UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ANDREI ARIEL APPELT

ELISSON ROBERTO DE ANDRADE

DESENVOLVIMENTO DE APLICATIVO MOBILE E API DE INTEGRAÇÃO UTILIZANDO A ARQUITETURA DE MICRO SERVIÇOS

PONTA GROSSA

2021

ANDREI ARIEL APPELT

ELISSON ROBERTO DE ANDRADE

DESENVOLVIMENTO DE APLICATIVO MOBILE E API DE INTEGRAÇÃO UTILIZANDO A ARQUITETURA DE MICRO SERVIÇOS

Mobile application development and integration api using a microservices architecture

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Richard Duarte Ribeiro

Coorientador: Prof. MSc. Vinícius Camargo Andrade.

PONTA GROSSA

2021

ANDREI ARIEL APPELT

ELISSON ROBERTO DE ANDRADE

DESENVOLVIMENTO DE APLICATIVO MOBILE E API DE INTEGRAÇÃO UTILIZANDO A ARQUITETURA DE MICRO SERVIÇOS

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 24/novembro/2021

Richard Duarte Ribeiro
Doutor
Universidade Tecnológica Federal do Paraná

Vinícius Camargo Andrade
Mestre
Universidade Tecnológica Federal do Paraná

Simone de Almeida
Doutora
Universidade Tecnológica Federal do Paraná

Geraldo Ranthum
Mestre
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2021

AGRADECIMENTOS

Primeiramente agradecemos a Deus que sem ele não estaríamos aqui e não teríamos dom e capacidade para escrever este trabalho.

O desenvolvimento deste trabalho contou com a ajuda de diversas pessoas que de alguma forma se envolveram um pouco na nossa rotina durante alguns meses para que esse trabalho fosse concluído, dentre elas os professores MSc. Vinícius Camargo Andrade e Dr. Richard Duarte Ribeiro que com muita dedicação nos auxiliaram em todos os momentos no decorrer do projeto.

Também agradecemos a nossa família que sempre nos apoiou em todos os momentos em que deixamos ela um pouco de lado para focar em nossos objetivos, e a toda comunidade da UTFPR, desde professores, funcionários e nossos colegas de classes que desde o princípio nos auxiliaram.

RESUMO

A grande maioria dos restaurantes ainda utilizam comandas feitas em papel, fato este que não é seguro, pois o papel pode ser extraviado, perdido, ou até mesmo difícil de entender, pois na maioria das vezes o garçom escreve com muita pressa. Também não se tem um controle de qual garçom é mais eficiente por exemplo. Com base nestas dificuldades foi visitado um restaurante e realizada uma entrevista com a proprietária. Foram levantados os requisitos necessários para o projeto de um aplicativo para realização dos pedidos, substituindo a caneta e papel. Após a análise dos requisitos, foi desenvolvido um aplicativo mobile, para realização de pedidos utilizando o Framework Ionic. O intuito deste trabalho é demonstrar uma arquitetura de Micro serviços e discutir como ela pode ser utilizada para simplificar as interações com diferentes tipos de tecnologias, neste caso mobile, desktop e web. Para o desenvolvimento dos micro serviços foi utilizado o Framework Spring. O banco de dados usado foi o MySQL, que é utilizado pelo sistema de ponto de venda do estabelecimento desenvolvido em Delphi. Os resultados obtidos foram um aplicativo Android, com cadastro de garçons, cadastro de comandas, lista de produtos, e realização de pedidos, atribuindo a comanda os produtos selecionados pelo cliente.

Palavras-chave: Aplicativo mobile; Micro serviços; Framework; Spring; Ionic.

ABSTRACT

The vast majority of restaurants still use paper commands, a fact that is not safe, as the paper can be misplaced, lost, or even difficult to understand, as most of the times the waiter writes in a hurry. There is also no perfect control of which waiter is more efficient, for example. Based on these difficulties, a restaurant was visited and, based on an interview with the owner, the necessary requirements were raised for the design of an application for placing orders, replacing pen and paper. After analyzing the requirements, a mobile application was developed to carry out orders using the Ionic Framework. This work purpose is to demonstrate a Microservices architecture and discuss how it can be used to simplify interactions with different kinds of technologies, in this case mobile, desktop and web. For the development of micro services, the Spring Framework was used. The database used was MySQL, which is used by the establishment's point of sale system, which is developed with Delphi. The results obtained were an Android application, with registration of waiters, registration of orders, list of products, and placing of orders, assigning the orders to the products selected by the customer.

Keywords: Mobile; Microservices; Framework; Spring; Ionic.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BD	Banco de Dados
CAGR	*Compound Annual Growth Rate*
CORS	*Cross-origin Resource Sharing*
CRUD	*Create, Read, Update, Delete*
CSS	*Cascading Style Sheets*
ERP	*Enterprise Resource Planning*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
ISV	*Independent Software Vendor*
J2EE	*Java 2 Enterprise Edition*
JMS	*Java Message Service*
JPA	*Java Persistence API*
JTA/JCA	*Java Transaction API/Java EE Connector Architecture*
MIT	*Massachusetts Institute of Technology*
NFC-e	Nota Fiscal de Consumidor Eletrônica
OEM	*Original Equipment Manufacturer*
PDA	*Personal digital assistants*

SUMÁRIO

1 INTRODUÇÃO

A evolução da tecnologia está ocorrendo rapidamente. Isto é evidenciado pela quantidade de novas tecnologias que vem surgindo em todas as áreas do conhecimento, ferramentas estas que abrangem técnicas e equipamentos como dispositivos móveis, sistemas distribuídos, processamento paralelo, para citar algumas delas. Os dispositivos móveis tais como *smartphones*, *tablets*, *Personal Digital Assistant* (PDA) podem ser integrados com outros dispositivos como *notebook*, ou outros *smartphones*, estabelecendo assim uma rede de tecnologia da informação e comunicação (AULIA, 2017).

Android é um sistema operacional de código aberto para dispositivos móveis, é um projeto liderado pela Google. Como um projeto de código aberto, o objetivo do Android é evitar qualquer ponto central de falha em que um participante da indústria possa restringir ou controlar as inovações de qualquer outro participante. Para este fim, Android é um sistema operacional completo com qualidade de produção para os produtos consumidores, completo com código fonte que pode ser customizado e portado para praticamente qualquer dispositivo, com documentação publica que está disponível para todos (ANDROID, 2019).

Em um restaurante, por exemplo, as etapas do pedido do menu, geralmente apresentam alguns problemas, por exemplo: pode levar muito tempo porque o pedido é realizado de forma manual, podendo também não ser entendido pelos colaborados da cozinha, devido a grafia incorreta, assim como existe a possibilidade de troca do pedido com o de outros clientes. Outro problema seria a disponibilidade do item em estoque (AULIA, 2017).

Este sistema tem sido utilizado na maioria dos restaurantes. Nestes, os cardápios oferecidos aos clientes são feitos de papel, garçons utilizam notas para registrar os pedidos dos clientes, e sempre que o cliente visita o restaurante seleciona seu pedido por uma carta de menu.

Dessa maneira é possível que o pedido seja danificado por contato com água, por mau manuseio, ou extraviado. Para Poonan et al (2016) existe ainda o desperdício de tempo, dinheiro e papel. Outro fator relevante é que depois de alguns dias de uso os menus impressos perdem seus atributos visuais.

Do ponto de vista do cliente, este sistema é demorado, já que ele tem que aguardar que o garçom venha pegar o pedido. Existe ainda a possibilidade de

confusão por parte do garçom, já que as pessoas podem confundi-lo ao falar ao mesmo tempo. Outra falha pode ocorrer quando o pedido é entregue, já que pode ser enviado um prato errado (POONAN et al, 2016).

Neste contexto, este trabalho é concebido com o objetivo de utilizar as tecnologias disponíveis para o desenvolvimento de um aplicativo móvel e uma API utilizando uma arquitetura de micro serviços para integrar os dados dos pedidos de um restaurante com um sistema de gestão empresarial, denominado *Enterprise Resources Planning* (ERP). O aplicativo possui o foco voltado ao sistema operacional Android visto que atualmente é o sistema mais utilizado, com mais de 2,5 bilhões de usuários ativos (CURRY, 2021), além de ser um projeto de código aberto liderado pela Google [1], pode ser customizado e portado para qualquer dispositivo e sua documentação é pública (ANDROID, 2019).

1.1 Objetivos

Nas próximas duas seções serão descritos o objetivo geral e também os objetivos específicos do projeto desenvolvido.

1.1.1 Objetivo Geral

Desenvolver um sistema que facilite o trabalho em um estabelecimento de venda e consumo de produtos alimentícios, como restaurante ou lanchonete.

1.1.2 Objetivos Específicos

Para atingir o objetivo geral desta pesquisa, os seguintes objetivos específicos foram definidos:

- Estudar os problemas de atendimento em um determinado estabelecimento de vendas de produtos alimentícios, no qual o pedido e comanda são feitos manualmente;
- Manter um cardápio informatizado, facilitando o acesso e atualização de preços e itens;

[1] Site da Google: https://www.google.com.br

- Desenvolver uma API para integração, que disponibilize os dados dos pedidos feitos por meio de um dispositivo móvel, para integração com um sistema ERP de terceiros para ser finalizado o processo de compra e pagamento.

2 REFERENCIAL TEÓRICO

O referencial teórico deste trabalho é constituído de cinco seções: *Enterprise Resource Planning*, engenharia de software, banco de dados, *frameworks* e sistemas similares.

2.1 Enterprise Resource Planning

Com o crescente avanço da tecnologia dentro das indústrias, e a necessidade de cada vez mais integrar setores para evitar que informações sejam perdidas entre um processo e outro surgiu a necessidade de utilizar um sistema informatizado que trabalhe integrando todos estes setores. Assim surgiu o conceito dos sistemas *Enterprise Resource Planin*g (ERP), ou sistema de gestão empresarial. Segundo Alsené (1999) o conceito de ERP existe desde a década de 60, quando surgiram os primeiros computadores. Porém os primeiros registros de sistemas que realmente utilizam a tecnologia são de 1972. Sua expansão foi aproximadamente no ano de 1990, após o bug do milênio, as empresas buscavam cada vez mais informatizar seus processos.

Hoje é muito utilizado em empresas de grande porte, que buscam uma melhor organização e padronização de seus processos, os sistemas ERP auxiliam nesta integração (DAVENPORT, 1998).

Segundo Souza (2000), para melhor organizar um sistema ERP, ele é dividido em quatro principais blocos:

- Financeiro: Consiste no conjunto de módulos que integram todas as formas de movimentação de valores. Os principais módulos do bloco financeiro são: Contas a pagar, contas a receber, movimento bancário e fluxo de caixa. (SOUZA, 2000);
- Recursos Humanos: O bloco de recursos humanos gerencia os dados e informações referentes aos colaboradores da organização. Ainda Para Souza (2000), os principais módulos de recursos humanos são: folha de pagamento, agendamento de férias, cálculo de horas de folha ponto, entre outros;

- Logística: No bloco de logística estão agrupados os módulos que gerenciam o movimento de cargas. Os principais itens que constituem este bloco são: controle de estoque, faturamento e planejamento de recursos. (SOUZA, 2000);
- Vendas e Marketing: Abrange itens referentes ao relacionamento entre o cliente e a organização. Para Souza (2000), os principais módulos são: processamento de pedidos e o controle de vendas.

A Figura 1 ilustra a estrutura típica de funcionamento de um sistema ERP e seus módulos integrados.

Figura 1 - Estrutura típica de funcionamento de um ERP

Fonte: Neil Patel (2018)

Os sistemas ERP integram todos os processos das organizações, e de uma complexidade grande, que possuem algumas vantagens e desvantagens.

Pode-se citar vantagens da utilização do sistema como a coleta das informações da organização em tempo real, melhora na comunicação entre diversos setores, economia de tempo, diminuição de tempo de espera entre a saída e uma informação de um setor e a chegada desta informação em outro setor, diminuição do

lead time de produção, economia de papel, controle do ciclo produtivo ERP (MESQUITA, 2000).

Devido ao sistema ERP ser uma solução com grandes dimensões e complexidades, ele exige uma grande atenção em alguns pontos, antes, durante e após a implantação para que o sistema por um todo funcione perfeitamente. Segundo Mesquita (2000), pode-se citar como desvantagens de sistemas ERP o Custo elevado, a demora na implantação e a complexidade de customização.

Integrado ao sistema ERP, existem diversos outros sistemas, como por exemplo o Sistema de Ponto de Venda (PDV) o qual está associado principalmente a parte de realizar o lançamento das vendas, como consequência, tem envolvimento também com estoque e financeiro.

2.1.1 Sistema de Ponto de Venda (PDV)

Sistema de Ponto de Venda, ou simplesmente PDV, é um termo utilizado para definir o local em que a venda de um produto ou serviço é efetivada. Segundo o SEBRAE (2018), PDV é o local onde o produto ou serviço é exposto, por tempo limitado, de forma clara a ser avaliado pelo consumidor. Além disso, um PDV, deve ser composto de vários itens indispensáveis que juntos o tornam um diferencial no momento da fidelização de um cliente, entre eles estão, fachada, iluminação, música, promoções, equipamentos e software.

Um sistema de PDV é uma função que pode ser integrada ao sistema ERP, conectando informações de venda com dados de controle gerenciais, como financeiro e estoque de um negócio. Em outras palavras o sistema PDV, é o módulo de um sistema em que a venda ao consumidor é finalizada, é o último contato do vendedor com o cliente (SANKHYA, 2021).

