Relatório de Estágio

Ricardo Dias Ferreira

Curso Técnico Superior Profissional em
Análise de Dados

jul | 2023

GUARDA
POLI
TÉCNICO

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

UNIVERSITY STUDENT CARD APP

RELATÓRIO DE ESTÁGIO
PARA OBTENÇÃO DO DIPLOMA DE TÉCNICO(A) SUPERIOR PROFISSIONAL
EM ANÁLISE DE DADOS

Ricardo Dias Ferreira
Julho / 2023

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

UNIVERSITY STUDENT CARD APP

RELATÓRIO DE ESTÁGIO
PARA OBTENÇÃO DO DIPLOMA DE TÉCNICO(A) SUPERIOR PROFISSIONAL
EM ANÁLISE DE DADDOS

Professor(a) Orientador(a): Paulo Alexandre Andrade Vieira

Professor(a) Supervisor(a): Carlos Eduardo dos Santos Fonseca

Ricardo Dias Ferreira
Julho / 2023

POLI TÉCNICO GUARDA

Ficha Técnica

Estagiário
Ricardo Dias Ferreira
N° - 1706995
GitHub: ricardof2303

Curso Técnico Superior Profissional
CTeSP-AD – Análise de Dados

Estabelecimento de Ensino
Escola Superior de Tecnologia e Gestão
Instituto Politécnico da Guarda
Morada: Avenida Dr. Francisco Sá Carneiro, 6300-559, Guarda
Telefone: 271 220 120

Instituição de acolhimento
Entidade: Centro de informática – Escola Superior de Tecnologia e Gestão
Localização: Avenida Dr. Francisco Sá Carneiro, 6300-559, Guarda
Email: ci@ipg.pt
Site: www.ci.ipg.pt
Supervisor: Engenheiro Carlos Fonseca

Duração do estágio
Início: 20/01/2023
Fim: 09/05/2023

Resumo

Este relatório de estágio de Ricardo Ferreira descreve a experiência deste autor durante o seu estágio no Centro de Informática do Instituto Politécnico da Guarda, no período de janeiro a maio de 2023. O estágio foi realizado no âmbito do desenvolvimento de software, tendo como objetivo desenvolver uma aplicação que permitiria a utilização de um cartão digital, destinado aos estudantes deste Instituto.

Neste relatório são descritas as atividades que o autor desenvolveu durante o seu estágio, tais como, os desafios enfrentados, o trabalho realizado, e as aplicações desenvolvidas. O estágio consistiu no desenvolvimento de 2 aplicações, uma afeta ao cartão dos estudantes, o Student Card Digital, e outra intitulada IpgService, que visa servir para o registo de eventos. A IpgService é direcionada para os gestores de eventos e permite o registo de presenças através de QRcodes, o que faz da app do Student Card Digital uma sua app cliente.

As apps foram desenvolvidas em Flutter e Dart e o GitHub foi usado como sistema de gestão e versionamento de código. Na Student Card Digital, a gestão e o uso da aplicação exigem o consumo de APIs para obter dados, e a gestão da aplicação em termos de segurança exige autenticação no servidor sempre que a app for aberta, juntamente com uma gestão local de geração de códigos QR. Um código QR gerado na app tem uso limitado ao tempo, obrigando sempre à geração de um novo para uso no Cartão. A Student Card Digital fornece também um conjunto de serviços do cartão: atualização do ecrã a cada segundo como forma de evitar fraudes por fotografias da aplicação, geração de códigos QR, geração de páginas de eventos, geração de páginas de notícias, ligação para o site Luop (que faz a gestão das cantinas do IPG). A app de gestão de eventos deve ser usada pelos criadores de eventos e permite monitorizar os eventos e registar a presença por meio de códigos QR. Isso significa que a app Student Card Digital e outras do mesmo género servem como apps clientes.

O relatório destaca a importância do estágio para o crescimento profissional do autor, que teve a oportunidade não só de colocar em prática os seus conhecimentos teóricos num ambiente real de trabalho, mas também de descobrir novos conhecimentos e de procurar ajuda nas questões que surgem durante o processo, por exemplo, a criptografia e a geração de códigos QR.

Finalmente, o relatório apresenta as conclusões e considerações finais do autor sobre sua experiência de estágio: A importância de manter boas relações com a equipa de trabalho, a necessidade de procurar constantemente o conhecimento, de enfrentar desafios e imprevistos e da aprendizagem adquirida em relação à metodologia ágil de desenvolvimento.

Palavras Chave:

Flutter, Dart, Cartão do Estudante Universitário, aplicação móvel, widget, QRcode, criptografia.

Abstract

This internship report by Ricardo Ferreira describes this author's experience during his internship at the Centro de Informática do Instituto Politécnico da Guarda, from January to May 2023. one that will allow the use of a digital card, intended for students of this Institute.

In this report, the activities that the author developed during his internship are described, such as the challenges faced, the work carried out, and the applications developed. The internship consisted of the development of 2 applications, one related to the students' card, the Student Card Digital, and another called IpgService, which aims to record events. The IpgService is aimed at event managers and allows attendance registration through QRcodes, which makes the Digital Student Card app your client app.

The apps were developed in Flutter and Dart and GitHub was used as a code management and versioning system. In Student Card Digital, the management and use of the application compensated for the consumption of APIs for data, and the management of the application in terms of security requires authentication on the server whenever the app is opened, together with local management of QR code generation . A QR code generated in the app has limited use, always requiring the generation of a new one for use on the Card. The Student Card Digital also provides a set of card services: updating the screen every second as a way to avoid fraud due to application photographs, generation of QR codes, generation of event pages, generation of news pages, link to the website Luop (who manages the IPG canteens). An event management application is intended for use by event creators and allows you to monitor events and register attendance via QR codes. This means that the Digital Student Card app and others like it serve as client apps.

The report highlights the importance of the internship for the professional growth of the author, who had the opportunity not only to put his theoretical knowledge into practice in a real work environment, but also to discover new knowledge and seek help with questions that arise during the internship. process, for example encryption and generation of QR codes.

Finally, the report presents the author's final considerations about his internship experience: The importance of maintaining good relationships with the work team, the need to constantly seek knowledge, to face challenges and unforeseen circumstances, and the learning acquired in relation to the agile methodology of development.

Key words:
Flutter, Dart, University Student Card, mobile app, widget, QRcode, encryption.

Índice

Índice de Figuras

Acrónimos

API - *Application Programming Interface*

APP - *Application (Aplicação)*

CSS - *Cascading Style Sheets*

ESTG - *Escola Superior de Tecnologia e Gestão*

GUI - *Graphical User Interface*

HTTPS- *Hyper Text Transfer Protocol Secure*

HTTP - *Hyper Text Transfer Protocol (Protocolo de Transferência de Hipertexto)*

IDE - *Integrated development Environment (ambiente de desenvolvimento integrado)*

IEEE - *Institute of Electrical and Electronic Engineers*

IPG - *Instituto Politécnico da Guarda*

IOS - *iPhone operating system*

ISIC - *International Student Identity Card (Cartão Internacional de Estudante)*

POO - *Programação orientada a objetos*

QRcode - *Quick Response Code*

RTP - *Rádio e Televisão de Portugal*

SDK - *software development kit*

TIC - *Tecnologias de Informação e Comunicação*

TCP/IP- *Transmission Control Protocol/Internet Protocol*

UCL - *University College London*

URL - *Uniform Resource Locator (Localizador Uniforme de Recursos)*

1. Introdução

Um grupo de antropólogos da University College London (UCL) estudou, durante mais de um ano, a relação da população com os smartphones e a importância que estes têm na rotina diária. A equipa concluiu que os telemóveis deixaram de ser apenas um dispositivo através do qual as pessoas comunicam, para se tornarem "um lugar onde agora vivemos" (Mariana Ribeiro Soares – RTP, 2021). Nos últimos anos, o acesso ao smartphone generalizou-se e o número de aplicações existentes em cada dispositivo é geralmente muito grande. As instituições de ensino como parte da sociedade, não podem ficar de fora dessa evolução, e certamente aproveitarão as ferramentas atuais naquilo que de alguma forma podem ajudar na comunicação e na aprendizagem. O que foi desenvolvido no estágio a que esse relatório descreve, certamente é um início.

O presente relatório de estágio tem como objetivo apresentar as atividades desenvolvidas durante o período de realização do estágio no centro de informática do Instituto Politécnico da Guarda. O estágio foi realizado como parte do curso técnico superior profissional em análise de dados e teve a duração de 750 horas.

O estágio foi realizado no Centro de informática da Escola Superior de Tecnologia e Gestão no Instituto Politécnico da Guarda.

O projeto de estágio está baseado no desenvolvimento de uma aplicação, que nesse primeiro momento chamamos de Student Card (Cartão do estudante). Essa aplicação poderá vir a ser o novo cartão do estudante do Instituto Politécnico da Guarda, o qual tem dentre as suas possibilidades de utilização, a identificação do estudante na instituição ou fora da mesma.

A aplicação foi desenvolvida utilizando o Flutter como ferramenta principal. Trata-se de um framework de desenvolvimento de aplicações móveis criado pela Google. Ele permite criar aplicações de alta qualidade e alto desempenho para iOS, Android, web e desktop usando uma única base de código. Esta aplicação conta com parâmetros voltados para a segurança que são a criptografia assimétrica da senha como forma de impedir a fraude da chave de segurança, e atualização do ecrã a cada segundo para impedir a clonagem por fotografia do mesmo.

