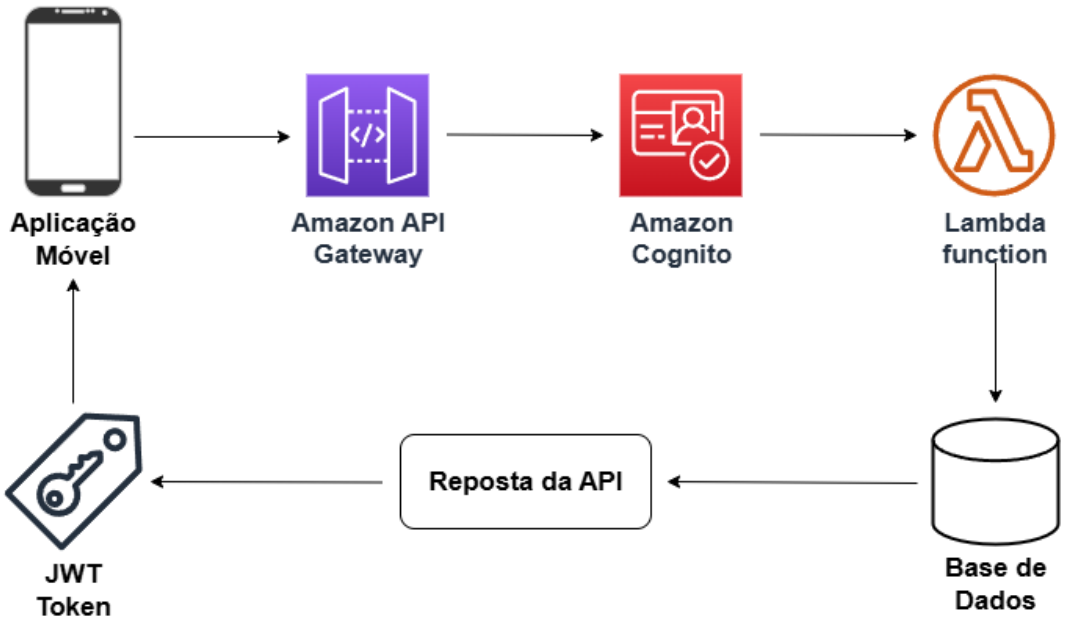

Figura 7: Fluxo de Autenticação

7.3 Interação com a API

A comunicação da aplicação com o backend foi implementada recorrendo à biblioteca `Ktor Client`, compatível com Kotlin Multiplatform. Este cliente HTTP foi utilizado para efetuar chamadas a uma API REST desenvolvida para suportar as funcionalidades da plataforma Court&Go, como autenticação, gestão de clubes, reservas e horários disponíveis.

A lógica de comunicação com a API foi centralizada na camada `commonMain`, onde foram definidos os serviços responsáveis por enviar pedidos HTTP e tratar as respetivas respostas. Esta abordagem permitiu reutilizar o mesmo código de acesso à API tanto na aplicação Android como numa futura versão iOS.

Cada serviço da aplicação (como `ClubService`, `ReservationService`, `UserService`, entre outros) encapsula as operações associadas a um conjunto de endpoints. Estes serviços utilizam o `HttpClient`, cuja instância é criada separadamente nas plataformas `androidMain` e `iosMain`.

São feitas as chamadas http, os dados recebidos são convertidos em modelos de domínio definidos também em `commonMain`, garantindo assim a consistência da informação em toda a aplicação.

7.4 Deploy

A aplicação móvel Court&Go está desenvolvida com Kotlin Multiplatform, permitindo partilhar a lógica de negócio entre plataformas. Contudo, para efeitos de execução e testes, o foco principal da fase de desenvolvimento centrou-se na plataforma Android, mas procurando sempre manter a compatibilidade para as duas plataformas.

O processo de "deploy"funciona apenas localmente no modo *dev*. Através da transferência de um ficheiro `.apk` instalável, que se encontra disponível no repositório do

projeto (`https://github.com/tiago15ts/court-go-app/releases`) e a ligação manual da API Serverless.