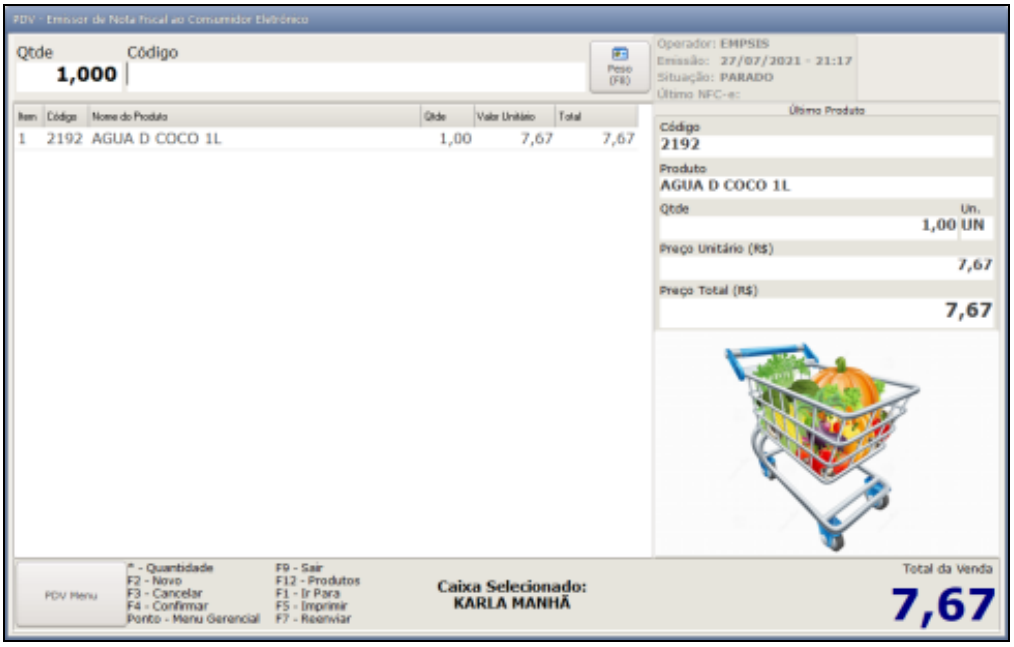
De forma geral o PDV possui várias funções que se não seguidas corretamente, acarretará em um mau funcionamento do negócio. Como exemplo, um sistema PDV deve permitir que o caixa seja aberto e fechado várias vezes ao dia, como exemplo, uma farmácia em que são três expedientes e, consequentemente, o caixa é aberto três vezes ao dia (SEBRAE, 2018).

Neste ponto também entram as retiradas de dinheiro do caixa, processo que geralmente é especialmente feito pelo PDV. Para que haja uma sincronia entre caixa

(gaveta) e caixa sistema, é necessário que o fluxo do caixa acompanhe as vendas físicas, assim, ao final do expediente, o relatório emitido pelo PDV deve ser igual ao físico realizado pelo operador do caixa (SEBRAE, 2018).

Segundo a Sankhya (2021), todo e qualquer sistema de PDV deve emitir a Nota Fiscal de Consumidor Eletrônica (NFC-e), e, posteriormente, realizar a impressão para entregar ao cliente do estabelecimento. A NFC-e é o comprovante de venda, nela constam as informações dos produtos adquiridos, estas devem ser transmitidas ao fisco a fim de recolhimento de impostos. Um PDV deve ser um sistema simples de utilizar, a Figura 2 apresenta um sistema de PDV para mercado.

Figura 2 - Sistema de PDV para mercado



Fonte: Autoria própria (2021)

É de responsabilidade do sistema de PDV realizar um controle de estoque correto dentro de uma empresa. Para isso, é necessário que o PDV trabalhe em sincronia com a gestão do estoque. Isso se dá pelas saídas automáticas nos produtos vendidos para que os almoxarifes consigam controlar o estoque de uma forma clara sem atrapalhar as vendas dentro do negócio (TOTVS, 2021).

O gerenciamento de cadastros e consultas de clientes e produtos também é de responsabilidade do sistema de PDV. Para facilitar a venda, é de fundamental importância que o operador consiga consultar o estoque de um produto ou de um cliente para verificar se este é um cliente fiel para, por exemplo, dar um desconto maior em uma venda. De acordo com Mordor (2021) o mercado de terminais de

ponto de venda está crescendo a um CAGR (Compound Annual Growth Rate) taxa de crescimento anual estimada em 8,3% no período entre 2021 e 2026.

Geralmente um PDV é composto por um balcão com banqueta, um computador, impressora térmica, leitor de código de barras e maquineta de cartão magnético (SEBRAE, 2018).

2.2 Engenharia de Software

O termo Engenharia de Software foi criado na década de 60, com o intuito de formalizar e padronizar os processos de planejamento, desenvolvimento e manutenção de um *software*. Todo o processo de desenvolvimento, desde elicitação de requisitos, visitas, reuniões, definições de ferramentas, linguagem de programação, tipo de banco de dados, entre outros pontos técnicos são definidos dentro da engenharia de *software*.

Segundo Sommerville (2007), a engenharia de *software* desempenha um papel importante não só nos conceitos técnicos de desenvolvimento, mas também no projeto do *software*, desenvolvimento de ferramentas e *frameworks*, e até de métodos de produção de *software*. Ou seja, é a área da tecnologia da informação que acompanha o processo de programação do sistema desde a análise e definição de requisitos até a manutenção e desenvolvimento de customizações da versão final do projeto.

Bourque e Fairley (2014) classificou a engenharia de *software* em diversas áreas de conhecimento, as quais ficaram conhecidas como *Software Engineering Body of Knowledge* (SWEBOK) *Guide*. Esta classificação visa definir limites entre a área de engenharia de *software* e demais disciplinas de computação, algumas das áreas definidas por (BOURQUE e FAIRLEY, 2014):

- Requisitos de software, é a etapa em que são feitos os levantamentos de todas as funções que o sistema deve desempenhar, é neste momento que são feitas entrevistas com o cliente do projeto;
- Projeto de *software*, aqui são elencados todos os requisitos e documentados, gerando diagramas e gráficos, é o momento que antecede o desenvolvimento;

- Construção de *software*, é nesta etapa que o programador entra em cena, em que são gerados os códigos-fonte do projeto, é neste momento que deve começar as primeiras entregas de versões;
- Teste de *software*, é nesta etapa em que são disponibilizadas as versões de testes para que o cliente inicie os testes;
- Manutenção de *software*, é a etapa em que são feitas adaptações que surgem para correção de *bugs*, ou melhorias nos requisitos do sistema.
- Gerência de configuração de software, é a área de engenharia que controla a mudança de versões do *software*, auditando alterações, e gerando novas documentações;
- Gerência de engenharia de *software*, coordena todas as etapas acima citadas, geralmente é composta por, pelo menos, um gerente, mas depende do tamanho da equipe de projeto;
- Processos de engenharia de *software*, aqui são controlados os processos de documentação, gestão de pessoas, controle de horas, e relacionamentos pessoais dentro da equipe;
- Ferramentas e métodos de engenharia de *software*, área em que são gerenciadas as ferramentas necessárias para projeto e desenvolvimento do *software*, incluindo controle de licenças, avaliação de viabilidade, entre outros;
- Qualidade de *Software*, é a área que fiscaliza e audita se o produto final atende aos requisitos funcionais e não funcionais definidos no setor de requisitos de *software*.

Segundo Jalote (2005), ao iniciar um projeto de *software*, quanto mais complexo for o sistema, é necessário um melhor gerenciamento do engenheiro de software. Jalote (2005), divide o processo de engenharia de *software* em quatro atividades:

- Análise de Requisitos, atividade na qual é necessária total atenção do analista de sistemas, pois é na análise de requisitos, que são definidos todas as funções que o sistema irá desempenhar, ela geralmente é feita em etapas, utilizando técnicas de visitas, entrevistas, *brainstorming* (reuniões com o cliente), e até mesmo vivenciando um dia de trabalho do cliente;

- *Design* do *software*, é no *design* de *software* que são documentados os requisitos, e gerados os diagramas de caso de uso, diagrama de classes, e são feitas as análises de tempo necessário para o desenvolvimento;
- Codificação, é a etapa de colocar em prática toda a análise anterior, e desenvolver o projeto planejado. É nela que são geradas as primeiras versões;
- Teste, momento que são realizados os primeiros testes em homologação junto ao cliente solicitante do projeto, caso os testes sejam satisfatórios, o projeto pode ser considerado como finalizado.

Na primeira etapa, Análise de Requisitos, ocorre o processo de engenharia de requisitos que visa coletar, junto ao cliente, todos os dados necessários, além das exigências por parte do cliente/usuário do sistema, que o *software* irá conter. Ou seja, é nesta primeira etapa que são determinadas as funcionalidades do produto final (SOMMERVILLE, 2007).

2.2.1 Engenharia de Requisitos

O processo de Engenharia de requisitos é essencial dentro do processo de desenvolvimento de um *software*, seja ele grande ou pequeno. É nessa etapa que se define suas funcionalidades e a capacidade de adaptação a cada alteração de funcionalidades reportada pelo cliente (SOMMERVILLE, 2003).

Para Pressman (2006) nesta etapa o engenheiro de *software* deve manter um relacionamento próximo com o cliente, dono do projeto, conservando assim um fluxo de reuniões semanais para captação de novos requisitos ou até mesmo melhorias em módulos já existentes.

Dentre os requisitos citados pelos engenheiros de *software*, estes podem ser classificados em requisitos funcionais e não funcionais. Os requisitos funcionais, são aqueles que são funções que o aplicativo deve realizar, ou seja, são ações que um sistema deve ser capaz de executar, como por exemplo, cadastro de pessoas físicas, controle de estoque, edição de funcionários, entre outros. Em contrapartida, os requisitos não funcionais são as restrições as quais o sistema deve operar, como por exemplo, responsividade, característica a qual um sistema adapta-se em diferentes resoluções e tamanhos de telas e variados dispositivos, confiabilidade,

capacidade do software em executar funcionalidades requeridas sob condições específicas, disponibilidade, o aplicativo deve estar disponível sempre que necessária a sua utilização, manutenibilidade, o sistema deve ser de fácil manutenção, seja para inserir novas funcionalidades ou corrigir erros encontrados, entre outros.

O processo de elicitação de requisitos, segundo Sommerville (2007), é constituído por quatro etapas: descoberta de requisitos, classificação e organização de requisitos, priorização e negociação de requisitos, e especificação de requisitos.

Descoberta de requisitos: Etapa em que são feitas as visitas, para levantamento dos requisitos solicitados pelo cliente.

Classificação e organização de requisitos: É a etapa em que o engenheiro de *software* separa os requisitos funcionais e não funcionais, e avalia o que é viável ou não fazer.

Priorização e negociação de requisitos: Etapa em que são separados os requisitos conforme prioridade definida junto com o cliente.

Especificação dos requisitos: Processo em que os requisitos são documentados e explicados com mais clareza para a equipe de desenvolvimento.

Segundo Sommerville (2004), para uma correta e completa elicitação de requisitos, é necessário seguir técnicas. As mais conhecidas são as técnicas de entrevistas abertas e fechadas, *workshops* e etnografia.

Para Sommerville (2003) a entrevista é uma das técnicas mais tradicionais e mais utilizadas, a entrevista geralmente é composta por um ou dois entrevistados e um engenheiro de *software*, ela deve ser planejada, para não ser muito longa e cansativa, o engenheiro deve elaborar um questionário para que sejam feitas perguntas, mas também deve estar pronto para ouvir todos os questionamentos do entrevistado.

Ao contrário da entrevista, o *workshop* geralmente é realizado com um grupo de pessoas, de um lado analistas da equipe de desenvolvimento e de outro os *stakeholders* do negócio ou da área de empresa em que está sendo desenvolvido o projeto, geralmente os *workshops* são realizados com um organizador principal, e utilizam a técnica de *brainstorming*, em que cada um cita sua ideia para ser avaliada pelo grupo (SOMMERVILLE, 2003).

A etnografia consiste em o analista vivenciar um dia de trabalho no ambiente em que o sistema será utilizado, evidenciando assim os problemas e todas as atividades que poderão ser incorporadas ao *software* (SOMMERVILLE, 2003).

2.2.1.1 Requisitos funcionais

Requisitos funcionais são funções ou atividades que o sistema deve desempenhar, satisfazendo uma expectativa ou desejo do usuário para com o *software*, geralmente são indicados por verbos, como cadastrar, consultar e permitir. Geralmente os requisitos funcionais são avaliados pelos usuários no momento da entrega do *software*, através de testes de funcionalidades Sommerville (2007).

Os requisitos funcionais, ou seja, aqueles que são funções solicitadas pelo cliente e que o aplicativo deve realizar são: cadastrar novos pedidos/comandas, consultar e editar pedidos/comandas já cadastradas, consultar produtos já cadastrados, cadastrar usuários/garçons.

2.2.1.2 Requisitos não funcionais

Para os requisitos não funcionais, não importa o que o sistema deve fazer, mas sim, como ele deve fazer, entra neste conceito a parte de desempenho do sistema, em muitos casos não é avaliado diretamente pelo cliente, mas deve ser tomado muito cuidado com relação a estes requisitos. É com base nestes requisitos que se determina a qualidade do *software* (PRESMMAN, 2006).