Todos os estudantes poderiam ter acesso à aplicação da seguinte maneira: quando o estudante faz sua matrícula na instituição recebe o número de estudante, e no email cadastrado na matrícula, recebe uma senha. Com esses dois dados em mão, o aluno poderá ter acesso à aplicação Student Card.

O relatório está dividido em três secções que apresentam as tecnologias utilizadas, a descrição das atividades desenvolvidas com trabalhos práticos, o desenvolvimento da aplicação em si e ainda de uma segunda aplicação que utiliza a

aplicação principal como cliente, além de uma conclusão do trabalho realizado. É importante ainda referir que durante o estágio, foram desenvolvidas competências como trabalho em equipe, liderança, capacidade de resolução de problemas, comunicação e organização.

Por fim, é importante ressaltar que a experiência vivenciada no estágio foi fundamental para o desenvolvimento profissional e pessoal, permitindo uma maior compreensão sobre a área de atuação, além de possibilitar a aplicação prática dos conhecimentos adquiridos durante o curso.

1.1 Tecnologias utilizadas

Visual Studio Code

O Visual Studio Code é um editor de código-fonte desenvolvido pela Microsoft para Windows, Linux e macOS. Ele inclui suporte para depuração, controle de versionamento Git incorporado, realce de sintaxe, complementação inteligente de código, snippets e refatoração de código. Ele é customizável, permitindo que os utilizadores possam mudar o tema do editor, teclas de atalho e preferências. Ele é um software livre e de código aberto, apesar do download oficial estar sob uma licença proprietária. (Definição tirada do site da Wikipedia - pt.wikipedia.org/wiki/Visual_Studio_Code)

Ferramentas de pesquisa

Google, YouTube, Sites, etc.

Aparelho telemóvel

Modelo – M2006C3MG
Versão do Android – 10QP1A. 190711.020

Scrcpy

Traduz-se "cópia de ecrã", é uma aplicação de espelhamento de ecrã gratuito e de código aberto que permite o controle de um dispositivo Android a partir de um computador desktop Windows , macOS ou Linux .

Android Studio

O Android Studio oferece um ambiente unificado onde é possível criar aplicações para telefones, tablets, Android Wear, Android TV e Android Auto. Os módulos de código estruturados permitem a divisão do projeto em unidades de funcionalidade que você pode construir, testar e depurar de forma independente. (https://developer.android.com/studio/features?hl=pt-br)

1.2 Metodologia

As metodologias de gestão de projetos são conjuntos de princípios, ferramentas e/ou técnicas que são usados para planear, executar e gerir projetos. Elas ajudam os gestores de projetos a liderar as suas equipas, facilitando a colaboração entre todos os membros.

1.2.1 Método utilizado

Nesse projeto decidi aplicar uma abordagem ágil pois, a abordagem ágil tem demonstrado ser melhor aplicada para entregas de software, onde a falta de fisicalidade e a variação de requisitos é tratada através da sua abordagem leve para a execução do projeto.
Como um projeto ágil seguimos o seguinte modelo de desenvolvimento:

- Estabelecem-se os fundamentos do projeto:
  - A visão do negócio para o projeto;
  - Os objetivos;
  - Uma lista dos requisitos/funcionalidades;
- O projeto foi dividido em parcelas;
- Foram desenvolvidas as funcionalidades;
- As funcionalidades foram testadas e integradas;

Como num desenvolvimento ágil, cada aspeto do desenvolvimento do projeto – requisitos, desenho, arquitetura, construção, testes, etc. – foi continuamente revisitado ao longo do ciclo de vida do desenvolvimento.

O projecto foi acompanhado com reuniões semanais, o que possibilitou a condução numa direção diferente, quando necessário.

A combinação dos ciclos de desenvolvimento curtos e de duração fixa (iterações semanais) e a priorização dos requisitos permitiu salvaguardar o projecto de riscos de desalinho com os objectivos. Um aspeto fundamental da abordagem ágil é a adaptabilidade: no desenvolvimento tem-se em atenção o objectivo e a forma de o alcançar. Na abordagem ágil os processos de desenvolvimento são divididos em iterações de curta duração (sprints, no nosso caso semanal).

1.3 Etapas do desenvolvimento (Gantt)

Durante todo o processo de desenvolvimento houve reuniões semanais com o professor orientador Paulo Vieira e com o professor Supervisor Carlos Fonseca, para acompanhamento e auxílio na elaboração do projeto, na imagem abaixo temos o cronograma que seguimos (Figura 1).

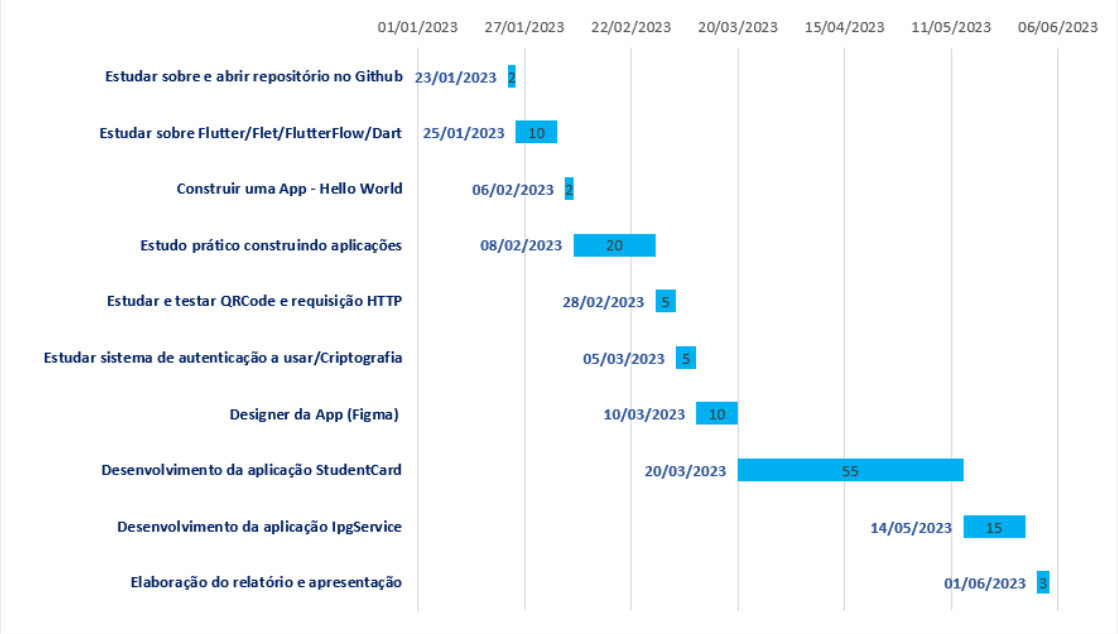

Figura 1 – Etapas do desenvolvimento (Gant)

Temos também uma segunda imagem (Figura 2) que mostra um cronograma que estava previsto para o estágio.

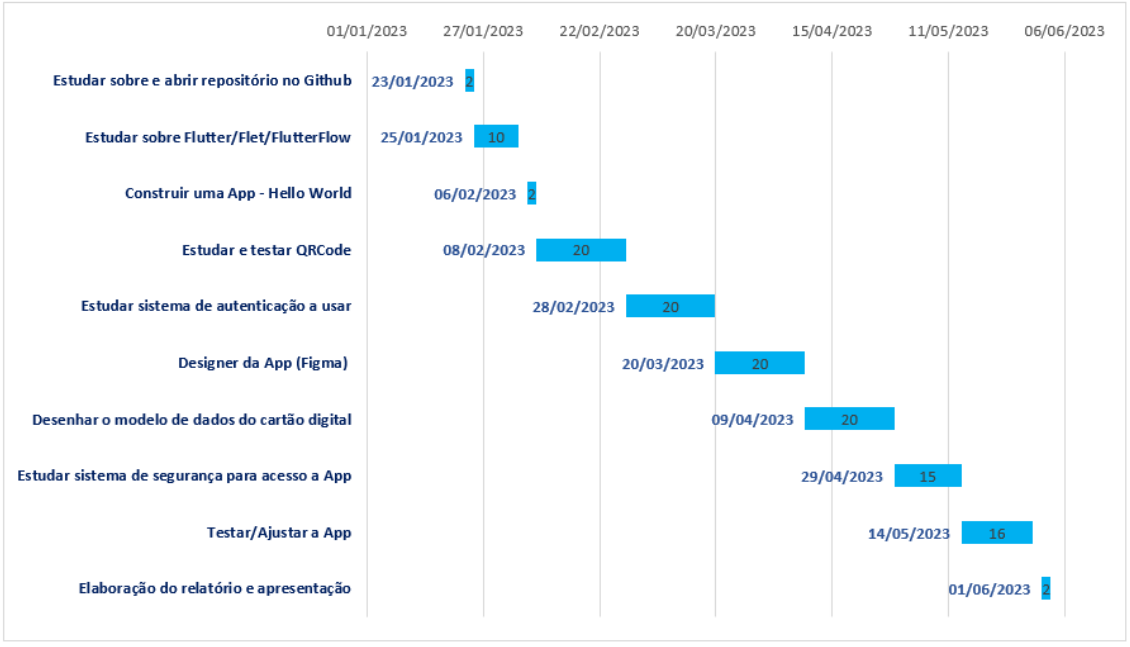

Figura 2 – Etapas previstas para o do desenvolvimento (Gant)

