UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

Brenda Sotero Ferreira

FRAMEWORK LARAVEL:
UM ESTUDO DE CASO FULL STACK DEVELOPMENT

Ouro Preto, MG
2021

Brenda Sotero Ferreira

FRAMEWORK LARAVEL:
UM ESTUDO DE CASO FULL STACK DEVELOPMENT

Monografia II apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Carlos Frederico M. C. Cavalcanti
Coorientador: Especialista e Engenheiro Rodolfo Siviero Stein

Ouro Preto, MG
2021

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

BRENDA SOTERO FERREIRA

FRAMEWORK LARAVEL: UM ESTUDO DE CASO FULL STACK DEVELOPMENT

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 27 de Agosto de 2021.

Membros da banca

Carlos Frederico M. da Cunha Cavalcanti (Orientador) - Doutor - Universidade Federal de Ouro Preto
Roberto Steiner (Coorientador) - Especialista - Masterix Informática
Ricardo Augusto Rabelo Oliveira (Examinador) - Doutor - Universidade Federal de Ouro Preto
Luciano Lage Torres Alves (Examinador) - Especialista - Masterix Informática

Carlos Frederico M. da Cunha Cavalcanti, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 27/08/2021.

Documento assinado eletronicamente por **Carlos Frederico Marcelo da Cunha Cavalcanti**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 03/09/2021, às 14:30, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0213301** e o código CRC **EB95E9F4**.

Referência: Caso responda este documento, indicar expressamente o Processo nº 23109.008217/2021-98 SEI nº 0213301

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35400-000
Telefone: 3135591692 - www.ufop.br

A minha família: meu início, meio e fim.

Agradecimentos

São tantas pessoas que cruzam nosso caminho ao longo da vida e, principalmente, no decorrer de uma graduação. Que fica difícil resumir em poucas linhas. Mas sou grata a Deus, por criar tudo da melhor forma que poderia ser. Aos meus pais e irmãos, que sempre apoiaram e acreditaram em mim. A República Quarto Crescente, que me acolheu e me fez crescer individual e coletivamente, enquanto vivia os melhores anos universitários que poderia existir. Aos amigos que estavam nos tempos bons e também nos difíceis. Ao professor Carlos Frederico Cavalcanti, Luciano Torres, Rodolfo Stein e Alexandre Péret, mentores que me deram a oportunidade de vivenciar o aprendizado na prática. E por fim a Universidade Federal de Ouro Preto, em especial ao Departamento de Ciência da Computação com seus mestres e doutores, por proporcionarem um ensino público e de qualidade.

Resumo

A web é uma poderosa ferramenta na sociedade atual. Sua estrutura possibilita, além de outras coisas, interligar pessoas e serviços de forma fácil e prática. Fazendo com que elas economizem tempo e dinheiro. Com isso, autônomos e empresários tiveram que se reinventar e criar formas de ofertarem seus produtos e serviços pela Internet. Este trabalho tem como ponto de partida a necessidade de desenvolver uma aplicação web que una um serviço de aluguel de quadras ao seu cliente final. Foram analisadas as ferramentas essenciais para o desenvolvimento e os requisitos fundamentais para a aplicação. Como resultado tem-se, além da entrega de uma ferramenta que condiz com as especificações requeridas, o avanço na compreensão e entendimento no que diz respeito ao *framework* Laravel.

Palavras-chave: *Framework* Laravel. Aplicação Web. *Full Stack*. Serviços na Web. Aluguel de Quadras.

Abstract

The Web is a powerful instrument in today's society. Its structure makes possible to connect people and services in an easy and practical way, providing time and money savings. As a result, self-employed people and entrepreneurs had to reinvent themselves and create ways to offer their products and services over the Internet. This study aim is to develop a web application that combines a court rental service to its final customer. The essential tools and the fundamental requirements for the development of the application were analyzed. As a result we have a tool with the required specifications and the advance in understanding regarding the Laravel framework has been achieved.

Keywords: Framework Laravel, Web Application, Full Stack, Web Services, Court Rental.

Lista de Ilustrações

Figura 2.1 – Funcionamento de um adquirente | Fonte: <https://www.formasdepagamento.com/artigo/adquirentes-subadquirentes/> 8
Figura 2.2 – Funcionamento de um gateway | Fonte: <https://www.formasdepagamento.com/artigo/adquirentes-subadquirentes/> 9
Figura 2.3 – Funcionamento de um subadquirente | Fonte: <https://www.formasdepagamento.com/artigo/adquirentes-subadquirentes/> 9
Figura 2.4 – Padrão MVC | Fonte: Luciano e Alves (2011) 11
Figura 2.5 – Comparação dos termos de Pesquisa | Fonte: Google *Trends* 12
Figura 2.6 – Comparação entre Servidores Web *Server-Side* | Fonte: <https://w3techs.com/technologies/overview/web_server> 13
Figura 2.7 – Dinâmica dos Processos em um Servidor Web Apache | Fonte: (ABREU et al., 2013) 14
Figura 2.8 – Sites que usam linguagem *Server-Side* | Fonte: <https://w3techs.com/technologies/overview/programming_language> 15
Figura 2.9 – Exemplo de Uma página simples em HTML | Fonte: Henke (2009) 17
Figura 2.10–Linha de Código com atributos HTML | Fonte: Schulz (2017) 18
Figura 2.11–Linha de Código com Propriedades CSS | Fonte: Schulz (2017) 18
Figura 2.12–Comparação de Páginas Web com e sem CSS | Fonte: Schulz (2017) 19
Figura 2.13–Plataforma VSCode em Execução 21
Figura 2.14–Estrutura de Pastas do Laravel 22
Figura 3.1 – Tela de Cadastro 26
Figura 3.2 – Tela de Cadastro 27
Figura 3.3 – Área do Cliente - Tela *HOME* 28
Figura 3.4 – Área do Cliente - Selecionar Quadra 29
Figura 3.5 – Área do Cliente - Selecionar Horário Disponível 30
Figura 3.6 – Área do Cliente - Confirmação do Dados Selecionados 31
Figura 3.7 – Área do Cliente - Agendamento Concluído 32
Figura 3.8 – Área do Administrador - Tela *Home* 33
Figura 3.9 – Área do Administrador - Lista dos Clientes Cadastrados 34
Figura 3.10–Área do Administrador - Listas das Quadras Cadastradas 35
Figura 3.11–Área do Administrador - Cadastro de Quadras 36
Figura 3.12–Área do Administrador - Listas das Reservas 37
Figura 3.13–Modelagem do Banco de Dados da Aplicação 39
Figura 3.14–Exemplo Arquivo *.env* 40
Figura 3.15–*Migration* - Criação da Tabela Recurso 41
Figura 3.16–Comando para Criação da *Migration* Recursos 42

Figura 3.17–Comando para Salvar a *Migration* Recursos 42
Figura 3.18–Comando para Criar um RecursoSeeder 43
Figura 3.19–Classe RecursoSeeder 43
Figura 3.20–Comando para Salvar as Informações do RecursoSeeder 44
Figura 3.21–Classe DatabaseSeeder 44
Figura 3.22–*Model* da Tabela Recurso 46
Figura 3.23–Continuação do *Model* da Tabela Recurso 47
Figura 3.24–*View* da Tabela Recurso 48
Figura 3.25–*View* da Tabela Recurso 48
Figura 3.26–*View* da Tabela Recurso 49
Figura 3.27–Comando para a Criação do RecursoController 49
Figura 3.28–*Controller* da Tabela Recurso - Função *List* 50
Figura 3.29–*Controller* da Tabela Recurso - Funções *Create* e *Store* 51
Figura 3.30–Rotas 52
Figura 3.31–Rotas da Tabela Recurso 52
Figura 4.1 – Área de Login - Tela Principal. 55
Figura 4.2 – Área de Login - Tela Principal Responsíva. 56
Figura 4.3 – Área de Login - Recuperação de Senha. 57
Figura 4.4 – Área de Login - Inserir Nova Senha. 57
Figura 4.5 – Área de Login - Cadastro de Novo Usuário. 58
Figura 4.6 – Área de Login - Verificação do Endereço de *e-mail*. 59
Figura 4.7 – Área do Cliente - Página Principal. 60
Figura 4.8 – Área do Cliente - Página Principal Responsiva. 61
Figura 4.9 – Área do Cliente - Exibe Informações da Reserva Desejada. 62
Figura 4.10–Área do Cliente - Confirmar Exclusão de Reserva. 63
Figura 4.11–Área do Cliente - Confirmar Exclusão de Reserva. 64
Figura 4.12–Área do Cliente - Confirmar Exclusão de Reserva. 65
Figura 4.13–Área do Cliente - Confirmar Dados da Reserva. 66
Figura 4.14–Área do Cliente - Protocolização da Reserva. 67
Figura 4.15–Área do Gestor - Página Principal. 68
Figura 4.16–Área do Gestor - Página Principal Responsiva. 69
Figura 4.17–Área do Gestor - Clientes - Lista de Clientes que Possuem Registro. 70
Figura 4.18–Área do Gestor - Clientes - Editar Informações do Cliente. 71
Figura 4.19–Área do Gestor - Clientes - Pesquisa de Clientes. 72
Figura 4.20–Área do Gestor - Clientes - Realização de Reserva para um CLiente. 73
Figura 4.21–Área do Gestor - Quadras - Instancia uma Nova Quadra. 74
Figura 4.22–Área do Gestor - Quadras - Lista Quadras Registradas. 75
Figura 4.23–Área do Gestor - Quadras - Edita as Informações de uma Quadra. 76
Figura 4.24–Área do Gestor - Reservas - Lista as Reservas Existentes no Sistema. 77

Figura 4.25–Área do Gestor - Reservas - Mostra as Informações à Respeito da Reserva Selecionada. 78
Figura 4.26–Área do Gestor - Reservas - Altera o *status* da Reserva. Considerando a Presença ou não do Cliente. 79
Figura 4.27–Área do Gestor - Reservas - Realiza o Cancelamento da Reserva. 80
Figura 4.28–Área do Gestor - Reservas - Pesquisa Reservas Compatíveis com as Informações Inseridas. 81

Lista de Abreviaturas e Siglas

API	Interface de Programação de Aplicações
ASF	*Apache Software Fundation*
CSS	*Cascading Style Sheets*
HTML	*Hyper Text Markup Language*
HTTP	*HyperText Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
MVC	*Model - View - Controller*
NASC	*National Center for Supercomputing Applications*
ORM	*Object Relational Mapping*
PHP	*Hypertext Preprocessor*
REST	*Representation State Transfer*
UFOP	Universidade Federal de Ouro Preto
VSCode	*Visual Studio Code*

Sumário

1 Introdução 1
1.1 Justificativa 2
1.2 Objetivos 4
1.2.1 Objetivo Geral 4
1.2.2 Objetivos Específicos 4
1.3 Organização do Trabalho 4
1.3.1 Estrutura da Monografia 5
2 Revisão Bibliográfica 6
2.1 Fundamentação Teórica 6
2.1.1 Sistemas de Pagamento *On-line* 6
2.1.2 Framework Laravel 8
2.1.2.1 Primeiros Passos com o Laravel 11
2.1.2.1.1 Servidor Apache 12
2.1.2.1.2 Linguagem PHP 14
2.1.2.1.3 HTML5 16
2.1.2.1.4 CSS 17
2.1.2.1.5 Gerenciador de Pacotes *Composer* 19
2.1.2.1.6 VSCode 19
2.1.2.1.7 Jetstream e Tailwind 20
2.1.2.2 Iniciando um Projeto com Laravel 22
2.2 Trabalhos Relacionados 24
3 Desenvolvimento 25
3.1 Levantamento dos Requisitos 25
3.2 Modelagem do Banco de Dados 38
3.3 Implementação 40
3.3.1 Instanciando o Banco de Dados 41
3.3.1.1 *Migrations* 41
3.3.1.2 Seeders 42
3.3.2 Padrão de Projeto 44
3.3.2.1 *Model* 45
3.3.2.2 *View* 46
3.3.2.3 *Controller* 49
3.3.3 *Rotas* 51
4 Resultados 54
4.1 Área de Login 54
4.2 Área do Cliente 55

4.2.1 Tela Principal do Cliente 55
4.2.2 Agendar Quadra 56
4.3 Área do Gestor 58
4.3.1 Tela Principal do Gestor 58
4.3.1.1 Clientes 59
4.3.1.2 Quadras 59
4.3.1.3 Reservas 61
5 Considerações Finais 82
5.1 Conclusão 82
5.2 Trabalhos Futuros 82

Referências 84

1 Introdução

Para Latorre (2018), a web é “um conjunto de documentos interconectados por enlaces de hipertexto e disponíveis na Internet que podem se comunicar através da tecnologia digital”. Definimos o conceito de hipertexto como sendo um texto eletrônico que possui informações localizadas fora do texto e que se tem acesso através de referências digitais chamadas *hiperlinks*. Em outras palavras, são documentos com referências a outros documentos e recursos (inclusive em outros formatos e mídias) através de referências digitais de localização do documento / recurso (*hiperlinks*). .

O termo web é muitas vezes confundido com a rede mundial Internet. A Internet é uma rede pública (qualquer um pode se conectar) onde informações originadas e destinadas dos computadores conectados à esta rede são trafegadas (CONTE; MENDES; TRAVASSOS, 2005). A web apenas utiliza desse meio de transporte para difundir suas aplicações. São duas as categorias existentes de aplicações web: hipermídia, que junta características de um sistema hipertexto e é disponibilizada através da web; e software web, que, por meio de um software convencional, manipula a infraestrutura web para sua execução.

A base de um projeto de aplicação web, também conhecido como *web application*, deve respeitar três pilares, tratados por Conte, Mendes e Travassos (2005) como dimensões, são eles:

- Estrutural: trata-se da forma como o projeto será modelado, definindo suas funcionalidades e requisitos;
- Navegacional: trata do acesso a essas informações;
- Apresentação: como se dará a interação dos usuários com essas informações.

Os autores acima afirmam:

> “Cada uma dessas dimensões define diferentes visões para o projeto da aplicação. O objetivo principal da Engenharia Web é desenvolver aplicações corretas, nas quais suas estruturas, funcionalidades, aspectos navegacionais e de interação com o usuário estejam representados de forma apropriada.”

Para Silva e Carvalho (2017) uma aplicação web é dividida em *back-end* e *front-end*. O *front-end* representa o visual do software, ou seja, o que a pessoa vê e interage na tela. E é estruturada de forma com que se adapte a diferentes tipos de ecrãs. O *back-end* trabalha de forma oculta, recebendo, processando e respondendo às requisições feitas pelo usuário.

O fato é que a web, em conjunto com a Internet, desempenham um forte papel social na comunidade. Xavier (2009) aponta que a web "constitui uma nova realidade que temos que saber enfrentar e usufruir". A web tem o poder de transformar um expectador passivo, um mero receptor e consumidor de informações vindas da televisão e outros mecanismos, em um elemento ativo, uma fonte direta de informações onde ele produz e transmite seu conteúdo. O cuidado com essa tecnologia deve ser redobrado. Afinal de contas, quando todo mundo gera conteúdo a todo momento, torna-se difícil a garantia da qualidade do mesmo.

Mas os pontos positivos da web são infinitamente maiores. Ela é uma eficiente ferramenta para a globalização fazendo com que milhares de quilômetros de distância entre duas pessoas se findasse em apenas uma vídeo chamada. Ou, que alguém pudesse usar um serviço inexistente na sua região, mas acessível graças a tecnologia web. O fato é que, a cada minuto, milhares de novos web sites são criados e outros extintos(XAVIER, 2009) fazendo com que essa produção de conteúdo e ferramentas seja dinâmica. Da mesma forma, novos segmentos de negócio *on-line* são criados a cada momento, aumentando cada vez mais a cobertura de serviços e aplicações disponíveis na web.

O conjunto de serviços disponibilizados na web cresce constantemente. Hoje em dia, todos os negócios podem e devem ter pelo menos uma parte do empreendimento focada na Internet. No mínimo, informações de local e telefone para contato de uma empresa devem ser disponibilizadas na web. As pesquisas por um produto ou serviço que uma pessoa necessita, primeiro se dá na Internet, para depois ser fechado de fato, seja na própria aplicação web ou presencialmente. Por isso a importância de se fazer presente neste meio.

O usuário poderá fazer mais do que uma pesquisa por um serviço ou produto, poderá realizar a compra, isto é, realizar o negócio na própria plataforma de uma forma *on-line*, interativa e remota. Este método proporciona a compra por um preço justo, devido a rapidez de acesso à informação e a possibilidade de comparação instantânea (LOPES, 2018). Assim, uma interação de comércio eletrônico, do inglês *e-commerce*, tem total capacidade de atingir o limiar máximo de satisfação tanto para o usuário quanto para o dono do empreendimento. Isso desconsiderando quaisquer outros pormenores que possam existir nessa negociação.

1.1 Justificativa

O processo de urbanização de uma cidade começa a existir quando há uma migração da população das zonas rurais para zonas urbanas. Esse processo pode acontecer por inúmeros motivos e vai de acordo com as necessidades da população. Quando se abre uma fábrica em determinada região, por exemplo, cria-se uma movimentação maior naquela área. Pessoas que trabalham ali começam a buscar moradias mais próximas, comprar em lojas e mercados próximos e até abrir novos empreendimentos no local.

Assim, como uma característica das zonas urbanas, os espaços vão ficando mais cobiçados

e disputados, fazendo com que cada metro quadrado seja valorizado monetariamente. E pouco a pouco, árvores são derrubadas para darem lugar a prédios. Ou seja, a possibilidade de lucro pela venda daquela área vale mais do que o lazer e o bem estar que este local poderia oferecer para a sociedade. Nesta perspectiva, percebe-se que as mudanças advindas da urbanização geram a necessidade de criação de mecanismos para a socialização e interação e uma delas é o esporte, ponto crucial para este trabalho.

A necessidade de uma área adequada para a prática de esportes vai de encontro ao que foi dito sobre as relações entre espaço e monetização já que, com a falta de espaço em uma área urbana, as pessoas não têm mais liberdade e nem estrutura adequada para praticar seus esportes onde bem entenderem. Isto justifica, por exemplo, a criação de clubes de tênis, quadras de futsal e de basquete e campos de futebol, onde as pessoas fazem uma permuta, oferecendo dinheiro em troca de um período de tempo para usufruir desses ambientes.

Com menos de 100.000 habitantes de acordo com o IBGE[1], Mariana é uma cidade de Minas Gerais considerada de pequeno porte. Assim como a maioria das outras cidades do país e do mundo, Mariana já conhece a realidade dos efeitos da urbanização. Para amenizar este impacto, a cidade oferece dezenas de quadras espalhadas no município, sendo algumas particulares. O serviço de aluguel procura promover praticidade e melhor estrutura para quem deseja desfrutar de uma experiência mais completa e organizada. Um exemplo desse serviço é a oferecida pelo clube Guaranix, que já faz parte da história da cidade. Este trabalho foi idealizado exatamente para auxiliar no gerenciamento dessa demanda para este clube, sendo sua motivação principal.

A ideia é que qualquer pessoa que tenha interesse de fidelizar ao clube tornar-se-á seu cliente. Esse poderá acessar a agenda no site e escolher uma quadra e o horário desejado, desde que o mesmo esteja vago. Para finalizar o aluguel da quadra desejada, basta inserir seus dados e, após processado, receberá a confirmação do aluguel em seu e-mail. Já o gestor terá todas as informações do agendamento feito: horários, clientes e seus dados, cancelamento de reservas e comparecimento das mesmas, bem como a possibilidade de realizar e excluir cadastros e agendamentos. Também será responsável pelas informações essenciais como preço e disponibilidade das quadras. Finalmente e não menos importante, poderá inserir e excluir funcionários que terão acesso a maioria dessas informações.

Sendo assim, a gestão de dados e pessoal ficará inteiramente à cargo do gestor enquanto o processo para o usuário ocorre de forma descomplicada.

[1] https://cidades.ibge.gov.br/brasil/mg/mariana/panorama

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver os conhecimentos acerca do *framework* Laravel. E para isso será implementado uma aplicação que realizará o agendamento de horários para o uso das quadras do Guaranix e permitirá que seja efetuado o pagamento do serviço prestado pela plataforma. Além de prover um ambiente onde agendamentos podem ser feitos, o sistema permitirá gerenciar quadras, horários e clientes de acordo com as necessidades do gestor.

1.2.2 Objetivos Específicos

A aplicação a ser desenvolvida será um sistema Web e terá toda a sua arquitetura detalhada, contando com *back-end* e *front-end*. As demandas do projeto são:

- Entender a API Laravel e seus componentes;
- Levantar os requisitos necessários para modelagem da aplicação;
- Fazer a modelagem do sistema e do banco de dados;
- Realizar pesquisas relacionadas a formas de pagamento *on-line*;
- Aplicar os conhecimentos adquiridos na concepção do projeto;
- Testar a ferramenta, trazendo os resultados.

.

1.3 Organização do Trabalho

Este trabalho se dará em duas etapas. A primeira parte compreende a busca pelos referenciais teóricos, escolha das ferramentas adequadas para o desenvolvimento e o levantamento e estudo dos requisitos necessários para o sistema. Já a segunda parte contará com a implementação e análise dos resultados obtidos.

A introdução conta com um breve histórico sobre o sistema web e a oferta de serviços que surgiram a partir dela. O referencial teórico vai apresentar os estudos relacionados ao tema, bem como as ferramentas que serão utilizadas no trabalho. Na metodologia se dará início ao projeto, com a análise dos requisitos, modelagem do banco de dados e implementação do sistema. E, por fim, os resultados, demonstrando tudo o que foi feito, a conclusão e os trabalhos futuros.

1.3.1 Estrutura da Monografia

A estruturação deste trabalho se apresenta da seguinte forma:

Capítulo 1: Introdução.

Capítulo 2: Revisão Bibliográfica.

Capítulo 3: Metodologia.

Capítulo 4: Resultados.

Capítulo 5: Considerações finais e conclusão.

2 Revisão Bibliográfica

2.1 Fundamentação Teórica

Este trabalho está sendo desenvolvido baseado no *framework* Laravel. Assim, um dos principais requisitos é conhecer os fundamentos e funcionalidades deste, bem como suas linguagens de desenvolvimento, em especial PHP, e algumas de suas extensões, a exemplo de Jetstream. Também é necessário entendimento sobre levantamento de pré-requisitos, modelagem de banco de dados para MySQL e integração de uma plataforma para pagamentos *on-line*. Existem outras ferramentas que serão usadas no decorrer do projeto, mas que podem ser modificadas de acordo com as preferências de cada programador. São elas: o software de virtualização VirtualBox, o sistema operacional Ubuntu, o editor de código Visual Studio Code, o servidor Apache e o MySQL Workbench [1], para modelagem do banco de dados. Todas estas ferramentas e os principais conceitos associados à elas serão explanadas no decorrer deste capítulo.

2.1.1 Sistemas de Pagamento *On-line*

Para Lopes (2018) o *e-commerce* "nada mais é do que a conversão de qualquer negócio *off-line* na sua versão *on-line*". Negociar significa transacionar e se faz necessário que o negócio (transação) seja realizado dentro do ambiente *on-line*, facilitando a obtenção de um produto ou serviço com apenas um clique. Das vantagens desta forma de transação comercial, Moreira (2020) cita:

> "O *e-commerce* apresenta inúmeras vantagens para as empresas e para os clientes. Para as empresas podemos referir a possibilidade de colocar detalhes e informações sobre os produtos e os custos fixos serem menores. Para além disso, uma vez que a Internet é uma ferramenta utilizada em nível mundial, permite a compra / venda em qualquer lugar e hora ajudando a empresa na sua internacionalização. Para o cliente apresenta como vantagens a possibilidade de variedade dos produtos levando a uma maior facilidade de comparação de artigos, vários meios de pagamento e serviço 24 horas por dia para qualquer lugar".

Um dos principais fatores para o crescente sucesso do *e-commerce* é a disponibilidade de diferentes formas de pagamento para adquirir o produto ou o serviço ofertado. Isto, potencialmente, torna o produto mais acessível para os diversos públicos, pois flexibiliza as condições de pagamento e atende as condições financeiras de cada indivíduo. Alguém que esteja precisando de um eletrodoméstico, por exemplo, e que não tenha condições de pagar à vista, pode usar a

[1] https://www.mysql.com/products/workbench/

função de crédito, parcelando e diluindo o valor ao longo dos meses. Assim como uma pessoa que não possui cartão de crédito e quer fazer uma compra *on-line*, basta gerar um boleto e realizar o pagamento em alguma agência credenciada.

Existe toda uma estrutura por trás dos pagamentos *on-line*. Instituições que prestam serviços e que estão envolvidos no processo de compra e venda (transação) se comunicam entre si para garantir a segurança e a confiabilidade das transações. Marboni (2019) explica os cinco principais conceitos relacionados a essas operações:

- **Bandeira de Cartão**: responsáveis pela comunicação entre o adquirente e os bancos emissores de cartão. Elas processam as requisições, realizam uma análise do perfil de consumo e repassam as informações para os bancos. As mais conhecidas no Brasil são Visa e Mastercard;

- **Bancos Emissores de Cartão**: são bancos como Itaú, Caixa Econômica Federal, Credicard e, para efeito de pagamento online, são responsáveis por verificação do saldo e limites;

- **Adquirente**: As adquirentes são empresas como a Stone, a Cielo e a Rede cujo papel é liquidar as transações financeiras por meio de cartão de crédito e débito e são popularmente conhecidas pelo "aluguel" dos terminais financeiros (vulgarmente conhecidos como TFs ou "maquininhas de cartão"), porém o aluguel de TFs não é sua principal função. A adquirente estabelece uma comunicação segura entre os estabelecimentos, as bandeiras e os bancos emissores de cartão. A Fig.2.1 ilustra o processo de transação *on-line* com o adquirente, não sendo muito diferente do meio físico, sendo da seguinte maneira: o cliente insere seus dados do cartão, o adquirente recebe essas informações e encaminha para as bandeiras, que irão verificar se o cliente tem limite, caso seja no crédito, ou saldo suficiente para débito. Caso tudo esteja certo, o adquirente dá como retorno a liberação da conta.

  Muito simples, porém costuma sair caro. As taxas cobradas são geralmente em cima do valor de cada compra e podem ir de 2 a 6% dependendo do plano contratado e da forma de pagamento escolhida pelo cliente;

- **Gateway**: é o responsável por interligar as lojas virtuais com os meios de pagamentos citados e faz uso de uma API (*Application Programming Interface*), que é um conjunto de funções geralmente baseados em uma interface RESTFul, para processar as informações. Representation State Transfer (REST), ou Transferência de Estado Representacional, é definido por Lecheta (2015) como uma arquitetura de software que tem como principal função "criar serviços web e auxiliar na integração de sistemas". E o termo RESTFul é utilizado para indicar sistemas que seguem o princípio REST (LECHETA, 2015).

  A Fig.2.2 exemplifica como funciona o gateway: o cliente insere seus dados para a compra, o *gateway* transfere esses dados para uma adquirente, que realiza seu processo de forma normal e retorna se a transação está aprovada ou não. As taxas cobradas pelo *gateway* são

Figura 2.1 – Funcionamento de um adquirente | Fonte: <https://www.formasdepagamento.com/artigo/adquirentes-subadquirentes/>

baseadas no volume e valor das transações e não em taxas fixas como anuidade. Outra vantagem do *gateway* é que o dinheiro da compra, independente de como o cliente escolheu pagar, já vai direto para a conta da empresa. Mas sua implementação depende da contratação de outros serviços, o que pode acabar saindo mais custoso;

- **Subadquirente**: considerada a melhor opção para *e-commerce* de menor porte, as empresas subadquirentes funcionam como uma plataforma intermediadora. Elas não fazem nenhum tipo de processamento, apenas repassam informações da loja para os meios de pagamento e fazem o caminho contrário, repassando o parecer desses meios, como mostrado na Fig. 2.3. Seu atrativo é o custo inferior, já que possuem parcerias com várias adquirentes.

Além dessas empresas que fornecem a estrutura necessária para validação dos pagamentos *on-line*, também existem as formas com que os clientes desejam quitar suas compras que são os boletos bancários, cartões de crédito e de débito e transferências eletrônicas entre fundos (MARBONI, 2019).

2.1.2 Framework Laravel

Os *frameworks* vem tomando cada vez mais espaço nos dias de hoje, apesar de não serem novidade. Isso se deve ao fato de que, além de integrar várias ferramentas úteis num mesmo ambiente de programação, eles também são diversificados e mantêm uma estrutura que facilita o

Figura 2.2 – Funcionamento de um gateway | Fonte: <https://www.formasdepagamento.com/artigo/adquirentes-subadquirentes/>

Figura 2.3 – Funcionamento de um subadquirente | Fonte: <https://www.formasdepagamento.com/artigo/adquirentes-subadquirentes/>

manuseio das mesmas. Isto torna a programação mais prática e o desenvolvimento de sistemas e aplicações bem mais produtivo.

(SILVA; CARVALHO, 2017) explica o que é um *framework* da seguinte forma: "é o agrupamento de várias bibliotecas buscando promover uma solução genérica, ou seja, um conjunto de códigos que possam ser utilizados frequentemente para resoluções de diferentes problemas". E ainda completa dizendo que um *framework* é responsável por gerenciar o fluxo de controle da aplicação, função que antes era responsabilidade exclusiva aos desenvolvedores.

Laravel, como diz seu slogan *The PHP framework for Web Artisans*, é um *framework* PHP para artesãos (de software) e tem o sentido de levar a mensagem de que um artesão mistura a técnica com a arte. Laravel é uma ferramenta para criação de sistemas web complexos e completos. Gabardo (2017) afirma que em pouco tempo de prática com o *framework*, é nítido sua clareza e bom desenvolvimento além de ter um código limpo e elegante. Essas características também são vistas na documentação da ferramenta, que são bem detalhadas, com bons exemplos de todas as funcionalidades e é bem completa. Para além dessas qualidades, ainda tem uma comunidade ativa, que torna mais fácil e ágil a resolução de problemas que possam ser encontrados no decorrer do desenvolvimento.

O Laravel "é um *framework* PHP MVC sob o paradigma de orientação a objetos" (GABARDO, 2017). Ou seja, ele é baseado na linguagem PHP e tem sua arquitetura MVC - *Model, View, Controller*. De acordo com Luciano e Alves (2011) o padrão MVC surgiu na década de 80 e se popularizou na criação de aplicações web. Seus três pilares têm uma dinâmica muito simples: o *controller* é onde passam todas as requisições do sistema, ele acessa a camada *model* para processar essas requisições e utilizam a *view* para exibir os resultados. Tendo então de forma separada as camadas de apresentação, de lógica de negócio e de gerenciamento de fluxo da aplicação. Facilitando a reutilização e manutenção de um projeto.

A Fig.2.4 apresenta um diagrama que exemplifica o relacionamento *Model-View-Controller*, tendo as linhas tracejadas como associação indireta e as sólidas como associação direta (LUCIANO; ALVES, 2011).

Com a simplicidade e a facilidade da arquitetura MVC, dá para entender um pouco mais os motivos que fazem do Laravel uma aplicação que está em ascendência. O Google *Trends*, uma plataforma da google que é especializada em realizar comparações entre nome, termos e afins, usando as informações das procuras na web pelo buscador da Google. A Fig.2.5 apresenta um comparativo desta plataforma. Os termos de pesquisa usados foram: Laravel, Symfony, CodeIgniter, Zend e Node.js. Como resultado ela mostra que o Laravel, sendo uma das maiores apostas da atualidade, se destaca em termos de popularidade. No gráfico, quanto mais próximo de 100, mais popular é o termo. É bom ressaltar que o gráfico analisa a quantidade de vezes que o termo foi pesquisado em função dos demais termos.

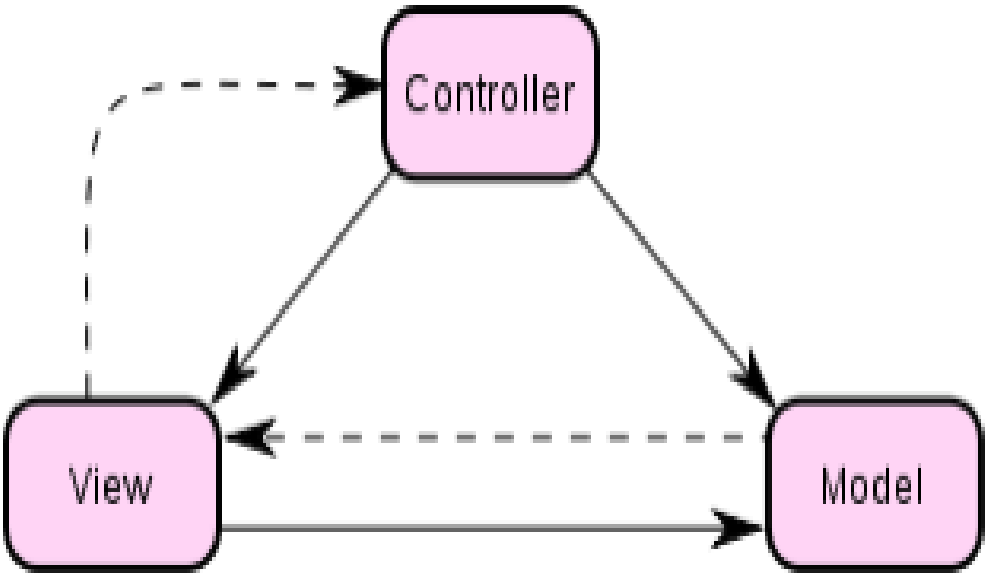


Figura 2.4 – Padrão MVC | Fonte: Luciano e Alves (2011)

2.1.2.1 Primeiros Passos com o Laravel

Para dar início a um projeto no Laravel, antes é necessário que algumas configurações no ambiente local sejam feitas. Como já dito anteriormente, este projeto fará uso de uma máquina virtual VirtualBox[2] com o sistema operacional Ubuntu versão 20.04.1. É nesta máquina que serão feitas as instalações das ferramentas necessárias e nela que os arquivos ficarão salvos.

Antes de instalar o Laravel, é necessário que a máquina já contenha as seguintes ferramentas:

- PHP, com algumas extensões. Aqui será usado a versão 7 da linguagem;
- Servidor, que será o Servidor HTTP Apache;
- Gerenciador de Pacotes Composer;
- Editor de código, neste caso o Visual Studio Code;
- Jetstream e Tailwind.

Após as configurações desses itens, o Laravel já pode ser instalado. Junto com ele também será instalado o Jetstream, um kit de aplicação feito para o Laravel que fornece funcionalidades como login e autenticação de dois fatores, tornando este trabalho mais simples e seguro.