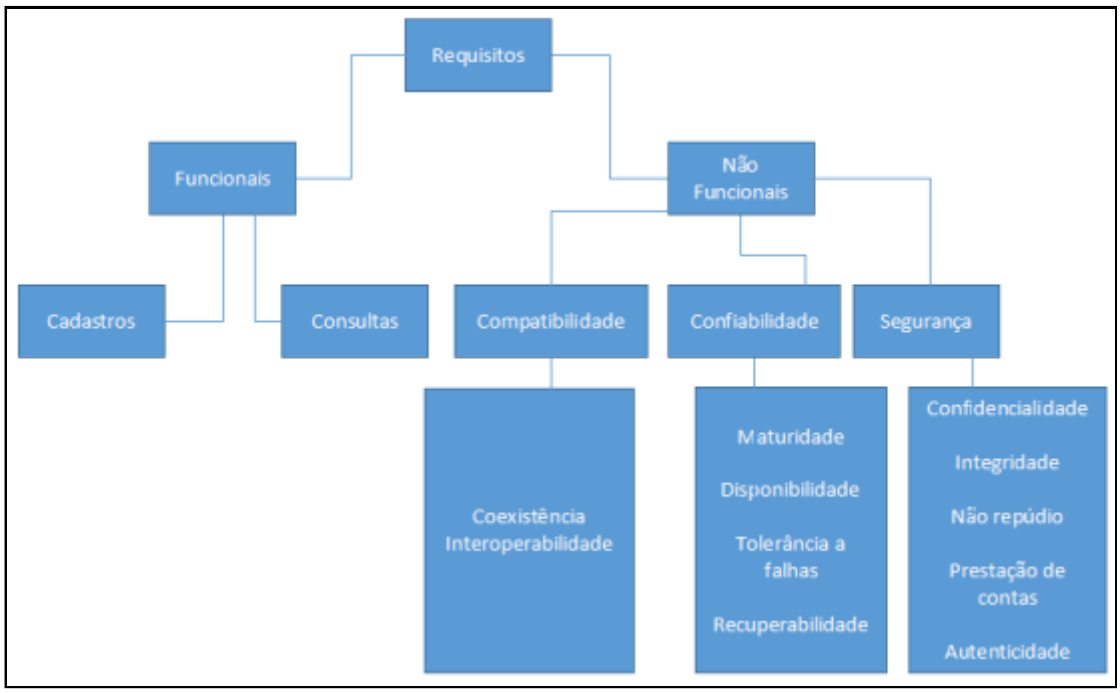
Para que os requisitos funcionais sejam atendidos de maneira adequada, devem ser considerados alguns requisitos não funcionais que fazem com que a regra de negócio do aplicativo seja cumprida com sucesso, dentre eles:

- Ser responsivo, adaptando-se assim em diferentes resoluções e tamanho de telas de variados de *smartphones* e *tablets*.
- Confiabilidade, o *software* deve ser capaz de realizar as suas funções requeridas sob condições específicas.
- Disponibilidade, o aplicativo deve estar disponível sempre que necessária a sua utilização.

- Manutenibilidade do sistema, ele deve ser fácil de efetuar manutenções com custos baixos, para que o custo-benefício seja melhor que sem a sua utilização.

A Figura 3 apresenta alguns exemplos de requisitos funcionais e não funcionais, separados por categorias.

Figura 3 - Comparação entre requisitos funcionais e não funcionais



Fonte: Autoria própria (2021)

2.2.2 Metodologias de desenvolvimento

No outono de 2000, em um encontro entre Robert C. Martin e Martin Fowler surgiu da ideia de reunir apoiadores de processos simplificados com o objetivo de lançar um manifesto unificado. Assim foram enviadas cartas-convite e em caso de concordância a reunião seria realizada em uma estação de esqui de Snowbird, Salt Lake City (MARTIN, 2020).

Com dezessete pessoas reunidas, adeptos de variadas metodologias, foi dado início a reunião, com a missão de criar um manifesto que descrevesse os pontos em comum de todos os processos leves de desenvolvimento. Foi então concebido o que ficou conhecido como Manifesto Ágil, o qual contem quatro valores (CUNNINGHAM, 2001):

Valores do método ágil:

1. Pessoas e interações, em detrimento de processos e ferramentas.
2. Validação do *software*, em vez de uma documentação exaustiva e longa.
3. Colaboração com o cliente, em detrimento da negociação de contrato.
4. Resposta à mudança, em vez de seguir um plano cegamente.

Também no Manifesto Ágil foram definidos 12 princípios fundamentais, listados a seguir (CUNNINGHAM, 2001):

1. Nossa maior prioridade é satisfazer o cliente, por meio da entrega adiantada e contínua de software agregado;
2. Aceitar mudanças de requisitos, mesmo ao término do desenvolvimento, processos ágeis permitem a adequação a mudanças, para que o cliente possua vantagens em todas as etapas do desenvolvimento;
3. Entregar muitos softwares funcionando com frequência, na escala de semanas até meses, dando preferência aos períodos mais curtos;
4. Os desenvolvedores e os analistas devem trabalhar em conjunto e diariamente, durante todo o curso do projeto;
5. Trabalhar ao redor de uma equipe motivada. Dando a ela o ambiente e suporte necessário, e confiar que farão seu trabalho da melhor maneira possível.
6. A melhor e mais eficaz maneira de expor ideias e informações para um time de desenvolvimento são reuniões presenciais.
7. O maior progresso acontece quando se entrega um software totalmente funcional.
8. Processos ágeis promovem um ambiente sustentável. Todas as equipes que envolvem o projeto, desde desenvolvedores, até usuários, devem caminhar no mesmo rumo e na mesma velocidade.
9. É necessário ter uma atenção síncrona para permitir à técnica perfeita e o bom design, para aumentar a agilidade.
10. Às vezes ser simples é necessário para maximizar a quantidade de trabalho que não precisa ser feito.
11. As melhores arquiteturas, códigos, requisitos e designs emergem de times que se organizam por conta própria.
12. É necessário que a equipe reflita sobre melhorias contínuas, para se ajustar e otimizar seu processo de desenvolvimento.

2.2.2.1 Scrum

Scrum é um framework de gestão e planejamento de projetos de software, geralmente utilizada para projetos complexos, porém com uma equipe pequena, a metodologia é classificada como ágil. Foi criada pelos desenvolvedores Ken Schwaber e Jeff Sutherland, e divide o desenvolvimento em várias pequenas tarefas chamadas de *sprints*. Um *Sprint* possui um conjunto de funcionalidades em que devem ser executadas dentro de uma determinada interação (DE PAULA, 2016).

Segundo De Paula (2016), o conjunto de vários *sprints* e que formam uma parte do produto final, passível de uma entrega para o *product owner* é chamado de *backlog* de produto. Antes de finalizar um *backlog* de produto é necessário concluir os *sprints backlog* que são as pequenas funcionalidades dentro de um *backlog* de produto.

Para o controle de uma equipe Scrum, é necessário um *product owner*, este faz parte da gerência do projeto, é ele que toma decisões e, define prioridades dentro da equipe, ele deve estar no centro do projeto e possuir disponibilidade para responder a dúvidas da equipe (DE PAULA, 2016).

Antes de iniciar cada *Sprint*, é feita uma reunião chamada de *Sprint Planning Meeting*, na qual o *product owner*, define quais são as atividades prioritárias para o próximo *Sprint*, e a equipe de desenvolvimento indica se é capaz de cumprir estas missões. Durante a ocorrência do *Sprint*, são realizadas reuniões diárias, chamadas de *Daily Scrum Meeting*, em que cada membro do time, deve descrever o que fez no dia anterior, quais foram as dificuldades e define a agenda para as próximas etapas a seguir, ao final de cada *sprint*, é realizada a *Sprint Review Meeting*, reunião de fechamento do *sprint*, em que a equipe apresenta ao *product owner* os resultados do *sprint* que se finalizou De Paula (2016).

Uma equipe *scrum* deve possuir como peça importante um *scrum master*. Para Rehkopf (2017) o *scrum master* faz um papel de facilitador sobre as técnicas do *scrum* para com a equipe. É ele que implementa e fiscaliza os métodos utilizados.

Jeff Sutherland estabelece que o Scrum valoriza a qualidade da equipe e das ferramentas utilizadas no projeto, com menos gente, é possível fazer mais, desde que o time esteja preparado para o projeto (LEYA, 2016).

O Scrum possibilita que o produto seja entregue em etapas, ou seja, a equipe não precisa esperar até que o produto final esteja finalizado para entregar algo ao *Product Owner*, as entregas são feitas ao final de cada Sprint, e as correções e modificações são feitas no decorrer do desenvolvimento.

2.3 Micro serviços

É uma abordagem para desenvolver uma única aplicação como um conjunto de pequenos serviços, cada um executando em seu próprio processo e se comunicando com mecanismos leves, geralmente uma API de recurso HTTP. Estes serviços são construídos sobre funcionalidades de negócios e implantáveis de forma independente por máquinas de implantação totalmente automáticas (LEWIS, 2014).

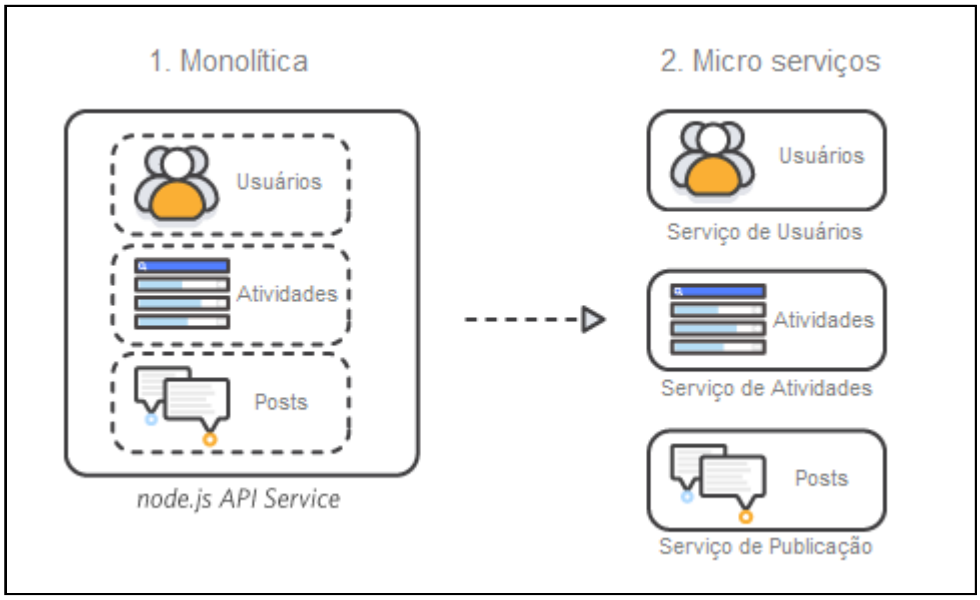
Ainda Lewis (2014) afirma que existem um mínimo de gerenciamento centralizado desses serviços, que podem ser escritos em diferentes linguagens de programação e usar diferentes tecnologias de armazenamento de dados.

Nas arquiteturas monolíticas todos os processos são acoplados e executam em um único serviço, com isto é necessário aumentar toda a arquitetura, caso alguma funcionalidade do aplicativo alcance uma determinada demanda (Amazon, 2019).

Segundo a Amazon (2019), as arquiteturas monolíticas aumentam o risco de disponibilidade de aplicativos, pois muitos processos dependentes e altamente acoplados aumentam o impacto da falha de um único processo.

Em contrapartida, na arquitetura de micro serviços, um aplicativo é construído como componente independente e executa uma funcionalidade em forma de serviço, este fica disponível para ser utilizado por outros aplicativos por meio de APIs. Por serem executados independentemente, cada serviço pode ser atualizado, implantado e escalado individualmente de acordo com a demanda especifica de algum aplicativo. A Figura 4 ilustra um aplicativo monolítico versus micro serviços.