A respeito do cronograma de previsão, há duas observações importantes a se fazer. A primeira é a respeito do desenho do modelo de dados que a aplicação utilizaria. Por motivo de segurança em relação ao mantimento dos dados dos alunos salvos na aplicação, o meu supervisor Carlos Fonseca que cuidou da parte de back-end da aplicação, entendeu que o melhor era trabalharmos mantendo os dados dos alunos de forma segura nas bases de dados do IPG, de forma que para ter acesso a esses dados trabalhamos com requisições HTTPS e sempre que apetece ao aluno apagar seus dados da aplicação, basta carregar no botão preparado pra esse objetivo. Isso pode ser mudado no futuro e desenvolver-se uma base de dados para a aplicação, mas foi assim que trabalhamos nesse início. Na imagem a seguir (Figura 3) temos um modelo de um arquivo JSON que é retornado após requisição de dados pela aplicação, no caso requisição de dados do aluno. Sempre que nesse relatório falar a respeito de dados, falo de um JSON como este que traz os dados que são requisitados da base de dados do IPG.


```
[{'result':'1',
'curso':'Análise de Dados',
'escola':'Escola Superior de Tecnologia e Gestão',
'estado':'Inscrito',
'grau':'Tesp',
'nome':'Ricardo',
'numero':'1706995',
'linkFoto':'cloud.sysnovare.pt/ipg/FOTOGRAFIAS_SERVICE.foto?pct_cod=',
'AnoLetivoEstado':'22/23'}]
```


Figura 3 - modelo de ficheiro JSON com dados de aluno

A segunda observação a respeito do cronograma de previsão, é sobre a tarefa de testar e ajustar a aplicação. Fizemos (eu e meu orientador) alguns testes com a aplicação em nossos telemóveis, mas um teste com um grupo de pessoas utilizando a aplicação ainda não foi feito, está previsto para o próximo semestre letivo no IPG com uma das turmas coordenadas pelo professor e coordenador Paulo Vieira, para em seguida entrar em produção.

2. Descrição de estudos práticos para desenvolvimento da App

2.1 Estado da Arte

É muito comum nas instituições de ensino superior o cartão do estudante no formato físico, inclusive com ligação a alguma instituição bancária, onde o cartão é utilizado para além da identificação como estudante daquela instituição, mas também pode-se fazer depósitos de valores financeiros e utilizar o cartão como um cartão de débito (https://www.cgd.pt/Particulares/Cartoes/Cartoes-de-Debito/Pages/Cartao-Debito-Caixa-IU.aspx, 03/2023).

No caso do Instituto Politécnico da Guarda, a ligação é com a instituição bancária Santander, mas o depósito que os alunos fazem serve apenas para ser utilizado nos pagamentos das refeições feitas nas cantinas do IPG.

Pesquisando em alguns Institutos Politécnicos da região (nos três que envio o link abaixo em março de 2023), não encontrei nenhum que tivesse uma aplicação como essa que estamos a desenvolver.

https://www.ipcb.pt/ - Instituto Politécnico de Castelo Branco

https://www.ipv.pt/ - Instituto Politécnico de Viseu

https://www.ipleiria.pt/ - Instituto Politécnico de Leiria

Aprofundando o estudo do estado da arte, fiz uma pesquisa para ver a possibilidade de um modelo internacional e encontrei algo parecido, o ISIC.

O ISIC - Cartão Internacional de Estudante (International Student Identity Card) - é um documento que reconhece internacionalmente o estatuto de estudante.

No site onde a pesquisa foi feita, o mesmo é referenciado abaixo, os desenvolvedores do produto o colocam como sendo um passaporte para descontos em Portugal e em todo o mundo. Para além da identificação como estudante, ao adquirir o cartão ISIC o indivíduo passa a fazer parte de uma comunidade global que conta com cerca de 5 milhões de estudantes de 130 países que usam o cartão para aproveitarem as ofertas que têm em viagens, lojas, museus e mais.

Desenvolvido para proporcionar o entendimento intercultural e o intercâmbio internacional, originalmente o cartão ISIC proporciona acesso a descontos exclusivos em viagens, permitindo aos portadores do cartão compreender diferentes culturas e trocar experiências com estudantes de outros países, de outras culturas e de outras línguas.

Este cartão foi criado por estudantes para estudantes em 1953. A decisão de criar o cartão ISIC foi uma iniciativa da União de Estudantes Noruegueses e Holandeses, apoiado pela COSEC - Secretariado Coordenador dos Sindicatos Nacionais de Estudantes - da Dinamarca, na 3ª Conferência Internacional de Estudantes onde foi tomada a decisão de "tomar as medidas necessárias para garantir que um cartão internacional de estudante seja disponibilizado logo que possível no maior número possível de países."

Originalmente, o cartão ISIC era um cartão de identificação que garantia aos estudantes a tempo inteiro de todo o mundo o acesso a tarifas aéreas especiais. Para terem acesso a tarifas com desconto os estudantes compravam o cartão ISIC, que permitia aos portadores explorar o mundo e descobrir novos países e culturas.

Isto apoiava a missão pioneira do cartão ISIC “aumentar o entendimento internacional através da promoção das viagens e oportunidades de intercâmbio entre os estudantes, jovens e comunidade académica.”

O ISIC existe em formato virtual e físico. Após fazer o cartão de estudante físico, é possível baixar a aplicação da plataforma e ativar sua conta virtual com os dados que estão no cartão físico de forma a poder utilizar o cartão estudantil virtual. (https://www.isic.pt/pt/, 02/2023)

Embora o ISIC tenha essa modalidade virtual, não se iguala a aplicação que estamos a desenvolver nesse projeto, de forma que a mesma seria uma inovação e além de trazer um contributo para os alunos do IPG, pode também servir de modelo para futuros projetos nas instituições de ensino.

2.2 Git e GitHub

O primeiro passo para o desenvolvimento do projeto do estágio consistia na realização de uma investigação acerca do Git e do GitHub e na criação de um repositório, que foi então partilhado com o meu professor orientador, para acompanhamento e direção. O presente texto foi redigido com base nas anotações a partir desta pesquisa e são aqui identificadas as fontes dos canais de aulas assistidas e dos websites consultados para o efeito.

Git e GitHub são duas tecnologias que surgiram para resolver o problema de gestão de versões de ficheiros e é importante que todos os desenvolvedores a aprendam, independentemente da sua área.

O que é Git?

O Git é um sistema de controlo de versões que permite rastrear as alterações feitas nos seus ficheiros ao longo do tempo. Com o Git, as pessoas podem reverter para vários estados dos seus ficheiros (como se utilizasse uma máquina do tempo), também é

possível criar uma cópia idêntica de um ficheiro, trabalhar nele e, quando estiver satisfeito com as alterações, pode-se fundir "merge" a cópia no ficheiro original, o principal "main".

link de acesso ao git: https://git-scm.com/

O que é o GitHub?

O GitHub é um serviço de alojamento online para repositórios do Git. De forma resumida, o GitHub permite que as pessoas criem e guardem o seu repositório na plataforma. Outro recurso que vem com o GitHub é a possibilidade de colaborar com outros desenvolvedores de qualquer lugar.

Segundo Guilherme Oliveira em 2021 [1] "O funcionamento básico do Git pode ser resumido em 4 conceitos (ou comandos): commit, branch, pull request e merge. Além de outros três relacionados à interação do Git com o GitHub: clone, pull e push."

link de acesso ao github: www.github.com

GitHub Desktop

O GitHub Desktop é uma aplicação que permite que interaja com o GitHub através de uma interface gráfica, em vez de usar um ambiente de linha de comando ou um navegador web. É possível carregar e descarregar ficheiros, assim como clonar repositórios remotos com o GitHub Desktop e usar ferramentas para trabalho colaborativo como atribuir commits e criar pedidos de pull. O GitHub Desktop amplia e simplifica ao mesmo tempo o fluxo de trabalho no GitHub, tendo uma interface visual em vez de comandos de texto a usar na linha de comando.

2.3 Flutter e Dart

Um dos primeiros passos do projeto foi adquirir conhecimentos sobre a linguagem de programação que seriam utilizadas no seu desenvolvimento. Como já tinha estudado outras linguagens no curso de Análise de Dados (Java e Python), fiz um curso básico da linguagem dart e continuei a aprender ao estudar o framework (Flutter) que usaria para desenvolver a aplicação. Então fiz o curso básico do canal no YouTube Startto.dev e procurei mais informação na documentação do dart.dev. Seguem-se as minhas anotações.

2.3.1 Dart

link da linguagem Dart: https://dart.dev

Dart é uma linguagem otimizada para que os clientes desenvolvam aplicações rápidas em qualquer plataforma. O seu objetivo é fornecer a linguagem de programação mais produtiva para desenvolvimento multi-plataforma, juntamente com uma plataforma de tempo de execução flexível para estruturas de aplicações.

O Dart foi projetado para uma caixa técnica que é especificamente adequada para o desenvolvimento do cliente, dando prioridade ao desenvolvimento e às experiências de produção de alta qualidade em uma ampla variedade de destinos de compilação (web, móvel e desktop). O Dart fornece a linguagem e os tempos de execução que alimentam as aplicações Flutter.

Dart: A linguagem

Segundo Jonathan Sande em 2022 [2] "não foi por acaso que Flutter escolheu Dart como sua linguagem. A máquina virtual Dart permite reconstruções em tempo de desenvolvimento extremamente rápidas, e seu compilador avançado cria aplicações nativos para todas as principais plataformas. Como uma das linguagens mais versáteis do mercado atualmente, você pode usar o Dart para escrever qualquer coisa, desde aplicações de linha de comando e servidores de back-end até aplicações nativas para Android, iOS, Web, Mac, Windows, Linux e até mesmo dispositivos incorporados."