No caso da plataforma iOS, embora a base de código esteja preparada para compilação, o deploy público não foi realizado. A publicação nesta plataforma requer uma assinatura paga no programa de desenvolvedores da Apple [2] e o grupo não obteve o acesso de estudante. Então a única forma de utilização da aplicação em iOS é com recurso a um Mac com o programa Xcode instalado e com a utilização do emulador de iPhone do mesmo e também a ligação manual da API Serverless.

Com tudo, a aplicação encontra-se funcional e pronta para ser distribuída manualmente através de ficheiros `.apk` (Android).

Capítulo 8

Testes

Nesta secção descreve-se os testes utilizados para verificar a fiabilidade da aplicação, sendo este um passo crucial no desenvolvimento da plataforma.

8.1 Testes Manuais

Os testes manuais consistem na avaliação prática da aplicação por parte dos programadores, permitindo uma análise abrangente da interface e das funcionalidades do sistema do ponto de vista do utilizador. No desenvolvimento da plataforma Court&Go, foram adotadas as seguintes abordagens de testes manuais:

- **Mocks de serviços no frontend**: São utilizados mocks no frontend para substituir os serviços reais disponibilizados via API pelo backend. Esta abordagem permite testar os aspetos visuais da aplicação sem necessidade de realizar pedidos http à API. Através da simulação de diferentes cenários (como utilizadores com reservas futuras, ou clubes com horários distintos), torna possível avaliar a interface e a sua capacidade de adaptação de forma eficaz. Para testar o frontend para donos de clubes também foram utilizados mocks e conforme o avanço do projeto, passaram a testes em postman.
- **Postman para testes à API**: Durante o desenvolvimento do backend, foi utilizado o Postman para enviar pedidos HTTP para a API. Esta técnica permitiu validar o correto funcionamento das funcionalidades do implementadas em backend e garantir que a comunicação entre o frontend e o backend será estável e fiável.

8.2 Testes Automáticos

Durante o desenvolvimento da aplicação Court&Go, foram implementados vários testes para o Frontent, seguindo a estrutura habitual de Kotlin Multiplatform os testes também estão separados por código comum partilhado entre plataformas (*commonTest*) e testes específicos da plataforma Android (*androidTest*). Estes testes contribuíram para validar a lógica de negócio, garantir a integridade do estado da aplicação e verificar o correto funcionamento da interface gráfica.

Testes no módulo `commonTest`

Os testes incluídos no módulo `commonTest` focam-se na validação da lógica de negócio contida nos *ViewModels*, serviços e demais componentes partilhados entre as plataformas. Foram utilizadas as seguintes bibliotecas e abordagens:

Testes no módulo `androidTest`

Na plataforma Android, foram criados testes com o objetivo de validar a interface da aplicação construída com Jetpack Compose, garantindo que os ecrãs reagem corretamente às interações do utilizador e refletem adequadamente os estados da aplicação. Para esse fim, foram utilizadas as seguintes ferramentas:

- **Jetpack Compose UI Test**: Esta biblioteca permite a simulação de interações com a interface, incluindo ações como cliques em botões , introdução de texto, e verificação da presença de elementos visuais.
- **JUnit4**: Utilizado em conjunto com a anotação `@Rule` para instanciar a `ComposeTestRule`, essencial para a inicialização e teste de ecrãs em Compose.