Figura 4 - Monolítico versus micro serviços



Fonte: Adaptado de (AMAZON, 2019)

Os principais pontos positivos dos micro serviços são (AMAZON, 2019):

- Agilidade: permite equipes pequenas e independentes e que gerenciam seus próprios serviços, possuindo autonomia para trabalhar, obtendo assim resultados significativos mais rapidamente;
- Escalabilidade: Como cada serviço pode ser escalado individualmente, de acordo com a demanda, facilita a medição e dimensionamento apropriado dos recursos, mantendo assim a disponibilidade do serviço;
- Fácil Implantação: Por meio da integração, entrega continua e baixo custo da falha, facilita a atualização e entrega de novos recursos;
- Liberdade tecnológica: Cada equipe pode definir quais as melhores ferramentas para o desenvolvimento de seu trabalho;
- Código reutilizável: Com a divisão em pequenos módulos, funções de um componente podem ser reaproveitadas em outro componente;
- Resiliência: A falha de um único componente não causa a falha de todo o aplicativo;

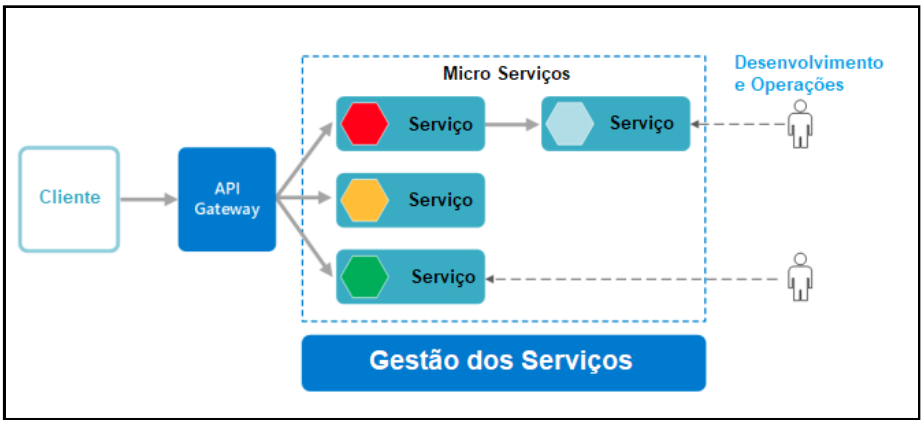
Os principais desafios desta arquitetura, além da cultura e do processo, a complexidade e a eficiência são dois grandes desafios de uma arquitetura baseada em micro serviços (REDHAT, 2017) :

- Dificuldade no gerenciamento do elevado número de entidades separadas;

- Controle de versão mais complexo devido necessidade de se manter a compatibilidade entre os serviços;
- O tráfego de rede entre micro serviços aumenta;
- Aumento da latência de rede;
- Sem ferramentas para ajudar a visualizar as dependências.
- É difícil ver o sistema como um todo;

Uma arquitetura de micro serviços consiste de uma coleção de serviços, pequenos e autônomos. Cada serviço é autocontido e deve implementar uma única funcionalidade dentro de um contexto limitado. Um contexto limitado é uma divisão natural dentro de um serviço e fornece um limite explícito dentro do qual um modelo de domínio existe (AZURE, 2021). A imagem da Figura 5 exemplifica uma arquitetura de micro serviços.

Figura 5 - Arquitetura de micro serviços



Fonte: Adaptado de (AZURE, 2021)

No desenvolvimento deste trabalho, temos o aplicativo de comandas e o sistema PDV como clientes, utilizando a Figura 5, situam se em Cliente.

Em se tratando de API Gateway, este é um ponto de entrada único para todos os clientes, trabalha com requisições de duas formas, um ponto de entrada único ou podendo ser especifico para cada cliente, roteando requisições para o serviço apropriado (REDHAT, 2019).

API Gateway trabalha com todas as tarefas envolvidas em receber e processar centenas de milhares de chamadas simultâneas, incluindo gerenciamento de tráfego, suporte a *Cross-Origin Resource Sharing* (CORS), um mecanismo baseado em cabeçalho HTTP que permite que um servidor indique qualquer origem

(domínio, *schema* ou porta) diferente de sua própria em que um *browser* (navegador) pode permitir carregar recursos (MOZILLA, 2021).

A maioria das APIs corporativas são disponibilizadas por meio de API Gateways, estes trabalham com tarefas comuns a um sistema de serviços de API, como autenticação de usuários, limite de requisições e estatísticas (REDHAT, 2019).

Na parte de micro serviços, é comum a utilização de um serviço de registro que mantém os dados dos serviços em execução, com suas instâncias e localização. Instâncias de serviços são registradas no servidor de registro em sua inicialização e excluídas em seu desligamento. Cliente do serviço e/ou roteadores consultam o serviço do servidor de registro para encontrar as instâncias disponíveis de um serviço. Um serviço de registro pode invocar a API de verificação de integridade de uma instância serviço para verificar se este está apto a trabalhar com requisições (REDHAT, 2019).

2.4 Banco de dados

Um banco de dados (BD) é um conjunto de dados e/ou informações, geralmente de um domínio específico, armazenados em um sistema de computador. Segundo Korth (1999), banco de dados “é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico”. Como exemplo de banco de dados, temos uma lista de convidados para um aniversário, uma agenda telefônica, entre outros.

Um banco de dados é composto por tabelas, e estas tabelas possuem linhas (tuplas) em que se encontram os registros, ou dados, e colunas (atributos), em que são determinadas as características de cada um dos dados.

Para um melhor controle destes dados, se faz necessário de um sistema gerenciador de banco de dados (SGBD). Como exemplo de SGBD, tem-se MySQL, SQL Server, Oracle, SQLite, PostgreSQL, entre outros.

Dentro de um *software* a função do banco de dados é de isolar o usuário dos dados e informações que são armazenados, de forma que este consiga efetuar pesquisas em tabelas, por intermédio do *software* que faz o acesso ao BD.

Estas consultas são feitas por meio de *querys* (consultas), que são as solicitações feitas ao BD para retornar os dados, para executar estas consultas é

utilizada uma linguagem SQL (*Structured Query Language*), Linguagem de Consulta Estrutura.

O CRUD, originário do inglês (*Create*, *Read*, *Update*, *Delete*) criação, consulta, atualização e destruição de dados respectivamente, são as quatro principais instruções a serem utilizadas em um banco de dados relacional.

Primeiramente temos a instrução *Create* que é utilizada para a criação de uma tabela, para isso é necessário informar o nome da tabela, e os campos com seus respectivos tipos de dados. Toda tabela deve conter um campo *primary key* (chave primária), este campo é utilizado para ser a identidade da tabela, ou seja, para que a tabela não contenha dados duplicados. No exemplo apresentado pela Figura 6, foi criada uma tabela de clientes com os campos “CPFCLI” que é o campo chave em que é informado o CPF do cliente e outro campo “NOMECLI” em que se deve informar o nome do cliente.

Figura 6 - SQL para criação de tabela

```
2
3   CREATE TABLE `cliente` (
4      `CPFCLI` CHAR(11) NOT NULL DEFAULT '',
5      `NOMECLI` CHAR(120) NULL DEFAULT '',
6      PRIMARY KEY (`CPFCLI`)
7   )
8   COLLATE='utf8_general_ci'
9   ENGINE=MyISAM
10  ROW_FORMAT=DYNAMIC
11  AVG_ROW_LENGTH=299
12  MAX_ROWS=9999999
13  ;
14
```

Fonte: Autoria própria (2021)

A instrução *Insert* é utilizada para a inserção de dados e informações nas tabelas, na Figura 7 tem-se a instrução que insere um cliente no BD:

Figura 7 - SQL para inserção de dados

```
INSERT INTO cliente
   (CPFCLI, NOMECLI)
   VALUES ('123.456.567-89', 'Pedro Paulo');
```

Fonte: Autoria própria (2021)

A instrução *Select* é utilizada para consulta dos dados inseridos nas tabelas, geralmente aparece acompanhada de filtros para que os dados selecionados sejam abstraídos de forma a melhorar a performance da consulta, na Figura 8 é exemplificado uma consulta na tabela cliente:

Figura 8 - SQL para consulta de dados

```
SELECT cliente.*
FROM cliente;
```

Fonte: Autoria própria (2021)

A instrução *Update* é utilizada para atualização de dados já cadastrados, utilizada quando, por exemplo, determinado cliente troca de endereço ou de cidade e o cadastro deve ser atualizado. A Figura 9 apresenta um exemplo de atualização do nome do cliente utilizando o campo “CPFCLI” como identificador para a atualização do cliente:

Figura 9 - SQL para alteração de dados

```
UPDATE cliente
   SET
      NOMECLI = 'Pedro Paulo Pernambuco'
   WHERE CPFCLI = '123.456.567-89';
```

Fonte: Autoria própria (2021)

A instrução *Delete* é utilizada para excluir algum registro já inserido no banco de dados, ou até mesmo excluir todos os dados de uma tabela já criada

anteriormente no BD. A Figura 10 é um exemplo de utilização desta instrução, utilizando o identificador “CPFCLI” para fazer a deleção de um cliente:

Figura 10 - SQL para remoção de dados

```
1
2
3  DELETE FROM cliente WHERE CPFCLI = '123.456.567-89';
4
5
```

Fonte: Autoria própria (2021)

A Figura 11 apresenta um exemplo de instrução *Drop*, em que a estrutura da tabela é excluída totalmente, juntamente com os dados:

Figura 11 - SQL para remoção de tabela

```
2
3  DROP TABLE cliente;
4
```

Fonte: Autoria própria (2021)

2.4.1 MySQL

O MySQL é um sistema de banco de dados relacional utilizado por diversas empresas desenvolvedoras de *software*, pelo fato de ser de código aberto e de fácil manipulação.

O serviço utiliza a linguagem SQL (Structure Query Language – Linguagem de Consulta Estruturada), que é a linguagem mais comumente utilizada por desenvolvedores. É desenvolvido em C e C++, o que possibilita ser multiplataforma, ou seja, é compatível com todo e qualquer sistema operacional, ele utilizada a licença GNU/GLP (General Public License), que permite alteração de seu código fonte.

2.4.2 SQLite

Construído para ser um banco de dados simples, leve e de fácil instalação e manipulação o SQLite é uma biblioteca desenvolvida em C, que implementa um banco de dados independente da aplicação.

Indicado para utilização em pequenas aplicações, pois não necessita de um serviço separado da aplicação, sem a necessidade de se utilizar de um sistema cliente/servidor.

Segundo a SQLite[2], devido ao seu pequeno porte e fácil configuração, por ser de código aberto, não possuir uma restrição de licença, ou seja, o código fonte do banco de dados pode ser copiado, e modificado sem a necessidade de permissões ou de licenças pagas ao desenvolvedor.

Por ser leve e de fácil instalação, é muito utilizado em aplicações desenvolvidas para celulares, *smartphones* e *tablets*, sua forma de leitura e gravação é diretamente em arquivos de disco. Uma grande vantagem é que também é multiplataforma, podendo executar em qualquer sistema operacional, e ser integrado em qualquer linguagem de programação.

2.5 *Frameworks*

Nesta seção serão abordados os *frameworks* utilizados neste trabalho. A Subseção 2.4.1 apresenta o *framework* Spring e a Subseção 2.4.2 descreve o *framework* Ionic.

2.5.1 Spring

O termo “Spring” tem diferentes significados em diferentes contextos. Pode ser usado para se referir ao próprio projeto Spring Framework, que é o seu ponto de início. Quando pessoas dizem “Spring” significa a família toda de projetos. (Spring, 2019).

Spring [3] foi criado em 2003 como resposta a complexidade das especificações Java Enterprise Edition (J2EE). Segundo Spring (2019) enquanto alguns consideram Java Enterprise Edition e Spring como sendo concorrentes, Spring é complementar ao Java EE. O modelo de programação Spring não segue a especificação Java EE, mas integra especificações individuais cuidadosamente selecionadas da plataforma EE Umbrella como por exemplo: Servlet API,

[2] Site do SQLite: https://sqlite.org/index.html
[3] Site do Spring: https://spring.io

WebSocket API, Concurrency Utilities, JSON Binding API, Bean Validation, JPA, JMS e também JTA/JCA. Também suporta especificações Dependency Injection e Common Annotations, caso os desenvolvedores escolham estas em detrimento a mecanismos específicos do Spring.

O Spring Framework pode ser dividido em três blocos, Spring Boot, Spring Cloud e Spring Cloud Data Flow, disponibilizando um conjunto de tecnologias para cada bloco.

O Spring Boot foi projetado para que comece a trabalhar o mais rápido possível com um mínimo de configuração previa, disponibilizando as bibliotecas básicas para iniciar o projeto (SPRING BOOT, 2019).

Ainda segundo Spring Boot (2019) os principais objetivos são:

- Proporcionar uma experiência introdutória rápida e acessível para todo desenvolvimento Spring;
- Ser opinativo mas sair do caminho rapidamente com os requisitos divergindo dos padrões;
- Proporcionar uma variedade de recursos comumente utilizados para uma grande gama de projetos, por exemplo segurança, métricas e configuração;
- Não gerar código e não requerer configuração XML;

O Spring Cloud foi criado para simplificar sistemas distribuídos, oferece um modelo de programação simples e acessível para os padrões mais comuns de sistemas distribuídos, é construído sobre o Spring Boot (SPRING BOOT, 2019).

Disponibiliza funcionalidades como: configuração versionada distribuída, descoberta e registro de serviços, roteamento, chamadas serviço para serviço, balanceamento de carga, quebra de circuito, menssageria distribuída, servidor de configuração, entre outras.

Os componentes principais do Spring Cloud são baseados no Netflix Open Source Software[4], um conjunto de bibliotecas e frameworks desenvolvidos pela Netflix com o objetivo de resolver problemas de sistemas distribuídos e em escala, como, por exemplo, balanceamento de carga, tolerância a falhas e descoberta de serviço.

[4] Netflix OSS: https://netflix.github.io

Este conjunto de bibliotecas e frameworks disponibilizados em código aberto pela Netflix foram adaptados e desenvolvidos pela comunidade e incorporados ao Spring com tamanho sucesso que a solução passou a ser utilizada internamente na própria Netflix.

2.5.2 Ionic

Ionic é um *framework* de código aberto que utiliza uma licença MIT, você pode utilizar o Ionic nos seus próprios projetos comerciais ou pessoais gratuitamente (IONIC, 2015). Criado no ano de 2012 com o objetivo de facilitar o desenvolvimento de aplicativos móveis e desktop, utilizando tecnologias *web*, como HTML, CSS e JavaScript, é uma poderosa ferramenta que auxilia desenvolvedores com experiência em aplicações *web* a também conseguir desenvolver aplicativos móveis, pois utiliza uma plataforma híbrida.