Dart: As bibliotecas

O Dart tem um vasto conjunto de bibliotecas principais, fornecendo o fundamental para muitas tarefas de programação quotidianas. Além das bibliotecas principais, muitas APIs são disponibilizadas através de um extenso conjunto de pacotes. A equipa do Dart publica muitos pacotes de adição úteis, e editores terceiros e a comunidade, em geral, publicam milhares de pacotes, o que nos deixa com muitas opções, se estamos a desenvolver a nossa aplicação Student Card.

2.3.2 Flutter

link flutter: https://flutter.dev/

O Flutter é um framework da Google que permite construir aplicações para Android, iOS, Desktop ou Web com apenas um código. O SDK (Software Development Kit) do Flutter tem código aberto e é gratuito.

O Flutter utiliza a linguagem Dart para construir aplicações. Algumas das características principais do Flutter incluem:

Widgets personalizados: O Flutter tem um conjunto completo de widgets totalmente personalizáveis, permitindo-lhe criar interfaces de utilizador únicas e impressionantes para as suas aplicações. A Hot reload do Flutter permite-lhe visualizar alterações em tempo real durante o processo de programação, aumentando assim drasticamente a velocidade do desenvolvimento.

Alto desempenho: o Flutter é altamente otimizado para o desempenho e usa um mecanismo de renderização próprio chamado Flutter Engine. Isso permite que suas aplicações funcionem suavemente e com rapidez, mesmo em dispositivos mais antigos.

Suporte a várias plataformas: com o Flutter, é possível criar aplicações para iOS, Android, web e desktop. Isso significa que você pode escrever um único código e implantá-lo em várias plataformas.

Comunidade ativa: o Flutter tem uma comunidade de desenvolvedores muito ativa e em crescimento, o que significa que há muitos recursos e suporte disponíveis.

Em resumo, o Flutter é uma ótima opção para quem deseja criar aplicações móveis multiplataforma com alta qualidade e alto desempenho. Com sua arquitetura moderna, recursos poderosos e comunidade ativa, o Flutter é uma escolha popular para muitos desenvolvedores de aplicações móveis.

Comecei o projeto com o estudo da linguagem de programação a ser utilizada no desenvolvimento, e continuei a aprender a linguagem Dart, mas agora em conjunto com o estudo do framework Flutter, onde passei a mesclar o estudo com a prática ao desenvolver algumas aplicações.

As seguintes anotações são de aplicações que foram criadas estudando com o professor Carlos Duarte do canal “Polimorfismo” no Youtube, março 2023 [16].

2.3.3 App Olá_Mundo

Esta foi a minha primeira aplicação (figura 4). Seguindo o que já é quase um padrão no meio dos desenvolvedores (Hello World), e já comecei a aprender como funcionava o Flutter, o que era e como era o funcionamento dos widgets, como eles se dividiam em widgets de layout e de interface. Esses dois tipos de widgets são utilizados em conjunto, basicamente, a construção de interfaces é compostos por widgets de interface[1] inseridos em widgets de layout[2].

[1] Widgets de Interface: Widgets responsáveis por criar componentes visuais que serão exibidos para os utilizadores. Estes widgets servem para determinar os componentes que comporão a interface do aplicativo.

[2] Widgets de Layout: Widgets responsáveis por determinar a organização e posicionamentos de outros widgets. Estes widgets servem para delimitar áreas em nossas telas que serão preenchidos por outros widgets.

O que esta aplicação faz quando é chamada com um click, é apenas exibir a mensagem "Olá Mundo!" no ecrã do telemóvel em que está instalada.

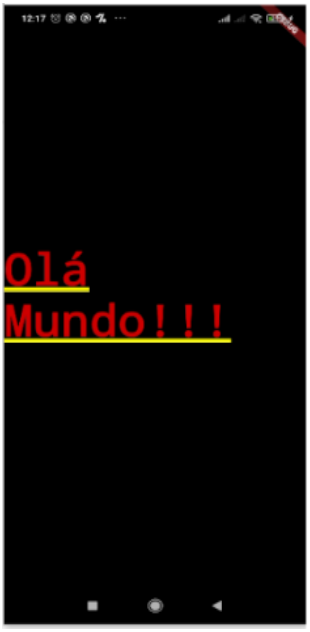

Figura 4 - AppOlaMundo

2.3.4 App EuSouRico e App EuSouProgramador

Quando a aplicação “eu sou rico” (figura 5) é aberta no telemóvel, aparece no ecrã uma imagem de um diamante, com o título "Eu sou Rico" na parte superior.

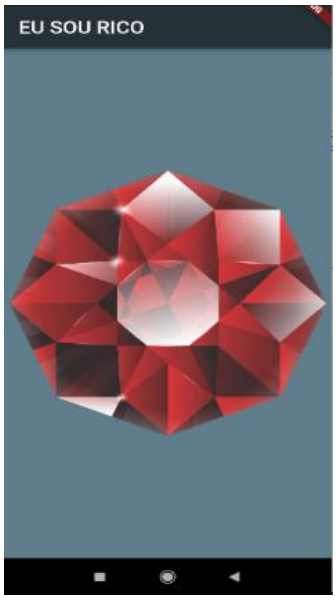

Figura 5 - AppEuSouRico

A aplicação eu "vou dominar o flutter" (figura 6) foi mais como que um treinamento do que havia aprendido na anterior, quando aberta no telemóvel, mostra no ecrã uma imagem relacionada ao flutter e na parte de cima o título "Eu vou dominar o flutter".

Figura 6 - AppEuDomFlutter

Nessa segunda e terceira aplicação, os principais conhecimentos foram sobre os widgets Scaffold que é o widget que implementa a estrutura do layout visual do Material Design básico e permite definir outros widgets dentro de si, o widget AppBar que trata-se de um dos componentes mais usados em todos os tipos de aplicações. Ele pode ser usado para abrigar um campo de pesquisa, botões para navegar entre as páginas ou simplesmente o título da página, o widget Body que pode conter qualquer widget de propósito geral e, a escolha vai depender das informações que quero exibir em minha aplicação, e ainda o widget Image que é usado para exibição de imagens, e que é algo fundamental para a maioria das aplicações móveis e o Flutter.

2.3.5 App Calculadora e App MyCard

Tudo o que aprendi nessa aplicação "MyCard" (figura 7) coloquei em prática no projeto. A utilização do CircleAvatar vem integrado com o SDK do flutter. É simplesmente um círculo no qual podemos adicionar cor de fundo, imagem de fundo ou apenas algum texto; ou a utilização do Card que é um widget embutido em flutter que deriva seu design da biblioteca de design de materiais do Google.

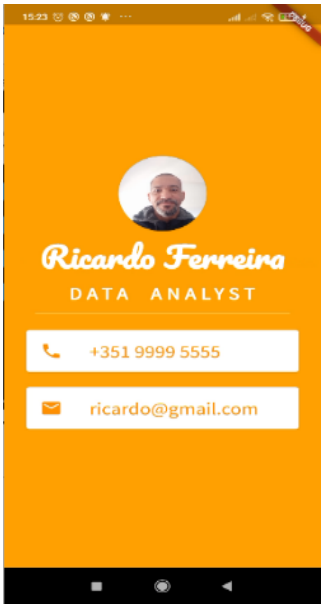

Figura 7 - AppMyCard

A aplicação "calculadora" (figura 8), monta no ecrã uma calculadora, embora esta ainda não seja funcional, isto é, não está com programação para fazer cálculos, pois nesse momento eu estava desenvolvendo aprendizagem sobre como construir um ecrã, foi muito útil os entendimentos sobre os widgets: o SafeArea é um widget importante e útil no Flutter, ele torna a interface do utilizador dinâmica e adaptável a uma ampla variedade de dispositivos; o Container que é uma classe de widget que permite colocar dentro de si outros widgets e personalizar esses widgets filhos; e ainda dois dos widgets mais importantes e poderosos do Flutter: Row e Column, esses widgets permitem alinhar os *"filhos"* horizontal e verticalmente de acordo com o requisito.

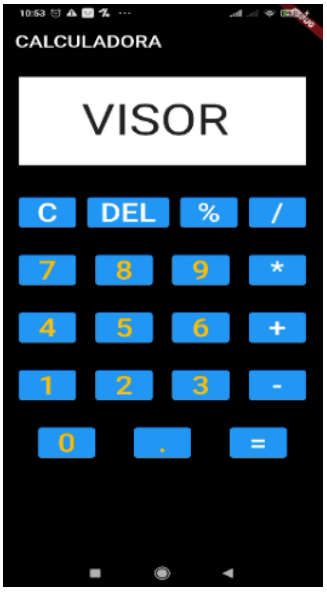

Figura 8 - AppCalculadora

2.3.6 App Dados

Nessa app fomos adiante nos conhecimentos, começando com o básico sobre funções que empacota uma quantidade de código dentro de chaves, e quando queremos executar todo esse código, chamamos a função pelo nome especificado, e fomos mais adiante com estudos sobre os widgets TextButton e IconButton.

O TextButton é um widget embutido no Flutter que deriva seu design da Biblioteca de design de materiais do Google, ele possui uma propriedade de estilo que aceita ButtonStyle como valor, usando esta propriedade de estilo podemos customizar o TextButton como quisermos.

O IconButton é a classe mais diferente fornecida no flutter. Um botão de ícone permite que os utilizadores executem ações como pesquisar, navegar, adicionar, etc., simplesmente pressionando o botão. Esta classe não possui um botão normal com algum texto, mas um ícone na forma de um botão. O ícone serve como um identificador.

A App "Dados" (figura 9), poderia ser utilizado para um jogo de adivinhação, a aplicação está carregada com seis imagens diferentes de dados, e sempre que um desses dados é tocados, no ecrã aparece duas imagens de dados que são chamadas por funções.