[2] https://www.virtualbox.org/

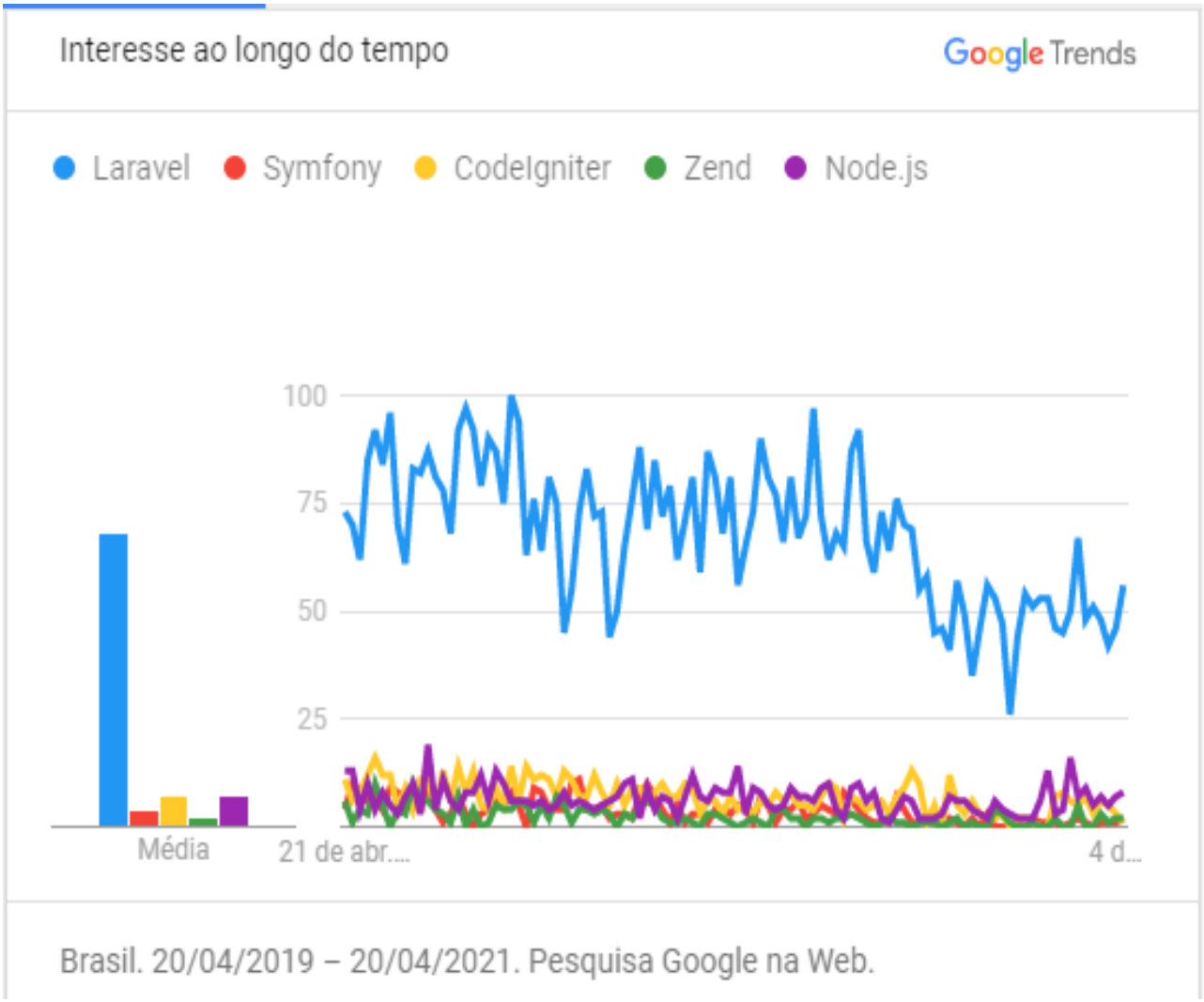


Figura 2.5 – Comparação dos termos de Pesquisa | Fonte: Google *Trends*

2.1.2.1.1 Servidor Apache

O servidor Apache é um servidor web (*web-server*). Martins (2015) define esse tipo de servidor como o resultado "da combinação de equipamento informático e do programa informático nele instalado", sendo o equipamento *hardware* e programa *software*. Ele é responsável por receber, processar e responder as requisições e solicitações feitas por um endereço web. Essa comunicação é estabelecida pelos protocolos de comunicação entre sistemas de hipermídia, HTTP ou HTTPS, e pelos protocolos de comunicação entre computadores TCP/IP, feita através de uma rede. Uma outra característica do servidor web é que ele utiliza o sistema operacional para dividir a função de gerenciar as conexões TCP (MARI, 2004).

Marcelo (2005) explica que o Apache surgiu da iniciativa para reviver o antigo NSCA Web *Server*, que tinha sido deixado de lado por seus desenvolvedores. E assim foi criada a Fundação Apache. Marcelo (2005) considera que esse seja "um dos mais robustos e seguros programas de desenvolvidos para ambientes TCP/IP"e, por isso, sua aceitação no mercado é tão

elevada.

A Fig.2.6 mostra que o Apache é usado por 34% dos sites que tem os servidores conhecidos pela W3Tech. De acordo com Abreu et al. (2013), essa popularidade se deve ao fato de que por ser *open source*, ou código aberto em português, houve uma maior colaboração por parte de outros desenvolvedores, deixando o software ainda mais funcional. E outra "grande"vantagem do servidor Web Apache consiste na existência de diferentes versões do programa, as quais permitem que o servidor funcione em diferentes sistemas operacionais"(ABREU et al., 2013).

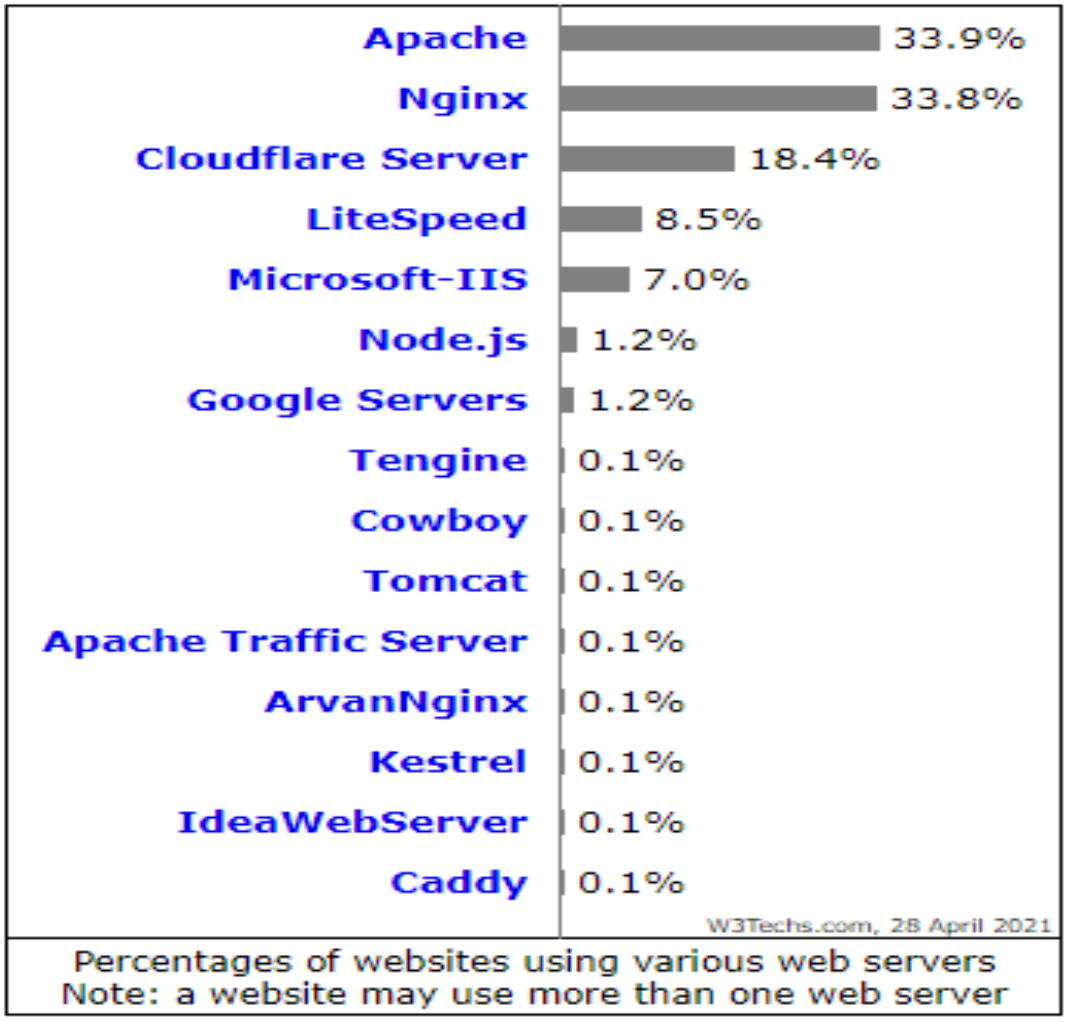


Figura 2.6 – Comparação entre Servidores Web *Server-Side* | Fonte: <https://w3techs.com/technologies/overview/web_server>

É importante entender como as requisições são tratadas pelo Apache. A Fig. 2.7 mostra a dinâmica deste processo. Assim que um usuário envia uma requisição, essa requisição entra em uma fila de espera que norteia o gerenciador do servidor quando e para qual *worker* ou processo essa requisição deve ir.

Os processos têm três estágios: *busy* que é quando o processo está ocupado; *idle* quando está ocioso e *wait*, quando não está atendendo a uma requisição, mas está esperando que o cliente,

já conectado, a envie.

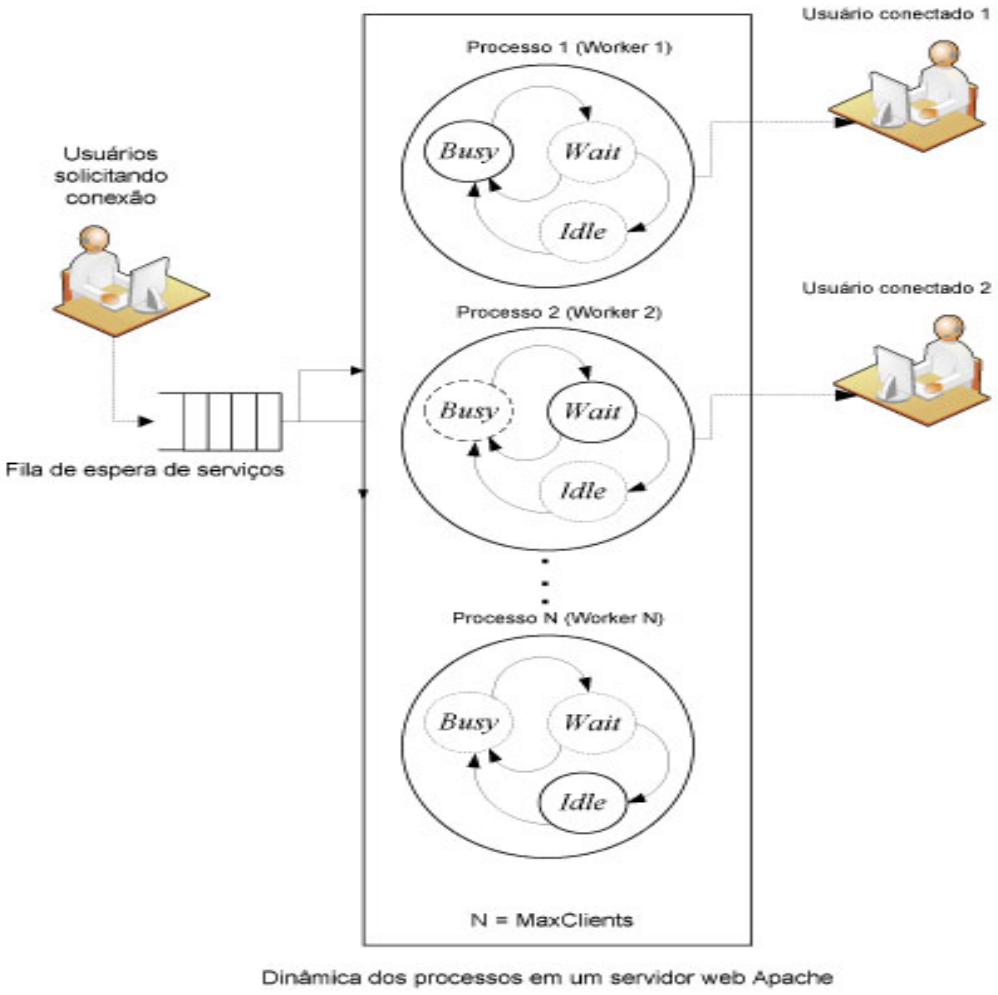


Figura 2.7 – Dinâmica dos Processos em um Servidor Web Apache | Fonte: (ABREU et al., 2013)

Como servidor escolhido para este projeto, a versão usada será a Apache/2.4.41 (Ubuntu). O download pode ser feito pelo próprio site [3] da Fundação Apache.

2.1.2.1.2 Linguagem PHP

A linguagem PHP: HyperText Preprocessor surgiu em meados dos anos 90 e foi desenvolvida por Ramus Lerdof. De acordo com Canto (2011), a versão 3.0 resultou numa reestruturação significativa na linguagem, tornando-a ainda mais funcional, com foco além da criação de páginas dinâmicas na Web, como inicialmente idealizado por Ramus. Canto (2011) também mostra que foi a partir dessa versão que surgiu a possibilidade da coparticipação de outros desenvolvedores com novas extensões.

[3] https://httpd.apache.org/download.cgi

Atualmente o PHP é uma das linguagens mais utilizadas no mundo para desenvolvimento WEB. De acordo com a *W3Techs*[4], site responsável pelo fornecimento de relatórios e informações acerca de tecnologias na web, até fevereiro de 2021 o PHP estava sendo utilizado por 79,1% dos sites que fazem uso de linguagem server-side. A Fig.2.9 mostra a disparidade em relação às demais linguagens.

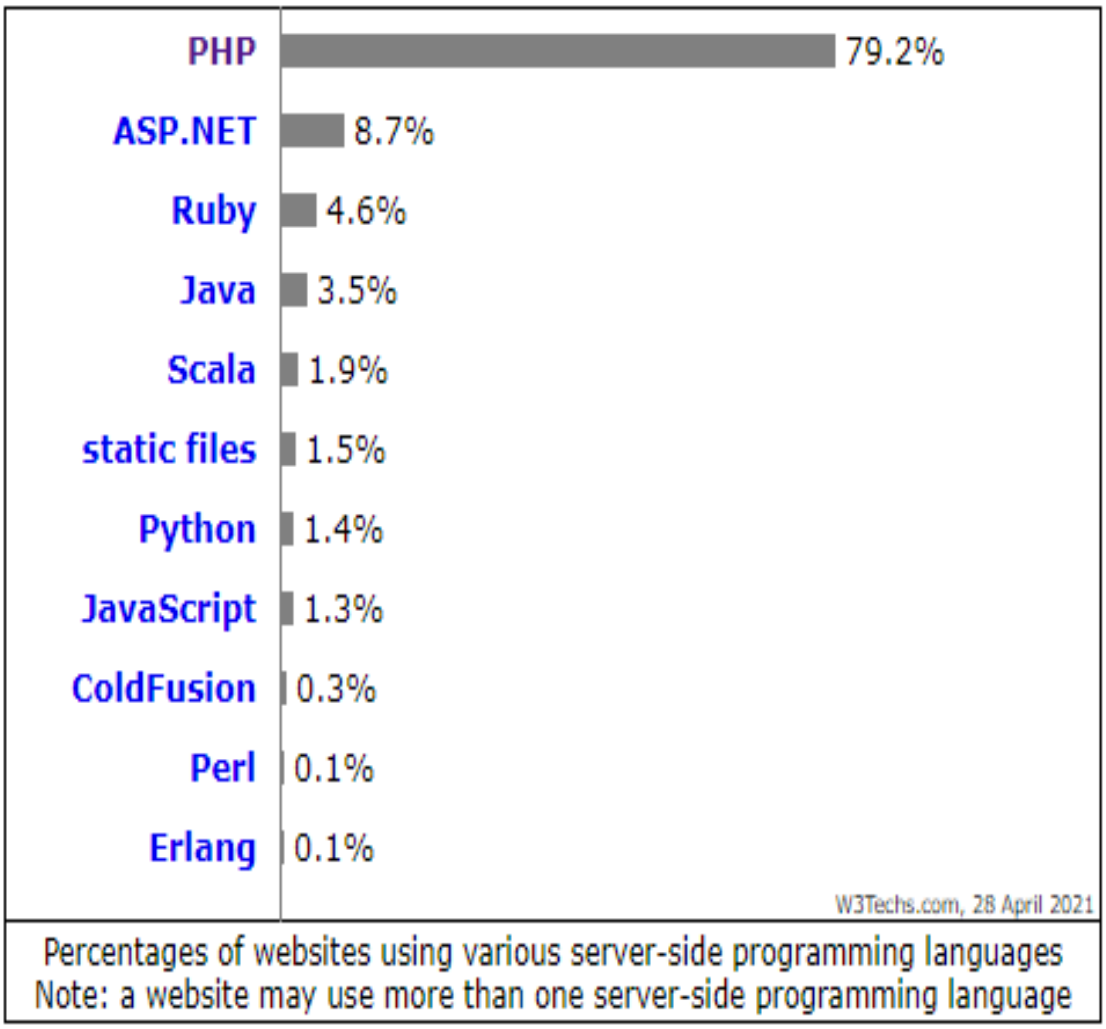


Figura 2.8 – Sites que usam linguagem *Server-Side* | Fonte: <https://w3techs.com/technologies/overview/programming_language>

Uma linguagem é dita *server-side*, ou seja, do lado do servidor, quando o processamento de uma requisição acontece diretamente no servidor e retorna uma resposta. Diferente do *client-side* em que o navegador fica responsável por fornecer essa resposta. Basicamente, o cliente não entra em contato com o PHP, tudo o que ele vê é o resultado em HTML de uma aplicação PHP.

Canto (2011) classifica o PHP como uma linguagem de tipagem fraca e dinâmica, isso porque na sua execução as variáveis não são instanciadas com um tipo específico. Para se declarar uma variável em PHP, basta atribuir um valor e assim ela passará a existir e o tipo será inferido,

[4] https://w3techs.com/

no momento da instanciação, a partir da notação usada em sua definição. Isso também faz com que as variáveis sejam fluidas, podendo receber hora um inteiro, hora uma string.

Converse e Park (2003) afirmam que, quando se trata de servidor, o "PHP é um módulo oficial de servidor HTTP Apache [...]. Isso significa que o mecanismo de *script* do PHP pode ser construído no próprio servidor web, tornando a manipulação de dados mais rápida.". Tanto o Apache, quanto o PHP, fazem parte da Apache Software Foundation (ASF), uma fundação sem fins lucrativos que conta com a participação de desenvolvedores de várias partes do mundo para a criação de conteúdo.

Converse e Park (2003) também ressaltam os motivos para aderir à linguagem: além de ser gratuito e de código aberto, indo de acordo com os propósitos da ASF. PHP não tem base em *tags* como o HTML, por exemplo, o que deixa o código mais dinâmico e sem a necessidade de ficar preso em *tags* já preestabelecidas. É estável, rápido, tem uma boa comunicação com outros programas e protocolos e integra novos recursos com facilidade e rapidez.

A versão do PHP que vai ser utilizada neste projeto é a 7.4. Além da instalação do PHP, que pode ser feita através do site php.net (<https://www.php.net/manual/pt_BR/install.php>), as seguintes extensões também devem ser baixadas:

- OpenSSL PHP *Extension*;
- PDO PHP *Extension*;
- Json PHP *Extension*;
- BCMath PHP *Extension*;
- Ctype PHP *Extension*;
- MBstring PHP *Extension*;
- Tokenizer PHP *Extension*;
- XML PHP *Extension*.

Considerando a linguagem PHP, é fundamental outras duas linguagens muito presentes no Laravel: HTML e CSS.

2.1.2.1.3 HTML5

Hyper Text Markup Language é uma linguagem de marcação formada por um conjunto de *tags*. Henke (2009) conta que a linguagem surgiu por uma necessidade que Tim Berners-Lee encontrou de criar um novo sistema de hipertexto, menos complexo e mais acessível. Assim ele

desenvolveu o Hyper Text Transfer Protocol, o HTTP e utilizou, como linguagem de formatação para estes documentos, o HTML.

Sua estrutura é simples e tem a capacidade de gerar documentos de vários tipos, como textos, mídias, links, entre outros ((SILVA; CARVALHO, 2017) apud MARCONDES, 2007). As *tags* servem como um modelo predefinido, usadas entre colchetes angulares e que precisam ser fechadas a cada nova ação. Como exemplificado na figura 2.2, onde uma tag <html> é aberta para iniciar um arquivo e fechada com </html> para marcar seu fim.

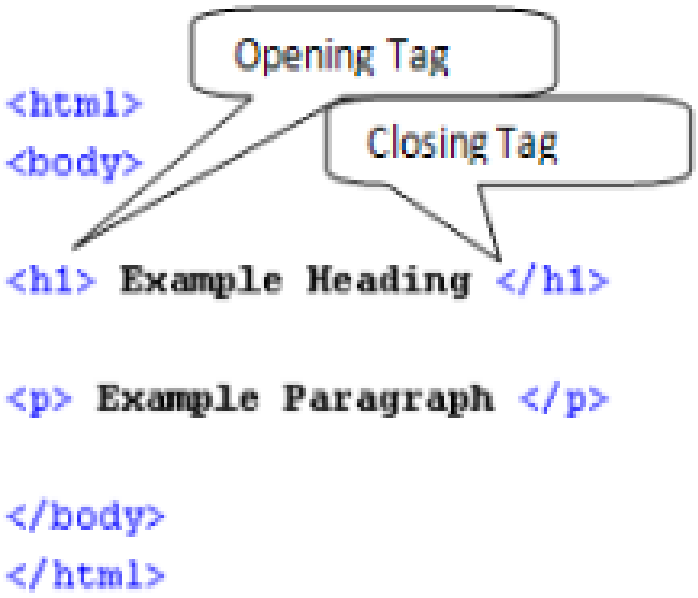


Figura 2.9 – Exemplo de Uma página simples em HTML | Fonte: Henke (2009)

Nas páginas Web, o HTML é responsável pela forma com que as informações vão ser exibidas. Sua funcionalidade está ligada ao *front-end* e ao *layout*. Não é objetivo principal do HTML trazer um *layout* agradável e com todos os requisitos necessários para satisfazer visualmente o usuário. Mas ele possui muitas ferramentas que podem ser usadas nessa área

O HTML5 vem sendo desenvolvido desde 2004 (a versão quatro é de 1999), e trouxe grandes mudanças, adicionando novas *tags*, funcionalidades e tecnologias e excluindo algumas que já não eram mais pertinentes. Apesar da versão 5 ser compatível com as demais, os atributos excluídos não estão mais disponíveis (HENKE, 2009).

2.1.2.1.4 CSS

CSS é a abreviação para *Cascating Style Sheet*, que traduzindo para o português é Folhas de Estilo em Cascata. Silva (2008) diz que a linguagem CSS existe para trazer de volta a principal funcionalidade do (X)HTML, que de acordo com ele é "estruturar um documento web marcando com o elemento apropriado cada tipo de conteúdo que compõe o documento". Schulz (2017)

complementa dizendo que o CSS tem a capacidade de ampliar a linguagem HTML. Melhorando visualmente as páginas web, através de suas propriedades.

Ao explicar a sintaxe da linguagem, Moskovitz Joseph (2010) compara com a forma como uma frase é formulada em português, por exemplo. Ele fala da estrutura gramatical de uma frase, considerando sujeito, verbo e objeto. Com isso ele tem a intenção de mostrar que o CSS funciona de forma similar e estruturada, pois:

> "Existem elementos, cada um sendo um bloco de construção ou componente de um todo maior. Um elemento pode ter certas propriedades que especificam mais profundamente seus detalhes. Essas propriedades normalmente são atributos, mas também podem ser o próprio conteúdo do elemento. Por fim, um elemento tem certa posição dentro do documento relativa a todos os outros elementos, dando-lhe um contexto hierárquico." (MOSKOVITZ JOSEPH, 2010)

Schulz (2017) apresenta como umas das principais vantagens do CSS, o seu código enxuto. As linhas de códigos que antes eram feitas de forma manual e repetitiva, agora são especificadas em blocos que podem ser usados no decorrer da aplicação. A Fig.2.10 mostra duas linhas de código HTML que atribui características para o *<h3>*, mas toda vez que for necessário o uso do *h3* deverão ser feitas novamente. Deixando o manuseio do código cansativo para o desenvolvedor. Já a Fig. 2.11 apresenta um bloco de código CSS com as mesmas funcionalidades que as linhas acima, mas com o diferencial de que toda vez que o *h3* for utilizado, só é preciso citá-lo.

```
<h3><font face="Arial" size="5" color="#454545">Un texto
cualquiera</font></h3>
```

Figura 2.10 – Linha de Código com atributos HTML | Fonte: Schulz (2017)

```
h3 {
font-family: Arial;
font-size: 1.3em;
color: #454545;
}
```

Figura 2.11 – Linha de Código com Propriedades CSS | Fonte: Schulz (2017)

Sendo assim, o uso do CSS se tornou imprescindível em sistemas WEB modernos e é considerada uma ferramenta que nenhum programador quer abrir mão. A Fig.2.12 apresenta um comparativo das páginas Web com e sem o CSS, deixando ainda mais visível o que as figuras anteriores queriam demonstrar: o quão econômico e prático a linguagem CSS pode ser.

	100 páginas		10.000 páginas	
	Sin CSS	Con CSS	Sin CSS	Con CSS
Código fuente en Mb	3,8	1,6	380	160
Líneas de código fuente	85.000	32.900	8.500.000	3.200.000
Tablas	4900	400	490.000	40.000

Figura 2.12 – Comparação de Páginas Web com e sem CSS | Fonte: Schulz (2017)

2.1.2.1.5 Gerenciador de Pacotes *Composer*

O Composer é um gerenciador de pacotes e dependências em nível de projeto, ou seja, se adicionada uma biblioteca em um programa, as funcionalidades dela só estarão disponíveis para o mesmo. Não é uma ferramenta de uso global como outros gerenciadores, a exemplo do YUM e APT, por exemplo (GABARDO, 2017).

A sua documentação o descreve como:

> "Composer é uma ferramenta para gerenciamento de dependências em PHP. Ele permite que você declare as bibliotecas das quais seu projeto depende e ele as gerenciará (instalará / atualizará) para você."[5]

De acordo com Salomão (2018), além de realizar instalações e atualizações para o projeto, "ele também fornece recursos de carregamento automático para bibliotecas para facilitar o uso de códigos de terceiros". Também em sua documentação é exemplificado de forma simples o seu funcionamento.

Supondo um projeto com várias bibliotecas que, por sua vez, também dependem de outras bibliotecas, o Composer vai permitir que seja declarada as bibliotecas das quais depende. Descobre e instala as versões dos pacotes necessários e permitidos. Permite atualizar as dependências com apenas um comando.

Para a instalação do Composer, a exigência feita é que a linguagem PHP esteja numa versão superior a 5.3.2. Garantindo isso, basta seguir os passos indicados no site da documentação para o sistema operacional adequado.

2.1.2.1.6 VSCode

Visual Studio Code é um editor de códigos *open source* e é disponibilizado de forma gratuita pela Microsoft. É uma plataforma nova, lançada em 2015, mas poderosa. Com suporte

[5] https://getcomposer.org/doc/00-intro.md

para mais de 30 linguagens de programação e também para os três principais sistemas operacionais: Linux, MacOS e Windows (WANDERLEY, 2017).

Kahlert (2015), no próprio site da Microsoft, cita alguns dos principais recursos da plataforma:

- Organização do código em pastas de fácil manuseio;
- Atalho de teclado ajustável possibilitando uma navegação mais rápida;
- Coloração das sínteses e correspondência de colchetes para diversas linguagens;
- *IntelliSense* ;
- Poderoso com aplicações Node.js e C#;
- Suporte para Git;
- Pesquisa de texto em arquivos e pastas;
- Salva arquivos de forma automática;
- Diferentes temas e uma ampla configurabilidade.

A maioria desses itens podem ser vistos assim que o editor é aberto. A Fig.2.13 demonstra algumas de suas funcionalidades de forma prática e visual. Assim como foi dito por Kahlert (2015). O principal menu é o vertical, localizado na esquerda da página. Nele é feita a maior parte dos projetos desenvolvidos. Seguindo os ícones temos: as pastas, a lupa de pesquisa que abrange as pastas do projeto, o gerenciador de compartilhamento, o *run and debug*, o explorador e conexões remotas, as extensões, o *GitLens*, o *Live Share*, a conta do usuário e as configurações.

A melhor forma para entender seus recursos e funcionalidades é na prática. Para isto basta fazer o *download* e instalação pelo próprio site [6]. É de fácil compreensão, mesmo sem nenhum projeto para treino. Mas para um estudo mais abrangente, a Microsoft oferece instruções básicas com os passos iniciais.

2.1.2.1.7 Jetstream e Tailwind

O Tailwind é um *framework* CSS "que fornece um catálogo profundo de classes e ferramentas CSS que permitem que você comece a estilizar facilmente seu site ou aplicativo"(ANDRADE, 2020). Ao invés de criar uma classe com vários atributos do CSS, o Tailwind permite criar várias classes a uma classe já existente de forma rápida e pouco custosa. Além disso, de acordo com sua documentação [7], que é robusta e intuitiva, Tailwind oferece um *design* inteiramente responsivo.

[6] https://code.visualstudio.com/download
[7] <https://tailwindcss.com/docs>

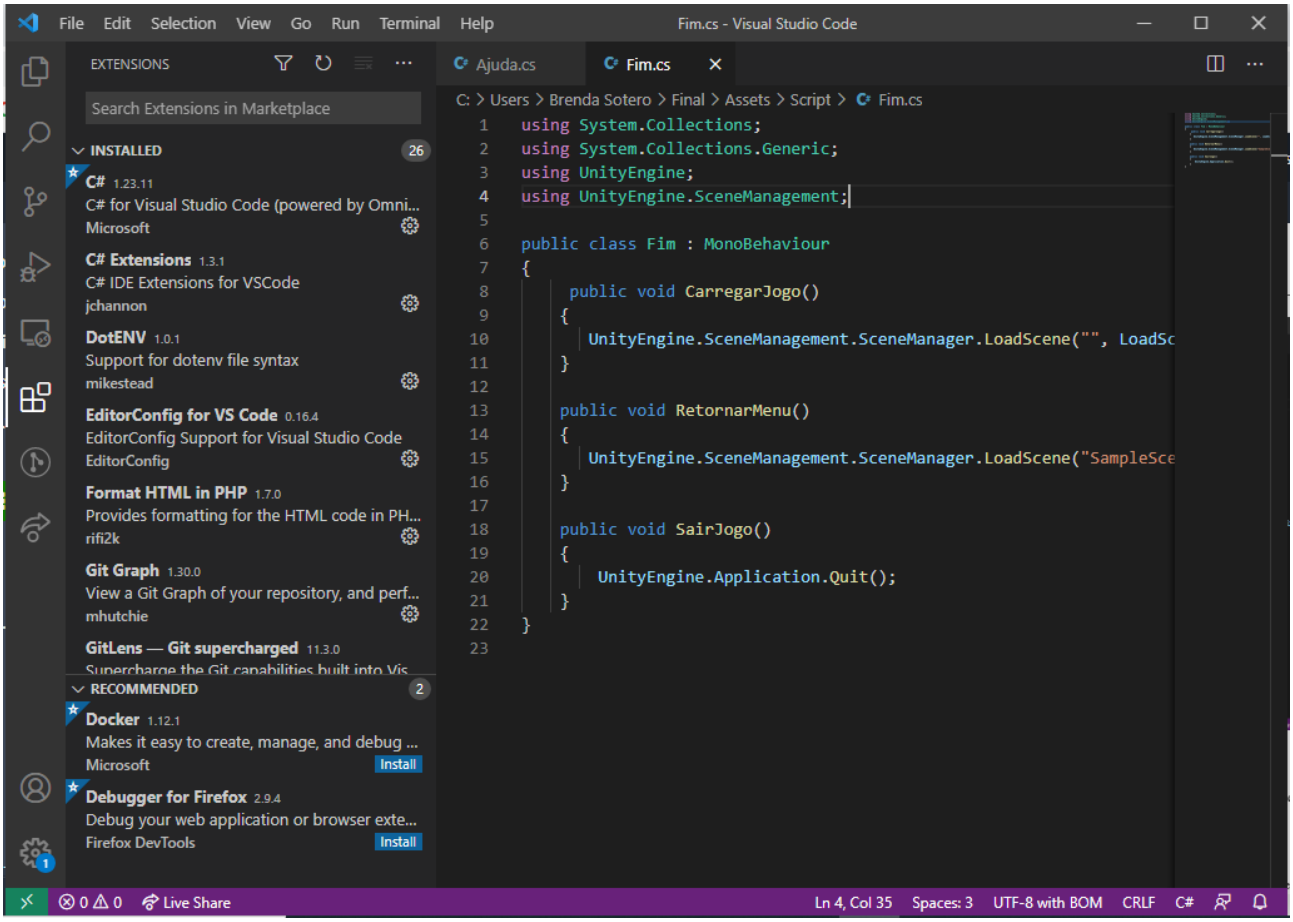


Figura 2.13 – Plataforma VSCode em Execução

Isso quer dizer que, usando suas classes de forma responsiva, ele se adapta a diferentes tipos e tamanhos de tela.

E o Jetstream, uma ferramenta projetada com o Tailwind, é definido pela própria documentação[8] como sendo "um kit inicial de aplicativo lindamente projetado para o Laravel e fornece...a implementação para login, registro, verificação de e-mail, autenticação de dois fatores, gerenciamento de sessão..."e outros recursos. É um importante instrumento em um projeto Laravel, porque através das suas funcionalidades ele garante mais segurança ao usuário.

Essas duas ferramentas serão usadas neste projeto, mas as instalações devem ser feitas depois de iniciar com o Laravel. A instalação do Jetstream pode ser seguida através da sua documentação. E sua versão vai depender da versão do Laravel. O Tailwind tem suas instruções na página principal de seu site [9].

[8] <https://jetstream.laravel.com/2.x/introduction.html>
[9] <https://tailwindcss.com/>

2.1.2.2 Iniciando um Projeto com Laravel

Com o ambiente local pronto, a instalação do Laravel já pode ser feita. Para realizar esta tarefa, basta consultar as documentações [10] do Laravel e seguir os passos indicados para cada sistema operacional. A página mostra uma visão geral do *framework*, as configurações iniciais e como iniciar um projeto. Também conta com uma opção para alterar as versões, mudando para a documentação da versão que desejar. Lembrando que a versão aqui utilizada será a versão 8.34.0.