Segundo a Ionic (2015), hoje o *framework* é a plataforma líder mundial na construção de aplicativos móveis multiplataforma, utilizando JavaScript, com mais de 5 milhões de desenvolvedores espalhados em mais de 200 países. Números que deixam a plataforma com mais de 20% de todos os aplicativos disponíveis nas principais lojas de aplicativos.

Segundo Bradley (2015) em Ionicframework[5], o ionic foi criado para não apenas funcionar, mas para funcionar rapidamente e com qualidade em todos os dispositivos móveis mais recentes. Mesmo se tratando de um *framework* multiplataforma o Ionic permite acessos a recursos nativos com detalhes de interface de usuário, por utilizar JavaScript, o Ionic permite a utilização de componentes de tela semelhantes aos de aplicativos nativos, não reconhecidos pelo usuário, mas apenas pelo desenvolvedor, eu outras palavras, para o usuário, seria como se o aplicativo tivesse sido desenvolvido utilizando de uma plataforma nativa de desenvolvimento de aplicativos, como exemplo o AndroidStudio da Google.

Em IonicFramework (2015), diz que o Ionic permite o acesso a recursos nativos do dispositivo com poucas linhas de código em JavaScript, utilizando bibliotecas prontas com funções pré-programadas que acessam, diversas funções dos dispositivos, como câmera, *bluetooth*, lista de contatos, entre outros.

---

[5] Ionicframework: https://ionicframework.com

O Ionic disponibiliza de uma equipe de suporte sempre disposta a auxiliar o desenvolvedor a esclarecer dúvidas sobre suas ferramentas. Em seu site a Ionic (2015) diz sobre seu framework “Não trabalhamos mais, trabalhamos de forma mais inteligente, pois nosso *software* funciona para nós e não o contrário”.

Para a companhia Ionic, o objetivo principal da plataforma é tornar o desenvolvimento de aplicativos mais acessível a todos os desenvolvedores e empresas de desenvolvimento, fornecendo ferramentas e serviços de código aberto.

2.6 Sistemas similares

Foram feitas pesquisas na loja virtual de aplicativos da Google, a Google Play, em busca de aplicativos com funcionalidades semelhantes as desenvolvidas neste projeto, por se tratar de um aplicativo comercial, a grande maioria não disponibiliza uma versão de testes, foram instalados e analisados dois aplicativos com algumas particularidades semelhantes às do projeto em questão, porém com características diferentes as tecnologias utilizadas para desenvolvimento.

Também em ambos os casos a arquitetura é totalmente diferente, nos dois aplicativos o banco de dados é SQLite, não permitindo assim uma gestão do negócio, visto que o banco de dados não é distribuído para compartilhamento em outros dispositivos.

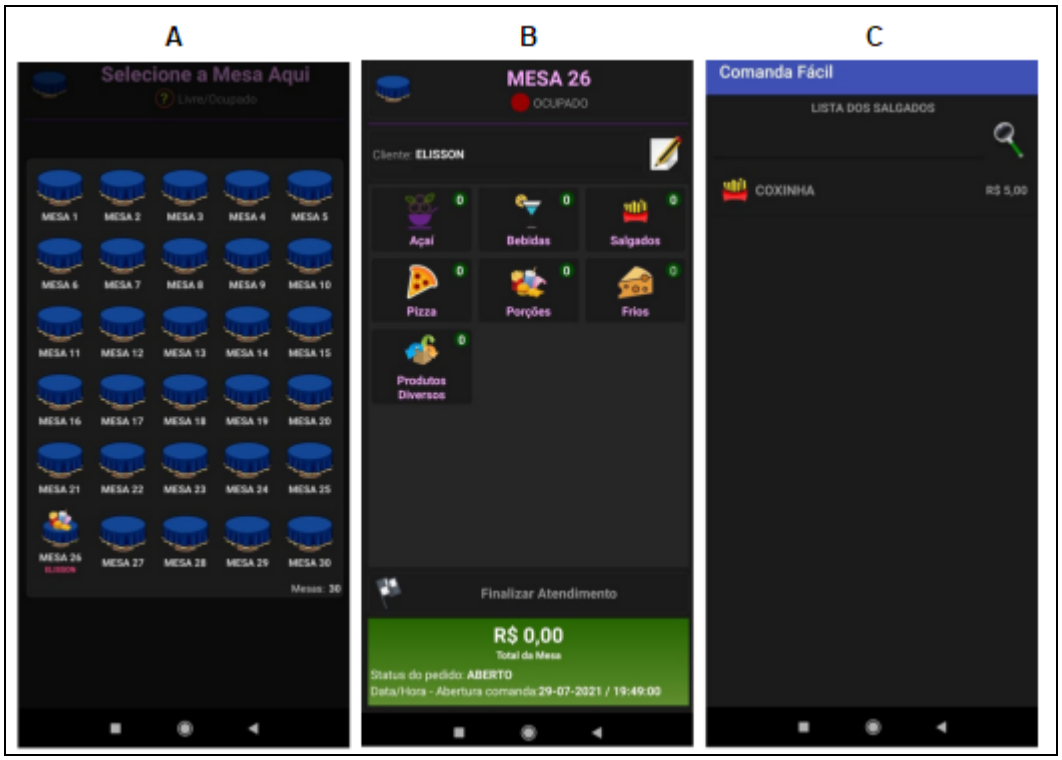
Outro ponto avaliado é que os dois aplicativos não permitem integração com nenhum outro tipo de software, fator este determinante para reforçar o desenvolvimento do aplicativo que disponibilize os dados para integração.

O primeiro aplicativo é o Comanda Fácil, ele não possui uma tela de *login* na versão demonstrativa, fator este que pode deixar o aplicativo vulnerável, pois em sua configuração necessita que digite o IP do servidor de banco de dados para integrar os dados.

Outro ponto avaliado é que o Comanda Fácil finaliza o pedido, e manda um ticket para a impressora, porém não possui integração com qualquer banco de dados para emissão de Nota Fiscal de Consumidor. O aplicativo permite apenas 30 (trinta) mesas ou comandas, como pode ser visualizado na Figura 12-A, não podendo adicionar mais mesas que este número. Na Figura 12-B temos a imagem

de edição de uma mesa, e na imagem 12-C temos a adição de um item em uma mesa.

Figura 12 - Aplicativo Comanda Fácil



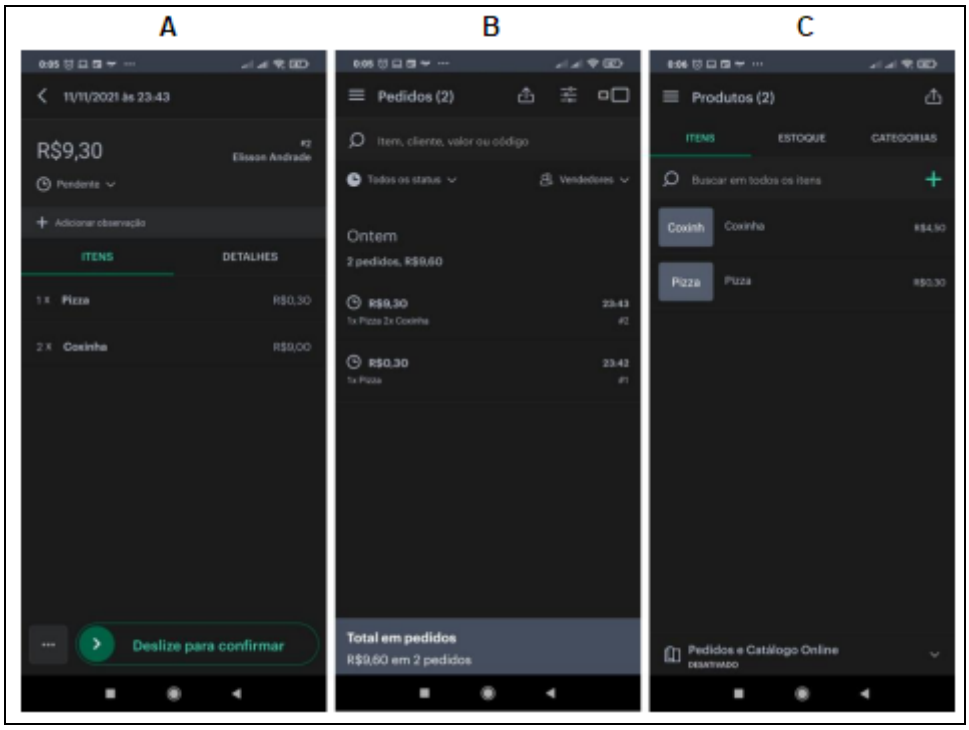
Fonte: Autoria própria (2021)

Baseados nestes pontos foi reforçada a ideia do desenvolvimento de um aplicativo que faça o pedido e disponibilize uma API de integração com o PDV, assim seriam otimizadas as tarefas tanto do garçom, quanto do operador de caixa no PDV, pois este não terá que digitar todos os itens manualmente, e sim apenas o número da mesa ou comanda, e assim poderá apenas finalizar a venda.

O segundo aplicativo analisado é o Kyte, é mais simples e com uma interface menos interativa se comparado ao anterior, porém com as mesmas funcionalidades, conforme é possível identificar na Figura 13-A em que mostra o fechamento de um pedido.

O gerenciamento de pedidos é mais simples que no anterior, como pode ser visto na figura 13-B, o aplicativo permite o cadastro de produtos e de clientes diretamente pelo celular. Conforme pode ser visto na figura 13-C.

Figura 13 - Aplicativo Kyte



Fonte: Autoria própria (2021)

O Kyte não permite integração com software de terceiros, porém nas versões para assinantes que possui planos a partir de R$ 19,00 (dezenove reais) mensais, ele disponibiliza módulos de catálogo online, controle financeiro e controle de permissões de usuários.

Após a análise dos dois aplicativos, foi possível extrais ideias, e incorporar melhorias nos requisitos que já vinham sendo propostos.

3 DESENVOLVIMENTO

Seguindo os princípios do Scrum o desenvolvimento deste projeto foi dividido em objetivos traduzidos em funcionalidades ou requisitos que software deveria atender, gerando o *Product Backlog*, este foi priorizado pelo *Product Owner* de acordo com seus critérios. Desta lista de funcionalidades, foi extraído um conjunto de pequenas tarefas, que gerou os *sprints*. A partir daí, foram definidas as ferramentas a se utilizar no desenvolvimento.

3.1 Elicitação de requisitos

Para a aplicação do projeto, foi escolhido um restaurante/lanchonete localizado na cidade de Irati-PR que atualmente ainda registra em papel os pedidos de seus clientes. Primeiramente foi entrado em contato com a proprietária para efetuar algumas perguntas referente ao processo diário atual.

Utilizando a técnica de elicitação de requisitos chamada de questionário, em que foi passado para a proprietária um questionário referente a rotina de atendimento do estabelecimento.

Após análise do questionário, foi utilizada a técnica de observação, em que foi definido um dia para uma visita ao cliente, para vivenciar o processo de trabalho do estabelecimento em um horário em que este tem o maior fluxo de atendimentos.

Com o levantamento de requisitos gerados das duas técnicas chegou-se aos seguintes resultados:

Diariamente um garçom realiza em média 30 pedidos de clientes, para atender um pedido um garçom gasta em média 4 minutos e 30 segundos, sendo o maior gargalo de atendimento entre as 11:00 e 14:00 horas.

Sobre as maiores dificuldades encontradas no processo, está o desperdício de papel que ocorre quando o garçom escreve errado, a perda de canetas, a dificuldade para entendimento da escrita pois em muitas vezes o garçom não tem uma forma adequada para apoiar o bloco para escrever, e por último a perda ou extravio da comanda pelo cliente. Para manter o atual sistema de comandas o custo mensal é em média de R$ 65,00 incluindo gastos com blocos de papel e canetas.

Por fim foi realizada uma reunião junto ao cliente, para definir o que ele espera do aplicativo, e seu principal requisito é de que o tempo de lançamento do pedido não ultrapasse mais que dois minutos, salvo indefinições do cliente, e que o custo não ultrapasse, os R$ 65,00 de custo mensal para manter o sistema atual.

3.2 Implementação do Sistema

A seção a seguir descreve as etapas do desenvolvimento, utilizando as tecnologias citadas nos capítulos anteriores. Para uma melhor organização o desenvolvimento foi dividido em etapas, neste processo é possível citar os *backlogs* de produtos e *sprints* para cada um, conceitos oriundos do SCRUM.

Antes do início do desenvolvimento foram elencados os *sprints* e relacionados em uma tabela para definir as prioridades e facilitar a tomada de decisão no *sprint* de desenvolvimento.

3.2.1 Priorização dos requisitos

A priorização dos requisitos foi definida com base no tempo disponível para desenvolvimento e utilizando as técnicas de metodologia ágil do Scrum, na qual devem se dividir os *sprints* em pequenas tarefas, para que seja possível fazer várias entregas durante todo o processo.

Como requisito principal, ficou definido a criação do servidor para a API, e a criação do banco de dados dentro do servidor.

A segunda prioridade é a construção dos micro serviços, dando prioridade para o micro serviço de registro, pois com ele, já é possível iniciar o desenvolvimento do aplicativo.

Como terceira prioridade, foi a construção das demais telas do aplicativo, com os micro serviços já em execução, é possível visualizar os dados disponibilizados pela API.