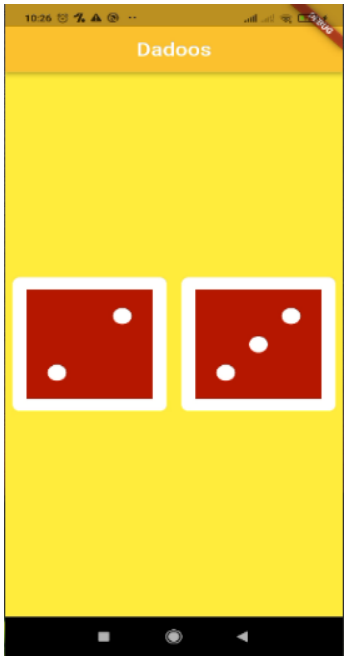

Figura 9 - AppJogoDados

2.3.7 App Music

Os pacotes têm marcadores de pontuação e popularidade, o que é um bom indicador para escolher qual pacote usar, pois quanto maior a pontuação, melhor. Muitos dos pacotes são desenvolvidos pela equipe do Flutter, o que pode ser outro

indicador de que se trata de um bom pacote para ser empregado. Para além do estudo da utilização das bibliotecas, eu também estudei as Arrow Functions e os Parâmetros das funções durante o desenvolvimento desta aplicação.

Um pacote Flutter é um conjunto de códigos escritos por outras pessoas com o objetivo de atingir uma meta, e essas pessoas têm a gentileza de partilhar o seu trabalho com a comunidade.

Nesta app (figura 10), aprofundei no meu conhecimento sobre como utilizar os pacotes do Flutter, pois era necessário usar um pacote para que a aplicação pudesse reproduzir músicas que existissem num determinado diretório, e é exatamente isso que esta aplicação faz, emitir som quando se clica numa das notas, de modo que se quem está a utilizá-la souber de música, é possível até mesmo compor uma melodia.

Figura 10 - AppNotasMusic

2.3.8 App SportsQuiz

Nesta aplicação (figura 11), aprofundei os meus conhecimentos sobre trabalho com listas, funções, estruturas condicionais, os pilares da Programação Orientada a Objetos e utilização de classes e objetos em Dart. Ela apresenta uma questão no ecrã com dois botões para o utilizador escolher a resposta correta. Se acertar ou errar a resposta, já é possível verificar imediatamente, pois é marcado um ícone na parte inferior do ecrã indicando o acerto ou o erro e, logo a seguir, surge uma nova questão. Além disso, no desenvolvimento desta aplicação, estudamos as estruturas condicionais e dois assuntos particulares do Dart, a utilização das palavras-chave 'const' e 'final'.

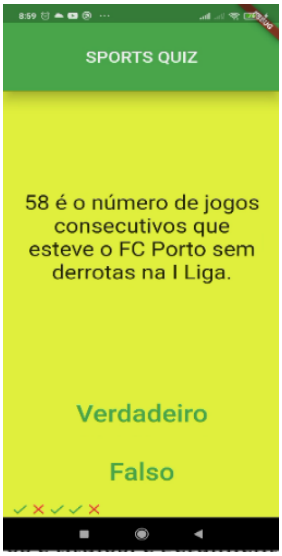

Figura 11 - AppSportsQuiz

2.4 Figma

Após o tempo investido no estudo na linguagem de programação e do framework que utilizei no desenvolvimento da aplicação, chegou o momento de estudar como desenhá-la, passamos então para o estudo do figma.

Sabe-se que o design é um elemento importante em uma aplicação, mas para que esse processo seja criado a contento é necessário organização e seleção das fontes certas. Dessa forma, a ferramenta Figma, apresenta um ambiente favorável para tal criação, conforme apresentado a seguir.

O Figma é uma das maneiras de criar ecrãs para produtos como aplicações, sites ou softwares, permitindo que a pessoa designer, crie todo o fluxo, estrutura e composição do projeto. Com essa ferramenta vamos construir nossa aplicação.

2.4.1 O que é o figma

O Figma é uma plataforma colaborativa para construção de design de interfaces e protótipos. É uma ferramenta gratuita e tem como objetivo trazer colaboração entre pessoas, possibilitando criar um produto para as mais diversas plataformas, mantendo a acessibilidade do sistema.

O Figma conta com as ferramentas que são usadas normalmente por designers, tais como formas geométricas, imagens, vetores, caixas de texto, entre outros. Trabalhando com esses elementos, temos a possibilidade de personalizar as cores, trabalhando tanto com cores sólidas como com diferentes tipos de gradiente, níveis e estilos de delineado, efeitos de sombra e desfoque, bem como as opções de textos.

O Figma é uma ferramenta voltada para designers, mas isso não significa que se destina apenas a esses profissionais. Designers de interface costumam trabalhar ativamente com programadores, pois é eles quem irão implementar os ecrãs que forem criados. Para facilitar esta colaboração, a ferramenta fornece uma área de inspeção que permite ver todos os detalhes de um elemento, como valores de largura, altura, espaçamento, cor, tipografia e até uma pré-visualização de código, gerado pelo programa, para CSS, Android ou iOS, o que facilita as tarefas de programação.

2.4.2 Figma Community

A forma de colaboração do Figma reflete-se não só em duas ou mais pessoas a trabalhar simultaneamente no mesmo ficheiro. A empresa criou uma forma para que o seu público colabore e contribua para o crescimento da ferramenta, através do partilhar e disponibilizar acesso a projetos prontos e de novas funcionalidades que ainda não existem no programa.

Assim, surgiu a Comunidade Figma, o meio pelo qual todas as pessoas podem publicar e partilhar ficheiros dentro da ferramenta. Aqui podemos encontrar diversos materiais adequados às nossas necessidades, como modelos, componentes e planos de design totalmente personalizados, quer pela própria equipa da ferramenta, quer por outras pessoas. Também podemos publicar e descarregar objectos da comunidade, entre os quais se incluem ficheiros, plug-ins e widgets.

2.4.3 Plugins

Os plugins são aplicações instaláveis da comunidade que adicionam funcionalidades novas, que, por defeito, não existem no Figma. O Figma é um dos programas de design de interface e prototipagem mais completos do mercado, mas para determinadas situações, é comum sentirmos falta de uma função ou outra. Por tal motivo, as pessoas desenvolvem plugins que criam novas funcionalidades, ou melhoram e aceleram recursos já existentes.

A instalação é simples, pois só preciso procurar por algo na comunidade e clicar no botão de instalação. Uma das vantagens é que a instalação de plugins não está ligada ao computador do utilizador, mas sim à sua conta Figma. Isto significa que para além do computador que esteja a utilizar, não precisa voltar a instalar o plugin, pois estará ligado ao seu perfil - a não ser que o desinstale.

O Figma é um programa baseado na web, o que significa que pode aceder ao site, fazer login na sua conta e usar a ferramenta diretamente a partir do navegador da mesma maneira que usaria se tivesse o software instalado, o que torna a sua utilização muito mais prática.

2.4.4 Trabalhando com o Figma

Depois de investir muito tempo a estudar a ferramenta, a ler artigos e posts e a assistir a aulas no YouTube, com fontes referenciadas, passei à prática e chegámos a algumas opções de modelos que podem ser vistas nas figuras dez a treze.

Esses primeiros conjuntos de imagens (figura 12), mostra uma aplicação com a cor predominante em branco, mas com detalhes em verde e nos encontros do verde com o branco tem um pouco de ondulação.

Figura 12 – Desenho da aplicação (Figma1)

O que diferencia esse próximo conjunto da próxima imagem (figura 13) é que tem cortes mais retos, menos arredondados como na imagem anterior.

Figura 13 - Desenho da aplicação (Figma2)

Já nesse próximo conjunto mostrado na próxima imagem (figura 14) temos a predominância da cor verde e uma aplicação mais reta, sem ondulações.

Figura 14 - Desenho da aplicação (Figma3)

No conjunto da próxima imagem (figura 15) continuamos com predominância da cor verde e já não temos a caixa de texto, mas linhas onde as informações são colocadas e também uma aplicação mais reta, sem ondulações.

Figura 15 - Desenho da aplicação (Figma5)

Esse tom de cor verde que se vê nos designs é exatamente o mesmo usado no site do IPG. Inicialmente, o objetivo era seguir o padrão de cores e linhas já definido em relação ao que vemos no site do Instituto Politécnico da Guarda.

Como se pode ver nas imagens, a nossa aplicação terá quatro páginas. A primeira será a página de boas-vindas, que ficará ativa alguns segundos, enquanto a aplicação é carregada no dispositivo do utilizador. A segunda seria a página de login

onde o utilizador entra com o número de aluno que é recebido quando se efetua a matrícula, e no email informado na matrícula recebe uma senha para acessar a aplicação. A terceira página será a página principal da aplicação StudentCard, onde aparecerão os dados relativos ao estudante que estariam no seu cartão de estudante.

Nesse ecrã temos um botão com uma figura de QRcode que leva para a quarta página onde é gerado um código com informações do estudante. Anteriormente eu havia feito também uma página de registo, mas fui alertado por meu supervisor Engenheiro Carlos Fonseca que não teríamos essa página, pois o aluno receberia a senha para acesso por email, como já é feito hoje em relação ao acesso ao portal Moodle.

Assim terminei os estudos no Figma e chegou a hora de começar a praticar o desenvolvimento da aplicação que é o alvo deste projeto de estágio, mas é importante destacar que esses modelos criados são algo para que possamos visualizar onde queríamos chegar; ao longo do caminho poderíamos ter várias alterações (como aconteceu) e a nossa aplicação acabou ficando um pouco diferente, mas desenhar algo foi essencial para começarmos o desenvolvimento.