Além do Laravel, para este projeto, será necessária a instalação do MySQL (que pode ser feita seguindo os passos da documentação[11], do kit Jetstream e do Tailwind. O Jetstream pode ser instalado via *composer* ou pelo próprio instalador do Laravel. É feito de uma forma muito simples, com poucas linhas de comando, como demonstrado no site [12]. Para evitar futuros conflitos no programa, essa instalação deve ser feita antes de dar início ao desenvolvimento da aplicação. E para o Tailwind as informações de instalação estão na sua documentação [13].

Instalações feitas e os passos para começar um novo projeto concluídos. Agora é só abrir o projeto, pelo VSCode e entender melhor o que é cada pasta contida na plataforma. A Fig.2.14 mostra todas as pastas e arquivos que são criados assim que um novo projeto é inicializado. Gabardo (2017) e Gomes (2019) detalham, de forma sucinta, o que cada uma faz:

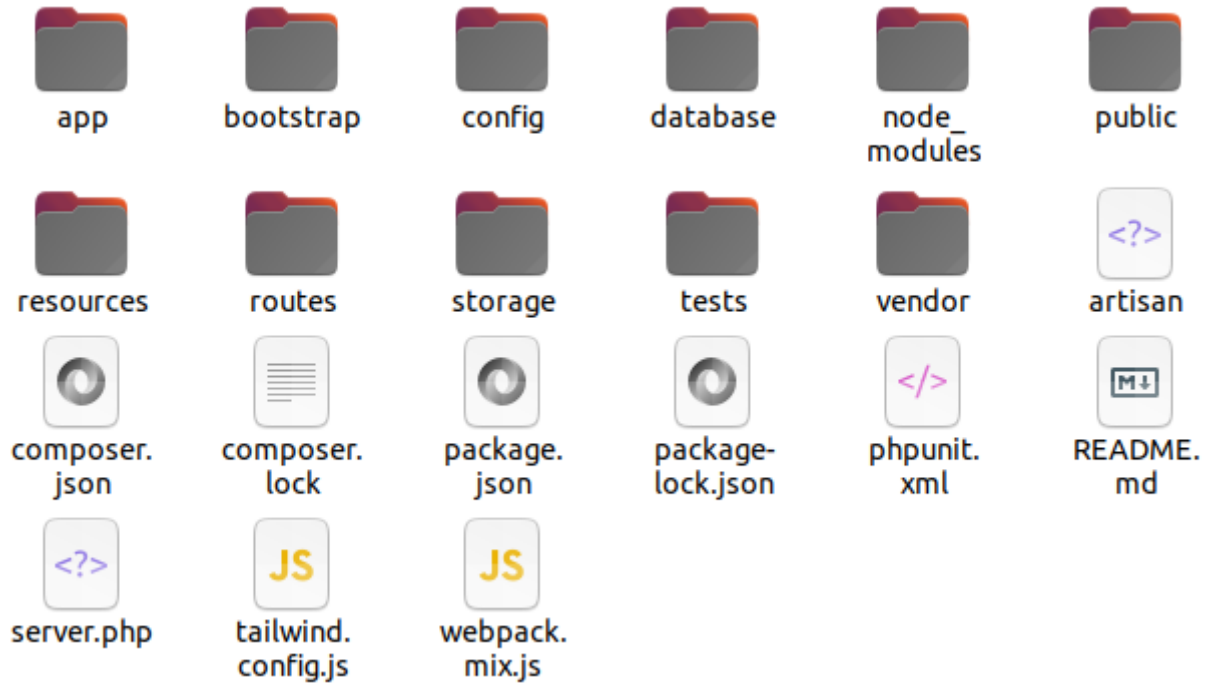


Figura 2.14 – Estrutura de Pastas do Laravel

[10] <https://laravel.com/docs/8.x/installation#environment-based-configuration>
[11] <https://dev.mysql.com/doc/>
[12] <https://jetstream.laravel.com/2.x/installation.htm>
[13] <https://tailwindcss.com/docs/guides/laravel>

- *app*: é a pasta que contém os *models* e *controllers*. É aqui que fica o código principal da aplicação;
- *artisan*: é o arquivo responsável por carregar, de forma automática, os recursos do Laravel;
- *bootstrap*: contém os arquivos responsável pela inicialização da estrutura. Não tem relação com o *framework Bootstrap*;
- *composer.json*: é o arquivo que define as dependências do projeto;
- *composer.lock*: gerado automaticamente quando as dependências são instaladas, guarda as informações como versão dos pacotes. Fazendo com que o projeto fique preso àquelas versões. Evitando possíveis conflitos;
- *config*: aqui ficam os arquivos de configuração da aplicação;
- *database*: é a pasta responsável pela integração da aplicação ao banco de dados, através das *migrations*, *seeders* e *model factories*;
- *.env*: arquivo com variáveis referentes à aplicação, como nome do projeto, acesso ao banco de dados, entre outras;
- *.env.example*: serve de exemplo para configurar o *.env*;
- *package.json*: tem as dependências para utilização do Node.js com o laravel;
- *package-lock.json*: também guarda as informações da dependência acima;
- *phpunit.xml*: contém as configurações para testes unitários;
- *public*: é uma pasta pública, ou seja, visível na web. Nela tem o arquivo *index.php*, que é por onde entram todas as solicitações para a aplicação e configurar o servidor Apache;
- *readme.md*: arquivo que contém diversas informações sobre o *framework*, licenças e indicadores de acesso a documentação;
- *resources*: onde ficam as *views* e os arquivos CSS, além de outros recursos;
- *routes*: dentre os arquivos existentes na pasta, tem-se o *web.php*, arquivo responsável por gerenciar as rotas da aplicação;
- *server.php*: servidor web embutido;
- *storage*: contém os arquivos gerados pelo *framework*, como modelos compilados de *blades* e caches;
- *tailwind.config*: contém as configurações para uso do Tailwind;

- *tests*: armazena os arquivos de testes unitários gerados;
- *vendor*: aqui ficam os arquivos do *framework*, dificilmente essa pasta precisará ser alterada;
- *webpack.mix*: empacotador de módulos para o Laravel.

Ainda existem arquivos e subpastas não citadas, mas elas serão vistas de acordo com a necessidade empregada pelo desenvolvimento do sistema. Com o ambiente já configurado e as informações básicas sobre o Laravel, suas ferramentas e funcionalidades. É hora de dar início ao desenvolvimento do projeto. Que será apresentado no capítulo seguinte.

2.2 Trabalhos Relacionados

Levando em conta o objetivo principal deste trabalho, outros serviços existentes e relacionados serviram de referência e estudo para a melhor idealização da aplicação a ser desenvolvida. Os serviços analisados seguem a mesma linha de lógica deste, que é alocar e gerenciar recursos considerando hora, dia, disponibilidade e preço. Além de incluir formas de pagamento *on-line* Eles são:

- Compras de passagens: A empresas de viação Pássaro Verde[14], por exemplo, realiza de forma mais complexa, as mesmas funcionalidades deste sistema;
- Agendar consultas: como exemplo o site Doutor Consultas[15];
- Marcar hora para ir a academia, como na SmartFit[16];
- E o mais parecido com o tema: Agendei Quadras[17] que apesar de ser um sistema *mobile*, ou seja, uma aplicação para celular, tem o mesmo propósito que este trabalho.

[14] <https://passaroverde.com.br/>
[15] <https://www.drconsulta.com/>
[16] <https://www.smartfit.com.br/reservas>
[17] <https://www.agendeiquadras.com.br/appAgendei/>

3 Desenvolvimento

Neste capítulo será abordado todo processo desenvolvimento da aplicação Agenda Guaranix. Desde o levantamento de pré-requisitos, que se dará em conjunto com os responsáveis pelo espaço, até a entrega de todas as funcionalidades e aprovação do mesmo. Passando pela modelagem de banco de dados, criação das tabelas através das *migrations* e *seeders*, criação das *blades*, *controllers* e rotas.

3.1 Levantamento dos Requisitos

O levantamento dos requisitos necessários para a aplicação foi feito de acordo com o *wireframe* já disponibilizado, que será apresentado a seguir. Um *wireframe* de um site é um protótipo de alta fidelidade, que tem por objetivo apresentar seu *design*, bem como o relacionamento entre suas páginas. Ou seja, ele é capaz de mostrar para um usuário como será o site, com seus campos e botões de acesso, mesmo sem ter o *backend* da aplicação.

As imagens que serão apresentadas em seguida estão divididas em três repartições:

- O *wireframe*;
- Os requisitos e funcionalidades identificadas a partir do *wireframe*;
- As mudanças que serão realizadas no desenvolvimento do software e que não estão inclusas aqui.

Além das mudanças elucidadas nas figuras Fig.3.1 até Fig.3.12, também será acrescentada uma tabela de preços. Que se distinguirá de acordo com a quadra escolhida e o dia e hora selecionado. Como visto também, o site terá as telas de *Login* e Cadastro em comum, e duas áreas distintas: a do cliente que só vai ter acesso ao seu histórico de agendamentos, ao seu perfil e a possibilidade de realizar uma nova solicitação na agenda; e a do administrador, que tem passe livre para as demais informações.

O próximo passo a ser realizado é a modelagem do banco de dados do sistema.

1. Login

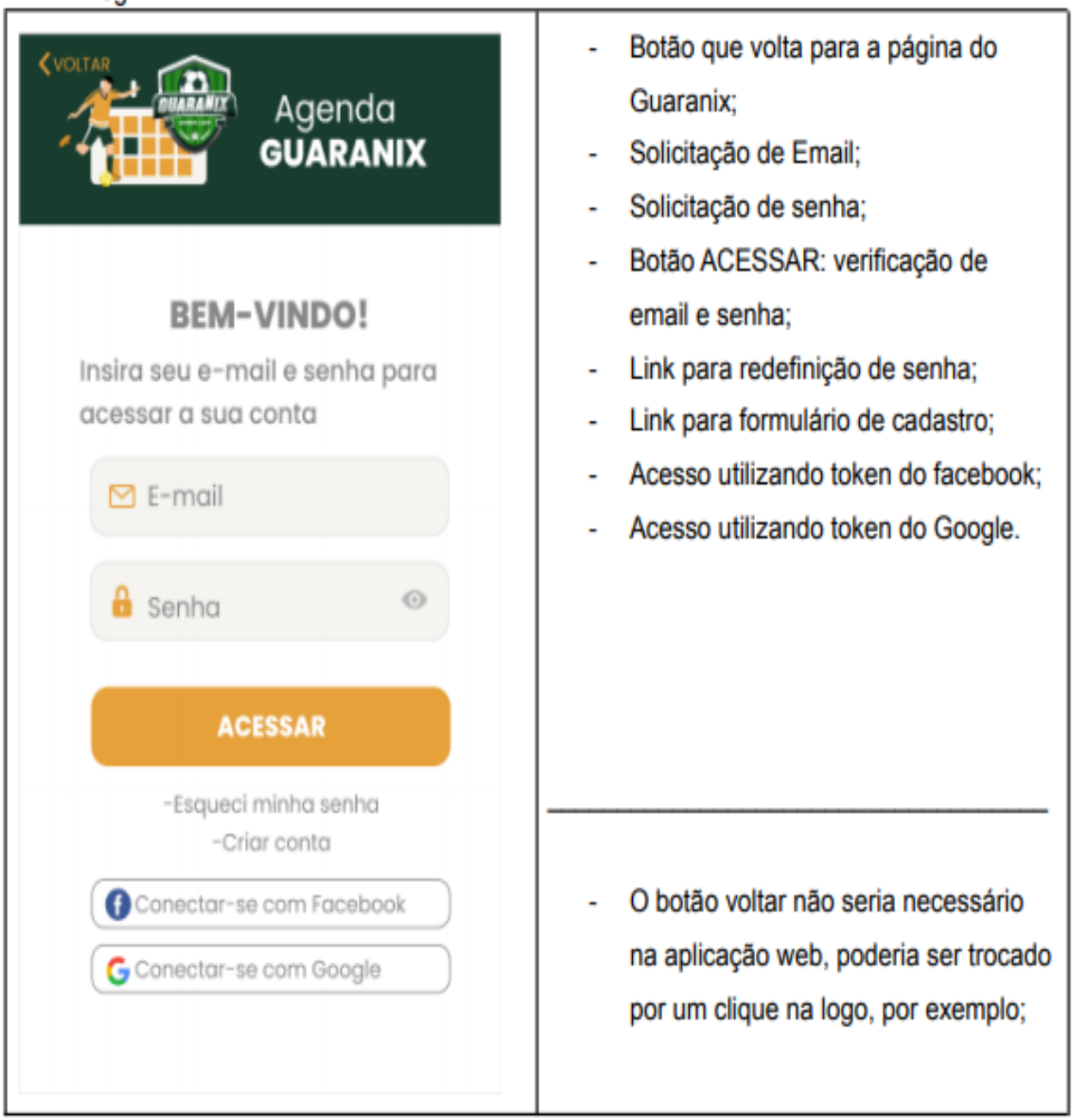


Figura 3.1 – Tela de Cadastro

2. Cadastro

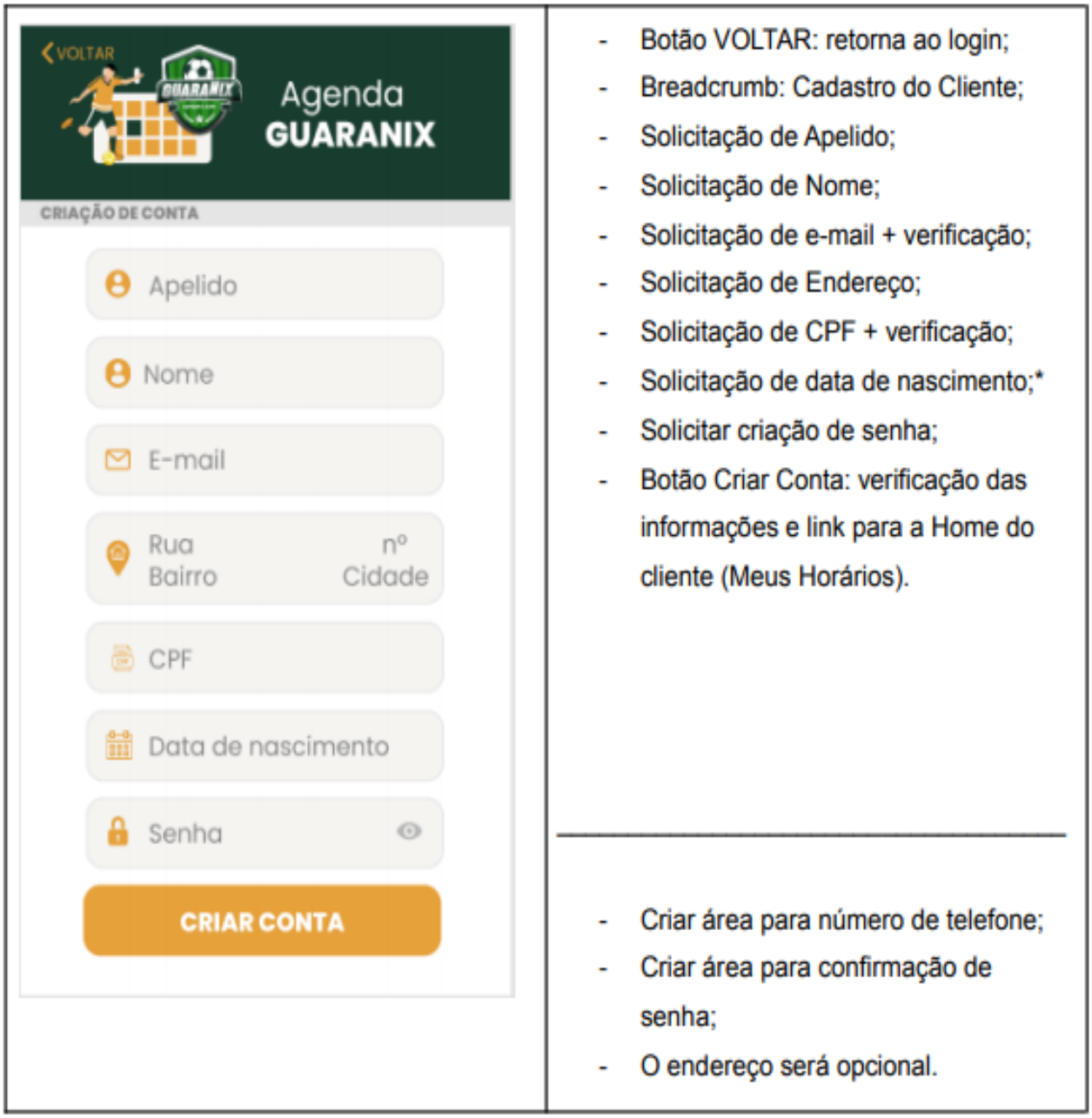


Figura 3.2 – Tela de Cadastro

3. Área do Cliente
    a. Meus Horários: (tela HOME)

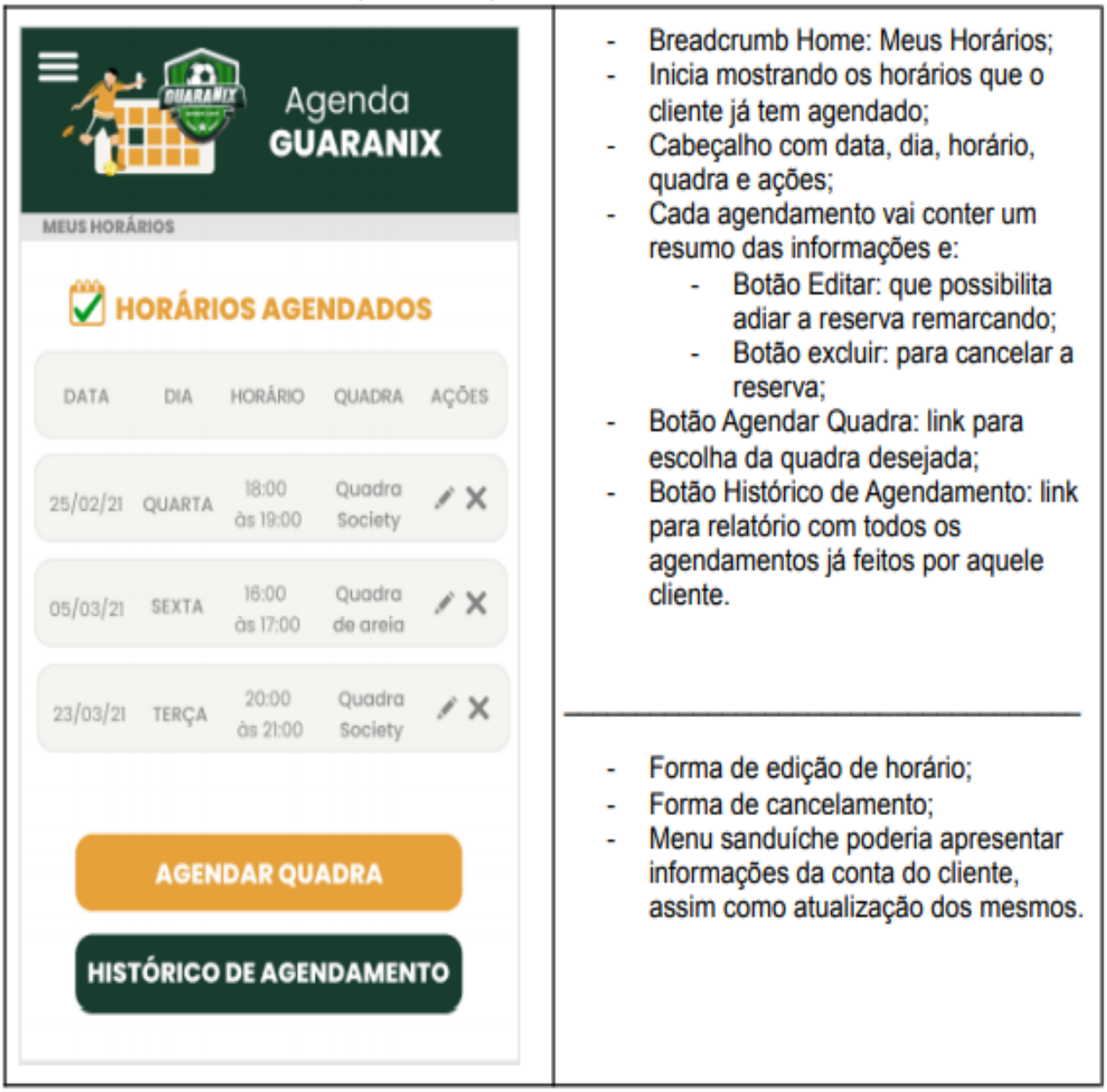


Figura 3.3 – Área do Cliente - Tela *HOME*

b. Quadras

Figura 3.4 – Área do Cliente - Selecionar Quadra

c. Agenda da Quadra

Figura 3.5 – Área do Cliente - Selecionar Horário Disponível

d. Confirmação dos Dados de Agendamento

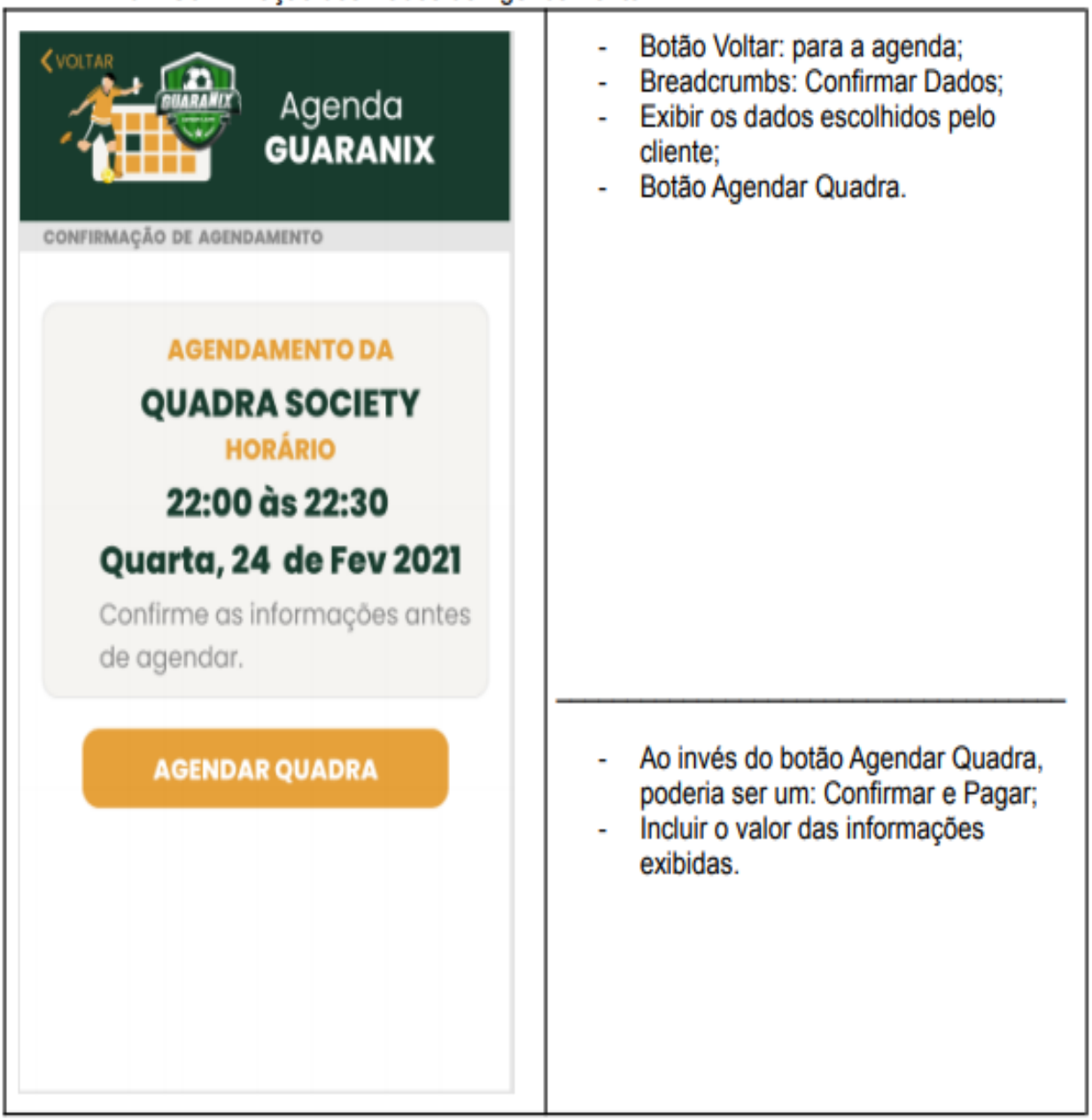


Figura 3.6 – Área do Cliente - Confirmação do Dados Selecionados

e. Protocolo de Agendamento

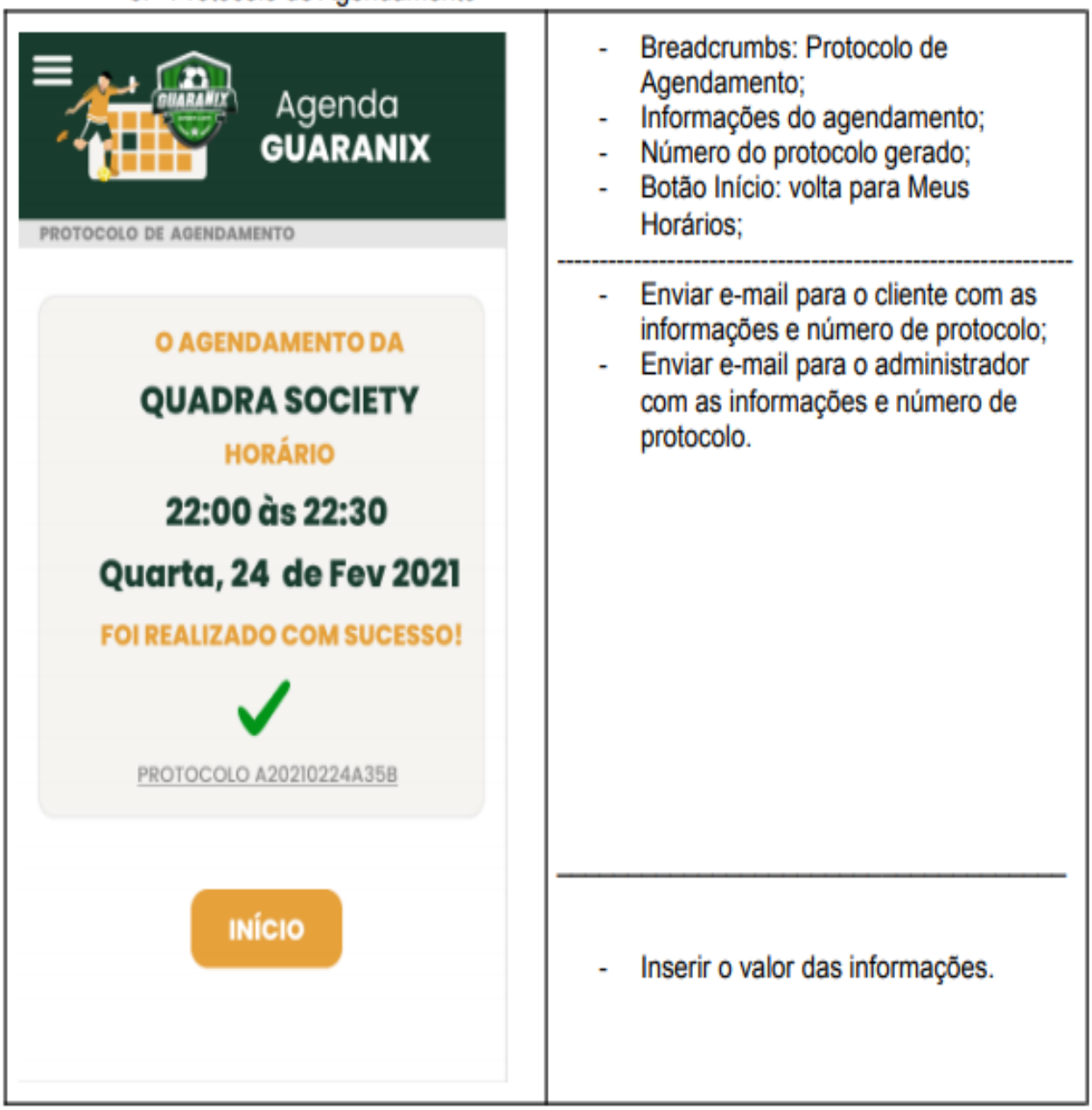


Figura 3.7 – Área do Cliente - Agendamento Concluído

4. Área do Administrador
    a. Início (Home)

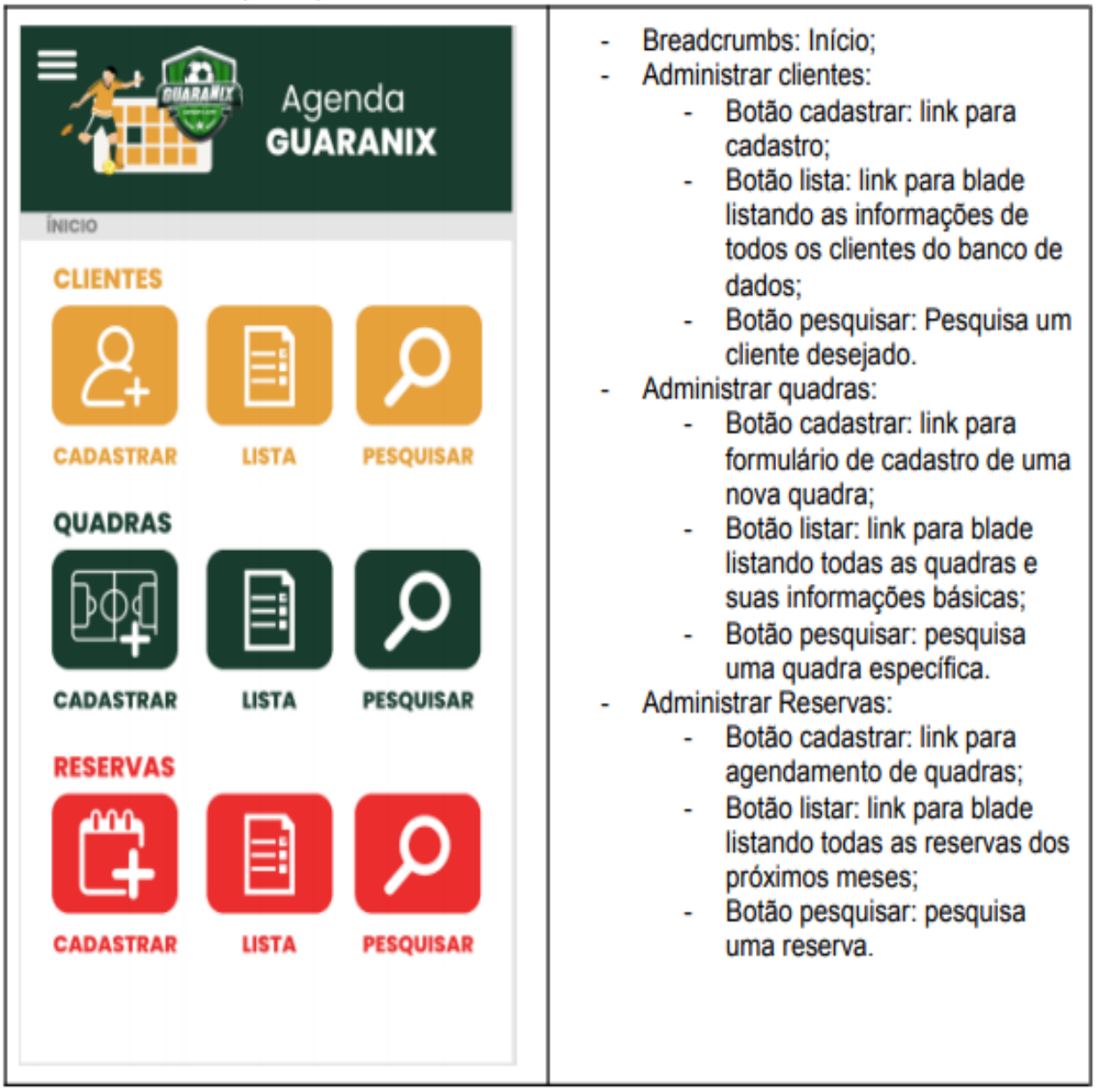


Figura 3.8 – Área do Administrador - Tela *Home*

b. Lista de Clientes

- Botão Voltar: volta para pagina inicial do admin;
- Breadcrumbs: Clientes;
- Botão Filtrar: podendo ser por nome, apelido e número;
- Cabeçalho das informações do cliente com nome ou apelido, telefone, última reserva e ações;
- Listagem da tabela de clientes que estão no banco de dados, podendo:
  - Botão editar: edita e atualiza as informações do cliente desejado;
  - Botão Excluir: exclui e apaga o cliente do banco de dados;
  - Marcação do status do cliente.

- A tela lista de clientes poderia conter uma funcionalidade de pesquisa no canto esquerdo superior;
- Também poderia conter um botão para adicionar um novo cliente no canto superior direito.

Simplificando a página inicial e a utilização do sistema.