Capítulo 9

Conclusões

O projeto desenvolvido tinha como objetivo criar uma aplicação móvel para reservas de campos de padel e ténis, um portal para administradores de clubes e uma API Serverless. Todos estes objetivos de implementações foram concluídos com sucesso, a única parte que o grupo não conseguiu concluir foi o deploy público do projeto dada a quantidade de problemas encontrados ao longo do processo de lançar a API Serverless e também o facto de que o projeto foi planeado para funcionar com o Free Tier da AWS [11] e o grupo acabou por ter uma despesa aproximada de 15$ o que também impediu um pouco o escalamento da aplicação na reta final para deploy público. Ainda assim perante todos os desafios o grupo considera o resultado atingido como positivo. No entanto a experiência com os serviços da AWS e desenvolvimento em SST [10] não foi a mais positiva visto que ainda há falta de alguma documentação para a última versão do SST [10], e existe uma mistura de informações entre as várias versões do SST [10].

9.1 Desafios

A exploração de novas tecnologias foi um dos principais desafios para o grupo, apesar de já existir alguma experiência com Kotlin Jetpack Compose, desta vez o grupo optou por utilizar Kotlin Multiplatform [1] de modo a permitir a escalabilidade da aplicação para as duas principais plataformas do mercado Android e iOS. Isto acabou por se tornar um grande desafio pois devido a ser ainda uma tecnologia recente há falta de algumas bibliotecas.

O IDE (Android Studio) [6] ainda não está completamente otimizado para o bom funcionamento da adição de novas bibliotecas, tornando o processo um pouco confuso inicialmente, pois o mesmo permite adicionar bibliotecas no espaço comum do projeto, mas por vezes as mesmas não eram compatíveis com Android e iOS simultaneamente. A estrutura do projeto também é bastante diferente do comum, tendo pastas direcionadas apenas para Android, outras para iOS e o espaço comum.

Explorar uma tecnologia nova como AWS [7] também foi um desafio, sendo praticamente tudo novo para o grupo. A escolha foi bastante ambiciosa, pois quis-se aprender uma tecnologia que é sabida como bastante influente no mercado atualmente. No entanto apareceram sempre vários desafios ao longo do projeto, mesmo com consulta de tutoriais e assistência de inteligência artificial o sistema da AWS [7] tem imensos menus que são atualizados constantemente e por vezes tornava-se complexo saber quais as opções selecionar para diversos serviços.

A configuração dos servidor SST foi a parte mais desafiante para o grupo. O facto de existir pouca documentação relacionada com SST, dificultou imenso o processo tal como a existência de três versões do mesmo, o que também não ajudava no uso de ferramentas de inteligência artificial pois a mesma baralhava as três versões gerando

versões inconsistentes. Depois de uma longa persistência o grupo conseguiu gerar uma versão estável do servidor mas apenas em modo dev, ou seja correr localmente o servidor. Toda este aglomerado de problemas impossibilitou o deploy público do projeto.

Inicialmente também tinha sido previsto haver login com Google e com conta Apple, mas o login com conta Apple depois de alguma pesquisa o grupo descobriu que o mesmo requeria uma subscrição no programa de desenvolvedores da Apple [2] com o custo de 99$ anualmente [3]. O grupo ainda tentou requerer a subscrição através do plano de estudantes, mas a mesma não foi concedida.

Isto prejudicou também no deploy da aplicação para dispositivos iOS, sendo também necessário a subscrição no programa de desenvolvedores na Apple [2] que iria permitir lançar a aplicação no TestFlight [4]. Como o acesso ao programa de desenvolvedores foi negado a única forma de testar a aplicação em iOS é ter um Mac com Xcode [5] instalado e executar a aplicação com emulador de iPhone que já vem integrado. Um dos membros do grupo tem um Mac e conseguiu testar a funcionalidade da aplicação, mas isto acabou por limitar muito a experiência em qualquer dispositivo iOS.

As notificações por SMS também não foi possível implementar pois todas as plataformas tinham custos, a principal plataforma seria o Twillio que tinha um free trial, mas depois de alguma pesquisa chegou-se à conclusão que o free trial [9] era muito limitado e não iria servir o propósito deixando de funcionar após alguns usos. O grupo ainda tentou implementar para notificações por Whatsapp mas deparou-se com o mesmo problema de subscrições pagas. No entanto ambas as implementações pareciam simples de se fazer, mas devido aos custos monetários das mesmas, foi abortada a implementação.