3. Desenvolvimento das Aplicações

3.1 Desenvolvimento da Aplicação StudentCard

A nossa aplicação principal StudentCard está divida em seis páginas: a página de capa, a de login, a página principal com as informações do aluno, a de eventos, a notícias e ainda uma página de QRcode. Neste capítulo vou explicar o desenvolvimento e as funcionalidades de cada página.

A seguir a figura 16 mostra em forma de diagrama como a nossa aplicação está organizada.

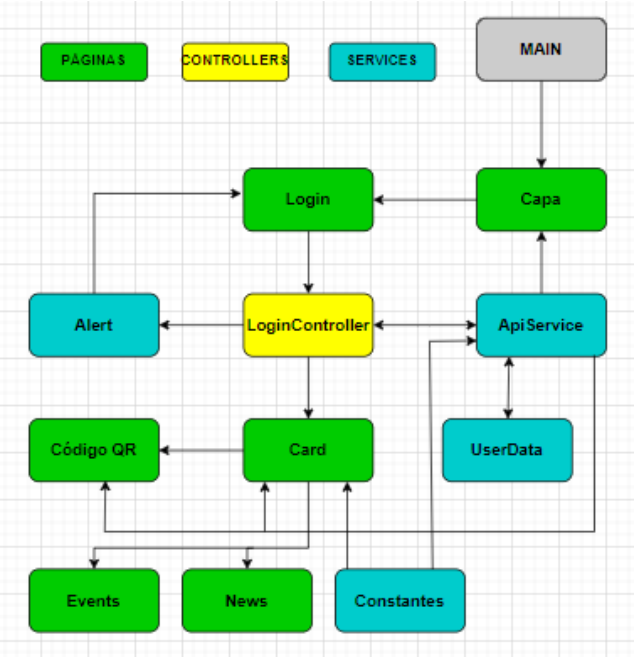

Figura 16-Diagrama da aplicação StudentCard

3.1.1 Main

A função MAIN define o ponto de entrada principal da aplicação, é nessa função que inicia a execução de toda aplicação, são configuradas as rotas de navegação, a interface gráfica é configurada e construída, são inicializadas as suas funcionalidades básicas, e é gerido o estado da aplicação.

Usamos na construção dessa função, no caso da nossa aplicação, uma propriedade chamada *routes*, ela define as rotas nomeadas para a aplicação e as mapeia para as páginas correspondentes. A propriedade-atributo *initialRoute* define a primeira

rota a ser exibida quando a aplicação é iniciada. Neste caso a primeira rota será a página da capa.

3.1.2 Páginas

Seguimos falando individualmente das seis páginas de nossa aplicação.

3.1.2.1 Capa

Para essa primeira página fizemos duas opções, uma com o logo do IPG, e outra com uma capa construída com o título da aplicação.

Nas duas páginas eu começo importando o pacote de material (Material.App) do flutter e o ficheiro de login_page.dart.; a função do pacote de material do flutter é disponibilizar os widgets que serão utilizados na criação da página, e o ficheiro login_page.dart está presente aqui porque ele será chamado após três segundos. A aplicação começa com uma capa, e após três segundos passa para a página de login, da qual vamos falar mais adiante.

Voltando para a página da capa. Para esta página foi criada uma classe chamada Capa_logo, quando a aplicação inicializa é uma instância desta classe que é chamada ao ecrã do telemóvel do utilizador. Há um método dessa classe que é executado initState() em que nele é definido um atraso de 3 segundos para que a rota de navegação da app siga para a página de login. Após a execução desse método, passados os 3 segundos, aparece no ecrã do dispositivo a página de login. Como corpo dessa classe temos um retorno de uma materialApp, com o conteúdo do widget principal envolvido em um *Scaffold*[3]. No corpo do *Scaffold* é definido um *GestureDetector* [4] para permitir a navegação de login quando tocar no ecrã. No corpo do *GestureDetector* há um widget *Image.asset* que contém a imagem do logo do IPG. Na outra opção de capa acontece a mesma coisa, exceto no nome da classe que é Capa_student, e no caso da imagem que ao invés de apresentar uma imagem (no caso anterior a logo do IPG), é colocado um widget do tipo *Text* com o seguinte texto 'Student Card'.

Em linguagem menos técnica, a capa aparece e pode receber um toque para navegar, ou após três segundos ocorre uma navegação automática para a página de login. Nas figuras 17, 18 e 19 temos as duas opções de páginas (17 e 18). Por fim, na imagem 19 é apresentada a página selecionada pelo supervisor Carlos, já com a nova cor que estamos a utilizar na aplicação.

[3] um scaffold é um widget no Flutter usado para implementar a estrutura básica de layout visual de material design. O material design é a linguagem de design da google, criada em 2014.
[4] O GestureDetector é um widget não visual usado principalmente para detectar o gesto do utilizador. Para identificar um gesto direcionado a um widget, o widget pode ser colocado dentro do widget GestureDetector o qual irá capturar o gesto e despachar vários eventos com base no gesto.

Figura 19 - CapaLS

Figura 17 - CapaLogo

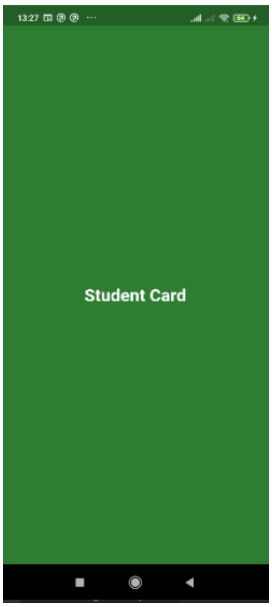

Figura 18 - CapaStudent

Uma coisa que já está muito clara neste momento é que o que esboçamos está a ser aperfeiçoado a partir de estudos, reuniões e uma melhoria de perspetiva ao ver novos exemplos. A seguir, na figura 20, apresento a árvore de widget das três opções de página para capa.

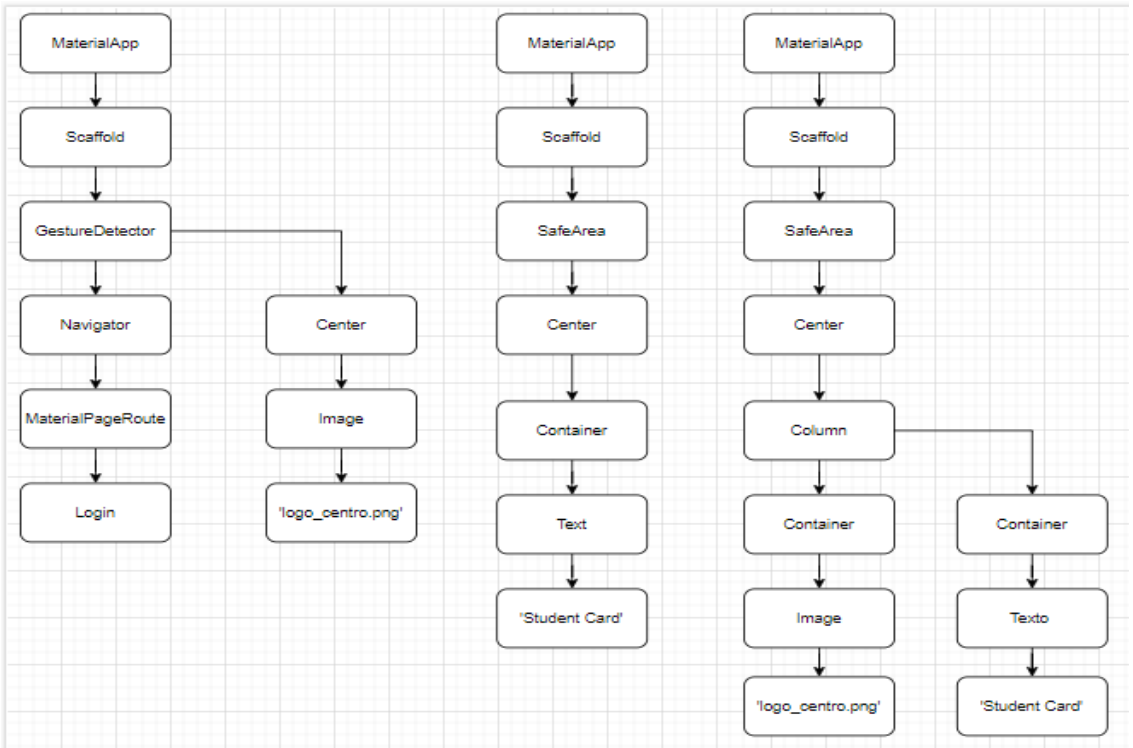

Figura 20 – Arvore de widgets das Capas

3.1.2.2 Login

Ao chegar na construção desta página, tinha vários aspectos para observar, estudar e planear para que tudo corresse bem.

O primeiro ponto era definir uma estrutura de autenticação. Antes de começar a codificar, era importante estabelecer a forma como a aplicação trataria a autenticação. Se usaria uma API de autenticação de terceiros, tal como a API do Google ou do Facebook, ou criávamos uma estrutura de autenticação. No entanto, após uma conversa com o professor e supervisor, o Engenheiro Carlos Fonseca, fiquei a saber que esta parte do back-end não faria parte do meu projeto, tendo o meu foco a parte de front-end, o que foi de certa forma um alívio, pois já havia bastante trabalho. O professor Carlos criou todo o backoffice do sistema de autenticação, e passou-me a incumbência de trabalhar como consumir uma API para autenticar e consumir os dados que seriam recebidos a partir do consumo desta API e seriam apresentados no cartão. Sobre isso vou falar mais adiante.