Figura 3.9 – Área do Administrador - Lista dos Clientes Cadastrados

Figura 3.10 – Área do Administrador - Listas das Quadras Cadastradas

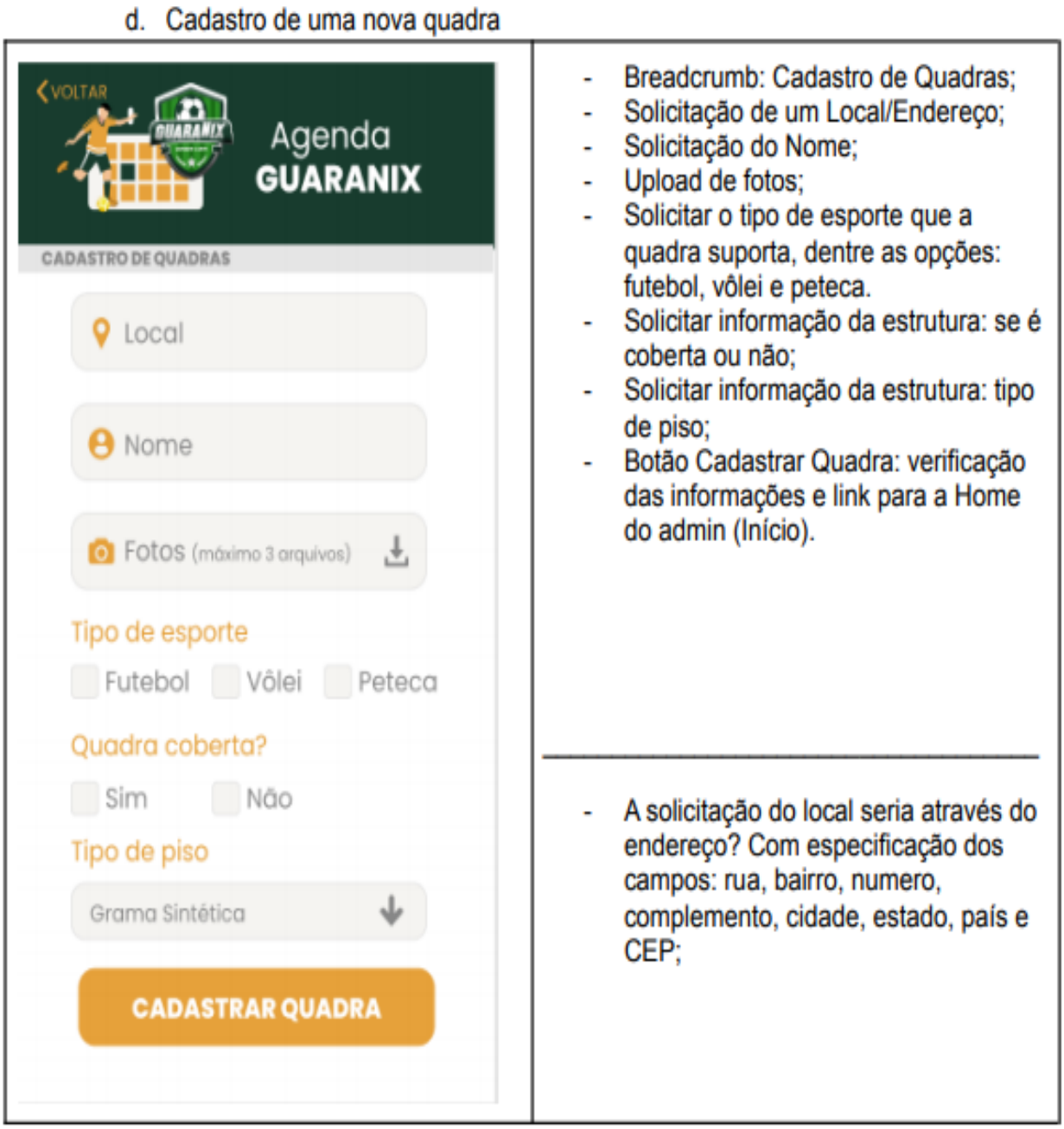


Figura 3.11 – Área do Administrador - Cadastro de Quadras

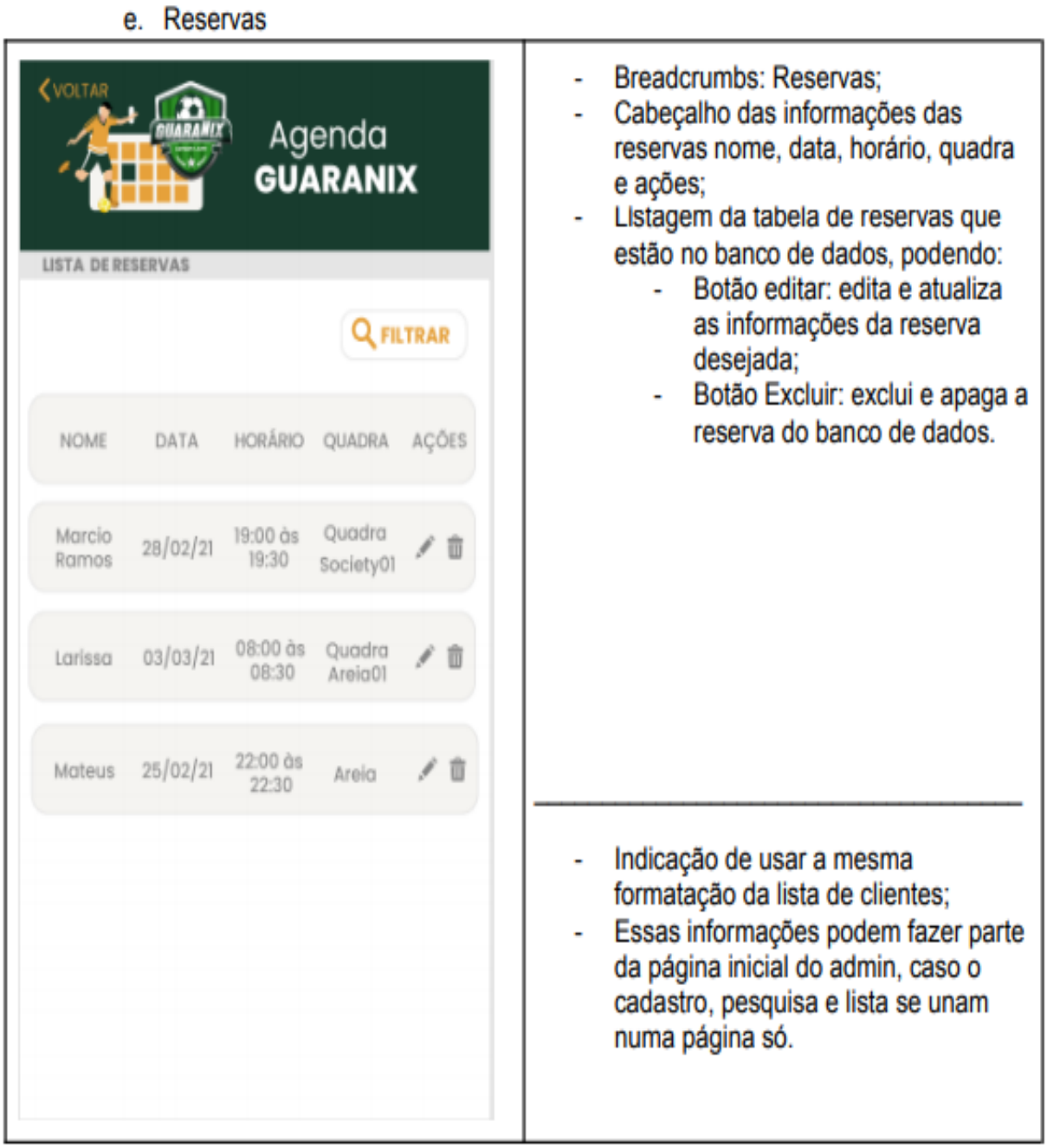


Figura 3.12 – Área do Administrador - Listas das Reservas

3.2 Modelagem do Banco de Dados

Como explanado no capítulo de Fundamentação Teórica, o sistema de gerenciamento de banco de dados que está sendo utilizado é o MySQL, da Oracle Corporation. Para realizar a modelagem das tabelas e seus relacionamentos foi usado como programa auxiliar o MySQL Workbench, de fácil compreensão e disponível a todos.

Para introduzir brevemente o MySQL, Bento (2021) o conceitua como sendo um banco de dados onde as informações são guardadas em estruturas chamadas tabelas, tendo cada linha de uma tabela como um novo registro. E compara, para exemplificar, com uma planilha de maior capacidade. O MySQL é baseado no modelo relacional e utiliza a linguagem SQL para fazer requisições ao banco (MARQUES; MEDEIROS; PEREIRA, 2018). A própria Oracle[1] caracteriza o banco de dados relacional como: "um tipo de banco de dados que armazena e fornece acesso a pontos de dados relacionados entre si".

Voltando ao desenvolvimento, a Fig.3.13 detalha como ficou a especificação do banco de dados do sistema, com as tabelas e suas relações. A seguir serão apresentados tópicos que darão uma breve descrição sobre como as tabelas do banco de dados agirão no sistema. Registros e relações entre tabelas não serão detalhados.

- Um usuário terá em seu nome contratos de locações com informações a respeito do agendamento, data, valores, pagamentos e afins;
- Cada locação consta um Agendamento de um Slot e cada slot tem seu preço. Os slots, na prática, são os horários a serem agendados;
- Os recursos, que representam as quadras neste problema, têm um tipo (se é quadra *society* ou de areia, por exemplo), um `tipo_esportes` que o recurso suporta. E também pode-se adicionar fotos só mesmo;
- A relação do recurso com os slots é bem simples: cada recurso gera sua própria tabela de slots. Isso significa que cada quadra terá seus próprios horários de locação;
- A tabela de Agendamento faz uma ponte das informações de slot, dia e recurso selecionados para a tabela de Locações;
- Cada contrato de locação tem a possibilidade de que seja efetuado o pagamento. E a tabela de Pagamentos também consta com tipos, como cartão de crédito, boleto e afins;
- Os usuários também são membros de um grupo. Todos os cadastros feitos colocarão o usuário no grupo de Clientes. A determinação de um gestor é feita internamente no sistema.

[1] https://www.oracle.com/br/database/what-is-a-relational-database/

É importante ressaltar que no decorrer do projeto, mudanças na modelagem do sistema são necessárias. Essa versão do Banco de Dados é a mais recente até a finalização deste trabalho. Mas possivelmente existirão outras narrativas com a evolução do projeto.

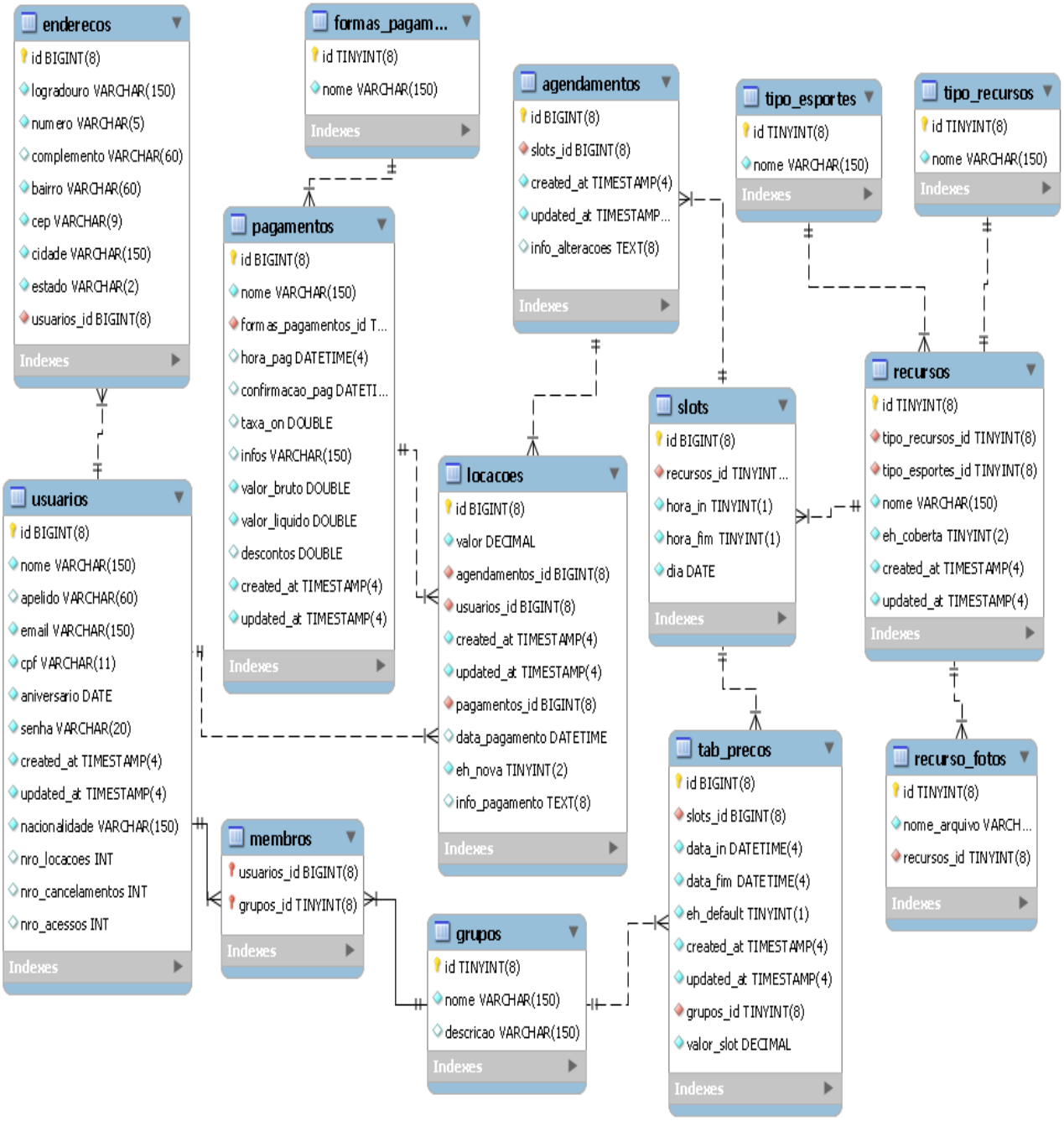


Figura 3.13 – Modelagem do Banco de Dados da Aplicação

3.3 Implementação

Com as instalações já feitas e o projeto criado de acordo com as especificações necessárias. Como início da implementação do sistema têm-se que criar um novo banco de dados especificando nome, senha e permissões de acesso. Este processo pode ser feito seguindo a própria documentação do MySQL[2].

A primeira configuração a ser realizada é no arquivo *.env*, que fica na raiz do projeto. Ele recebe as informações do seu ambiente de desenvolvimento local e do seu projeto. Quando se abre o projeto pelo VSCode, ao ir na área onde ficam as pastas, é fácil de encontrar um arquivo com o nome *.env.example*, como mostra na Fig.3.14. Depois basta fazer uma cópia e salvá-la como *.env* e inserir as informações da aplicação e do banco de dados. Também podem ser inseridos outros parâmetros como um e-mail alternativo para testes.

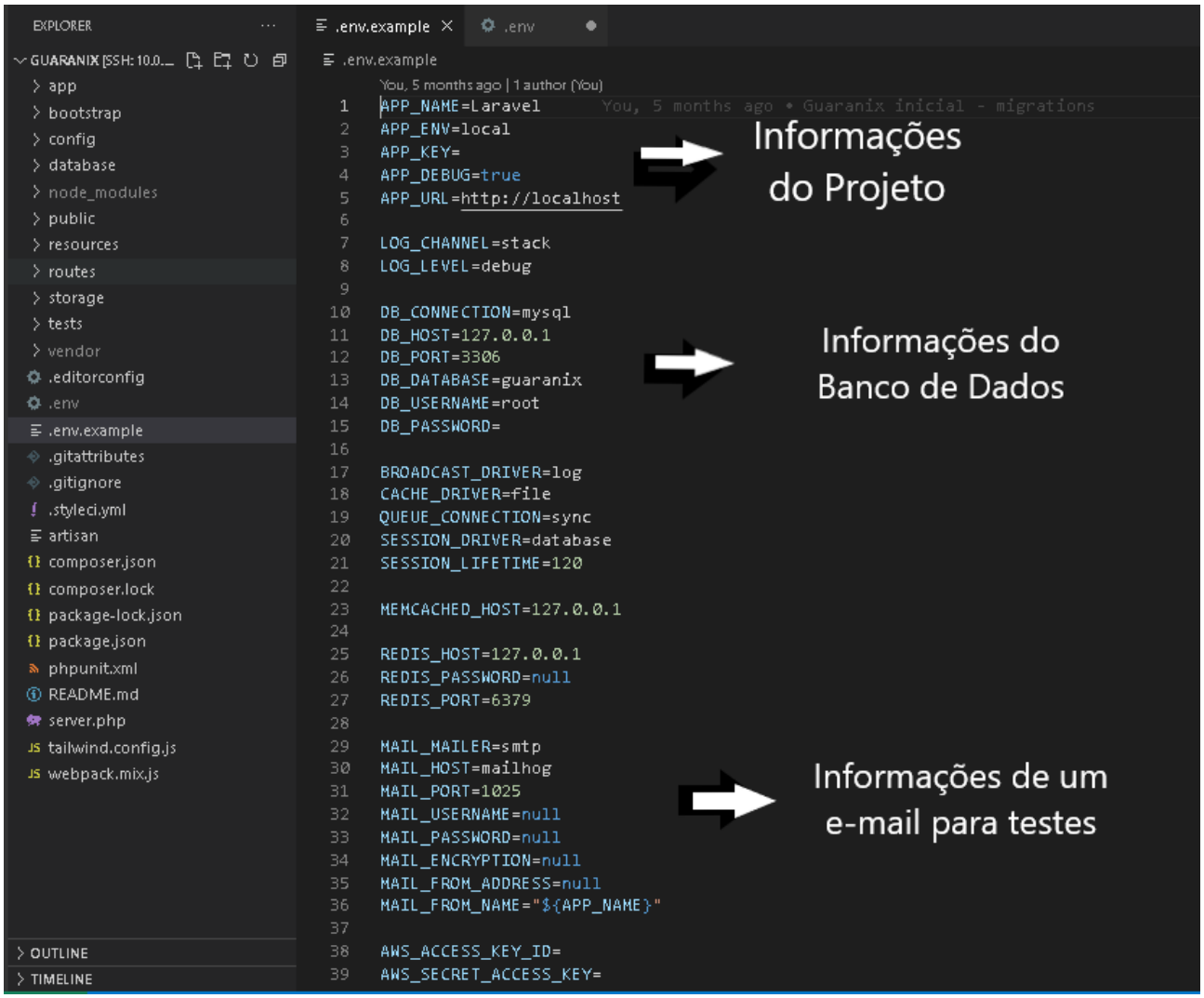


Figura 3.14 – Exemplo Arquivo *.env*

[2] <https://dev.mysql.com/doc/refman/8.0/en/tutorial.html>

3.3.1 Instanciando o Banco de Dados

3.3.1.1 *Migrations*

Para facilitar o entendimento do que será dito aqui, basta fazer a leitura acompanhando a Fig.3.15. Que é um exemplo de criação da tabela Recurso através do conceito de *migration*. A documentação do Laravel define *migration* como sendo um controlador de versões do banco de dados, que permite definir e compartilhar a definição do esquema do mesmo.

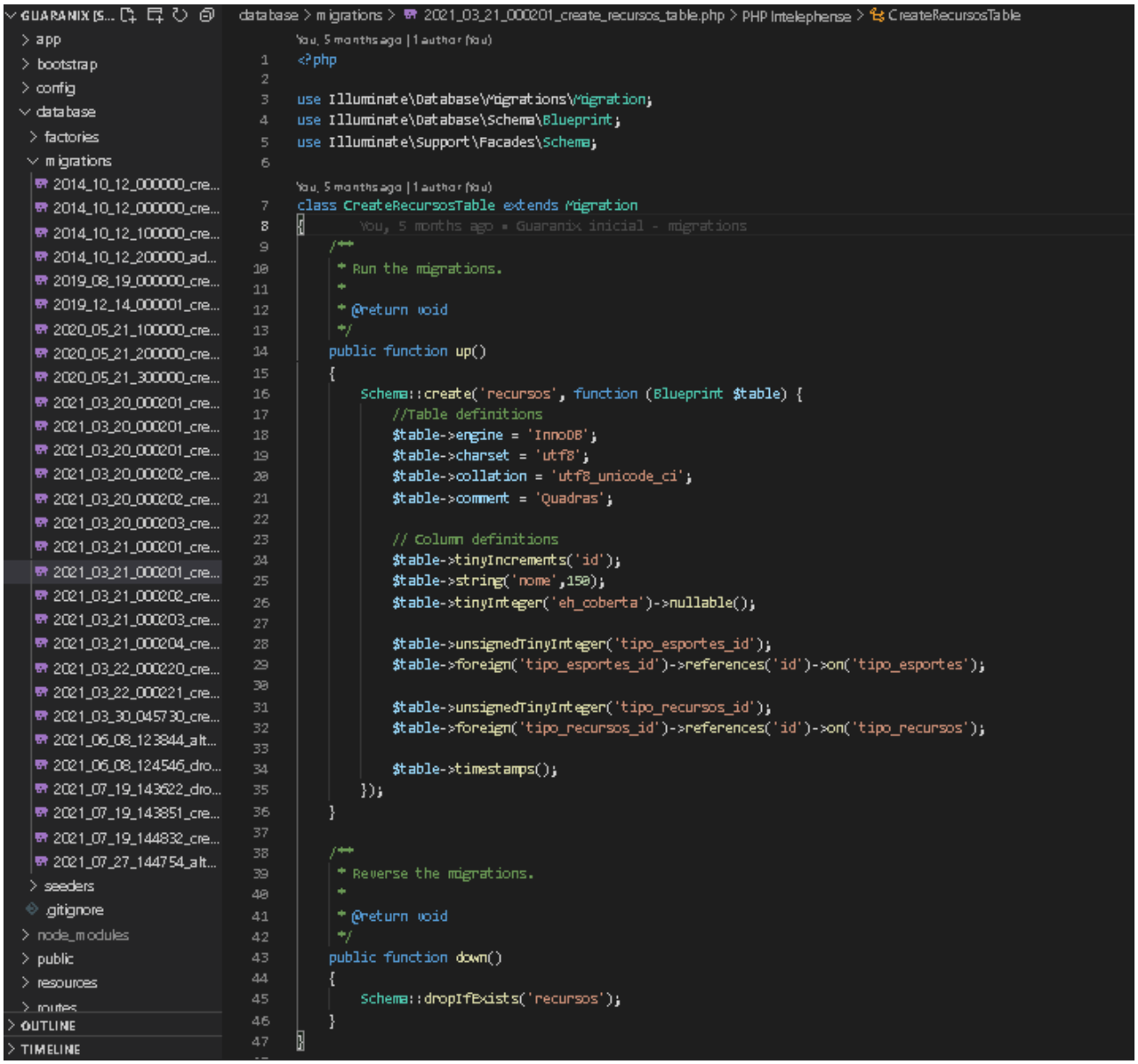


Figura 3.15 – *Migration* - Criação da Tabela Recurso

A subpasta *migrations* é acessada pela pasta *database*, onde ficam as informações do banco de dados do sistema e possibilita configurá-las. Aqui é importante que o *.env* esteja devidamente configurado para que seja possível produzir as tabelas que serão instanciadas.

Existem alguns comandos para a criação de uma nova classe de *migration*, neste trabalho

será usado o comando abaixo (Fig.3.16). Ele gera uma *model* com o nome especificado e uma *migration* com o mesmo nome. Ao rodar a linha de código no terminal, dentro do projeto, serão gerados arquivos nas pastas *app/models* e *database/migrations*.

```
php artisan make:model Recurso --migration
```

Figura 3.16 – Comando para Criação da *Migration* Recursos

As *Models* serão vistas posteriormente de forma mais aprofundada, mas basicamente ela é a ponte permitindo acessar, alterar e validar informações de uma determinada tabela no banco de dados (VASCONCELOS, 2017).

Quando um arquivo é gerado no Laravel, ele já costuma vir com uma pré-configuração definida. A *migration* já vem com as classes *Migration*, *Blueprint* e *Schema* importadas e possui duas funções:

- *Up*: cria tabela e atualiza informações do banco;
- *Down*: capaz de reverter alguma mudança ou atualização feita na tabela.

Na *migration*, além de definir as colunas da tabela, também é criado o relacionamento entre elas. Isso é feito através das *foreign keys* que sempre referenciam à coluna "id"da outra tabela. Como pode ser visto na Fig.3.15 onde a *foreign key* "`tipo_recursos_id`"referencia a coluna "id"da tabela `tipo_recursos`. E o mesmo acontece com a tabela `tipo_esportes`. Este processo deve ser feito para todas as tabelas especificadas na modelagem do banco de dados. Exceto algumas que já vêm configuradas no programa. Como a Users, que entraria no lugar da tabela de usuários. Para que cada tabela seja salva, é preciso executar a linha de comando da Fig.3.17;

```
php artisan migrate
```

Figura 3.17 – Comando para Salvar a *Migration* Recursos

3.3.1.2 Seeders

Por convenção, todas as colunas de uma tabela são instanciadas sem conter nenhum valor ou parâmetro. Para que seja possível adicionar informações às tabelas existe o *seeder*. Sua estrutura é básica e de fácil compreensão. E sua função é armazenar um *array* de dados gerados pelo próprio desenvolvedor que serão carregados no banco de dados quando for requisitado, como pode ser visto na Fig.3.19. Para criar um *seeder* basta executar o comando da Fig.3.18.

A pasta do *seeder* também fica no diretório *database*. Para que um *seeder* atue da forma desejada, não basta só criá-lo e preenchê-lo. Dentro de sua pasta já existe um arquivo

```
php artisan make:seeder RecursoSeeder
```

Figura 3.18 – Comando para Criar um RecursoSeeder

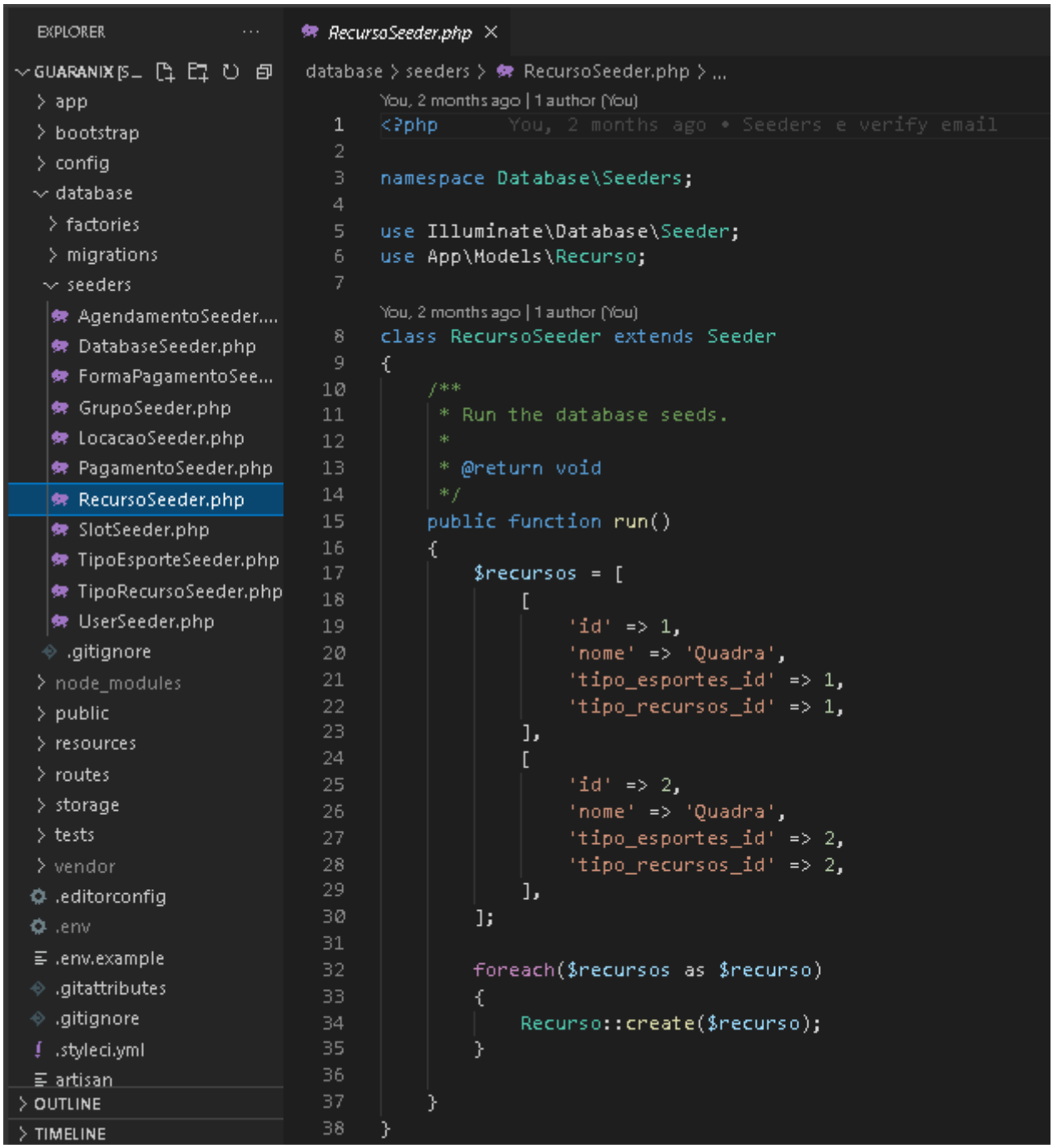


Figura 3.19 – Classe RecursoSeeder

chamado *DatabaseSeeder.php* que contém o método *call*. A classe de *seeder* recém criada deve ser adicionada dentro desta função (Fig.3.21) para que seja possível a execução desse *seeder*. Com essas configurações acertadas, para que as informações do *array* cheguem ao banco de dados, tem que rodar as linhas de comando da Fig.3.20. O primeiro é para dar um *refresh* no

banco de dados e o segundo é para, de fato, salvar os dados do seeder no banco.

```
php artisan migrate:fresh
[redacted]
php artisan db:seed
```

Figura 3.20 – Comando para Salvar as Informações do RecursoSeeder

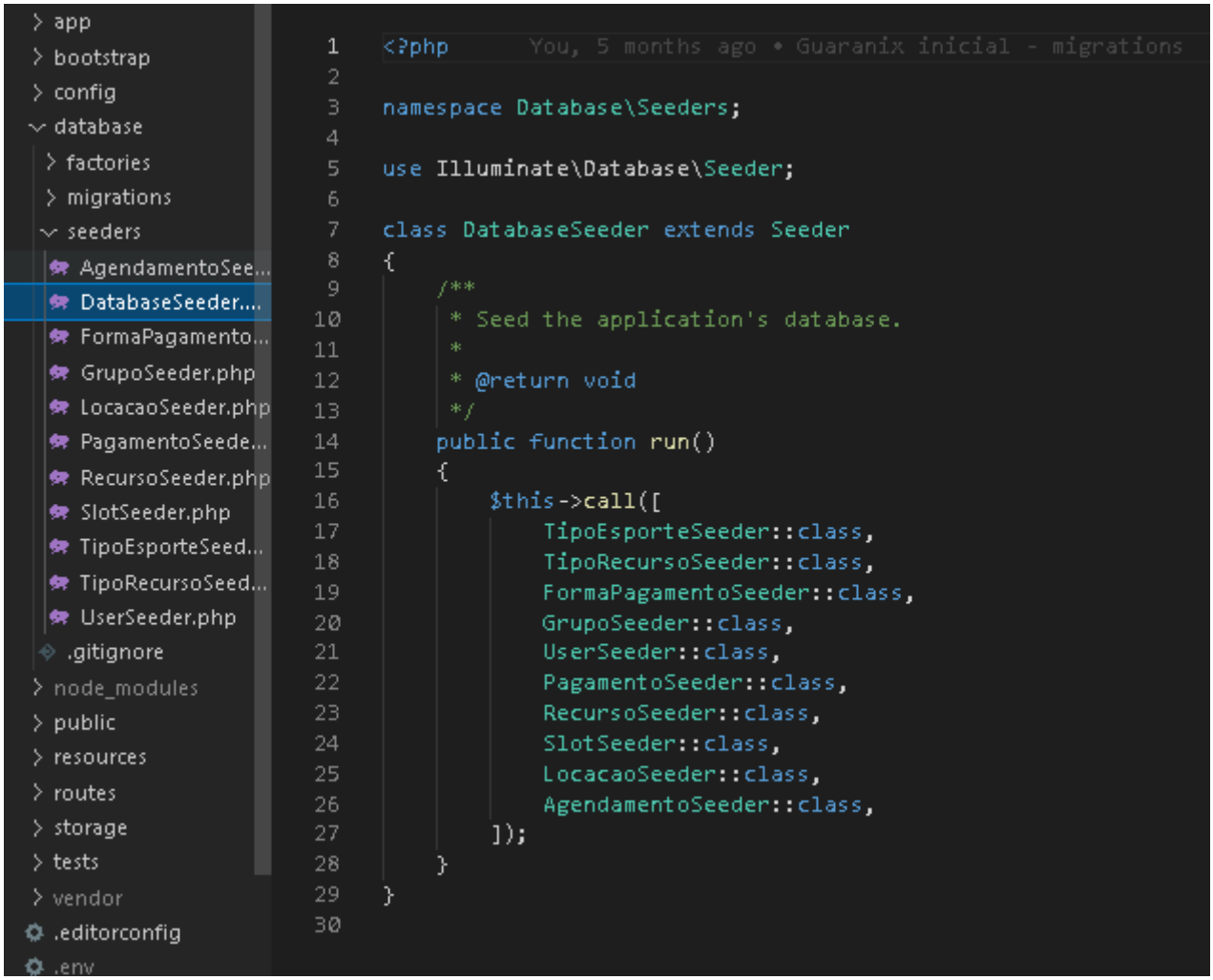


Figura 3.21 – Classe DatabaseSeeder

3.3.2 Padrão de Projeto

O Padrão de Projeto seguido pelo Laravel é o MVC, que já teve seu conceito elucidado no capítulo anterior. Sendo assim, esta aplicação foi implementada em cima destes três pilares: *models*, *views* e *controllers*. A seguir, além de aprofundar mais nas características destes três fundamentos, eles serão correlacionados com exemplos práticos do projeto. Para facilitar a compreensão tanto dos conceitos quanto no que foi desenvolvido no trabalho.

3.3.2.1 *Model*

Para falar de *model* primeiro é preciso introduzir o conceito de Eloquent, que é um *Object Relational Mapping* (ORM) ou mapeador de objeto-relacional, incluído no Laravel. Um ORM nada mais é que uma técnica que visa reduzir o tempo de resposta de desenvolvimento usando banco de dados relacional. E o Eloquent vai além de um ORM, abstraindo "toda a complexidade da interação com os bancos de dados utilizando as Models para interagir com cada tabela"(VASCONCELOS, 2017).

Com a execução do código apresentado pela Fig.3.16 a *model* para a tabela Recurso já está criada. Podendo ser acessada pelo caminho *app/models/Recurso*. Assim que ela foi criada, as classes *HasFactory* e *Model* já foram chamadas automaticamente, enquanto criou-se a classe Recurso estendendo a *Model*. As Figuras 3.22 e 3.23 mostram como ficou a *model* Recurso depois de definir os atributos e os relacionamentos.