Quadro 1 - Product Backlog 1 do projeto

Product Backlog				
Construção dos micro serviços				
ID Tarefa	Tarefa	Resultado Esperado	Estimativa (semanas)	Sprint
1	Criação do Banco de Dados em MySQL	Criar o banco de dados e estrutura de tabelas.	1	1
2	Criação do Servidor para disponibilização da API	Criar o servidor em nuvem, em que ficará o banco de dados e a API para comunicação Aplicativo/API	1	1
3	Micro serviço de Login de usuário (Garçom)	Desenvolvimento do micro serviço de consulta e autenticação de usuários (Garçom).	1	2
4	Micro serviço de busca de produtos	Desenvolvimento do micro serviço de consulta de produtos previamente cadastrados.	1	2
5	Micro serviço de pedidos e itens de pedidos	Desenvolvimento do micro serviço de pedidos, para criação, consulta e finalização do pedido.	1	2

Fonte: Autoria própria (2021)

3.2.2 Sprint 1

Item do *product backlog* 1 e 2. Para a criação do banco de dados foi utilizado o MySQL 5.7 que possui licença GPL, e para manipulação dos dados foi utilizada a ferramenta HeidiSQL, ferramenta de código aberto criada pelo alemão conhecido como Anse. O banco de dados foi criado em um servidor virtual da AWS - Amazon Web Services.

Apenas para o cadastro de usuários, também foi criado um banco de dados em SQLite, com a tabela de usuários com campo de usuário e senha.

Após a criação da estrutura de tabelas foi necessária análise das chaves primárias, para que na criação dos pedidos pelo aplicativo não ocorram inconsistências nos dados armazenados. Após a finalização da estrutura do banco de dados, foram lançados dados para testes das integrações.

3.2.3 Sprint 2

Item do *product backlog* 3,4 e 5. Para o desenvolvimento dos códigos fontes dos serviços, foi utilizada a ferramenta Spring Cloud Gateway para criação da estrutura do serviço e implementação dos códigos-fonte iniciais, a linguagem utilizada foi Java, utilizando a o Visual Studio Code como editor para os códigos.

Após criados os três micro serviços, foi utilizado o Insomnia, uma ferramenta para teste de APIs, disponibilizando as funcionalidades necessárias para a realização dos testes. O formato de dados utilizado foi o JSON, (JavaScript Object Notation - Notação de Objetos JavaScript) que é um formato de texto simples para troca de dados entre aplicações (JSON, 2015). Para Douglas Crockford (2002), programador conhecido por difundir e defender os conceitos de JSON, a ideia principal era uma linguagem fácil de ser escrita e lida por pessoas, e fácil de ser interpretada pela máquina (JSON, 2015).

Quadro 2 - Product Backlog 2 do projeto

Product Backlog				
Construção do Aplicativo Mobile				
ID Tarefa	Tarefa	Resultado Esperado	Estimativa (dias)	Sprint
1	Criação da tela de Login	Criar a tela de acesso ao aplicativo, em que o garçom irá selecionar seu nome e digitar uma senha.	1	3
2	Criação da tela de seleção de mesas/comandas	Desenvolvimento da tela em que serão selecionadas mesas ou comandas existentes, ou adicionadas novas.	1	3
3	Criação da lista de produtos	Desenvolvimento da tela responsável por listar os produtos previamente cadastrados	1	3
4	Criação de tela de lançamento de itens nos pedidos	Desenvolvimento da tela em que serão adicionados os itens a comanda do cliente, também irá mostrar o total do pedido	1	3

Fonte: Autoria própria (2021)

3.2.4 Sprint 3

Backlog Construção do Aplicativo Mobile. Item do *product backlog* 1, 2, 3,4.

Para a criação do aplicativo, foi utilizado o framework de desenvolvimento chamado Ionic, para a geração dos *layouts* de tela e execução dos códigos fontes foi utilizada a linguagem NODE.js, uma linguagem de programação baseada em Java Script que utiliza um interpretador de comandos, sem a necessidade do auxílio de um *browser*.

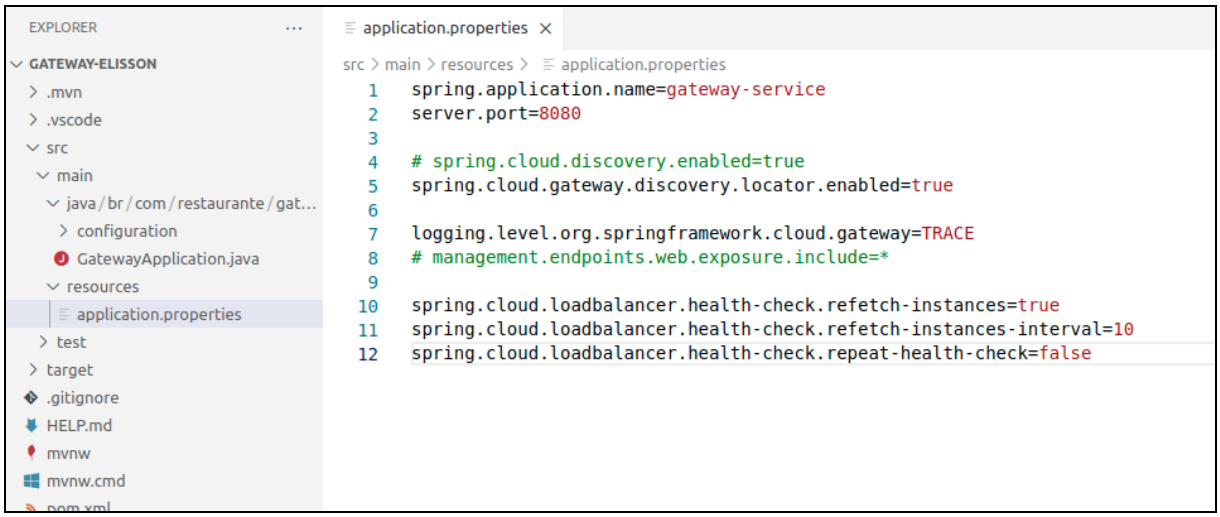
Para edição e manutenção dos códigos fonte, foi utilizada a ferramenta *Visual Studio Code (VSCode)* desenvolvida pela Microsoft, é um editor de códigos fonte de alto nível.

3.2.5 *API Gateway*

Foi utilizado o Spring Cloud Gateway para implementar nosso API Gateway, o Spring Cloud Gateway possui algumas funcionalidades como ser capaz de combinar rotas com qualquer atributo da solicitação, predicados e filtros específicos para rotas, quebra de circuito, integração com Spring Cloud DiscoveryClient, facilidade de escrever predicados e filtros, limitar taxa de requisições, reescrita de caminho (GATEWAY, 2021)

É necessário habilitar e configurar o serviço de API Gateway através do arquivo *application.properties* conforme apresentado na Figura 14, onde configuramos nome do serviço, porta de comunicação, detalhamento de *log* e propriedades da verificação da saúde dos micro serviços.

Figura 14 - Configuração do API gateway



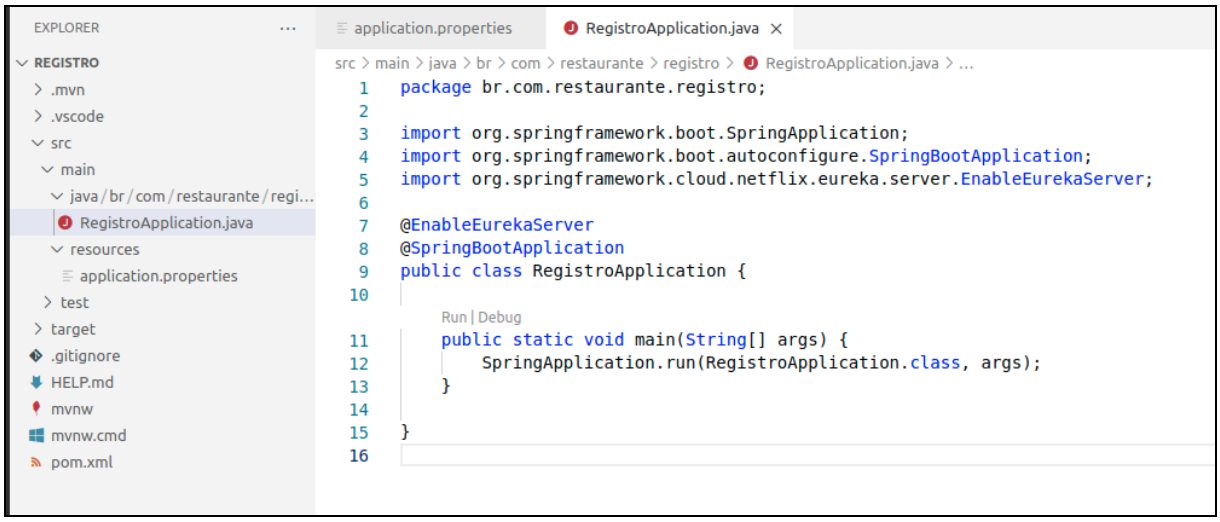
Fonte: Autoria própria (2021)

3.2.6 Servidor de registro

No framework Spring, foi utilizado como serviço de registro e descoberta o Netflix Eureka, um serviço baseado em REST (*Representational State Transfer*) que é utilizado primariamente na nuvem AWS para localizar serviços com o objetivo de balanceamento de carga e tolerância a falhas de servidores de camada intermediária (NETFLIX, 2014).

É necessário habilitar e configurar o servidor de registro, adicionar a anotação @EnableEurekaServer e importar a respectiva classe conforme apresentado na Figura 15 para habilitar o servidor.

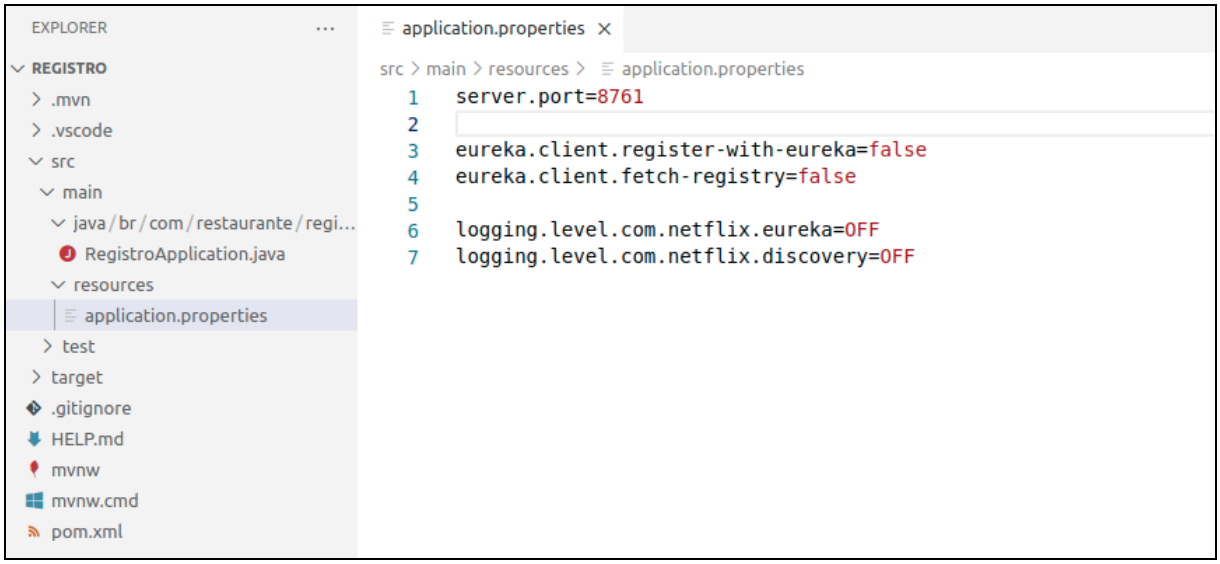
Figura 15 - Classe principal do servidor de registro



Fonte: Autoria própria (2021)

É preciso configurar algumas propriedades do serviço no arquivo *application.properties*, a Figura 16 apresenta as configurações realizadas, configurado a porta em que o servidor irá responder, configurado para o serviço não registrar a si próprio e também desabilitado os *logs* de registro e descoberta de serviços para manter uma saída mais limpa do console do serviço.

Figura 16 - Configuração do servidor de registro



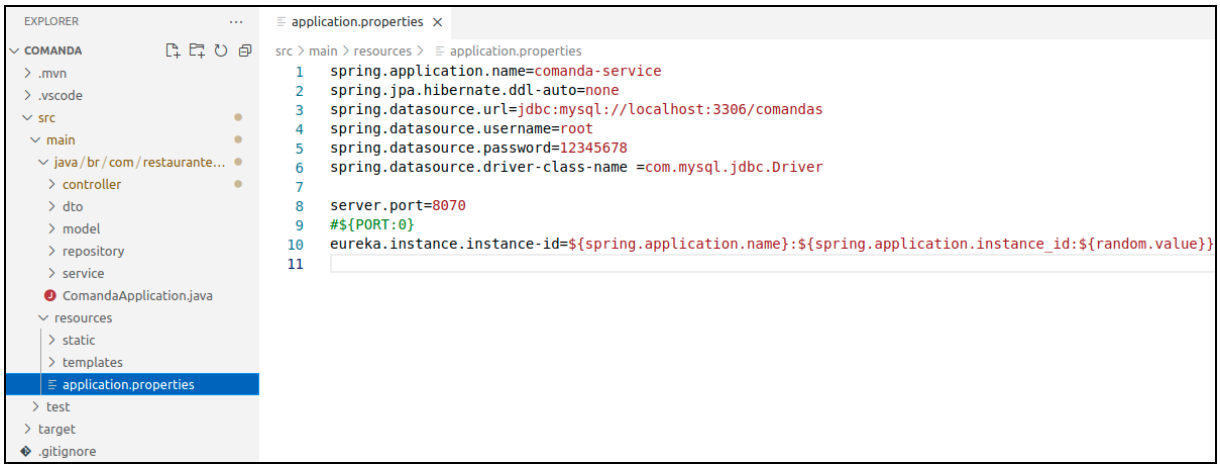
Fonte: Autoria própria (2021)

3.2.7 Serviço de comandas

Este serviço é responsável pelas operações relacionadas aos pedidos realizados pelos clientes aos garçons, também responsável por disponibilizar os dados necessários para que o ERP, por meio do PDV, possa tratar da parte de finalização das comandas, faturamento, entre outras atividades pertinentes a estes.