O segundo ponto era o ecrã de início de sessão. Uma vez que já estava definida uma estrutura de autenticação, eu precisaria criar um ecrã de início de sessão na minha aplicação. Esse ecrã deveria conter campos para o número de estudante e a password, além de um botão para conectar.

O terceiro ponto era cifrar a palavra-passe do utilizador. Quando utilizador introduzir a sua palavra-passe na aplicação, é importante que esta seja cifrada, isso protege a palavra-passe do utilizador caso ocorra uma violação de segurança. Mais adiante no projeto, decidimos além de cifrar a palavra-passe fazê-lo também para o número de estudante.

O quarto ponto era verificar as credenciais do utilizador. Quando o utilizador clicar no botão de login, a nossa aplicação deve verificar as credenciais do utilizador (número de aluno e senha) com as informações armazenadas na base de dados do IPG. Se as credenciais estiverem corretas, o utilizador deve ser redirecionado para a próxima página da aplicação. Caso contrário, o utilizador deve receber uma mensagem de erro informando que suas credenciais estão incorretas. Tomámos outra medida de utilizar o "widget SingleChildScrollView", que ajuda a questão de scroll quando o teclado sobe no ecrã, para não ter o problema de o teclado tomar o lugar dos campos que serão preenchidos. Como medida de segurança, também criei uma função para impedir que a palavra-passe do utilizador fique visível por um segundo após sua entrada. Há um botão para que o utilizador possa alterar essa visibilidade. Em seguida vou descrever um pouco mais sobre como ficou a página de login da nossa aplicação.

A página de login inclui um formulário com dois campos de entrada para o número de aluno e senha do utilizador. O formulário também inclui um botão para submeter, para iniciar o processo de login.

O widget importa várias dependências, como o Material.dart, que fornece vários widgets que são utilizados na aplicação. Também importa o Alerta.dart para exibir um alerta quando algo correr mal no processo de autenticação, e o login_controllers.dart para fazer chamadas à API.

Defini uma variável _formkey para acompanhar o estado do formulário e validá-lo. Estabeleceram-se também duas variáveis, númeroController e senhaController, que são usadas para controlar os campos relativos ao número do aluno e à respetiva senha, para garantir que a informação seja correctamente introduzida. Como medida de segurança, foi optado por um teclado numérico para o momento de digitar o número de aluno e um teclado com letras e números quando o cursor é posicionado no espaço para a introdução da senha.

Quando o botão de início de sessão é pressionado, a função LoginController é chamada. Falarei mais sobre esta função e de outras funções e ficheiros de serviços que lidam com criptografia e com todos os processos de início de sessão mais adiante, num subtópico dedicado a isso. Se tudo correr bem, o aluno é encaminhado para a página principal da aplicação; se algo estiver errado, recebe uma mensagem de aviso e a aplicação volta para a página de início de sessão.

Na figura 21 é possível ver o nosso ecrã de Login depois de um longo processo de desenvolvimento, enquanto a figura 22 mostra como ficou o ecrã até ao momento.

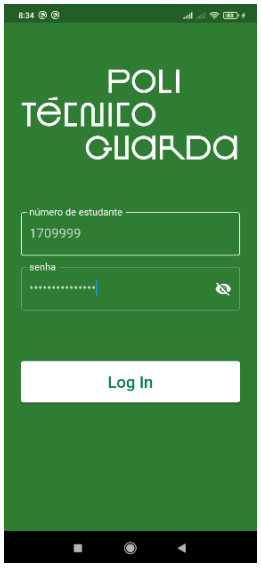

Figura 21 - LoginLogo

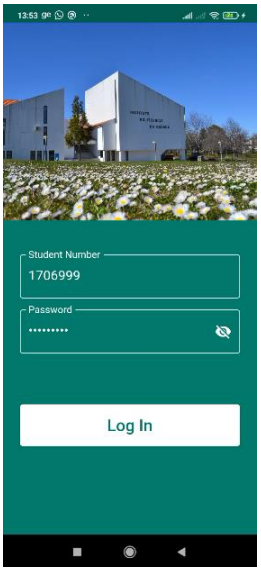

Figura 22 - LoginFoto

É importante referir que, quando um novo aluno se inscreve nas instalações do IPG, recebe o seu número de estudante e uma senha, que é enviada para o e-mail que é fornecido. Esta é a informação que o aluno precisa para aceder à aplicação StudentCard. Na Figura 23, será apresentada a árvore de widgets da página Login.

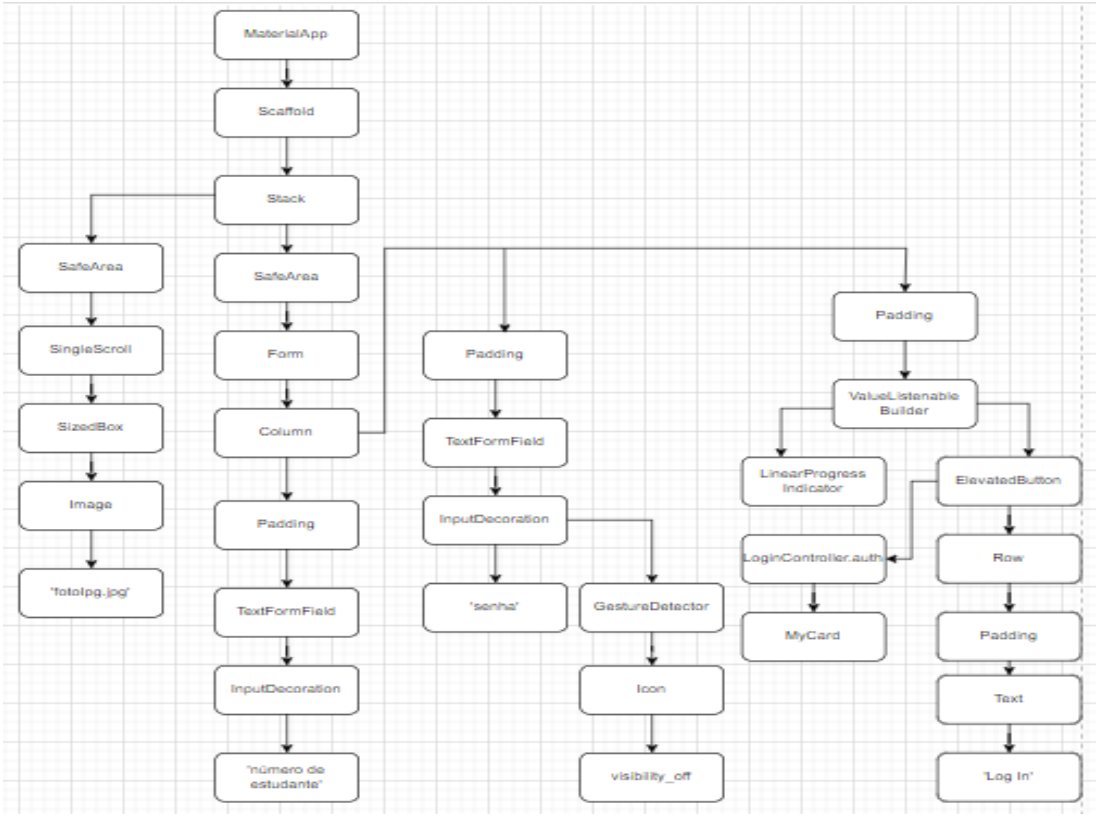

Figura 23 – Arvore de widgets do Login

3.1.2.3 Card

A página Card é a principal da nossa aplicação e, após o início de sessão, é para lá que o utilizador navega. Aqui são apresentadas as informações do estudante, dispondo de duas opções representadas nas figuras 24 e 25. A escolha recaiu sobre a figura 24, por isso foi a mais desenvolvida.

Figura 24 - CardPageNew

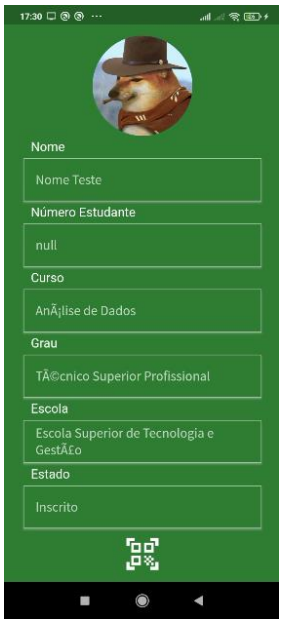

Figura 25 - CardPage

Assim como nas outras páginas, esta também é um widget e, para a construir, começámos por importar as bibliotecas e as classes do projeto que seriam necessárias. A classe principal é a MyCard, que foi herdada do StatefulWidget, o que significa que o estado atual pode ser alterado. Essa classe contém vários campos de texto que serão usados para armazenar informações sobre o utilizador, bem como um método assíncrono que retorna as informações do utilizador armazenadas quando, no momento do acesso, procedemos à procura de dados para autenticação, nos SharedPreferences.

No início temos o método initState(), que é invocado pelo flutter assim que o estado da classe é criado. Neste método, é chamado o método cardPage() que procura as informações do utilizador nos SharedPreferences e, posteriormente, é invocado o método setState() para atualizar o estado da classe com as informações do utilizador retornadas do cardPage().