Vasconcelos (2017) cita algumas convenções que devem ser usadas e que também estão na documentação do Laravel, mais precisamente na parte *Eloquent Model Convention*[3]. Vale ressaltar que essas convenções também servem para as *migrations*. Elas são:

- Os nomes das tabelas são padronizadas usando o plural do nome da classe. Ou seja, a classe Recurso tem como nome da sua tabela a palavra recursos. A linha 15 da Fig.3.22 mostra essa especificação do nome da tabela feita manualmente;
- Cada tabela tem uma coluna "id"como *primary key*, sendo sempre um valor positivo crescente;
- Toda tabela possui a coluna *timestamp* com a data de criação, *created_at*, e a de atualização, *updated_at*;
- Por padrão uma instância que acabou de ser criada, como as colunas 'nome' e '`eh_coberta`', não contém valor de atributo;
- As *models* irão se conectar com o banco de dados através do arquivo padrão *config/database.php*.

Para finalizar, a Fig.3.23 exemplifica a relação entre as tabelas através das funções *slots* e *tabelas*, suas classes devem ser chamadas no começo da *model*. No Laravel, as relações do Eloquent são feitas por esses métodos, no exemplo da classe Recurso tem-se dois tipos de relacionamento: o *hasOne* (um para um) que na prática significa que um recurso só pode estar ligado a um único tipo; e o *hasMany* (um para muitos, ou seja, um recurso pode ter vários slots. Outros tipos de relacionamento são definidos na documentação.

[3] <https://laravel.com/docs/8.x/eloquent>

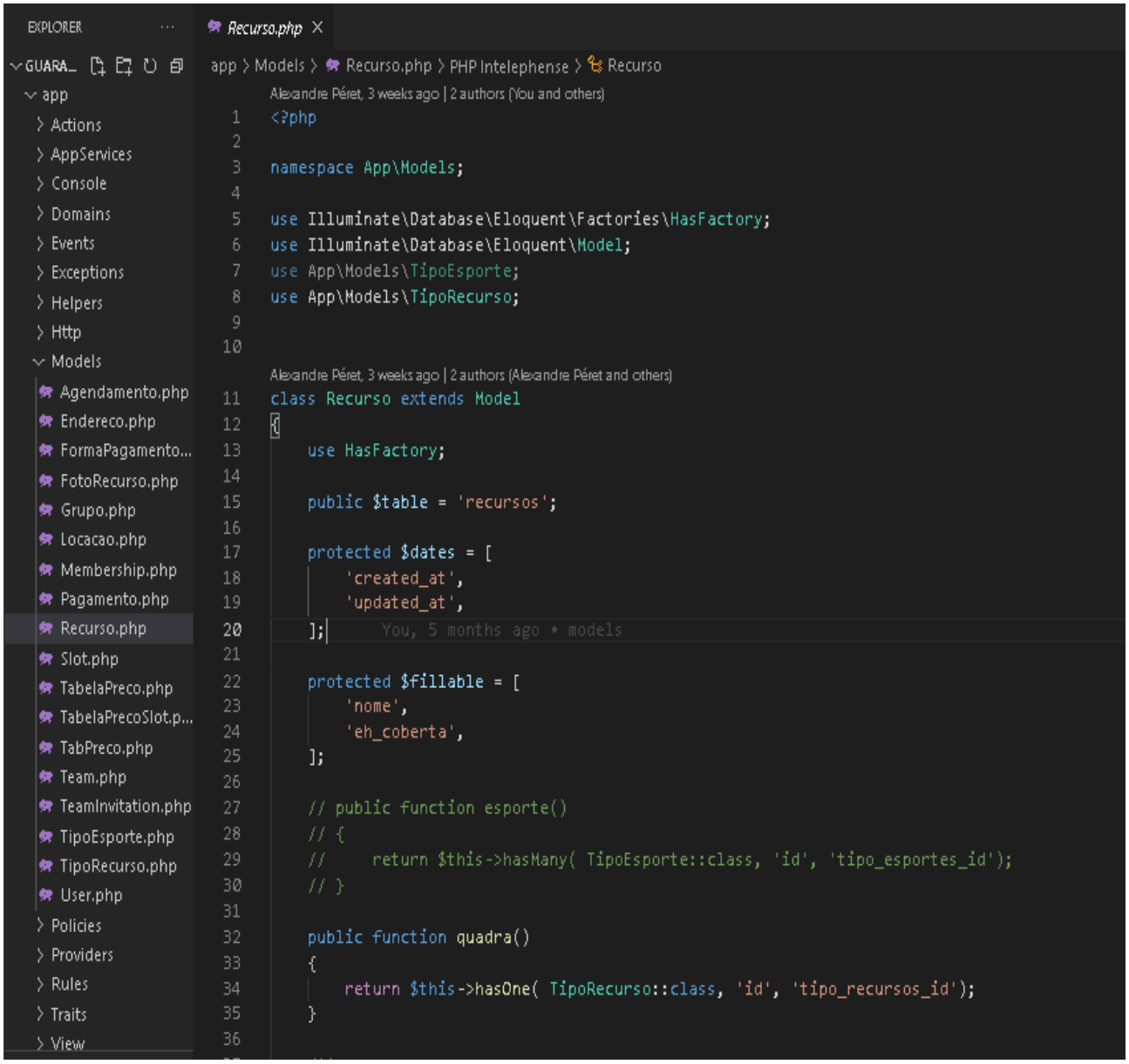


Figura 3.22 – *Model* da Tabela Recurso

3.3.2.2 *View*

Pela documentação do Laravel as *views* fornecem uma maneira mais simples de lidar com o HTML. Sua aplicação se baseia na separação da lógica dos *views* e da lógica de apresentação. Por padrão elas são armazenadas no diretório *resources/views*. Para iniciá-la não é necessário uma linha de comando, basta criar um novo arquivo na pasta com a extensão *.blade.php*.

O Blade é um padrão de template *engine* usado pelo Laravel. Ele é encarregado de facilitar a criação de páginas HTTP e outros processos relacionados, diminuindo sua complexidade. Este padrão também conta com a ajuda de outra ferramenta que compartilha do mesmo ideal facilitador, o Tailwind. Que facilita a criação dos templates e reduz consideravelmente as linhas de código.

A Fig.3.24 exemplifica a estruturação de uma *view*. Ela representa o visual da listagem de

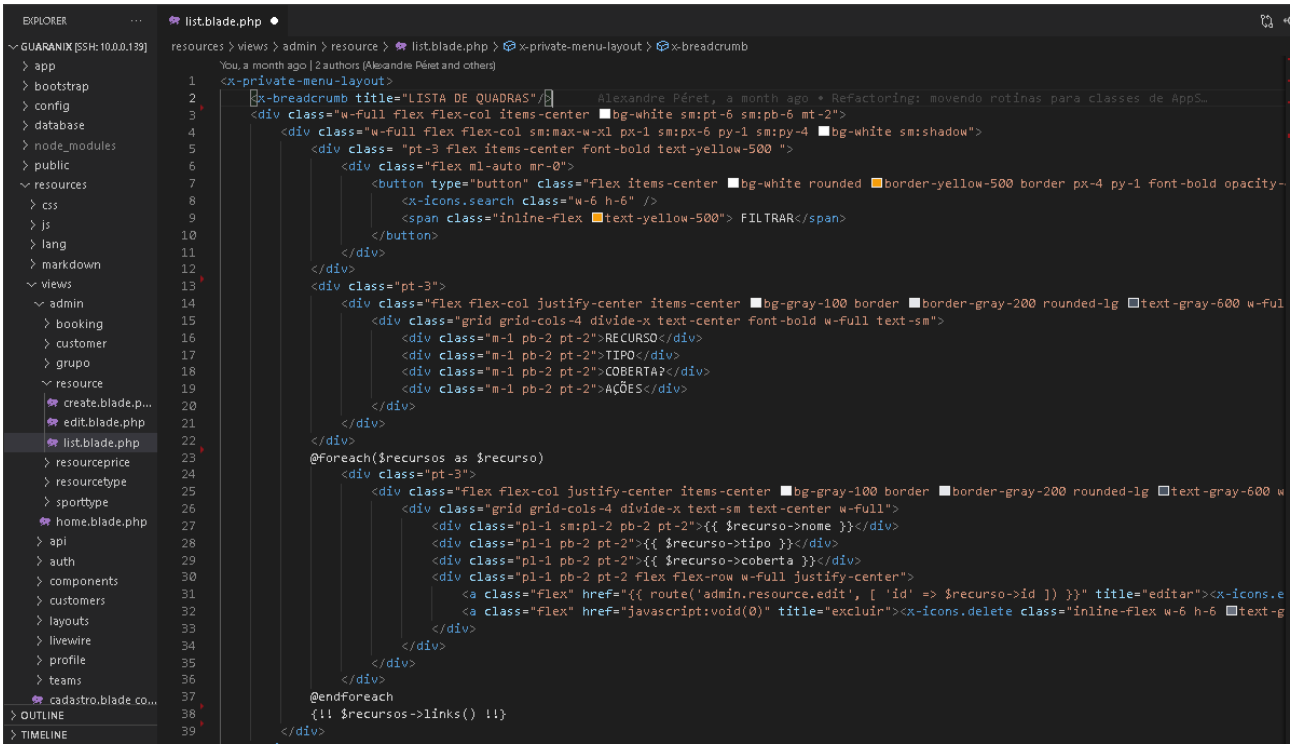
```php
    /**
     * Obtém os slots configurados para o recurso.
     *
     * Esta tabela funciona como um template de slots
     * para o recurso, contendo os horários para cada
     * dia da semana.
     *
     * @return void
     */
    public function slots()
    {
        return $this->hasMany(Slot::class, 'recursos_id', 'id');
    }

    /**
     * Obtém as tabelas de preços do recurso
     */
    public function tabelas()
    {
        return $this->hasMany(TabelaSlot::class, 'recursos_id', 'id');
    }

}
```

Figura 3.23 – Continuação do *Model* da Tabela Recurso

recursos e exibe as informações recebidas como parâmetro. Como pode ser visto nos resultados deste trabalho, pela Fig.4.22. Voltando à Fig.3.24, O Tailwind pode ser reconhecido pelos parâmetros que estão dentro da 'class' de uma . Um 'pt-6', por exemplo, significa *padding top*, ou seja, a margem superior. Em relação ao código, na primeira linha é chamado o componente que contém o *layout* principal do sistema. Nas linhas de 8 a 11 é inserido um botão de Filtro que está desativado, por isso não possui um 'href' contendo uma rota. E nas linhas 26 a 40 são listados os recursos, pelo *loop foreach*. Além disso, o código da linha 42 auxilia na paginação dos dados.

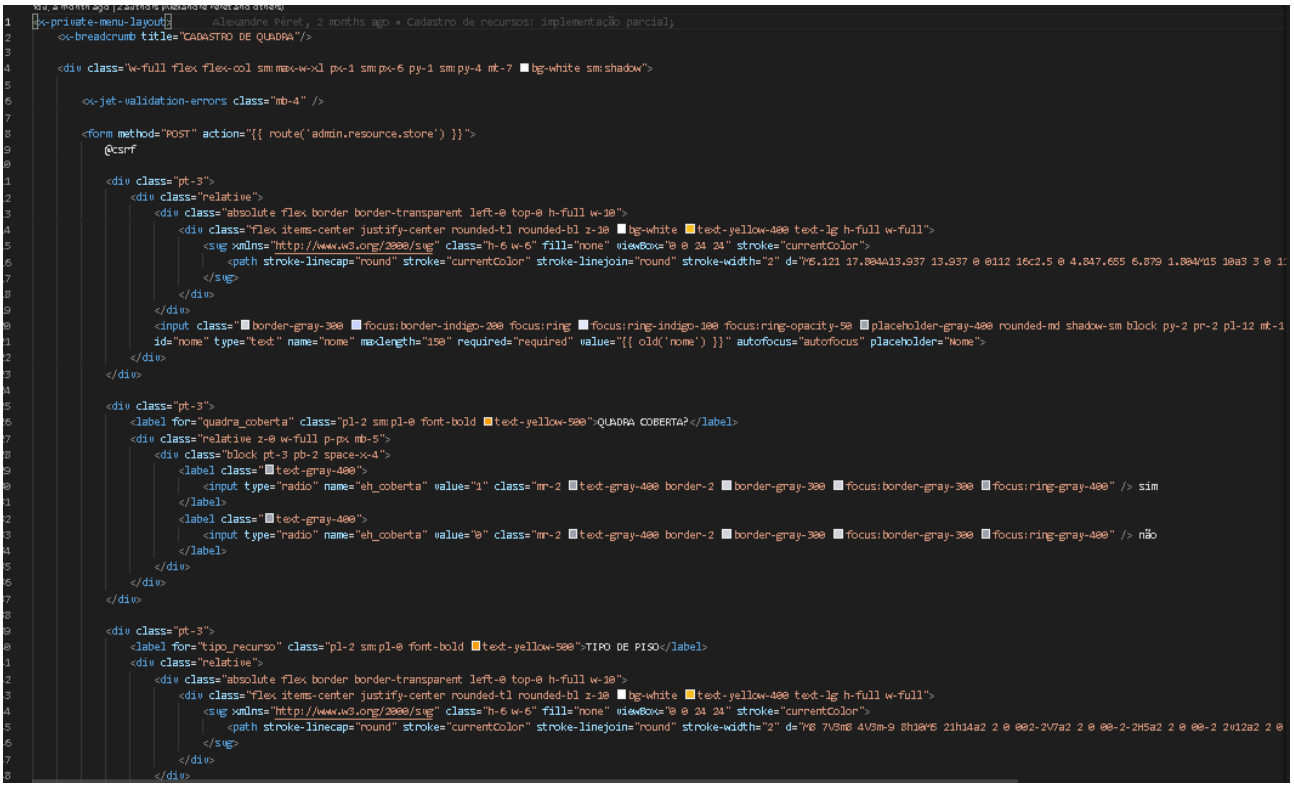
Outro exemplo de *view* com mais funcionalidades é a *view create*, mostrada nas Figuras 3.25 e 3.26. Ela consta com um *form* que chama o método 'POST' que direciona as informações recebidas aqui pelos *inputs* para a rota especificada. É importante que a declaração dessa rota contenha o mesmo método que este. O *form* é basicamente um formulário que recebe as informações e envia elas para uma *controller* através da rota especificada.

As Figuras 3.25 e 3.26 também serão discutidas como exemplo no tópico a seguir.



```
EXPLORER                                    list.blade.php ●
GUARANIX [SSH: 10.0.0.139]                  resources > views > admin > resource > list.blade.php > x-private-menu-layout > x-breadcrumb
  app                                       You, a month ago | 2 authors (Alexandre Péret and others)
  bootstrap                                 1  <x-private-menu-layout>
  config                                    2      <x-breadcrumb title="LISTA DE QUADRAS"/>      Alexandre Péret, a month ago • Refactoring: movendo rotinas para classes de AppS…
  database                                  3      <div class="w-full flex flex-col items-center bg-white sm:pt-6 sm:pb-6 mt-2">
  node_modules                              4          <div class="w-full flex flex-col sm:max-w-xl px-1 sm:px-6 py-1 sm:py-4 bg-white sm:shadow">
  public                                    5              <div class= "pt-3 flex items-center font-bold text-yellow-500 ">
  resources                                 6                  <div class="flex ml-auto mr-0">
    css                                     7                      <button type="button" class="flex items-center bg-white rounded border-yellow-500 border px-4 py-1 font-bold opacity-
    js                                      8                          <x-icons.search class="w-6 h-6" />
    lang                                    9                          <span class="inline-flex text-yellow-500"> FILTRAR</span>
    markdown                                10                     </button>
    views                                   11                 </div>
      admin                                 12             </div>
        booking                             13             <div class="pt-3">
        customer                            14                 <div class="flex flex-col justify-center items-center bg-gray-100 border border-gray-200 rounded-lg text-gray-600 w-ful
        grupo                               15                     <div class="grid grid-cols-4 divide-x text-center font-bold w-full text-sm">
        resource                            16                         <div class="m-1 pb-2 pt-2">RECURSO</div>
          create.blade.p...                 17                         <div class="m-1 pb-2 pt-2">TIPO</div>
          edit.blade.php                    18                         <div class="m-1 pb-2 pt-2">COBERTA?</div>
          list.blade.php                    19                         <div class="m-1 pb-2 pt-2">AÇÕES</div>
        resourceprice                       20                     </div>
        resourcetype                        21                 </div>
        sporttype                           22             </div>
      home.blade.php                        23             @foreach($recursos as $recurso)
    api                                     24                 <div class="pt-3">
    auth                                    25                     <div class="flex flex-col justify-center items-center bg-gray-100 border border-gray-200 rounded-lg text-gray-600 w
    components                              26                         <div class="grid grid-cols-4 divide-x text-sm text-center w-full">
    customers                               27                             <div class="pl-1 sm:pl-2 pb-2 pt-2">{{ $recurso->nome }}</div>
    layouts                                 28                             <div class="pl-1 pb-2 pt-2">{{ $recurso->tipo }}</div>
    livewire                                29                             <div class="pl-1 pb-2 pt-2">{{ $recurso->coberta }}</div>
    profile                                 30                             <div class="pl-1 pb-2 pt-2 flex flex-row w-full justify-center">
    teams                                   31                                 <a class="flex" href="{{ route('admin.resource.edit', [ 'id' => $recurso->id ]) }}" title="editar"><x-icons.e
    cadastro.blade.co...                    32                                 <a class="flex" href="javascript:void(0)" title="excluir"><x-icons.delete class="inline-flex w-6 h-6 text-g
OUTLINE                                     33                             </div>
TIMELINE                                    34                         </div>
                                            35                     </div>
                                            36                 </div>
                                            37             @endforeach
                                            38             {!! $recursos->links() !!}
                                            39         </div>
```

Figura 3.24 – *View* da Tabela Recurso



```
<x-private-menu-layout>      Alexandre Péret, 2 months ago • Cadastro de recursos: implementação parcial;
    <x-breadcrumb title="CADASTRO DE QUADRA"/>

    <div class="w-full flex flex-col sm:max-w-xl px-1 sm:px-6 py-1 sm:py-4 mt-7 bg-white sm:shadow">

        <x-jet-validation-errors class="mb-4" />

        <form method="POST" action="{{ route('admin.resource.store') }}">
            @csrf

            <div class="pt-3">
                <div class="relative">
                    <div class="absolute flex border border-transparent left-0 top-0 h-full w-10">
                        <div class="flex items-center justify-center rounded-tl rounded-bl z-10 bg-white text-yellow-400 text-lg h-full w-full">
                            <svg xmlns="http://www.w3.org/2000/svg" class="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor">
                                <path stroke-linecap="round" stroke="currentColor" stroke-linejoin="round" stroke-width="2" d="M5.121 17.804A13.937 13.937 0 0112 16c2.5 0 4.847.655 6.879 1.804M15 10a3 3 0 1
                            </svg>
                        </div>
                    </div>
                    <input class="border-gray-300 focus:border-indigo-200 focus:ring focus:ring-indigo-100 focus:ring-opacity-50 placeholder-gray-400 rounded-md shadow-sm block py-2 pr-2 pl-12 mt-1
                    id="nome" type="text" name="nome" maxlength="150" required="required" value="{{ old('nome') }}" autofocus="autofocus" placeholder="Nome">
                </div>
            </div>

            <div class="pt-3">
                <label for="quadra_coberta" class="pl-2 sm:pl-0 font-bold text-yellow-500">QUADRA COBERTA?</label>
                <div class="relative z-0 w-full p-px mb-5">
                    <div class="block pt-3 pb-2 space-x-4">
                        <label class="text-gray-400">
                            <input type="radio" name="eh_coberta" value="1" class="mr-2 text-gray-400 border-2 border-gray-300 focus:border-gray-300 focus:ring-gray-400" /> sim
                        </label>
                        <label class="text-gray-400">
                            <input type="radio" name="eh_coberta" value="0" class="mr-2 text-gray-400 border-2 border-gray-300 focus:border-gray-300 focus:ring-gray-400" /> não
                        </label>
                    </div>
                </div>
            </div>

            <div class="pt-3">
                <label for="tipo_recurso" class="pl-2 sm:pl-0 font-bold text-yellow-500">TIPO DE PISO</label>
                <div class="relative">
                    <div class="absolute flex border border-transparent left-0 top-0 h-full w-10">
                        <div class="flex items-center justify-center rounded-tl rounded-bl z-10 bg-white text-yellow-400 text-lg h-full w-full">
                            <svg xmlns="http://www.w3.org/2000/svg" class="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor">
                                <path stroke-linecap="round" stroke="currentColor" stroke-linejoin="round" stroke-width="2" d="M8 7V5m8 4V3m-9 8h10M5 21h14a2 2 0 002-2V7a2 2 0 00-2-2H5a2 2 0 00-2 2v12a2 2 0
                            </svg>
                        </div>
                    </div>
```

Figura 3.25 – *View* da Tabela Recurso

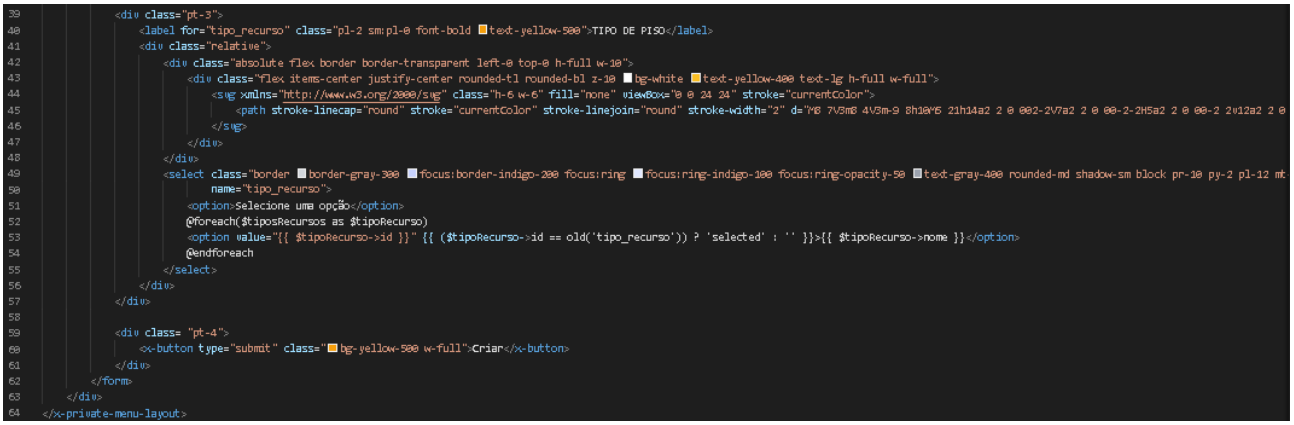


Figura 3.26 – *View* da Tabela Recurso

3.3.2.3 *Controller*

Como padrão do Laravel os *controllers* da aplicação ficam no diretório *app/Http/Controllers*. De acordo com Moraes (2018) um *controller* "é onde manipulamos a lógica de tratamento das requisições recebendo os dados do model e transmitindo-os para a view". Ele também diz que "a função dos Controllers é organizar e agrupar *requests* relacionadas, manipulando sua lógica em uma única classe".

Simplificando: os *controllers* fazem uso de métodos para estabelecer a comunicação entre as *models* (e o banco de dados) e as *views*, utilizando as rotas como caminho. Para criar uma classe *controller* basta seguir a linha de comando da Fig.3.27.

```
php artisan make:controller RecursoController
```

Figura 3.27 – Comando para a Criação do RecursoController

As Figuras 3.28 e 3.29 mostram como é a estrutura de *controller*. Assim, como na *model*, a cada classe que será usada, é necessário chamá-la no arquivo. Para explicar melhor como funciona a comunicação, serão usados os exemplos das funções *list*, *create* e *store*.

A função *list* basicamente gera um *array* com os recursos que estão no banco de dados, trazendo com eles as 'quadras', que é especificado na *model* (Fig.3.22) se referindo ao `tipo_recursos`. Formata alguns dos dados do *array* no *foreach* e envia essas informações para a *view 'admin.resource.list'*. Para entender melhor a linha 35 da Fig.3.28, o segundo parâmetro da *view* é o *array* com os dados e o primeiro parâmetro é o caminho para chegar até o arquivo *list* que está configurado para exibir os dados.

Outros exemplos de métodos da classe ResouceController são o *create* e o *store*, apresentados da Fig.3.29. Pode-se dizer que eles se complementam, pois:

- A função *create* apenas carrega os dados dos tipos de recursos existentes e envia para a *view*

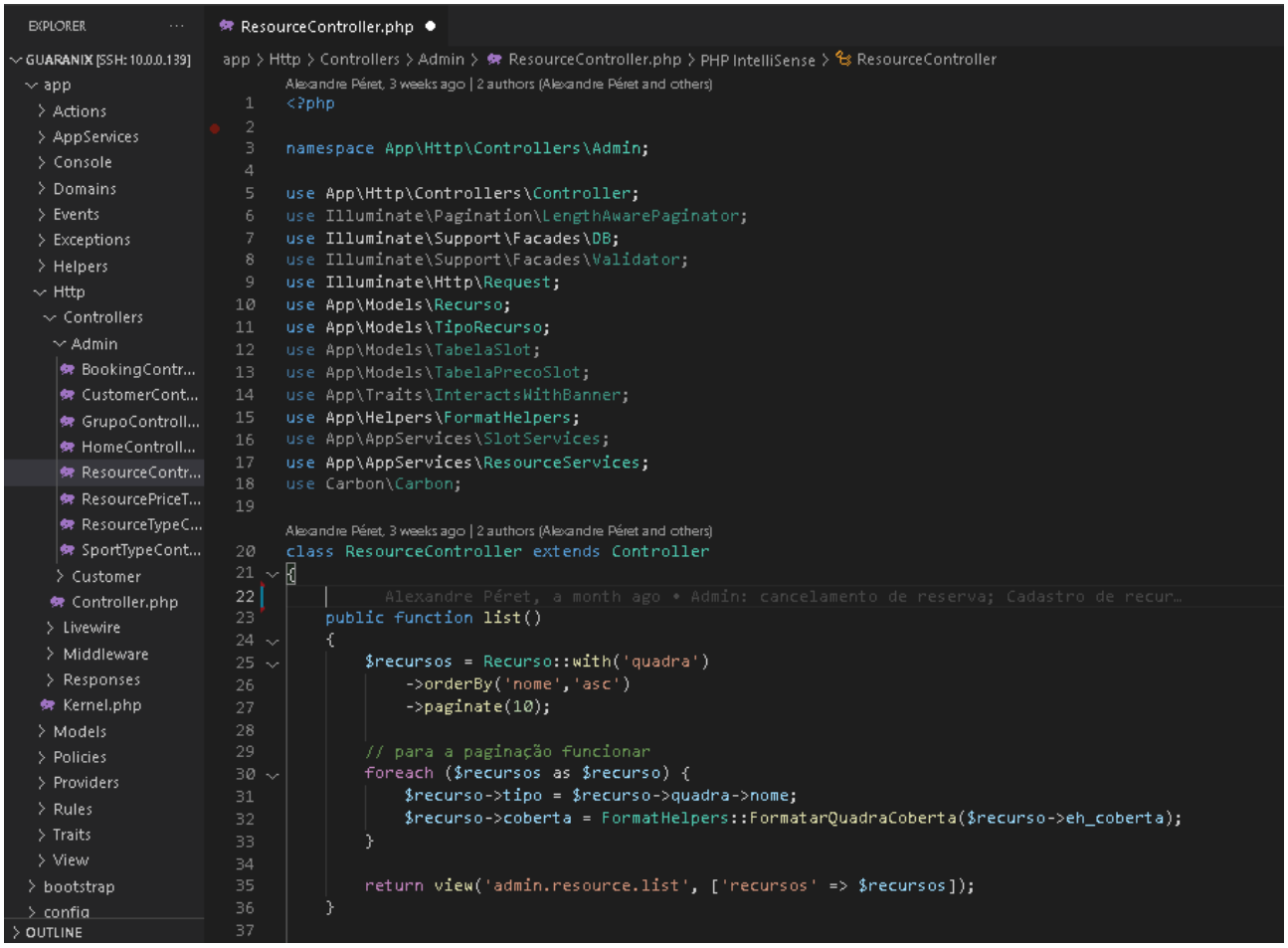


Figura 3.28 – *Controller* da Tabela Recurso - Função *List*

'admin.resource.create'. Nessa *view* existe um formulário com *inputs* para a entradas dos dados necessários para a criação de um novo recurso. Após a submissão deste formulário, as informações dos *inputs* são enviadas para a rota que contém o método *store* da classe em questão. Como indicado na Fig.3.25;

- A função *store* recebe como parâmetro a classe Request que é específica para requisições de entrada. E é aí que elas se unem, porque ela vai receber as informações que foram enviadas acima, validá-las, salvar no banco de dados e exibir uma mensagem de sucesso caso não encontre algum problema. E depois vai redirecionar para a página que lista os recursos, exibindo os novos dados.

```php
    public function create()
    {
        $tiposRecursos = TipoRecurso::orderBy('nome','asc')->get();

        return view('admin.resource.create', [ 'tiposRecursos' => $tiposRecursos ]);
    }       Alexandre Péret, 2 months ago • Banner: adicionando trait e banner global pa
    public function store(Request $request)
    {
        $request->validate([
            'nome' => ['required', 'string', 'max:150'],
            'eh_coberta' => ['required', 'int'],
            'tipo_recurso' => ['required', 'int', ],
        ]);

        $recurso = DB::transaction(function () use ($request) {
            $tipoRecurso = TipoRecurso::find($request->input('tipo_recurso'));

            $recurso = new Recurso;
            $recurso->nome = $request->input('nome');
            $recurso->eh_coberta = (bool)$request->input('eh_coberta');
            //TODO: avaliar como implementar usando ao associate
            // $recurso->quadra()->associate($tipoRecurso);
            $recurso->tipo_recursos_id = $tipoRecurso->id;
            //TODO: remover a relação tipo_esportes
            $recurso->tipo_esportes_id = 1;
            $recurso->save();

            $this->generateResourceDefaultSlots($recurso);

            return $recurso;
        });

        $this->banner('O novo recurso foi cadastrado.');

        return redirect()->route('admin.resource.list', $recurso->id);
    }
```

Figura 3.29 – *Controller* da Tabela Recurso - Funções *Create* e *Store*

3.3.3 *Rotas*

As rotas já foram citadas em alguns pontos deste texto e agora é o momento de defini-las. Elas se encontram no diretório *routes/web* e em suma são caminhos a serem seguidos dentro do projeto. Como por exemplo, ao receber uma requisição pela URL, são as rotas que decidem se redireciona para uma outra página ou se leva essa requisição para ser tratada por um *controller*.

Como nos demais arquivos citados, para cada classe referenciada nas rotas, é necessário chamá-la no início. Como pode ser observado na Fig. 3.30. Esta imagem e a Fig.3.31 servirão de exemplo para explicar como funciona a estrutura de uma rota.

Pode-se definir uma rota usando o `'$this'` chamando um método como *GET* ou *POST*, por exemplo. Ou da forma exemplificada, utilizando a própria classe (Route) e também chamado um destes (ou outros) métodos. É possível registrar rotas com todos os verbos do HTTP: GET, POST, PUT, PATCH, DELETE e OPTIONS. Os mais usados por este trabalho foram:

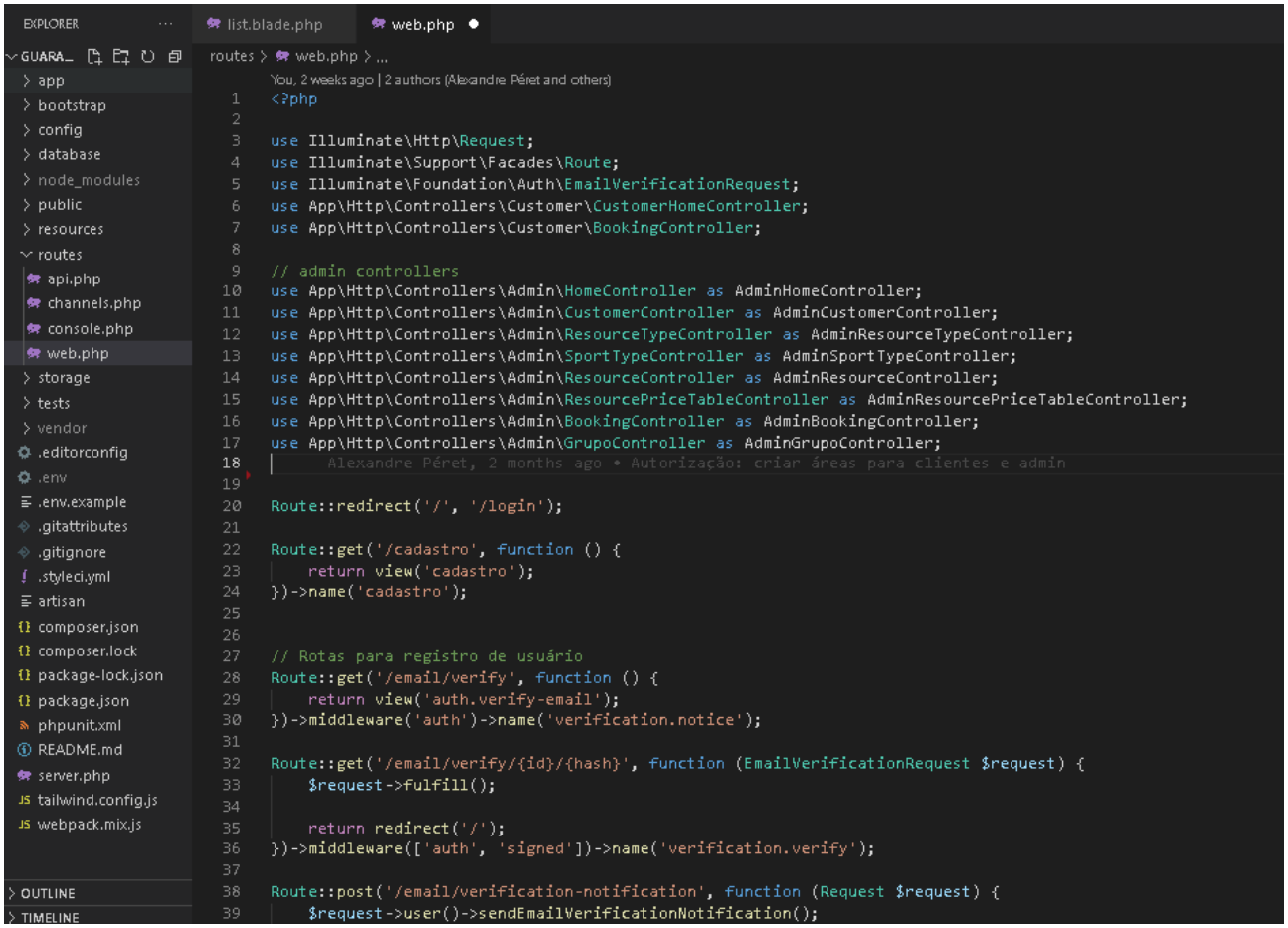


Figura 3.30 – Rotas

```
Route::get('/quadras', [AdminResourceController::class, 'list'])->name('resources.list');
Route::get('/quadras/pesquisar', [AdminResourceController::class, 'search'])->name('resources.search');
Route::get('/quadra/novo', [AdminResourceController::class, 'create'])->name('resource.create');
Route::post('/quadras', [AdminResourceController::class, 'store'])->name('resource.store');
Route::get('/quadra/{id}/editar', [AdminResourceController::class, 'edit'])->name('resource.edit');
Route::put('/quadra/{id}', [AdminResourceController::class, 'update'])->name('resource.update');
Route::get('/quadra/{id}/remover', [AdminResourceController::class, 'deleteConfirm'])->name('resource.delete-confirm');
Route::delete('/quadra/{id}', [AdminResourceController::class, 'delete'])->name('resource.delete');
```

Figura 3.31 – Rotas da Tabela Recurso

- Rota GET: normalmente são utilizadas para listar algum tipo de conteúdo. Tem os parâmetros passados pelo cabeçalho e são visíveis na URL;
- Rota POST: comumente usado para cadastro ou afins. As requisições passadas por elas não aparecem na URL, já que são passados no corpo da requisição;
- Rota DELETE: serve para deletar algo que foi solicitado.