9.2 Trabalho Futuro

Apesar das funcionalidades já implementadas, existem diversas melhorias e extensões que poderão ser consideradas em trabalhos futuros. Tais como:

- A principal melhoria seria conseguir fazer o deploy público do projeto.
- Realização dos requisitos opcionais, que eram criação de torneios e histórico, sistema de ranking, sistema de medalhas por objetivos. Apesar da base de dados estar preparada para receber torneios o grupo não teve tempo para implementar este requisito.
- Aumentar o número de testes no cômputo geral do projeto.
- Aperfeiçoar visualmente a interface web dos donos de clubes.
- Aumentar a documentação do projeto e ter a mesma tanto em português como em inglês.
- Expandir a aplicação para funcionar em iOS, tentando de novo obter o acesso ao programa de Desenvolvedores da Apple [2].
- Adicionar filtro por data nas reservas passadas e futuras.
- Permitir convites entre jogadores para as partidas. A base de dados já está preparada para esta implementação tendo uma tabela Player_Reservation, que já serve para o propósito.

Uso de Inteligência Artificial

Este relatório foi escrito com auxílio de ferramentas com IA, nomeadamente o *ChatGPT* da *OpenAI*. Os autores reconhecem que estas ferramentas foram utilizadas para gerar excertos de texto e para refinamento da linguagem. Os autores assumem a total responsabilidade pelo conteúdo e pelas conclusões apresentadas neste artigo.

Referências

[1] Kotlin Multiplatform. https://www.jetbrains.com/help/kotlin-multiplatform-dev/get-started.html (Visitado a 20/06/2025)

[2] Apple Developer Program. https://developer.apple.com/programs/ (Visitado a 01/07/2025)

[3] Custo do Programa de Desenvolvedores Apple. https://developer.apple.com/help/account/membership/program-enrollment/ (Visitado a 01/07/2025)

[4] TestFlight. https://developer.apple.com/testflight/ (Visitado a 01/07/2025)

[5] Xcode. https://developer.apple.com/xcode/ (Visitado a 01/07/2025)

[6] Android Studio. https://developer.android.com/studio/intro?hl=pt-br (Visitado a 01/07/2025)

[7] AWS. https://aws.amazon.com/pt/ (Visitado a 01/07/2025)

[8] KMPAuth. https://github.com/mirzemehdi/KMPAuth (Visitado a 02/07/2025)

[9] Twilio Free Trial. https://www.twilio.com/docs/usage/tutorials/how-to-use-your-free-trial-account (Visitado a 09/07/2025)

[10] SST. https://sst.dev/docs/ (Visitado a 05/07/2025)

[11] AWS Free Tier. https://aws.amazon.com/pt/free/ (Visitado a 17/07/2025)