A Figura 17 mostra as propriedades que foram configuradas para o serviço no arquivo *application.properties*, foi nomeado o serviço, foram informados os dados de conexão com a base de dados e a porta em que o serviço irá responder, neste caso 8070, o identificador da instância do serviço também foi configurado para que seja gerado de maneira aleatória.

Figura 17 - Configuração do serviço de comandas

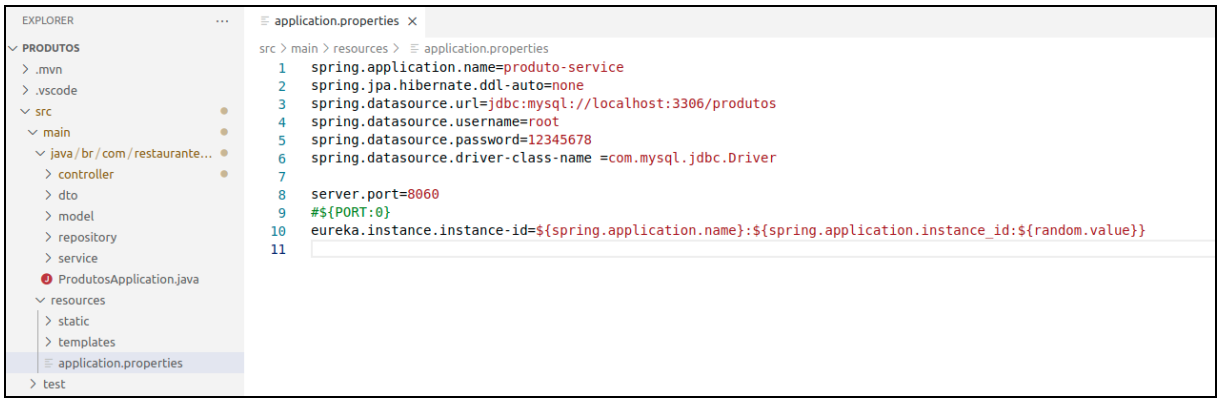


Fonte: Autoria própria (2021)

3.2.8 Serviço de produtos

Este serviço será responsável pelas operações relacionadas aos produtos disponibilizados para venda. Conforme apresentado na Figura 18, se faz necessário configurar algumas propriedades do serviço no arquivo *application.properties*, nomear o serviço, informar os dados de conexão com a base de dados e a porta em que o serviço estará disponível, neste caso 8060, também foi configurado para que o identificador da instância do serviço seja gerado randomicamente.

Figura 18 - Configuração do serviço de produtos

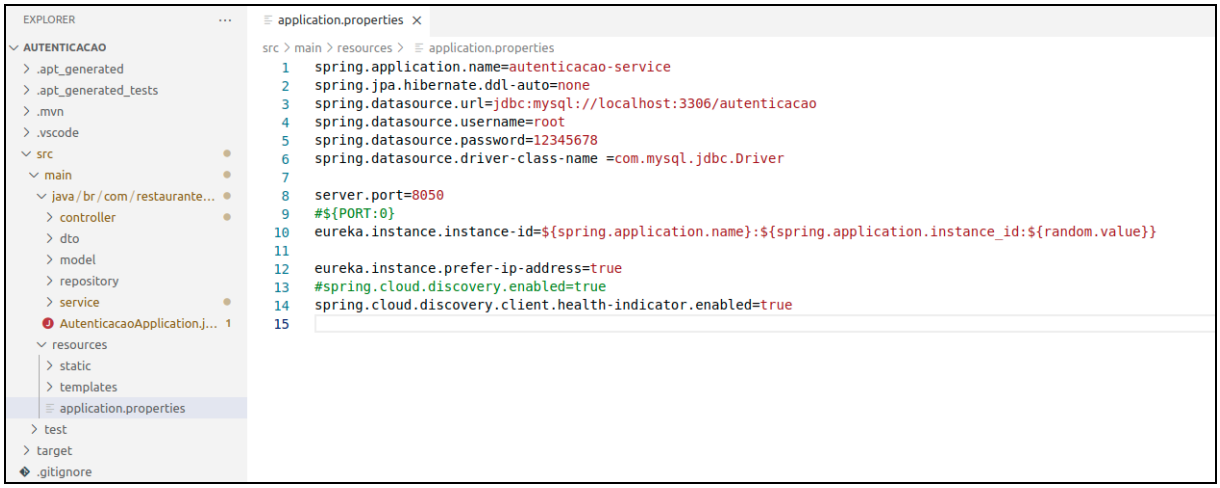


Fonte: Autoria própria (2021)

3.2.9 Serviço de autenticação

Este serviço será responsável pelas operações relacionadas ao controle de acesso dos garçons do estabelecimento. É necessário configurar algumas propriedades do serviço no arquivo *application.properties*, é preciso nomear o serviço, e informar os dados de conexão com a base de dados e a porta em que o serviço irá responder, neste caso 8050 conforme representado na Figura 19.

Figura 19 - Configuração do serviço de autenticação



Fonte: Autoria própria (2021)

4 RESULTADOS

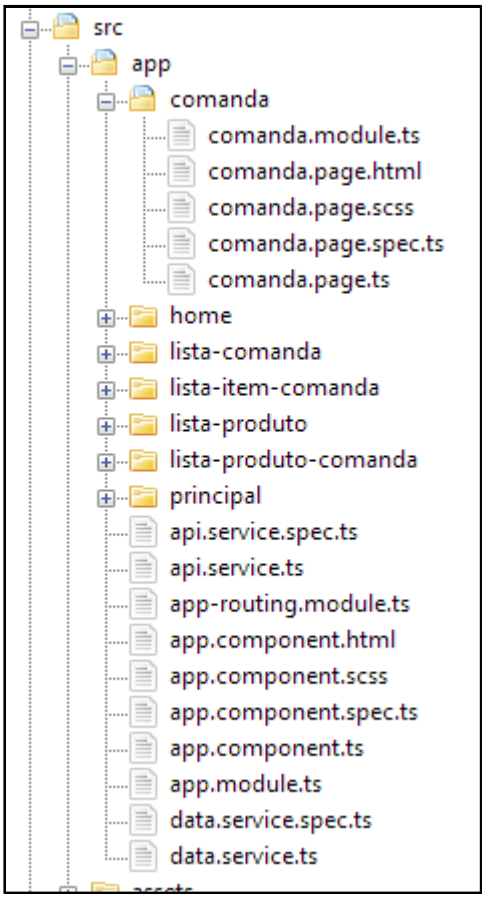
Os resultados obtidos com o desenvolvimento do projeto são abordados na seção 4.1 Aplicativo Android e 4.2 Arquitetura dos micro serviços.

4.1 Aplicativo Android

Para o desenvolvimento do aplicativo Android, foi utilizado o Ionic Framework, com NODE.js. Para manipulação dos códigos fontes, foi utilizado o Visual Studio Code.

A estrutura das pastas ficou dividida utilizando o conceito de Model View Controller – MVC.

Figura 20 - Estrutura de pastas do código do aplicativo



Fonte: Autoria própria (2021)

Para todas as tarefas de gravação dos dados por parte do aplicativo, foi utilizado o comando do HTTP POST, que é utilizado para salvar.

Figura 21 - Comando POST para gravação de Comanda

```
salvarComanda(numero, garcom){
  const httpOptions = {
    headers: new HttpHeaders({
      'Content-Type': 'text/plain',
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Methods':'POST, GET, OPTIONS'
    })
  };


  try {
    this.http.post(`http:localhost:8088/comanda?numero=${numero}&garcomNome=${garcom}`, httpOptions).subscribe(
        sucesso => {
    }
    );
    return true;
  } catch (error) {
    return false;
  }

}
```

Fonte: Autoria própria (2021)

Para a consulta de dados por parte do aplicativo, foi utilizado o comando do HTTP GET, em que são buscados os dados.

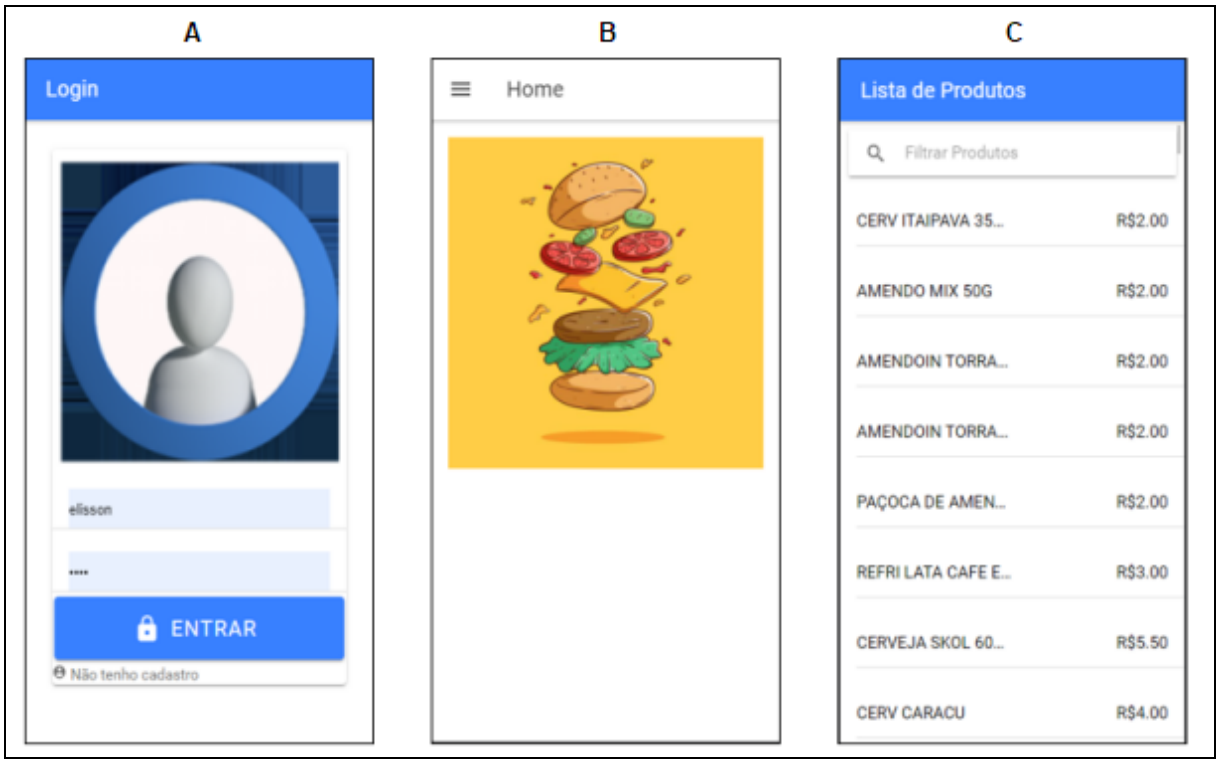
Figura 22 - Comando GET para consulta de um produto

```
    buscarProdutoFiltro(texto){
    let parametros = `${texto}`;
    return this.http.get(`http://localhost:8088/produto?descricao${parametros}`);
  }
```

Fonte: Autoria própria (2021)

As telas do aplicativo seguiram um padrão de *layout* minimalista, com uma baixa densidade de componentes, derivados de um CSS. Para abrir o aplicativo é necessário informar um usuário e senha, conforme a Figura 23-A, logo que iniciado, possui uma tela de menus, com uma imagem, que pode ser customizada com a logomarca da empresa, conforme a Figura 23-B. Para que os itens possam ser vinculados a uma comanda, é necessário visualizar a lista de produtos, que pode ser visualizada na tela da Figura 23-C.