Analisando a rota da linha três, na Fig.3.31, o primeiro parâmetro diz respeito à navegabilidade. Por exemplo, considerando a URL *http//www.guaranix.com.br* e ignorando outras configurações, se adicionasse o primeiro parâmetro a URL, ficando *http//www.guaranix.com.br/quadra/novo*, a página deverá ser direcionada para a criação de um novo recurso na aplicação. Esse exemplo

pode ser melhor acompanhado pelas imagens do Capítulo 4, mais precisamente a Fig.4.21. Os parâmetros também podem ser passados de forma dinâmica, como na linha cinco da Fig.3.31, que recebe o 'id' do recurso a ser editado.

O segundo parâmetro se refere a classe e a função que contém as especificações que processarão a requisição solicitada. Que já foi explicado como funciona nas subseções *View* e *Controller*. E, por fim, é possivel nomear uma rota, para distinguir e padronizar o uso da mesma durante o desenvolvimento.

Neste projeto outras pastas e funcionalidades do *framework* foram usadas e muitas também criadas. Mas com o que foi demonstrado até aqui já é possível criar uma aplicação funcional usando o Laravel. E para aprofundar ainda mais no assunto, além da documentação, existe muito conteúdo de valor nos fóruns de dúvidas pela internet.

4 Resultados

Dada a implementação do projeto, este é o momento de apresentar os resultados. Nos próximos subitens serão exibidas as telas resultantes do desenvolvimento da aplicação em conjunto com uma análise das funcionalidades contidas nela. A metodologia utilizada uniu desenvolvimento e testes, isso quer dizer que de acordo com que o projeto foi sendo desenvolvido, os testes foram sendo feitos para garantir o desempenho esperado. Ao final a aplicação foi colocada em um ambiente de homologação, onde o cliente e demais envolvidos puderam ter acesso e se familiarizar com o produto.

A demonstração a seguir será dividida em três principais grupos:

- Login: envolvendo todas as telas relacionadas às rotinas de login, cadastro, verificação de *e-mail* e recuperação de senha;
- Cliente: telas e funcionalidades que podem ser acessadas pelos clientes cadastrados;
- Gestor: com todas as funcionalidades que o administrador tem acesso.

Para cada imagem das páginas principais do Cliente, Gestor e de Login, também será apresentada uma imagem mostrando o *design* responsivo. Compatível com uma tela de *smartphone* com dimensões 360 x 640.

4.1 Área de Login

Para o usuário já cadastrado realizar o login, basta inserir seu *e-mail* e senha, como visto na Fig.4.1 e Fig.4.2. Caso ocorra algum problema, ele pode solicitar a recuperação de senha. Nessa etapa, como demonstra a Fig.4.3, basta ele inserir seu *e-mail* que um *link* para criar uma nova senha é enviado para o *e-mail* disponibilizado, encaminhando para a Fig.4.4. Vale ressaltar que para que essa etapa da recuperação de senha funcione o usuário deve ter registro no sistema.

A Fig.4.1 também indica que pode ser realizado o cadastro de um novo usuário. Levando então para a Fig.4.5, na qual as informações são requeridas e o usuário cadastrado vai automaticamente para o grupo de clientes do sistema. Depois de submetidas às informações pessoais, um *e-mail* é enviado para a verificação do mesmo (Fig.4.6). E após clicar no *link* contido nele, o cliente já consegue ter acesso à Área do Cliente.

Aqui é importante lembrar que todo esse processo de cadastro de novo usuário, login, verificação de *e-mail* e recuperação de senha já é disponibilizado pelo Laravel, bastando apenas adaptá-lo às necessidades do programa.

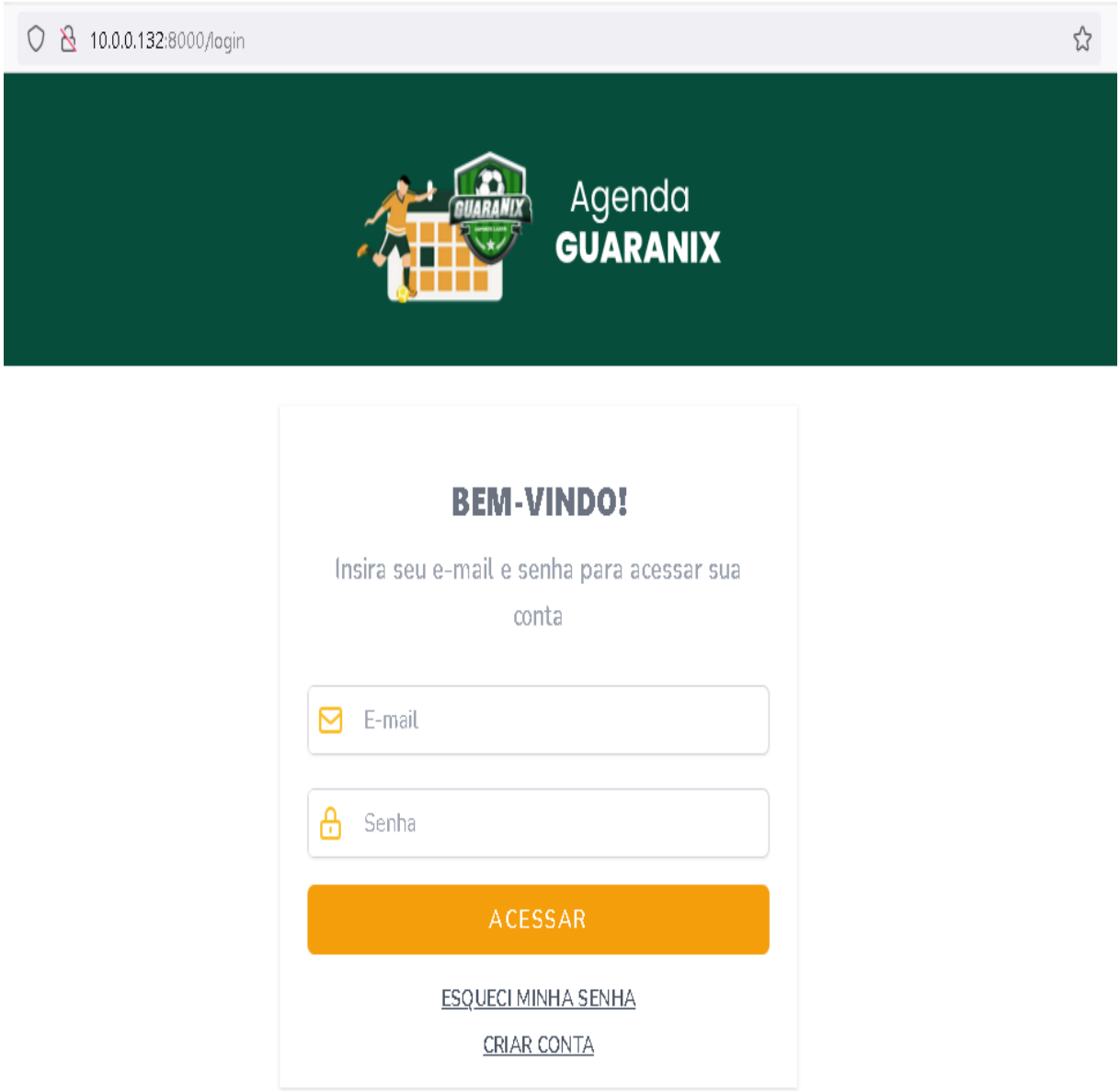


Figura 4.1 – Área de Login - Tela Principal.

4.2 Área do Cliente

Dando início, percebe-se (pela Fig.4.7) que a área do Cliente consta com um Menu Hambúrguer. Nele é possível sempre que quiser retornar a página inicial do cliente ou fazer o logoff do sistema, direcionando para a tela de Login. A seguir tem-se a descrição das telas desse ambiente.

4.2.1 Tela Principal do Cliente

Essa é a primeira tela que o cliente vê quando loga no programa e a Fig.4.8 representa a mesma tela, mas vista através de um *smartphone*. Nela, além do menu já citado, consta as informações das reservas que ele ainda tem em aberto, listadas pela data mais próxima. Nas

Figura 4.2 – Área de Login - Tela Principal Responsíva.

Ações, presentes na última coluna da lista, é possível visualizar todas as informações da reserva, como mostra a Fig.4.9. E também deleta-lá, através de uma rotina que pede a confirmação da ação (Fig.4.10).

4.2.2 Agendar Quadra

Clicando em Agendar Quadra na tela principal, o programa é direcionado para a tela representada na Fig.4.11. E ao selecionar uma das quadras disponíveis, vai para a página seguinte: a seleção de horários. A Fig.4.12 mostra para cada quadra a disponibilidade de horários dela, com *status* Livre ou Indisponível. Essa disponibilidade também segue por dia, podendo ser mudando clicando dos dias seguintes acima.

Essa rotina de seleção de horários mostra que: cada quadra tem sua disponibilidade

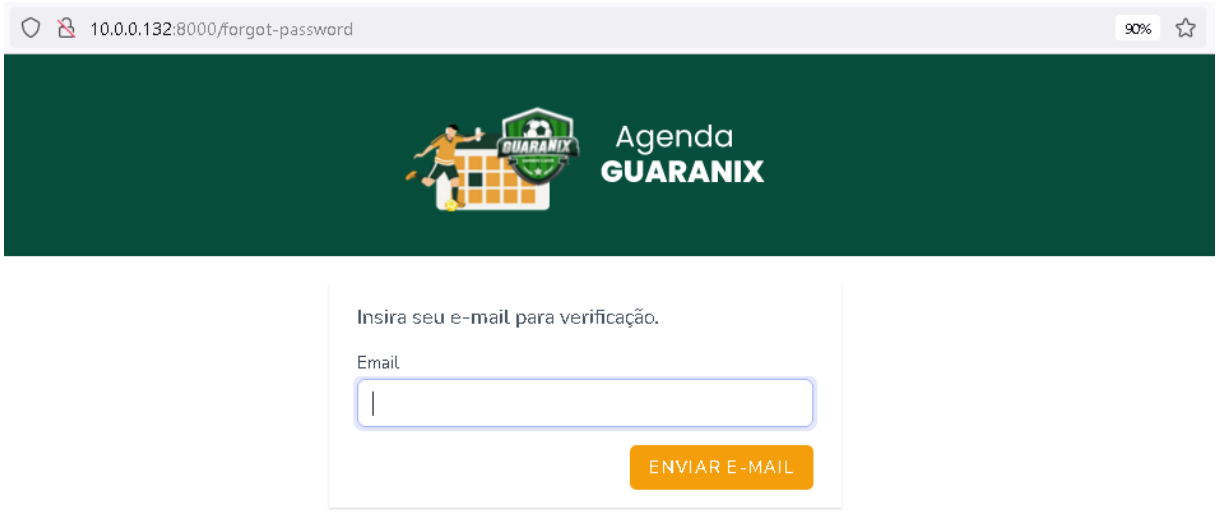


Figura 4.3 – Área de Login - Recuperação de Senha.

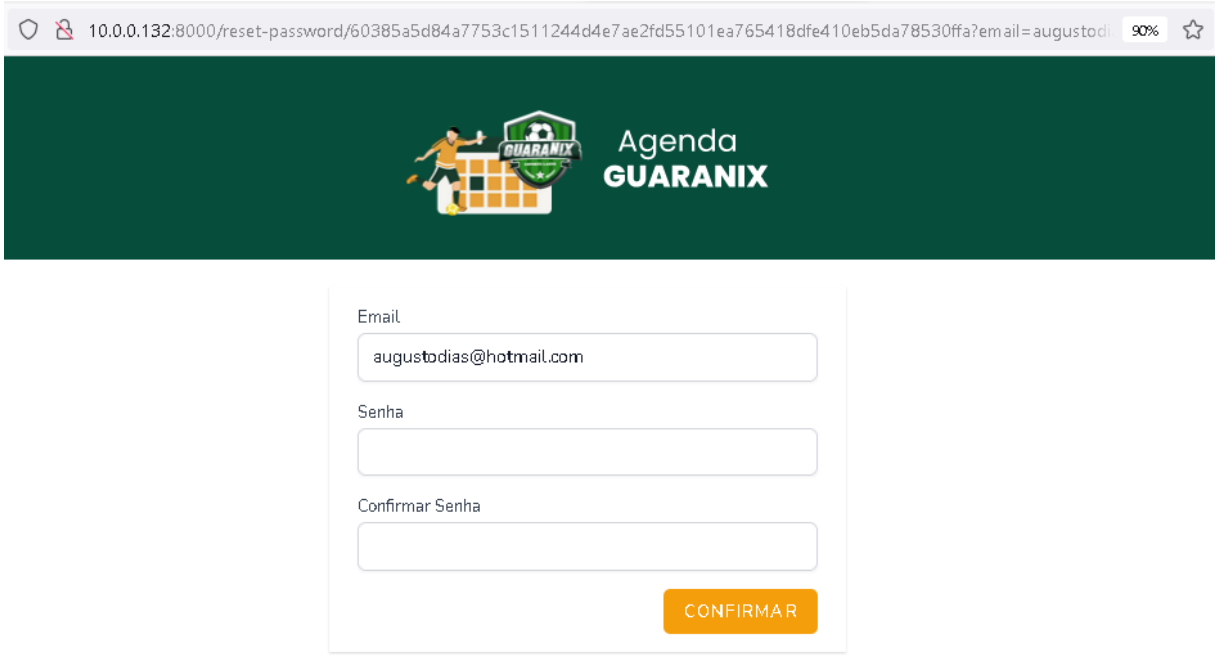


Figura 4.4 – Área de Login - Inserir Nova Senha.

de dias e horários distintos; os dias só são mostrados considerando o dia vigente mais 15 dias consecutivos; os horários são gerados de forma dinâmica e caso eles não tenham reservas, são apagados assim que a data passar.

Depois de escolhido o dia e horário clicando em cima dos mesmos, o próximo passo, como mostra a Fig.4.13, é a confirmação das informações selecionadas. Contendo o nome da quadra, dia e horários selecionados. Ao confirmar a reserva, os dados são gravados no banco e é gerado um protocolo para esta solicitação (Fig.4.14). E caso queira alocar mais um horário é só retornar ao início e refazer o processo.

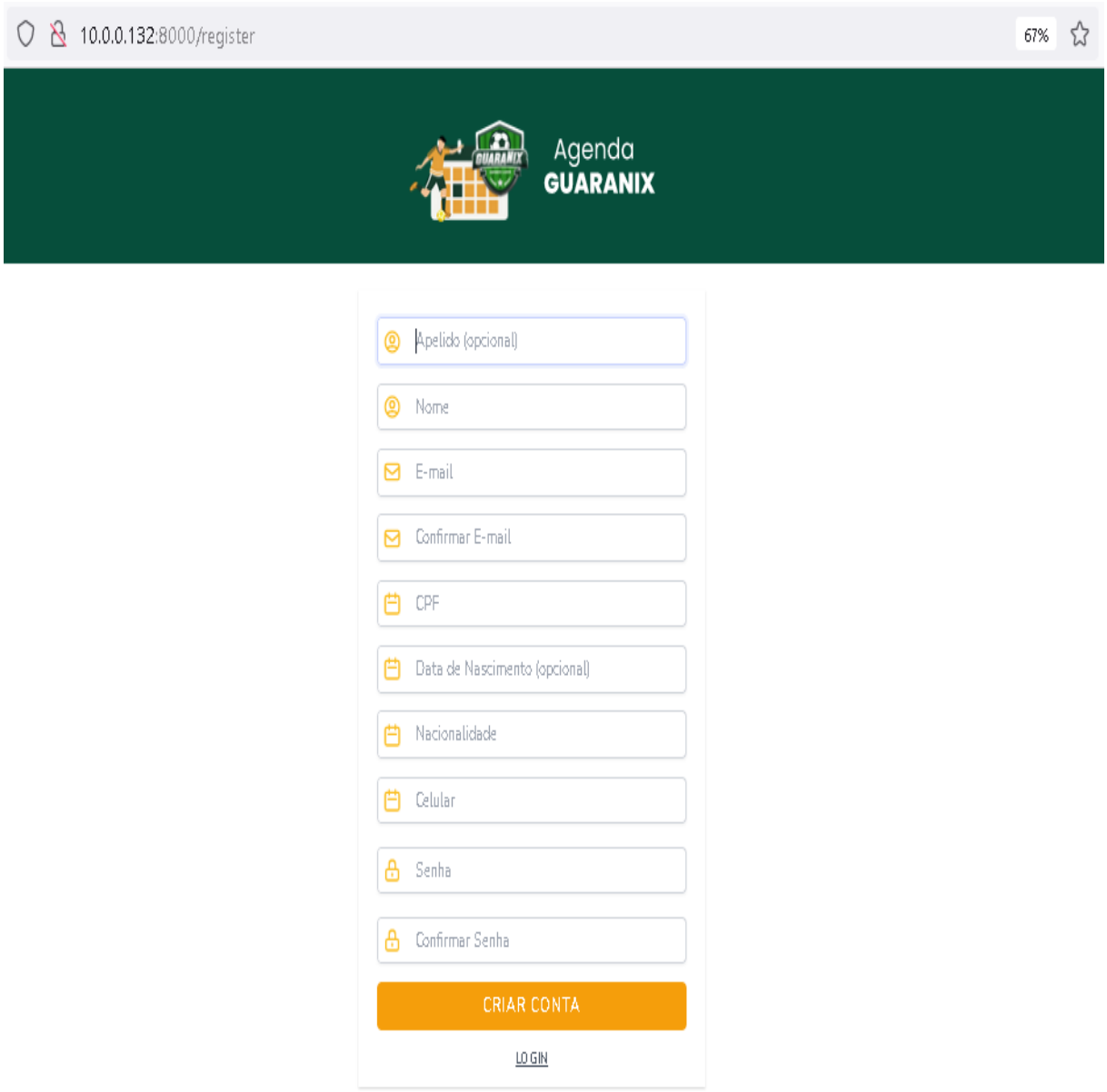


Figura 4.5 – Área de Login - Cadastro de Novo Usuário.

4.3 Área do Gestor

Essa área também consta com o Menu Hambúrguer citado na seção 4.2, contendo as mesmas funcionalidades. O ambiente em que o administrador tem acesso é a mais completa, contando com todas as permissões do programa. Um usuário só se torna um gestor através das configurações internas feitas no sistema.

4.3.1 Tela Principal do Gestor

As figuras 4.15 e 4.16 mostram como ficou a disposição da tela principal do Gestor. Com o *layout* e as funcionalidades seguindo de forma fiel ao que foi disponibilizado no Figma, visto no capítulo de desenvolvimento. São três subconjuntos representados por: Clientes, Quadras e

Figura 4.6 – Área de Login - Verificação do Endereço de *e-mail*.

Reservas.

4.3.1.1 Clientes

Aqui estão as funcionalidades relacionadas aos clientes. Permitindo cadastrar, pesquisar, listar, editar e também realizar uma reserva em nome de um determinado cliente.

Os requisitos e suas telas, são:

- Cadastrar: permite realizar o cadastro de um cliente da mesma forma que um usuário faria, visto na Fig.4.5. Enviando para o *e-mail* do cliente o *link* de verificação para que ele possa acessar posteriormente;
- Lista: lista os clientes cadastrados por ordem alfabética, como mostra a Fig.4.17. Permite editar um cliente direcionando para a página de cadastro com as informações do mesmo (Fig.4.18). Também é possível filtrar de acordo com as informações de nome, apelido, telefone ou última reserva (Fig.4.19). E por fim, clicando no ícone de livro visto na Fig.4.17 é possível realizar uma reserva. Seguindo a mesma rota de reserva feita pelo cliente, com a diferença do acréscimo do nome do cliente em que o nome da reserva está (Fig.4.20);
- Pesquisar: tem a mesma função do botão filtrar visto acima e retorna para a lista de clientes apenas com os clientes que cumpriram os requisitos pedidos (Fig.4.19).

4.3.1.2 Quadras

Essas são as funcionalidades relacionadas às quadras. Permitindo cadastrar, listar, editar e excluir.

Os requisitos e suas telas, são:

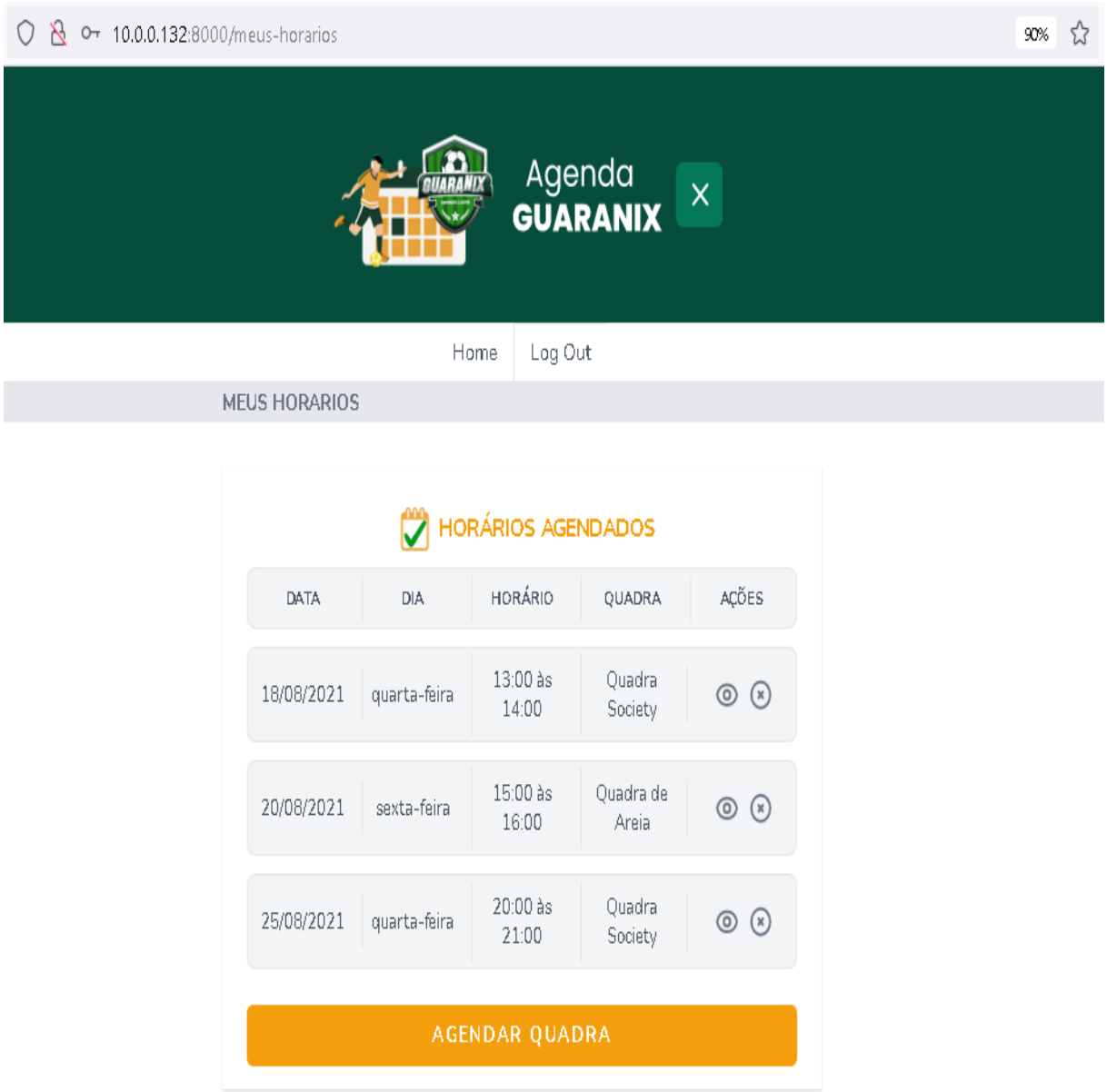


Figura 4.7 – Área do Cliente - Página Principal.

- Cadastrar (Fig.4.21): permite realizar o cadastro de uma quadra;
- Lista: mostra as quadras que estão cadastradas no sistema (Fig.4.22). Permite editar, indo para a mesma página de cadastro de quadras, carregando as informações já disponíveis (Fig.4.23). Também pode excluir uma quadra, caso ela não tenha pendências.

Os ítens de filtrar e pesquisar quadras não foram implementados devido ao pequeno número de quadras existente no clube. Fica como uma melhoria para o projeto caso ele ganhe outra proporções.

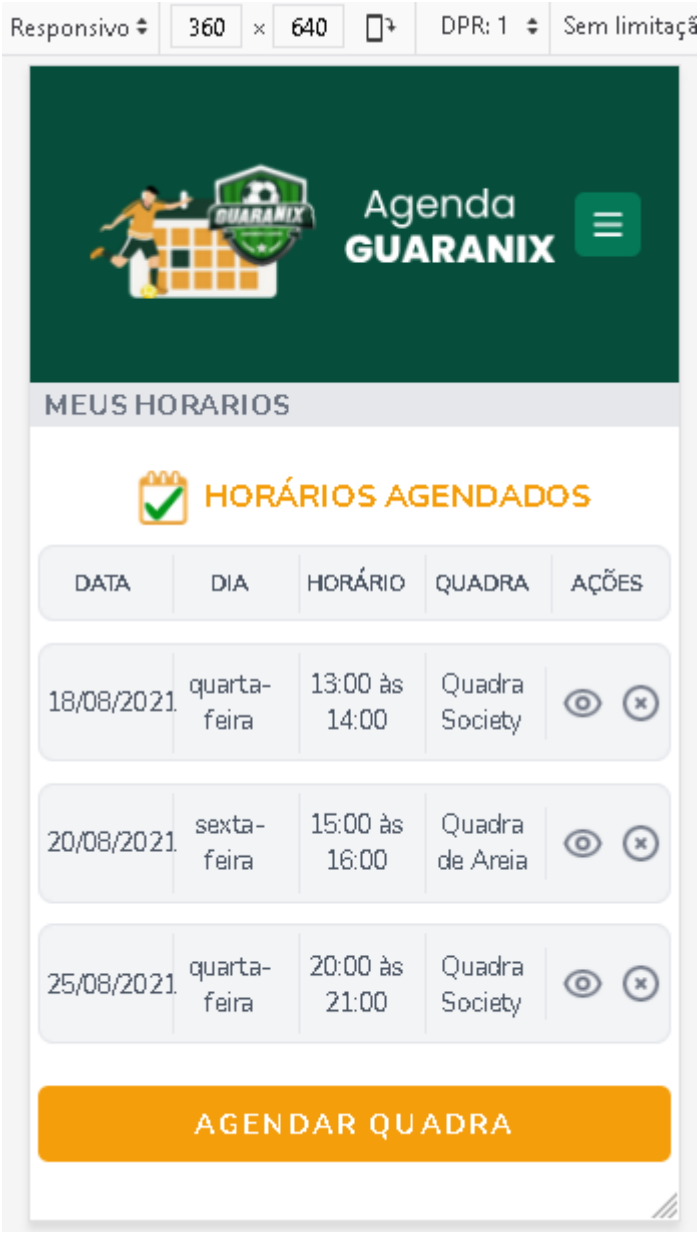


Figura 4.8 – Área do Cliente - Página Principal Responsiva.

4.3.1.3 Reservas

As funcionalidades relacionadas às reservas são parecidas com as do Cliente, como: cadastrar, pesquisar, listar, excluir e também tem a visualização e a confirmação de presença.

Os requisitos e suas telas, são:

- Cadastrar: ao clicar em cadastrar, ou seja, criar uma nova reserva, o programa redireciona para a página de listagem de clientes. Onde o atendente pode selecionar o ícone de livro e realizar uma nova reserva em nome daquele cliente específico, como mostrado na Fig.4.20;
- Lista: mostra as reservas que tem à partir do dia vigente, ordenado pela data mais recente (Fig.4.24). Nas ações é possível visualizar as informações completas da reserva selecionada

Figura 4.9 – Área do Cliente - Exibe Informações da Reserva Desejada.

(Fig.4.25); confirmar a presença do cliente através do ícone de caneta (Fig.4.26) e excluir uma reserva (Fig.4.27);

- Pesquisar: tem a mesma função do botão filtrar visto acima e retorna para a lista de reservas apenas com as reservas que cumprem os requisitos (Fig.4.28).

Figura 4.10 – Área do Cliente - Confirmar Exclusão de Reserva.

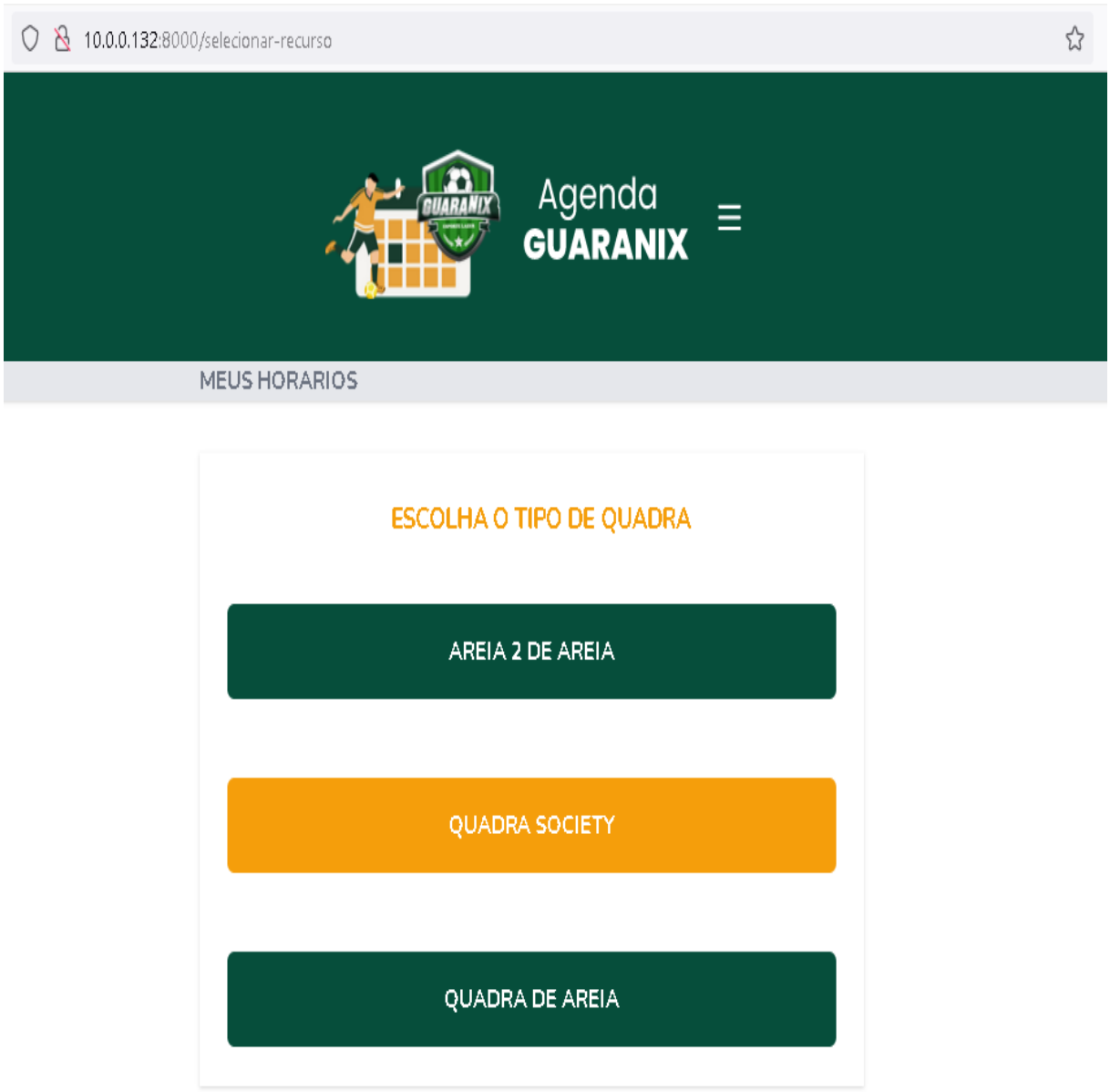


Figura 4.11 – Área do Cliente - Confirmar Exclusão de Reserva.