Anexos

Anexo 1

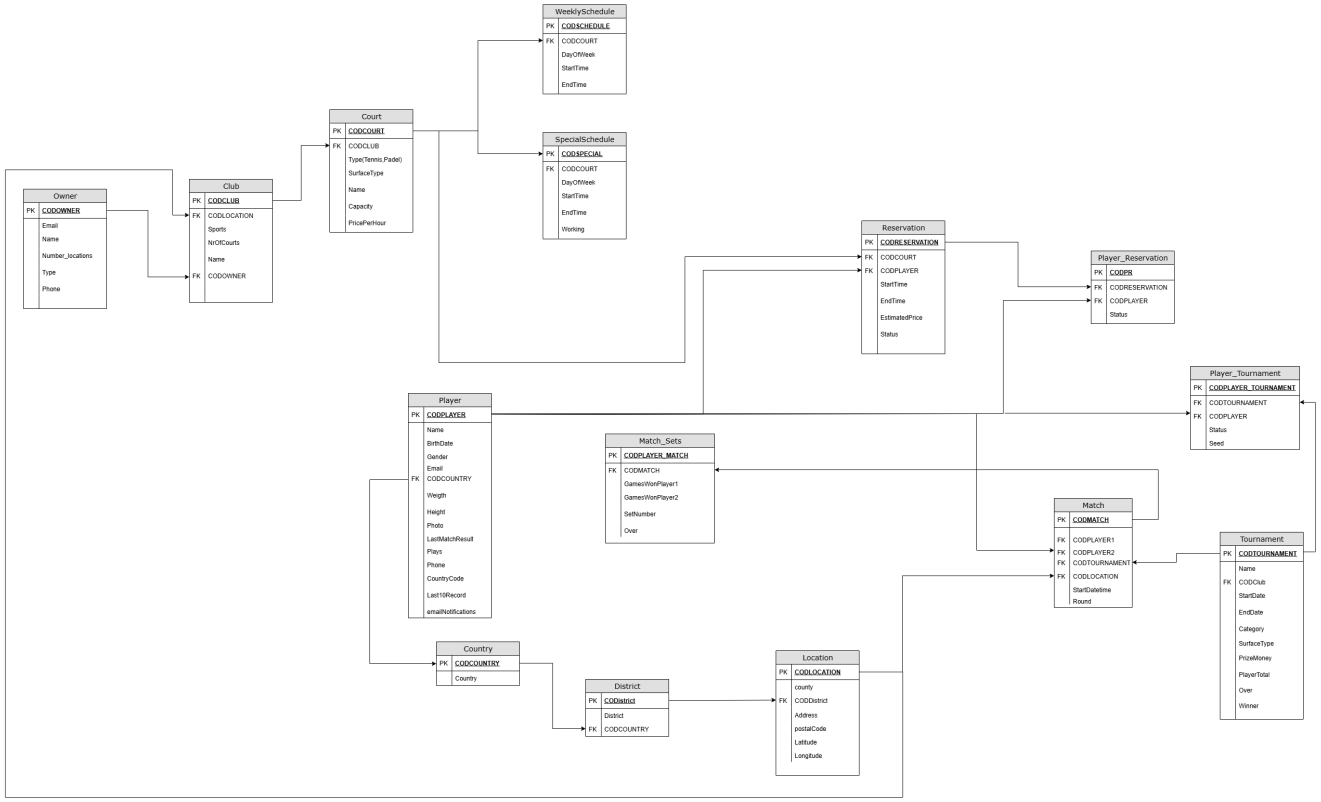

Figura 8: Modelo ER da base de dados

Anexo 2

```
//sst.config.ts file for API Backend

export default $config({
  app(input) {
    return {
      name: "CourtAndGo",
      removal: input?.stage === "production" ? "retain" : "remove",
      protect: input?.stage === "production", //protect: ["production"].includes(input?.
          stage),
      home: "aws",
    };
  },
  async run() {
    const vpc = new sst.aws.Vpc("MyVpc", { nat: "ec2" });

    const database = new sst.aws.Postgres("CourtAndGoDB", {
      vpc,
      dev: {
        username: "postgres",
        password: "1506",
        database: "local",
        host: "localhost",
        port: 5432,
      }
    });

    const api = new sst.aws.ApiGatewayV2("CourtAndGoAPI", {
      link: [database],
    });
    // === UserService ===
    api.route("POST /user/register", { handler: "packages/functions/user/register.
        handler" });
    api.route("POST /user/login", { handler: "packages/functions/user/login.handler" });
    api.route("POST /user/logout", { handler: "packages/functions/user/logout.handler"
        });
    api.route("GET /user/{id}", { handler: "packages/functions/user/getById.handler" });
    api.route("GET /user/email/{email}", { handler: "packages/functions/user/getByEmail.
        handler" });
    api.route("PUT /user/{id}", { handler: "packages/functions/user/update.handler" });
    api.route("POST /user/oauthregister", { handler: "packages/functions/user/
        oauthregister.handler" });
    api.route("GET /user", { handler: "packages/functions/user/getAll.handler" });
    api.route("PUT /user/emailnotification/{id}", {handler: "packages/functions/user/
        emailNotification.handler" });
    api.route("PUT /user/google/{id}", { handler: "packages/functions/user/
        updateFromGoogle.handler" });

    // === ScheduleCourtsService ===
    api.route("GET /schedule/weekly/{courtId}", { handler: "packages/functions/schedule/
        weekly.handler" });
    api.route("GET /schedule/special/{courtId}", { handler: "packages/functions/schedule
        /special.handler" });
    api.route("POST /schedule/weekly", {
      handler: "packages/functions/schedule/createWeekly.handler",
    });
    api.route("POST /schedule/special", {
      handler: "packages/functions/schedule/createSpecial.handler",
    });

    // === ReservationService ===
```

```
api.route("GET /reservations", { handler: "packages/functions/reservation/getAll.
    handler" });
api.route("GET /reservations/{id}", { handler: "packages/functions/reservation/
    getById.handler" });
api.route("GET /player/{playerId}/reservations", { handler: "packages/functions/
    reservation/getByPlayer.handler" });
api.route("POST /reservations", { handler: "packages/functions/reservation/create.
    handler" }); // Create a new reservation
api.route("PUT /reservations/{id}", { handler: "packages/functions/reservation/
    update.handler" }); // Update reservation by ID
api.route("DELETE /reservations/{id}/delete", { handler: "packages/functions/
    reservation/delete.handler" }); // delete reservation by ID
api.route("POST /reservations/{id}/confirm", { handler: "packages/functions/
    reservation/confirm.handler" }); // Confirm reservation by ID
api.route("GET /reservations/filter", {
  handler: "packages/functions/reservation/getByCourtIdsAndDate.handler",
});
api.route("POST /reservations/{id}/cancel", { handler: "packages/functions/
    reservation/cancel.handler" }); // Cancel reservation by ID

api.route("GET /reservations/{id}/ics", {