Figura 23 - Telas *login*, *home* e lista produtos

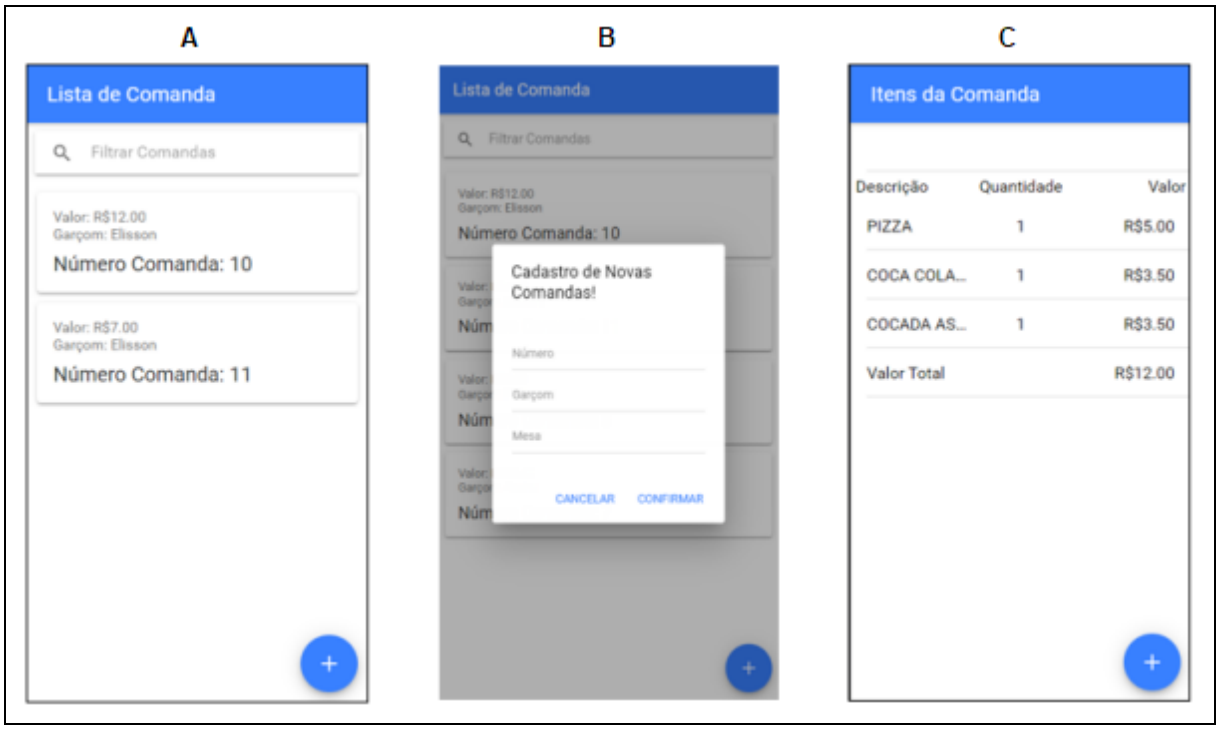


Fonte: Autoria própria (2021)

Ao selecionar a lista de comandas, é possível verificar as comandas com situação em aberto, conforme a Figura 24-A. Após as consultas é possível lançar uma nova comanda, processo este que é mostrado na imagem da Figura 24-B, utilizando um botão flutuante o usuário abre um alerta, neste é possível inserir o número da nova comanda, o nome do garçom e o número da mesa. Ao selecionar uma comanda, já existente é possível listar ou adicionar produtos a esta comanda, conforme podemos visualizar na Figura 24-C após adicionados os produtos é atualizado o valor total da comanda para visualização por parte do usuário.

Conforme novos produtos são cadastrados no ERP, a lista de produtos é atualizada automaticamente a cada consulta por parte do aplicativo. Quando alguma nova comanda é lançada, ela leva pelo menos 30 segundos para que seja carregada nos demais dispositivos, tempo este que foi definido em conjunto com o cliente.

Figura 24 - Telas lista comanda, cadastro comanda e itens comanda



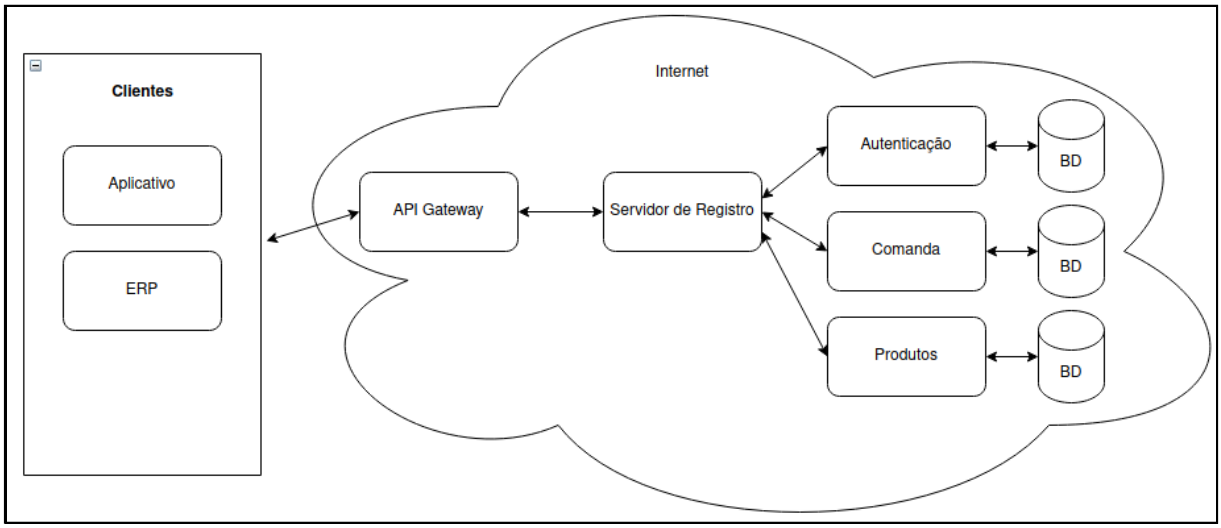
Fonte: Autoria própria (2021)

4.2 Arquitetura dos micro serviços

Os clientes enviam requisições para o API Gateway, que é responsável por prover roteamento dinâmico para os demais serviços da estrutura interna, sem expor assim estes serviços, possibilitando, por exemplo, atualização contínua de serviços internos sem gerar uma indisponibilidade, mesmo que temporária de uma funcionalidade, considerando que esta arquitetura permite escalar horizontalmente os serviços conforme necessidade.

O servidor de registro mantém uma lista com todos os serviços disponíveis para utilização. Os serviços de autenticação, comanda e produtos se registram e são monitorados pelo servidor de registro. Estes serviços também possuem suas próprias bases de dados independentes e podem se comunicar entre si, este processo é ilustrado pela Figura 25.

Figura 25 - Arquitetura dos micro serviços



Fonte: Autoria própria (2021)

Esta arquitetura é mais utilizada em aplicações de grande porte, principalmente pela flexibilidade e também por facilitar processos como o de integração contínua e entrega contínua CI/CD, nem por isto deixa de ser interessante para projetos menores, as características com maior destaque são a possibilidade de segregação de tarefas e uma entrega mais rápida de resultados.

Vale observar que esta arquitetura gera mais artefatos, que necessitam ser gerenciados, então esta é uma consideração importante a ser feita.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresentou um aplicativo para Android utilizando o *framework* Ionic, e uma API de integração construída em uma arquitetura de micro serviços.

Foi utilizada a metodologia de desenvolvimento Scrum, ela divide o desenvolvimento do trabalho em *sprints* que possuem pequenas tarefas, após a conclusão destas tarefas, tem se a finalização do *Sprint,* com isto é encerrada uma parte do projeto.

Para a elicitação de requisitos, foi visitado um restaurante na cidade de Irati, onde foram elencadas as principais dificuldades no atendimento, quais requisitos o aplicativo deveria atender, para ser oferecido ao estabelecimento.

Com a documentação pronta, foi iniciado o desenvolvimento da API, e dos micro serviços, para esse *sprint* foi utilizado o *framework* Spring e Java como a linguagem de programação.

No segundo *sprint*, foi desenvolvido o aplicativo Android, com as telas que utilizam os dados disponibilizados pela API, e também para envio de informações para o *gateway*.

O aplicativo possui um cadastro de usuários, após o *login*, o garçon visualiza a tela de consulta de comandas, nesta é possível editar uma comanda existente ou também inserir uma nova. Dentro das comandas possui uma consulta de produtos, nesta é possível adicionar um novo produto na comanda. Foi optado por não permitir cadastro de novos produtos, pois isso é função da gerência e deve ser feita através do sistema ERP.

Também foi determinado que os pedidos não serão finalizados pelo aplicativo, conforme requisito solicitado, função esta que deve ser realizada por meio do sistema PDV do caixa.

Após a finalização do aplicativo, foi oferecido ao restaurante uma versão para testes, em uma visita, com o aplicativo instalado no celular, foram realizados três pedidos, e com resultados satisfatórios, porém com uma ressalva por parte dos garçons, os operadores principais do aplicativo, referente a tela do celular disponibilizado ser pequena, dificultando a utilização durante o atendimento. Ficou definido que em um futuro próximo o estabelecimento irá adquirir um *tablet* para seguir com a implantação o projeto.

5.1 Trabalhos futuros

Prosseguir com a implantação do projeto no estabelecimento conforme mencionado, complementando assim com uma quantidade maior de em ambiente de produção, de maneira prática, com resultados de uso no dia a dia, também poderia ser desenvolvido um fechamento de pedidos.

Como melhoria para o aplicativo, com o objetivo de aumentar a robustez poderiam ser gravados todos os dados de pedidos diretamente no SQLite do próprio dispositivo, sincronizando em *background* os dados com a API, contornando assim casos de indisponibilidade temporária de rede.

Com relação aos micro serviços, existem algumas oportunidades interessantes que poderiam ser exploradas, como por exemplo a utilização do API Gateway utilizando balanceamento de carga, limite de requisições, filtros para tratamento e alteração de requisições e respostas, quebra de circuito e monitoramento de instâncias. Outro ponto interessante seria com relação a métodos para escalar horizontalmente os serviços.

REFERÊNCIAS

AMAZON, (2021). Microservices: Build highly available microservices to power applications of any size and scale. Disponivel em: https://aws.amazon.com/microservices/. Acesso em 12 nov. 2021.

AULIA, R.; et al. Mechanism of Food Ordering in A Restaurant Using Android Technology. In Journal of Physics: Conference Series, 930, 012030. 2017. Disponível em: https://doi.org/10.1088/1742-6596/930/1/012030. Aceso em 12 nov. 2021.

AZURE, (2021). Microservices architecture style. Disponível em: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices. Acesso em 11 nov. 2021.

CURRY, David. Android Statistics (2021), 2021. Disponível em: https://www.businessofapps.com/data/android-statistics/. Acesso em: 11 de nov. 2021.

DE PAULA, Gilles. Tudo sobre Metodologia Scrum: o que é e como essa ferramenta pode te ajudar a poupar tempo e gerir melhor seus projetos (2016), 2016. Disponível em: https://www.treasy.com.br/blog/scrum/. Acesso em: 12 nov. 2021.

GATEWAY, Spring Cloud Gateway (2021), 2021. Disponível em: https://spring.io/projects/spring-cloud-gateway. Acesso em: 12 nov. 2021.

LEWIS, James. (2014). Microservices - A definition of this new architectural term. Disponível em: http://martinfowler.com/articles/microservices.html. Acesso em 11 nov. 2021.

MORDOR (Mordor Intelligence). Global Point of Sale (POS) Terminal Market - Growth, Trends, COVID-19 Impact, and Forecasts (2021 - 2026). Disponível em: https://mordorintelligence.com/industry-reports/point-of-sale-terminal-market. Acesso em: 12 nov. 2021.

MOZZILA. Cross-Origin Resource Sharing (CORS). 2021. Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. Acesso em: 12 nov. 2021.

MYSQL, (2021) MySQL as an Embedded Database. Disponível em: https://www.mysql.com/oem/. Acesso em 12 nov. 2021.

NETFLIX, Eureka at a glance (2014). 2014. Disponível em: https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance. Acesso em: 12 nov. 2021.

PATEL, Neil. ERP: O Que É e Por Que Sua Empresa Precisa Desse Sistema (2018), 2018. Disponível em: https://neilpatel.com/br/blog/erp-o-que-e/. Acesso em: 11 nov. 2021.

PAULA FILHO, W.P. Engenharia de software: fundamentos, métodos e padrões. 3. ed. Rio de Janeiro: LTC,2009, 1248p.

REDHAT, (2017). What are microservices? Disponível em: https://www.redhat.com/en/topics/microservices/what-are-microservices. Acesso em 12 nov. 2021.

REDHAT, (2019). What does an API gateway do? Disponível em: https://www.redhat.com/en/topics/api/what-does-an-api-gateway-do. Acesso em: 12 nov. 2021.

REHKOPF, Max. O que é um scrum master?. Disponível em: https://www.atlassian.com/br/agile/scrum/scrum-master. Acesso em: 12 nov. 2021.

SANKHYA. Por que implementar um sistema PDV no seu varejo? Disponível em: https://www.sankhya.com.br/blog/pdv/. Acesso em: 12 nov. 2021

SEBRAE (Serviço Brasileiro de Apoio às Micro e Pequenas Empresas). Marketing. Ponto de Venda. 2016. Disponível em: https://www.sebrae.com.br/Sebrae/Portal%20Sebrae/UFs/RJ/Anexos/MARKETING_ponto_venda.pdf. Acesso em 11 nov. 2021.

SILBERSCHATZ, Abraham, KORTH, Henry F. e SUDARSHAN, S. Sistema de Banco de Dados. Editora Campus. 5a Edição, 2006.

SOMMERVILLE, I. Engenharia de software. 6. ed., São Paulo: Addison Wesley, 2003, 592p.

TOTVS. Sistema PDV: Por que implementar em seu negócio? (2021), 2021. Disponível em: https://www.totvs.com/blog/gestao-varejista/sistema-pdv/. Acesso em 12 nov. 2021.