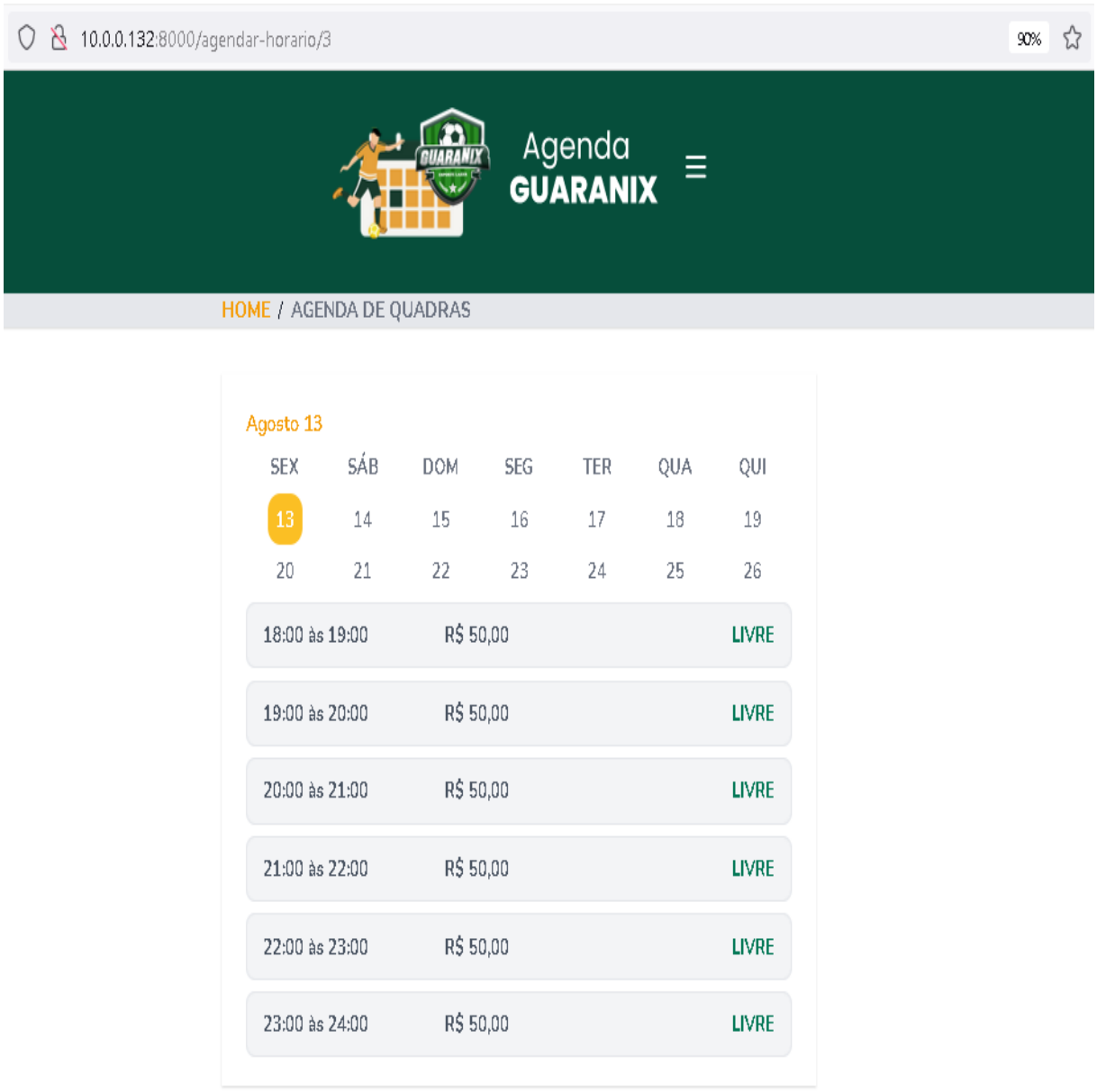


Figura 4.12 – Área do Cliente - Confirmar Exclusão de Reserva.

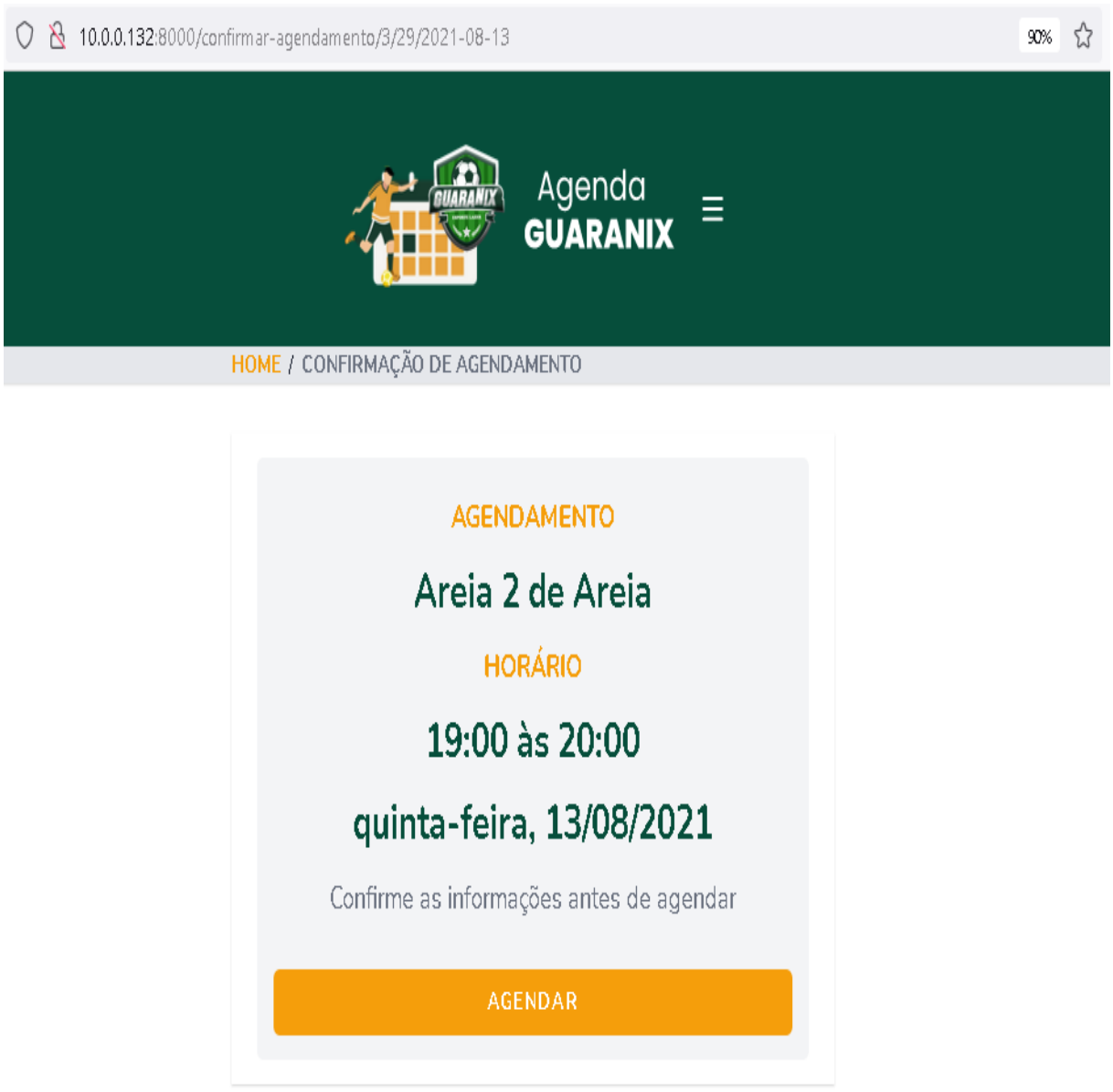


Figura 4.13 – Área do Cliente - Confirmar Dados da Reserva.

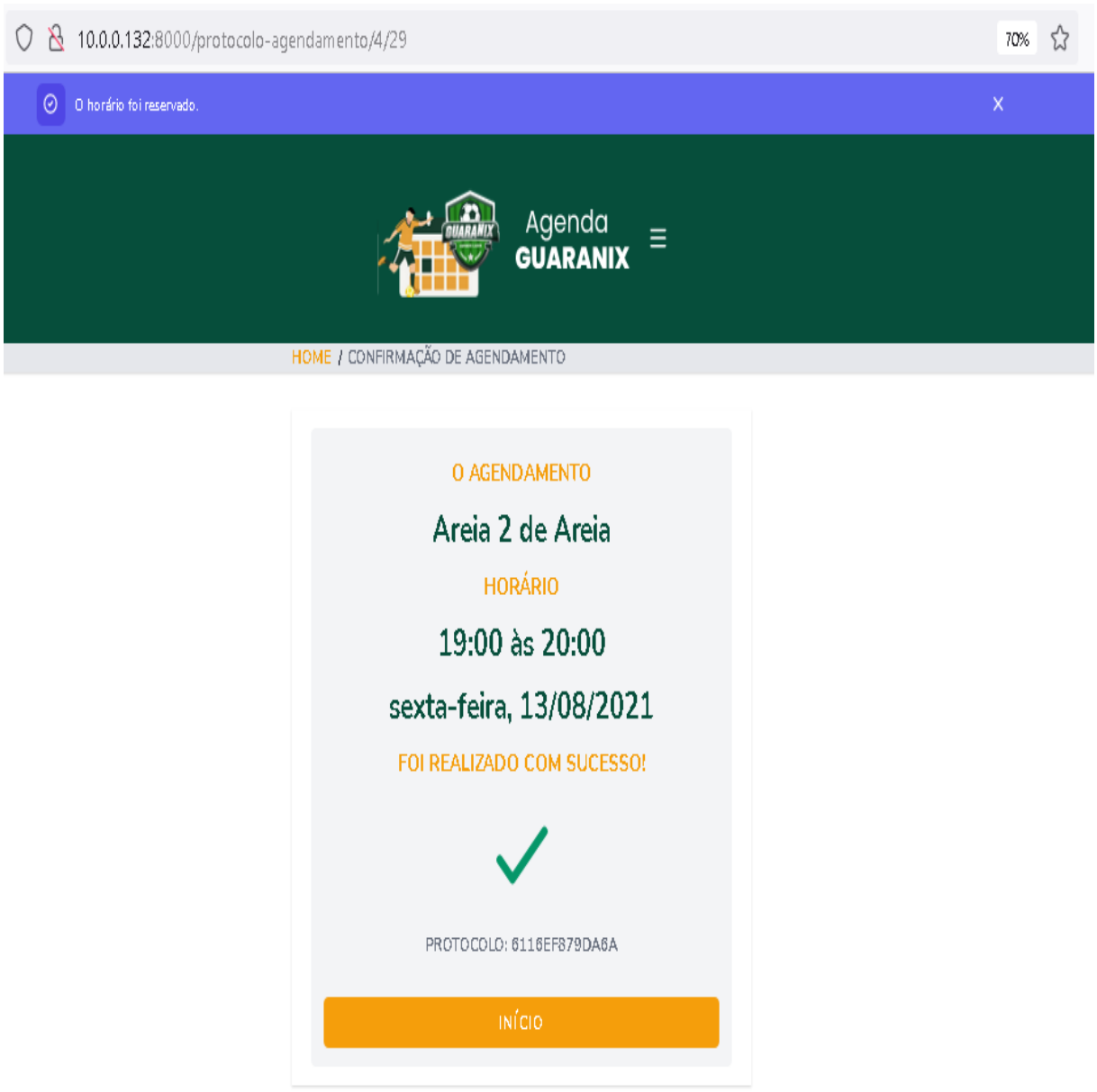


Figura 4.14 – Área do Cliente - Protocolização da Reserva.

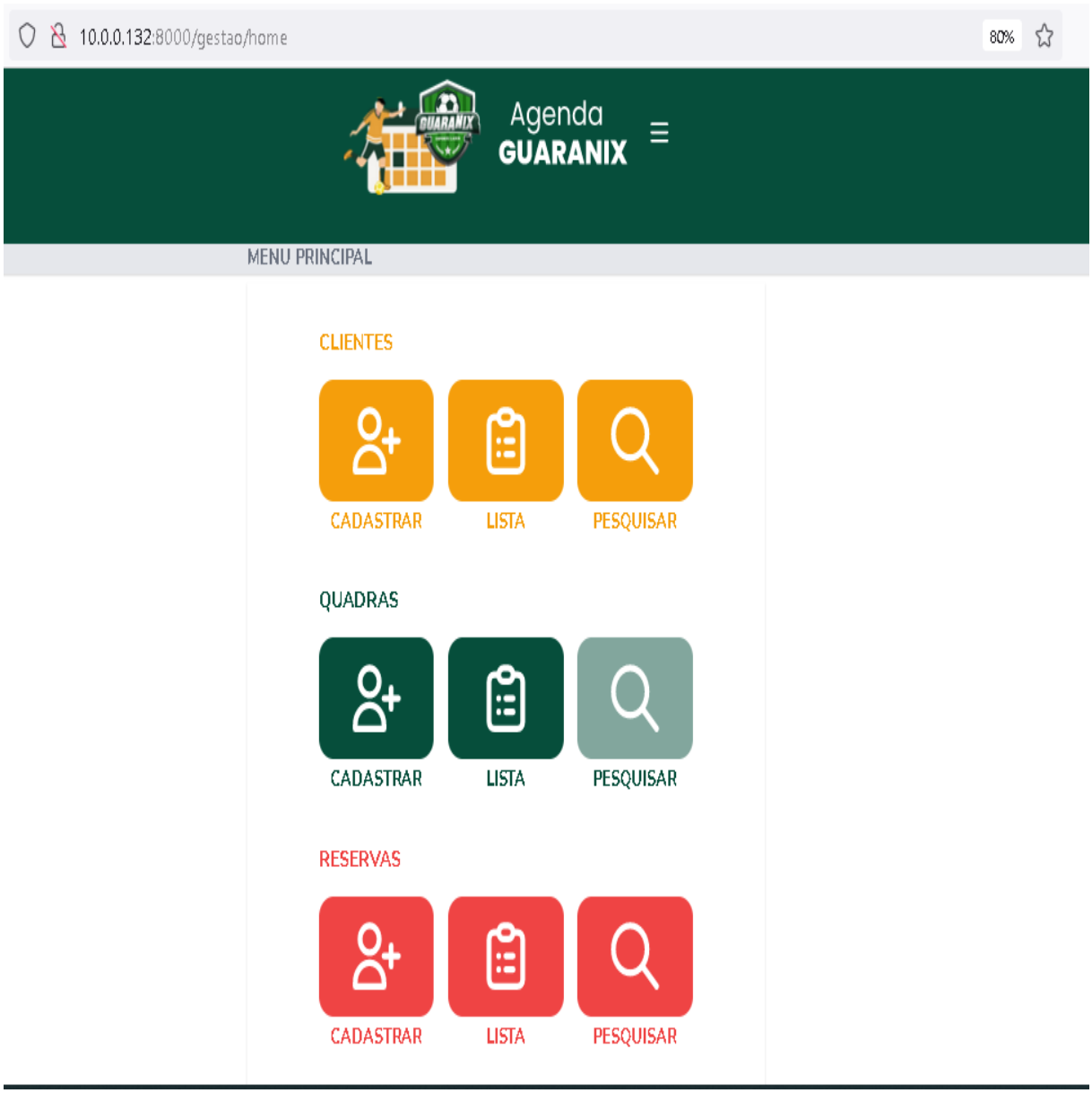


Figura 4.15 – Área do Gestor - Página Principal.

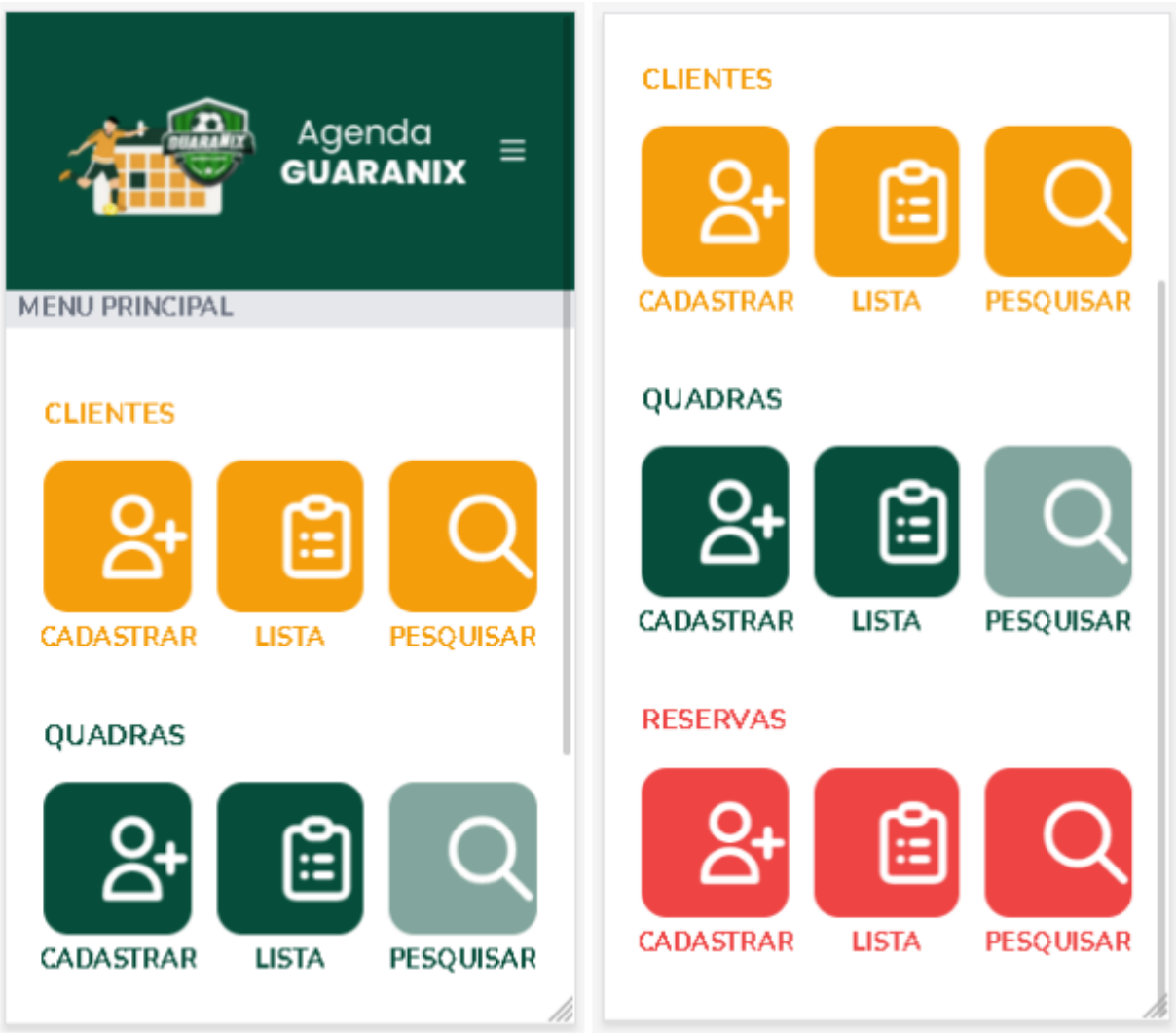


Figura 4.16 – Área do Gestor - Página Principal Responsiva.

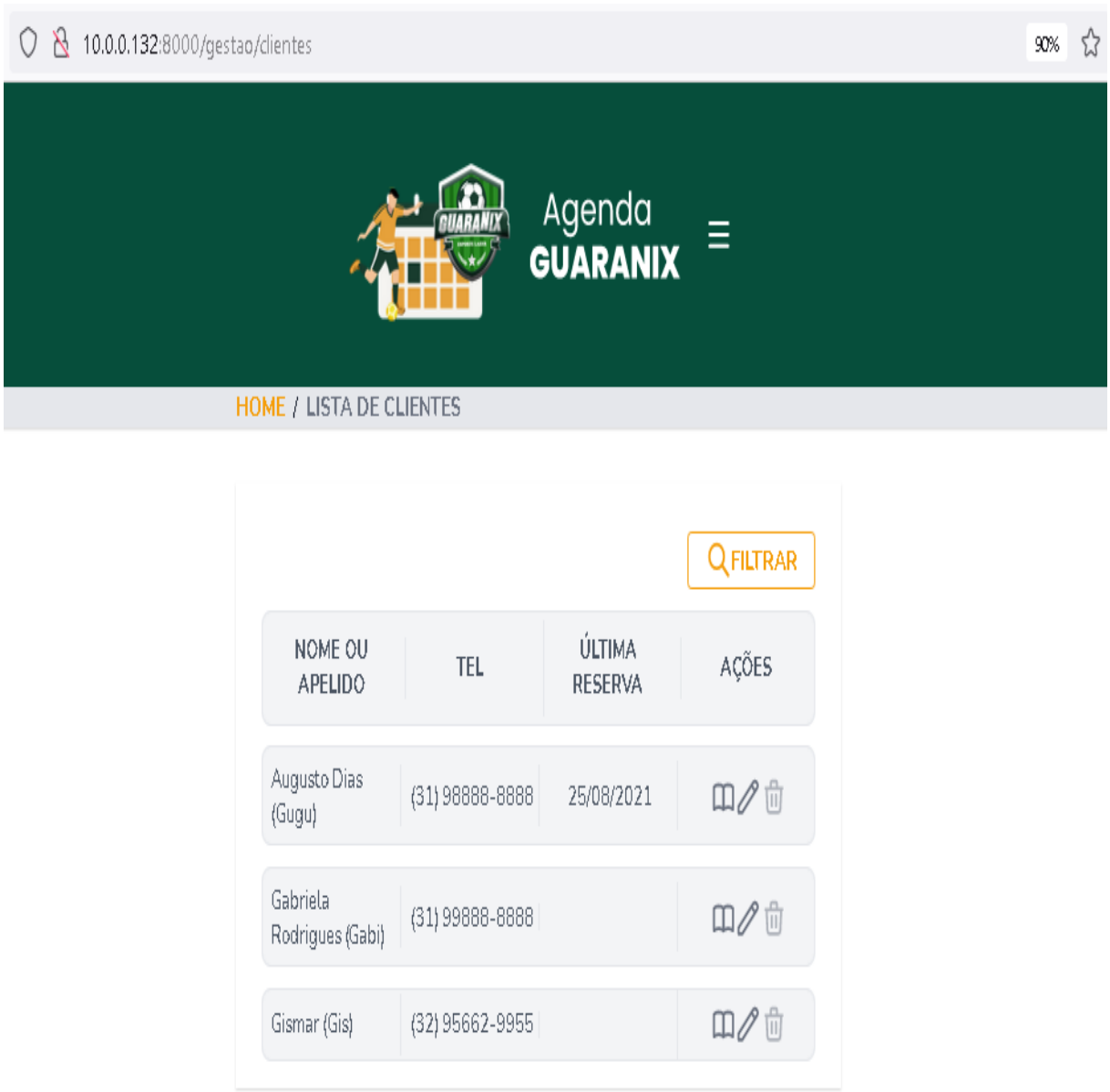


Figura 4.17 – Área do Gestor - Clientes - Lista de Clientes que Possuem Registro.

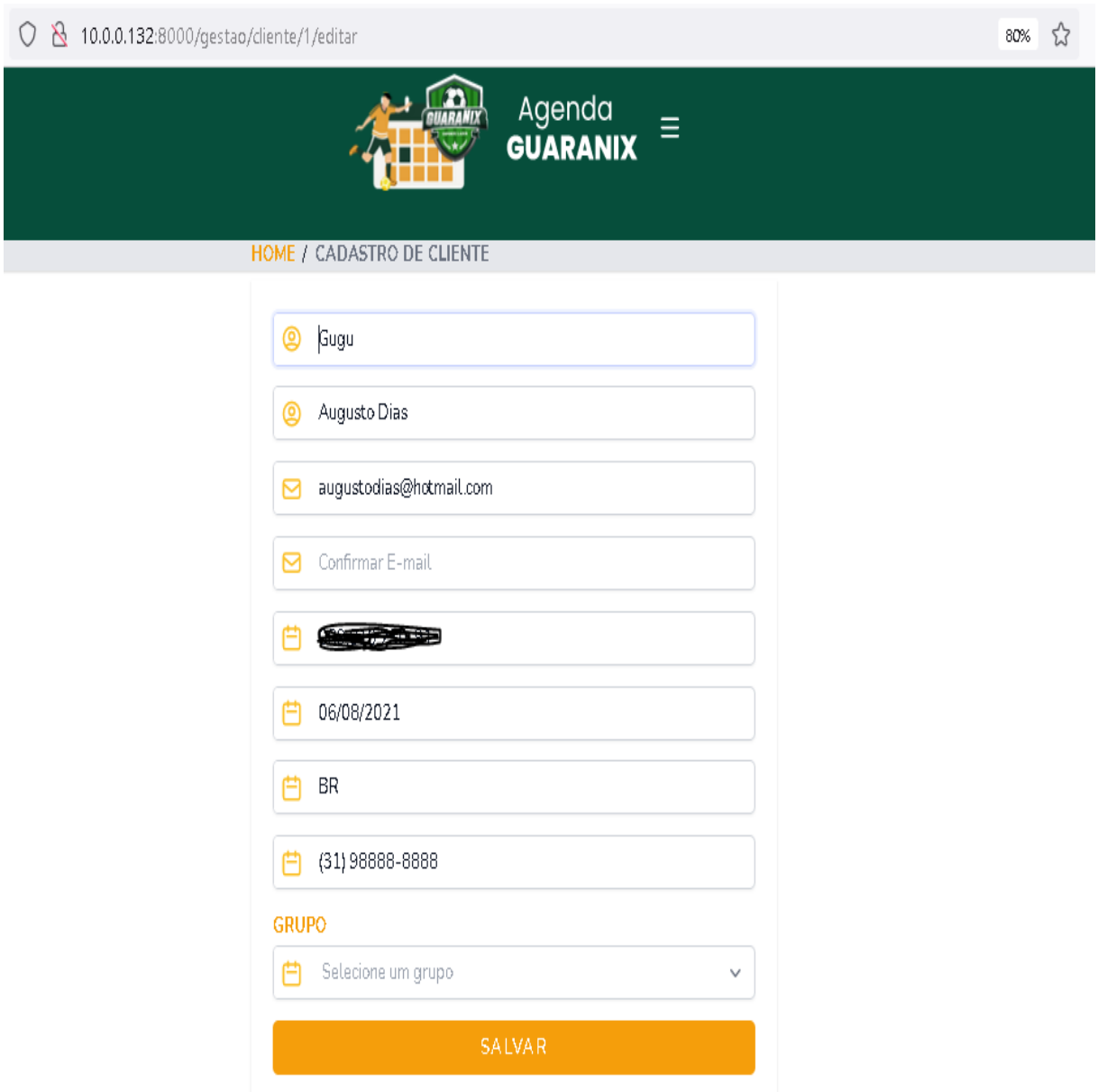


Figura 4.18 – Área do Gestor - Clientes - Editar Informações do Cliente.

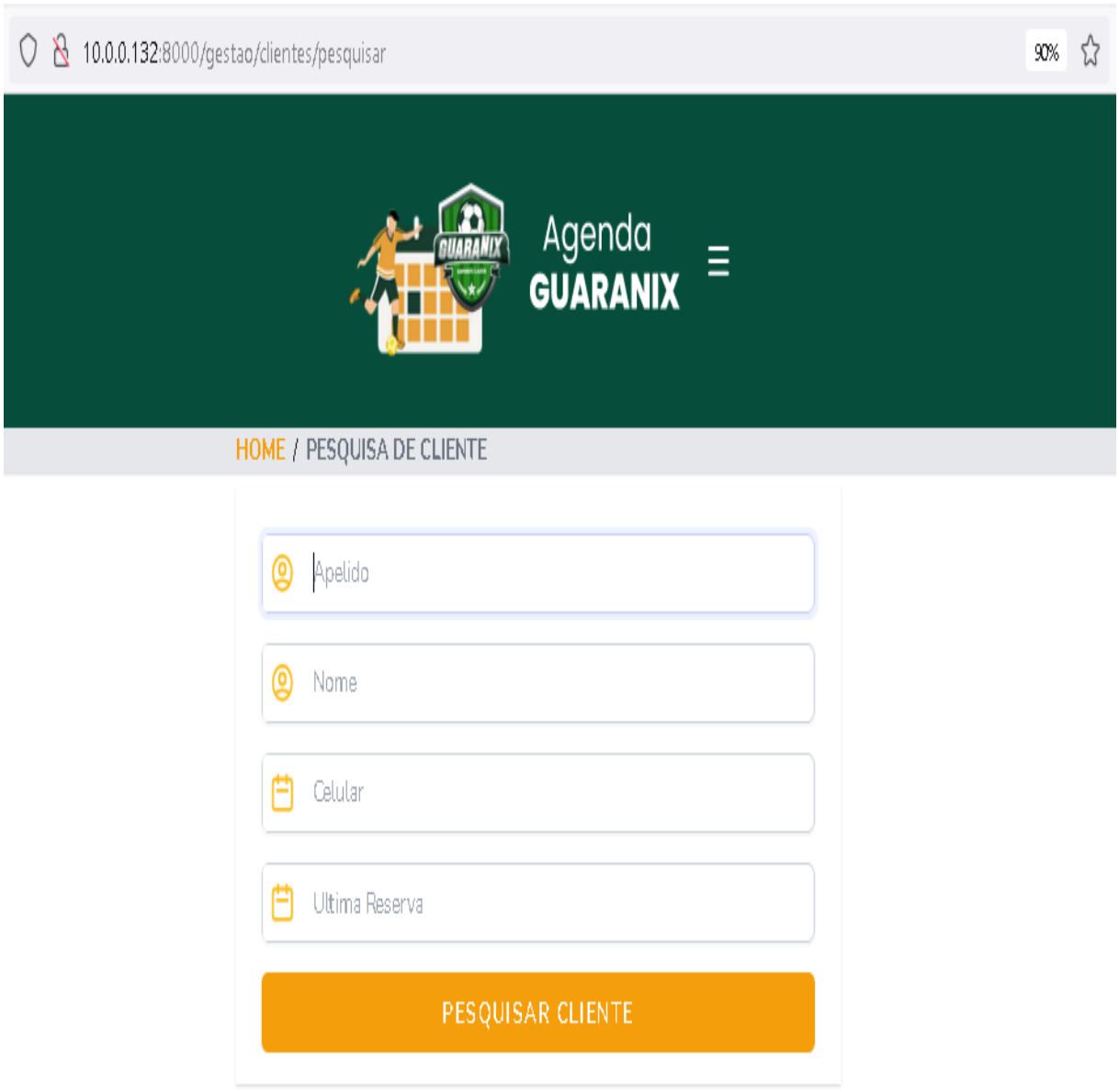


Figura 4.19 – Área do Gestor - Clientes - Pesquisa de Clientes.

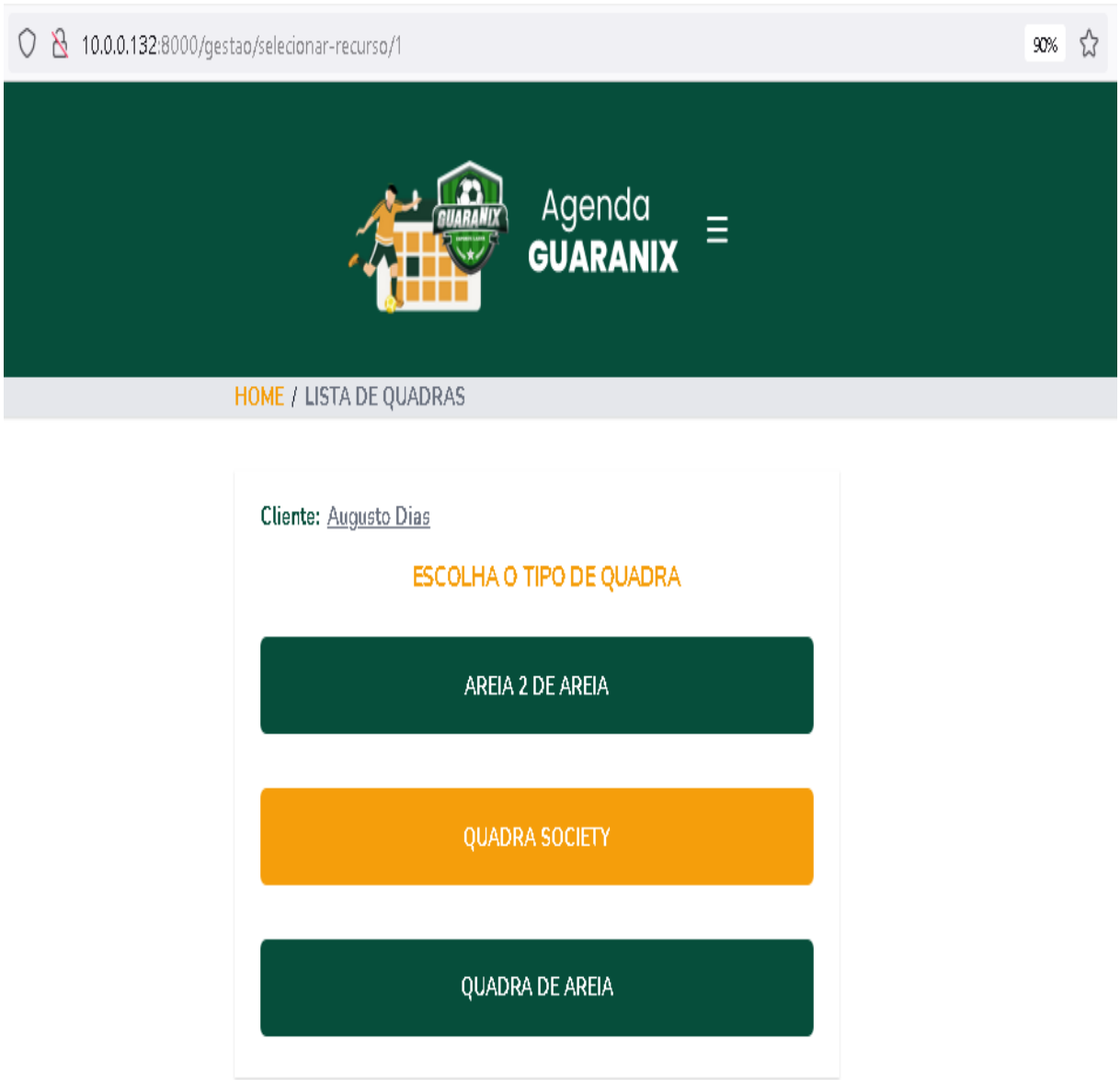


Figura 4.20 – Área do Gestor - Clientes - Realização de Reserva para um CLiente.