  handler: "packages/functions/reservation/getICS.handler"
});


// === CourtService ===
api.route("GET /courts", { handler: "packages/functions/court/all.handler" });
api.route("GET /courts/district", { handler: "packages/functions/court/byDistrict.
    handler" });
api.route("GET /courts/sport", { handler: "packages/functions/court/bySportType.
    handler" }); // Get courts by sport type
api.route("GET /courts/filter", { handler: "packages/functions/court/getFiltered.
    handler" });
api.route("GET /courts/{id}", { handler: "packages/functions/court/getById.handler"
    }); // Get court by ID
api.route("GET /courts/owner/{ownerId}", { handler: "packages/functions/court/
    byOwner.handler" });
api.route("POST /courts", { handler: "packages/functions/court/create.handler" });
    // Create a new court
api.route("PUT /courts/{id}", { handler: "packages/functions/court/update.handler"
    }); // Update court details
api.route("DELETE /courts/{id}", { handler: "packages/functions/court/delete.handler
    " });
api.route("GET /courts/club/{clubId}", { handler: "packages/functions/court/byClubId
    .handler" }); //getCourtsByClubId


// === ClubService ===
api.route("GET /clubs", { handler: "packages/functions/club/getAll.handler" });
api.route("GET /clubs/district", { handler: "packages/functions/club/getByDistrict.
    handler" });
api.route("GET /clubs/county", { handler: "packages/functions/club/getByCounty.
    handler" });
api.route("GET /clubs/country", { handler: "packages/functions/club/getByCountry.
    handler" });
api.route("GET /clubs/postal", { handler: "packages/functions/club/getByPostal.
    handler" });
api.route("GET /clubs/name", { handler: "packages/functions/club/getByName.handler"
    });
api.route("GET /clubs/sport", { handler: "packages/functions/club/getBySport.handler
    " });
api.route("GET /clubs/{id}", { handler: "packages/functions/club/getById.handler" })
    ; // Get club by ID
```

```
    api.route("GET /clubs/owner/{ownerId}", { handler: "packages/functions/club/
        getByOwner.handler" });
    api.route("GET /clubs/court/{courtId}", { handler: "packages/functions/club/
        getClubIdByCourtId.handler" });
    api.route("GET /clubs/filter", { handler: "packages/functions/club/getFiltered.
        handler" }); //confirmar se esta de acordo
    api.route("POST /clubs", { handler: "packages/functions/club/create.handler" }); //
        Create a new club
    api.route("PUT /clubs/{id}", { handler: "packages/functions/club/update.handler" });
         // Update club details


    api.route("POST /clubs/location", { handler: "packages/functions/location/create.
        handler" }); // Create a new location for a club
    api.route("PUT /clubs/location/{id}", { handler: "packages/functions/location/update
        .handler" }); // Update location details for a club
    api.route("GET /clubs/{clubId}/location", { handler: "packages/functions/location/
        locationByClubId.handler" }); // Get location by club ID

    // === OwnerService ===
    api.route("POST /owners/register", { handler: "packages/functions/owners/register.
        handler" });
    api.route("POST /owners/login", { handler: "packages/functions/owners/login.handler"
         });



    const site = new sst.aws.React("CourtAndGoAdminSite", {
      path: "frontend-admin",
      buildCommand: "npm run build",
      buildOutput: "dist",
    });

    return {
      host: database.host,
      port: database.port,
      username: database.username,
      password: database.password,
      database: database.database,
    };
  },
});
```