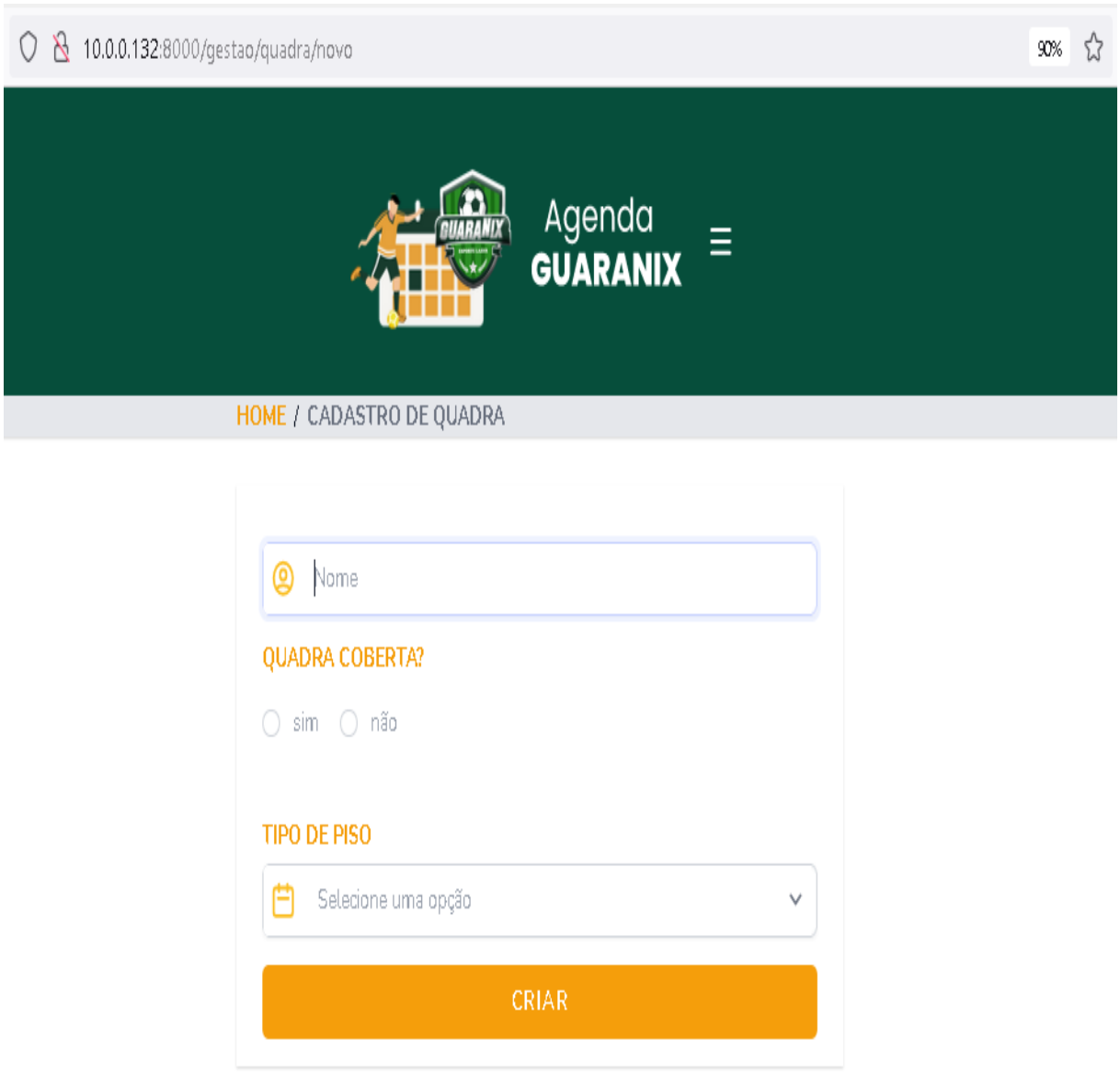


Figura 4.21 – Área do Gestor - Quadras - Instancia uma Nova Quadra.

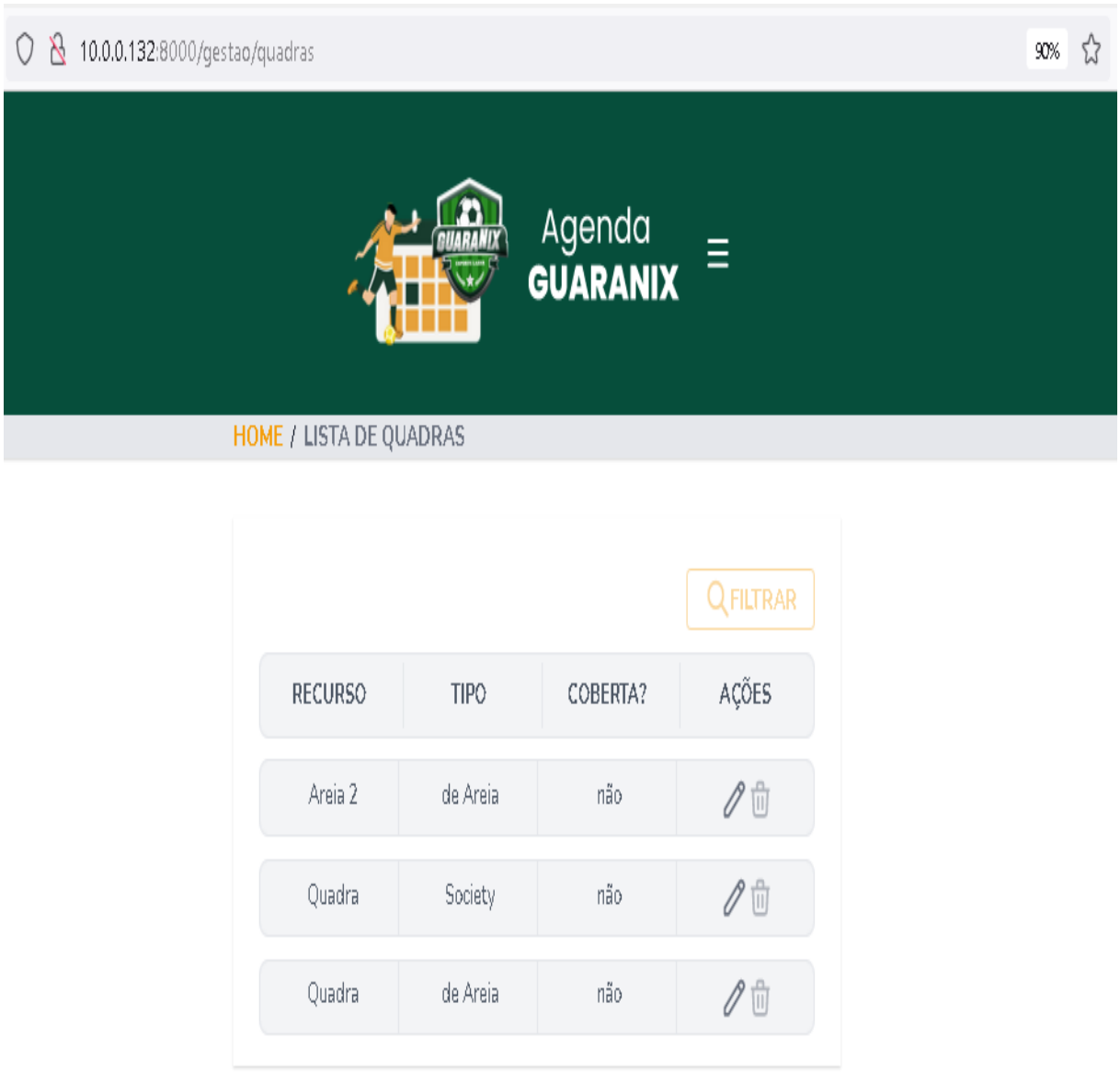


Figura 4.22 – Área do Gestor - Quadras - Lista Quadras Registradas.

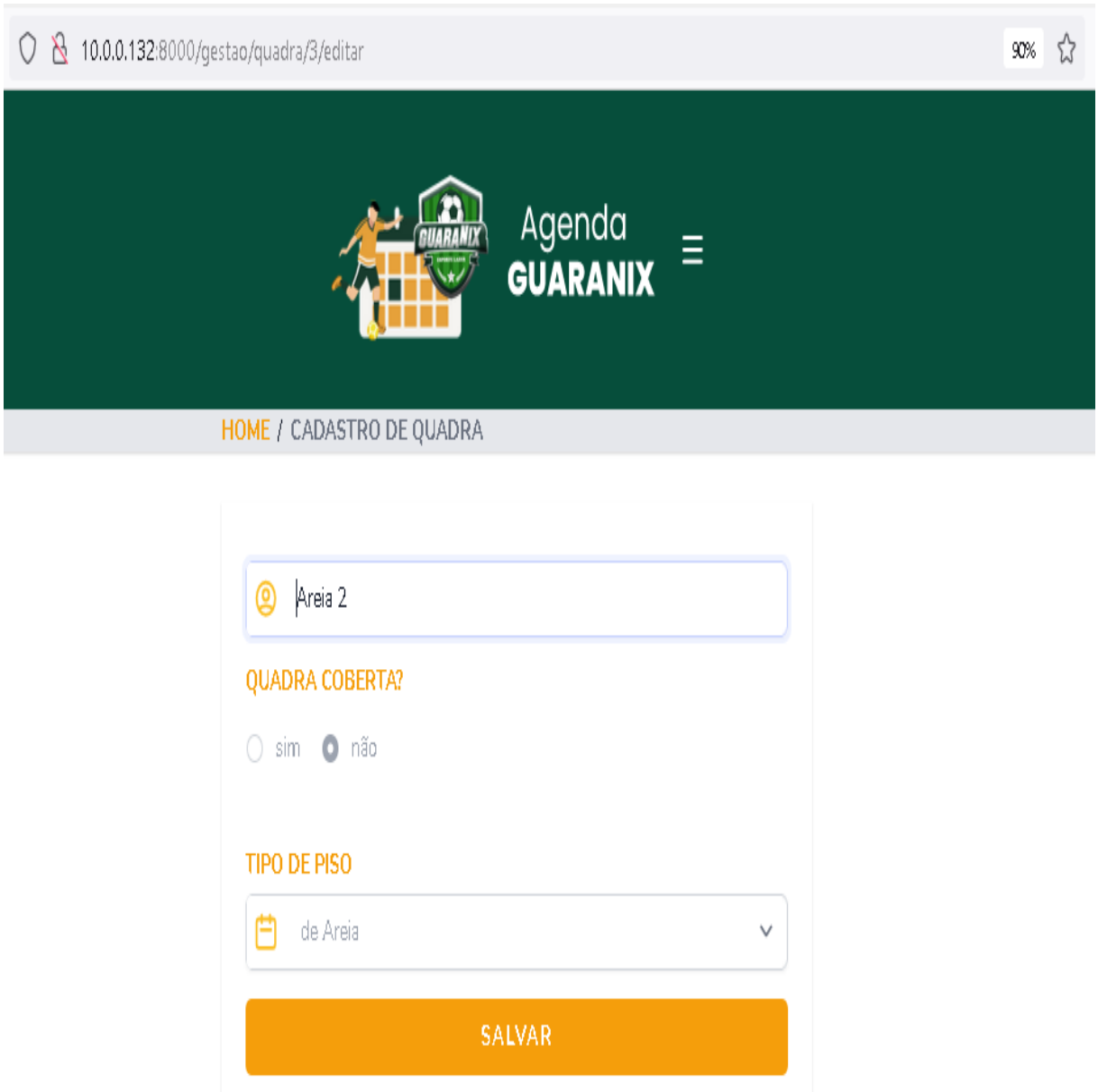


Figura 4.23 – Área do Gestor - Quadras - Edita as Informações de uma Quadra.

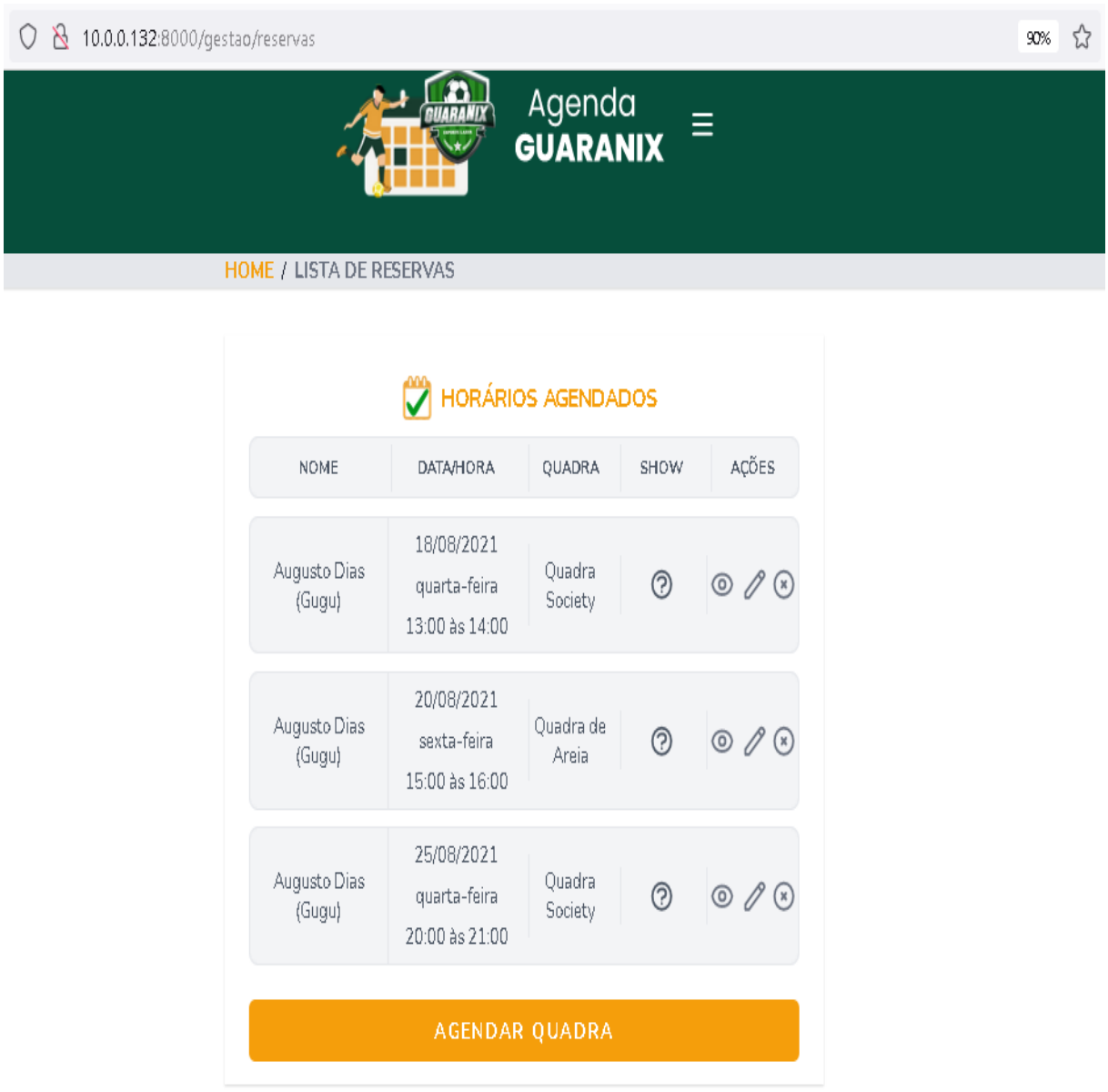


Figura 4.24 – Área do Gestor - Reservas - Lista as Reservas Existentes no Sistema.

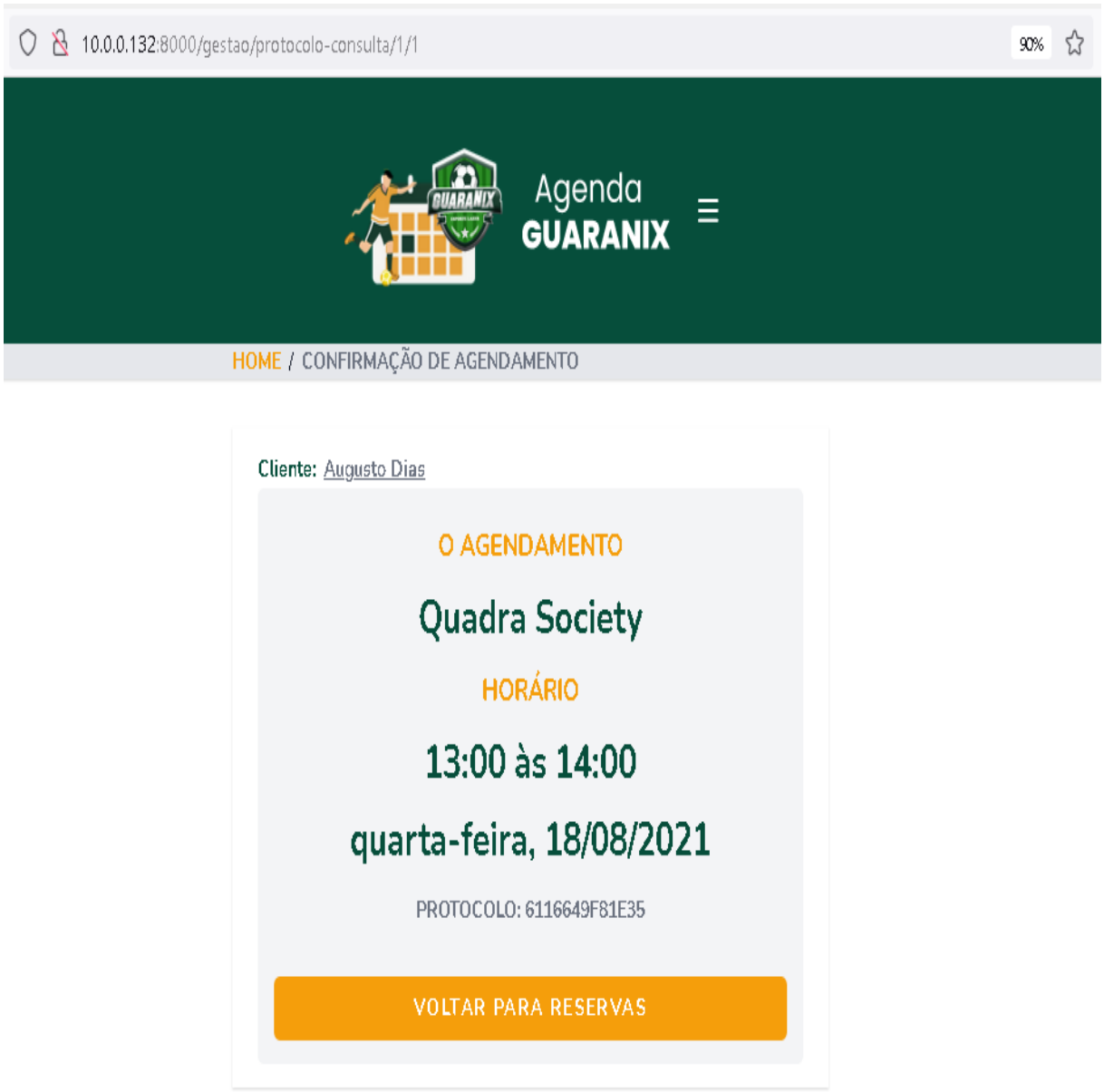


Figura 4.25 – Área do Gestor - Reservas - Mostra as Informações à Respeito da Reserva Selecionada.

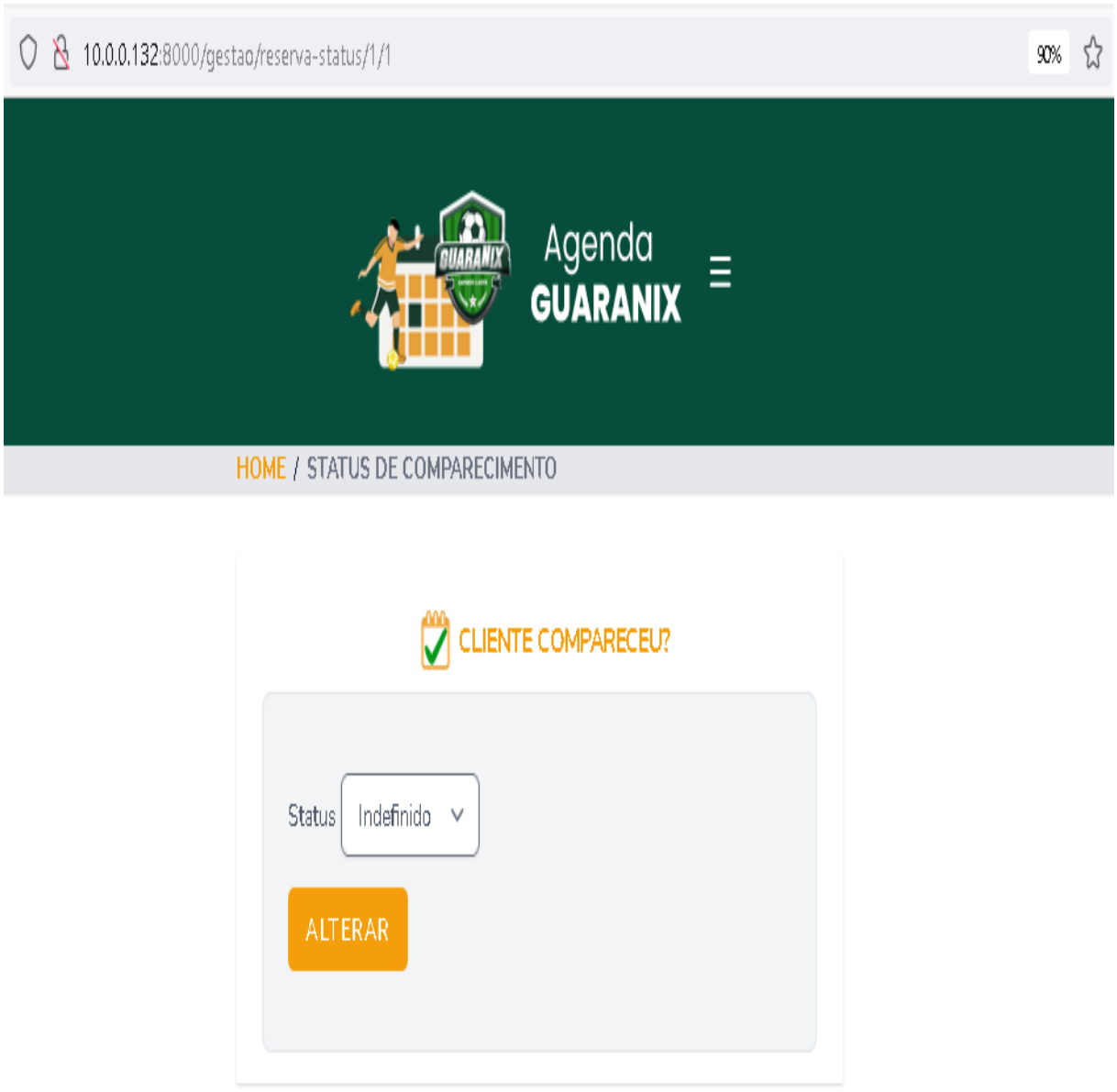


Figura 4.26 – Área do Gestor - Reservas - Altera o *status* da Reserva. Considerando a Presença ou não do Cliente.

Figura 4.27 – Área do Gestor - Reservas - Realiza o Cancelamento da Reserva.

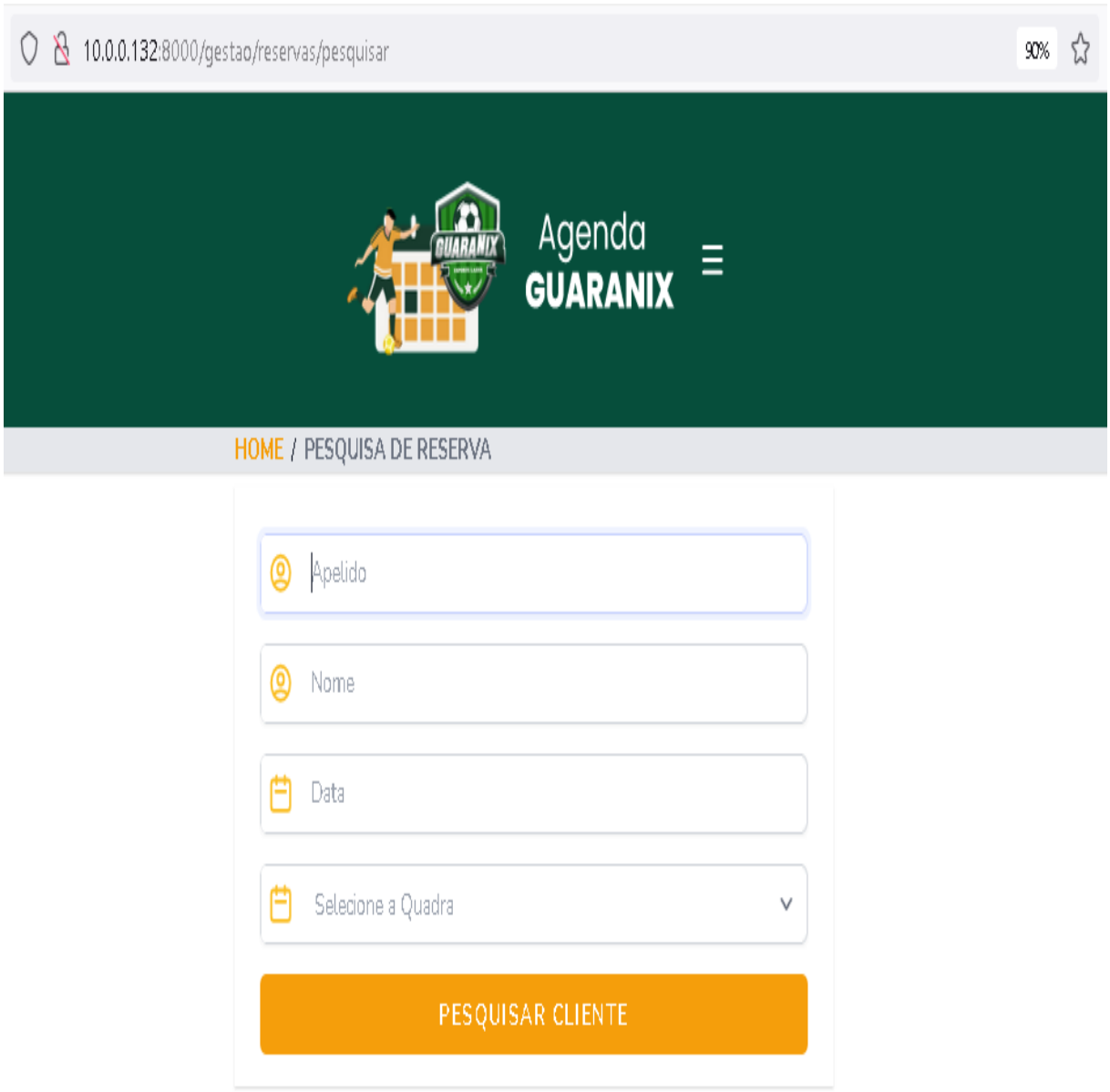


Figura 4.28 – Área do Gestor - Reservas - Pesquisa Reservas Compatíveis com as Informações Inseridas.

5 Considerações Finais

5.1 Conclusão

Os estudos acerca das ferramentas necessárias para desenvolver um projeto como este são de extrema importância. A análise destes materiais trás um bom entendimento do assunto e faz com que seja feito um planejamento ideal dos processos de desenvolvimento do software. E quando isso ocorre de forma correta, além do tempo economizado no decorrer do desenvolvimento, também ganha-se na qualidade do produto entregue.

A forma com que o projeto foi conduzido, com pequenas reuniões diárias de acompanhamento, se mostrou muito eficaz. Fez com que o processo de desenvolvimento não ficasse estagnado por um longo período de tempo, a cada encontro dúvidas eram sanadas e os afazeres atualizados. A evolução se tornou mais nítida no dia a dia e os resultados mais perceptíveis, além das entregas terem ocorrido mais rapidamente. E no decorrer de todo o processo foi vista a importância da fundamentação teórica. Que ajudou na superação de desafios e a saber em quais fontes procurar quando necessário.

E se tratando de desafios, muitos foram encontrados pelo caminho. O desenvolvimento do front-end acabou tomando mais tempo do que o esperado. Assim como o conhecimento restrito em Banco de Dados, que exigiu um pouco mais de tempo e esforço. Alguns problemas que pareciam facilmente solucionáveis se mostraram muito mais complexos do que se imaginava. Mas tudo isso reforça e ajuda a entender que nenhum software é fácil de desenvolver, não importa quão simples pareça o problema.

Outra característica significativa deste trabalho foi perceber que quando se trata de projetos viáveis, para o mercado e com um cliente acompanhando, mudanças no decorrer do processo são necessárias e bem-vindas. E neste ocorreram algumas com bastante impacto. Como a implementação do Pagamento Online, que deixou de ser uma exigência do cliente após perceber as dificuldades que isso traria para a aplicação com algumas decisões que teria que tomar e que impactam o cotidiano do clube.

Assim, os aprendizados foram para além do *framework* Laravel. Tratou-se de uma vivência de mercado, com prazos, entregas, decisões e muito trabalho em equipe.

5.2 Trabalhos Futuros

Para trabalhos futuros podemos considerar algumas ações, como:

- Implementar a tabela de precificação dos horários;

- Disponibilizar relatórios para os clientes contendo seu histórico de reservas;
- Adicionar mais informações como fotos e descrição das quadras;
- Implementação de um Registro Caixa para gerar relatórios que mostram a rentabilidade de cada quadra;
- Criar um CRUD para grupos, para que possa permitir que um grupo específico tenha uma tabela de preço específica;
- Implementar uma gestão de clientes e grupos, no qual o gestor pode definir qual grupo um determinado usuário fará parte;
- Implementação do Filtro de Quadras;
- Permitir que clientes editem o próprio perfil;
- Integrar um formato de pagamento online, se um dia vir a ser necessário.

Referências

ABREU, T. W. M.; BARRA, W. J.; BARREIROS, J. A. L.; COSTA, C. T. J. da. Estratégias de identificação paramétrica aplicadas a modelagem dinâmica de um servidor web apache. 2013.

ANDRADE, A. C. de. *O que é o CSS Tailwind e como posso adicioná-lo ao meu site ou ao React App?* 2020. Acesso em: 19/04/2021. Disponível em: <https://cibersistemas.pt/tecnologia/o-que-e-o-css-tailwind-e-como-posso-adiciona-lo-ao-meu-site-ou-ao-react-app/>.

BENTO, E. J. *Desenvolvimento Web com PHP e MySQL.* . ed. [S.l.]: Casa do Código, 2021.

CANTO, F. H. *Vulnerabilidade da Linguagem PHP*. Porto Alegre, Rio Grande do Sul: [s.n.], 2011. 36 p.

CONTE, T.; MENDES, E.; TRAVASSOS, G. *Processos de Desenvolvimento para Aplicação Web*: Uma revisão sistemática. 2005. 9 p. Acesso em: 23/03/2021. Disponível em: <https://docplayer.com.br/19139301-Processos-de-desenvolvimento-para-aplicacoes-web-uma-revisao-sistematica.html>.

CONVERSE, T.; PARK, J. *PHP 4*: A bíblia. . ed. [S.l.]: Edições Elsevier, 2003.

GABARDO, A. C. *Laravel para Ninjas*. 1. ed.. ed. [S.l.]: Novatec Editora Ltda, 2017. ISBN 9788575226063.

GOMES, R. G. Laravel: criando um *HelloHello Word*. 2019. Disponível em: <https://www.devmedia.com.br/laravel-criando-um-hello-world/40676>.

HENKE, G. Html 5.0. p. 12, 2009.

KAHLERT, T. *Código do Visual Studio (1): O básico*. 2015. Acesso em: 21/04/2021. Disponível em: <https://www.microsoft.com/de-de/techwiese/know-how/visual-studio-code-01-die-grundlagen.aspx>.

LATORRE, M. Historia de las web, 1.0, 2.0, 3.0 y 4.0. p. 8, 2018.

LECHETA, R. R. *Web Services Restful*: Aprenda a criar web services restful em java na nuvem do google. 1. ed.. ed. [S.l.]: Novatec Editora Ltda, 2015.

LOPES, S. D. C. *Os meios de pagamento online : o seu impacto na concretização da compra por parte de consumidor português*. Portugal: [s.n.], 2018. 96 p. Disponível em: <http://hdl.handle.net/11067/4496>.

LUCIANO, J.; ALVES, W. J. B. Padrão de arquitetura mvc: Model-view-controller. p. 6, 2011.

MARBONI, T. *Adquirente x subadquirente: entenda a diferença e escolha a melhor opção para o seu negócio*. 2019. Acesso em: 18/04/2021. Disponível em: <https://wirecard.com.br/blog/adquirente-ou-subadquirente/>.

MARCELO, A. *Apache*: configurando um servidor web para linux. 3. ed.. ed. [S.l.]: Brasport, 2005. ISBN 8574522538.

MARI, A. de. *Desenvolvimento de um Servidor HTTP/XML-RPC*. Florianópolis: [s.n.], 2004. 82 p.

MARQUES, G. V.; MEDEIROS, C. R.; PEREIRA, J. Q. Anáçise comparativa de desempenho de aplicação java com persistência em banco de dados mysql e mongodb. p. 8, 2018.

MARTINS, J. C. F. M. *QoS em servidores HTTP Apache*. 2015. 186 p.

MORAES, F. *Laravel Controller:*: 5 passos para desenvolver a lógica da sua aplicação. 2018. Acesso em: 17/08/2021. Disponível em: <https://www.dialhost.com.br/blog/laravel-controller/#3-8211-registrando-as-rotas>.

MOREIRA, M. *O e-commerce e o controlo dos meios de pagamento Online: caso de estudo Sonae*. Portugal: [s.n.], 2020. 52 p. Disponível em: <http://hdl.handle.net/10400.26/33622>.

MOSKOVITZ JOSEPH, R. L. e. M. *CSS Avançado*. 1. ed.. ed. [S.l.]: Novatec Editora Ltda, 2010. ISBN 9788575222201.

SALOMãO, G. H. S. *Desenvolvimento de aplicação web para gerenciamento de objetos de aprendizagem*. 2018. Acesso em: 21/04/2021. Disponível em: <https://repositorio.unesp.br/handle/11449/203632>.

SCHULZ, R. G. *Diseño web con CSS*. 1. ed.. ed. [S.l.]: Marcombo, S. A., 2017. ISBN 9788426714701.

SILVA, M. S. *Criando Sites com HTM*: Sites de alta qualidade com htm e css. 1. ed.. ed. [S.l.]: Novatec Editora Ltda, 2008. ISBN 9788575221662.

SILVA, N. H. da; CARVALHO, M. Análise das tecnologias requisitadas em vagas de desenvolvedor web de acordo com a localidade das empresas. Belém, Pará, p. 16, 2017.

VASCONCELOS, F. M. de. *Como criar as Models do seu projeto com Eloquent no Laravel*. 2017. Acesso em: 18/08/2021. Disponível em: <https://imasters.com.br/back-end/como-criar-as-models-do-seu-projeto-com-eloquent-no-laravel>.

WANDERLEY, G. S. M. *Desenvolvimento de aplicativo para relatório de sondagem SPT na plataforma Android*. 2017. Acesso em: 21/04/2021. Disponível em: <https://repositorio.ufpb.br/jspui/handle/123456789/13747>.

XAVIER, C. S. *CONFIANÇA NOS SERVIÇOS ONLINE*. 2009. Acesso em: 19/04/2021. Disponível em: <https://repositorio-aberto.up.pt/bitstream/10216/59962/1/000137176.pdf>.