Anexo 3

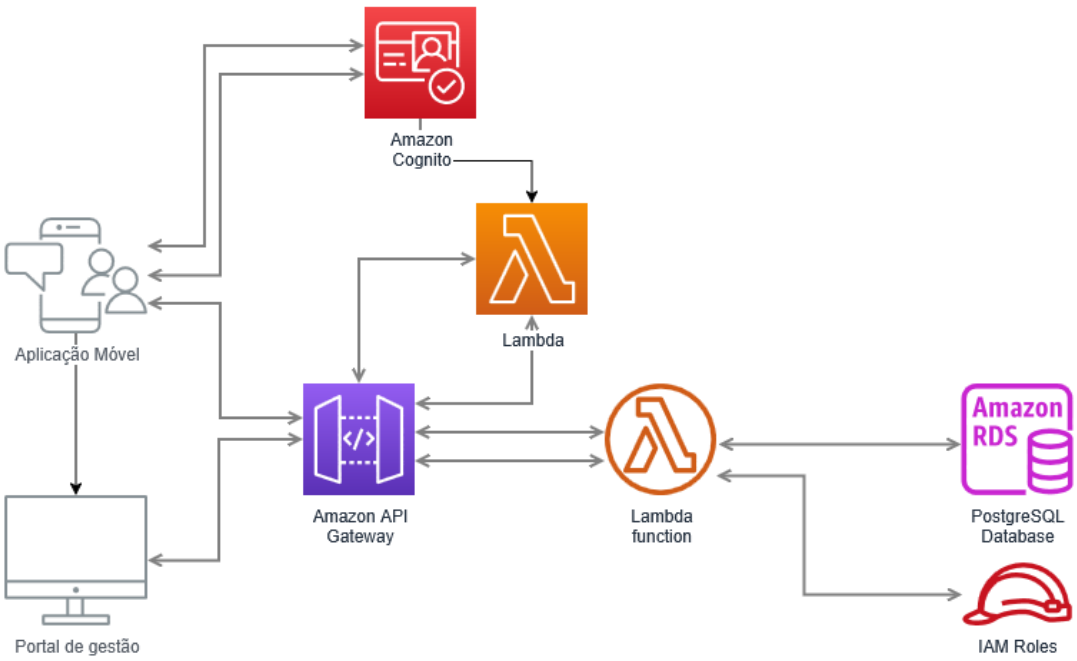

Figura 9: Infraestrutura de Serviços AWS