Finalmente temos a construção do widget que é retornado no ecrã do utilizador, onde é mostrado um novo widget MaterialApp, que é o widget principal de uma aplicação Flutter. Esse widget é responsável por criar uma estrutura básica da aplicação, incluindo a barra de título, gerenciamento de rotas e temas; dentro desse temos um scaffold que é um widget que fornece uma estrutura básica para um ecrã, é um esqueleto para construir os ecrãs, como corpo desse widget temos um stack, esse é um widget muito útil que permite sobrepor vários filhos de maneira simples, por exemplo, tendo algum texto sobre uma imagem.

O que temos na parte de cima do ecrã é uma imagem que foi colocada dentro de um widget chamado circleAvatar, o espaço destinado à foto do aluno. O professor Carlos desenvolveu uma API que nos dá a possibilidade de aceder à imagem do aluno no site

sigarra, e coloca-la na aplicação. Todos os alunos matriculados e com acesso ao site sigarra podem enviar uma fotografia com as especificações solicitadas para ser colocada no seu perfil de aluno.

Após a imagem temos duas caixas de texto separadas por um widget divider, que são espaços onde se devem inserir o nome da instituição e a identificação da aplicação. Logo abaixo, temos uma linha com dois elementos, sendo o primeiro a data e hora que são atualizadas a cada segundo. Esta caixa com a data e a hora atualizadas a cada segundo, é uma medida de segurança. Um ex-aluno não conseguiria pensar em 'tirar uma captura de ecrã' e mesmo depois de ter concluído o seu tempo de estudante na instituição, tentar continuar a utilizar o cartão para se identificar como estudante, o que deixa de ser possível devido a essa funcionalidade.

O segundo elemento da linha é um link para o Loupe, um site onde os alunos podem ver o cardápio e fazer a sua reserva de refeições. Seguidamente, temos as caixas de texto com as informações do utilizador (nome, número de estudante, ano do curso, estado da matrícula, nome do curso, grau do curso e nome da escola). Por baixo do texto temos a imagem do brasão do IPG, porém com uma grande transparência que não prejudica a visão dos dados do utilizador.

A respeito das informações do utilizador, como referido anteriormente, ao fazer uma requisição https, tendo os endpoints sido criado pelo responsável do back-end da aplicação, a aplicação recebe os dados do aluno em um ficheiro do tipo JSON. Na imagem a seguir (Figura 26) temos um modelo de um arquivo JSON que é retornado após requisição de dados pela aplicação, no caso requisição de dados do aluno.


```
[{'result':'1',
'curso':'Análise de Dados',
'escola':'Escola Superior de Tecnologia e Gestão',
'estado':'Inscrito',
'grau':'Tesp',
'nome':'Ricardo',
'numero':'1706995',
'linkFoto':'cloud.sysnovare.pt/ipg/FOTOGRAFIAS_SERVICE.foto?pct_cod=',
'AnoLetivoEstado':'22/23'}]
```


Figura 26 - Dados do utilizador como é recebida pela aplicação, tipo JSON

Por fim, temos quatro ícones na parte inferior da aplicação. O primeiro é de Código QR, esta página mostra um Código QR gerado com dados do aluno. O segundo é de uma página de Eventos, esta página mostra os eventos presentes no site do IPG. O terceiro é de uma página de Notícias, esta página mostra as notícias presentes no site do IPG. Quando um destes três ícones é clicado, é gerado um navegador para outra página de widget, usando a classe MaterialPageRoute, farei mais menção a estas páginas num

capítulo específico de cada uma delas. Por fim, temos ainda um quarto ícone que é o de Terminar Sessão. Como referi acima, no momento do login, as informações do aluno são guardadas no SharedPreferences. Com essas informações guardadas, o aluno pode ficar ligado à aplicação, podendo fechar e, ao voltar, não necessitar de fazer o login novamente. Mas, ao clicar neste ícone, as variáveis contendo as informações serão limpas e o utilizador será direcionado para a página de login.

A seguir apresento nas figuras 27, 28, 29, 30, e 31 a árvore de widgets da página card. A árvore foi organizada em partes para que fosse possível apresentar no relatório com boa visibilidade. A partir do segundo bloco, os widgets serão visualizados com partida da coluna (Column), que está colorida de azul para facilitar a identificação.

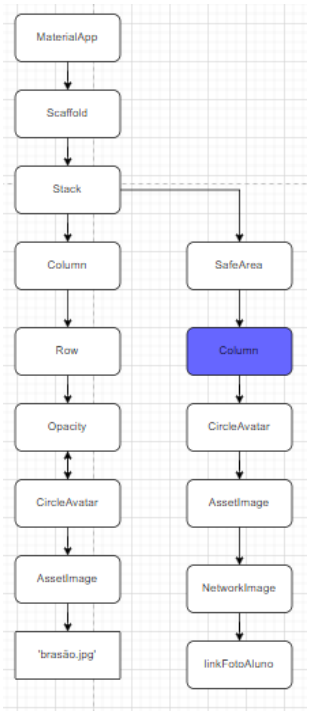

Figura 27 – Arvore de widgets de Card Bloco 1

Figura 28 - Arvore de widgets do Card Bloco 2

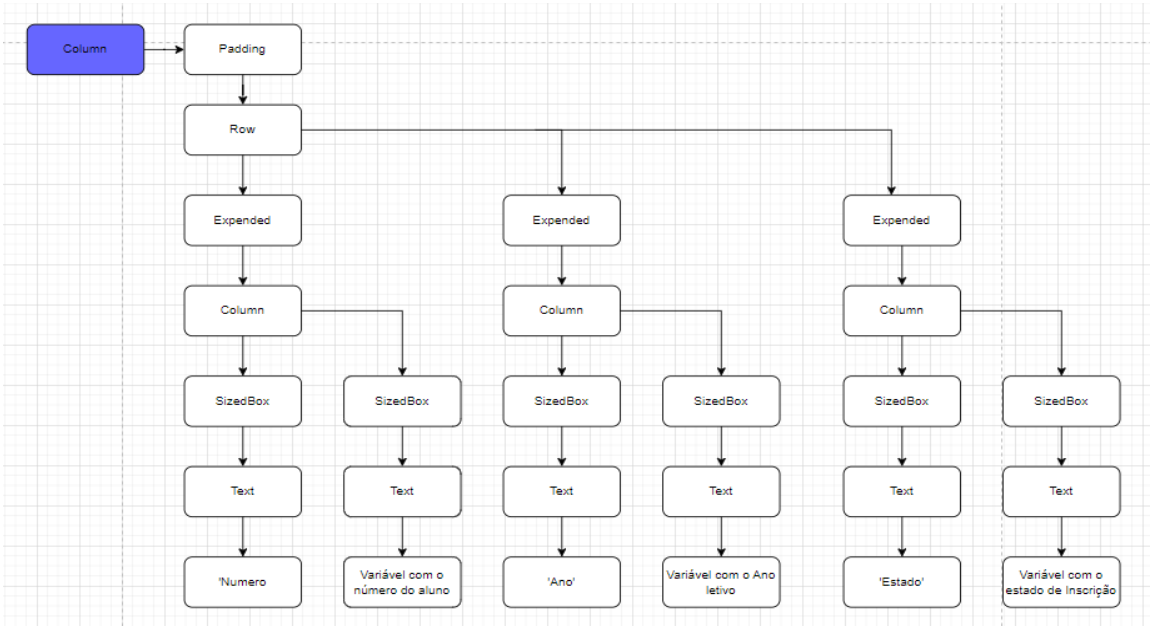

Figura 29 - Arvore de widgets de Card Bloco 3

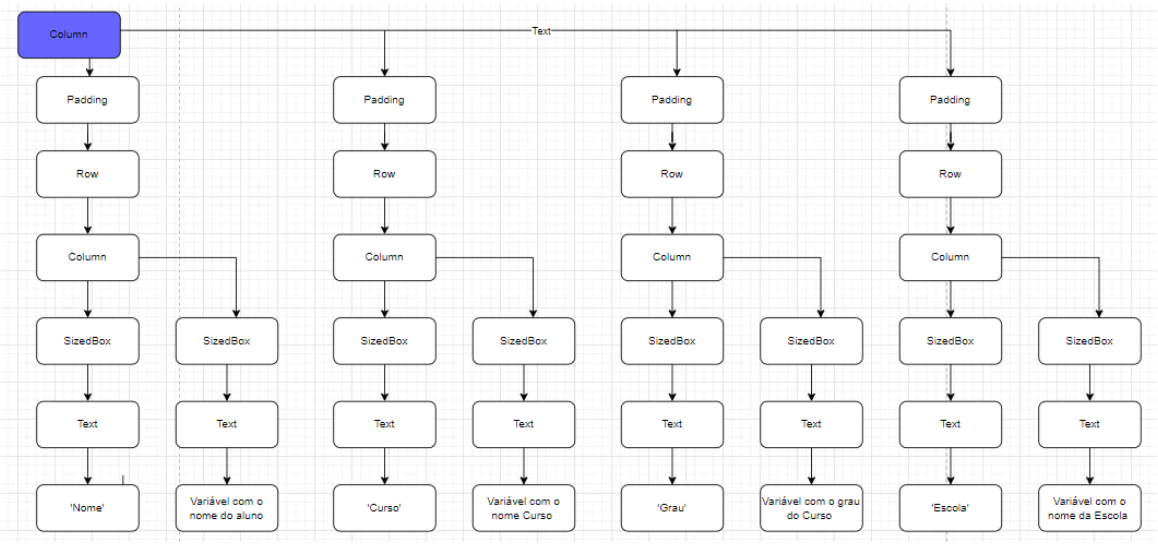

Figura 30 – Arvore de widgets de Card Bloco 4

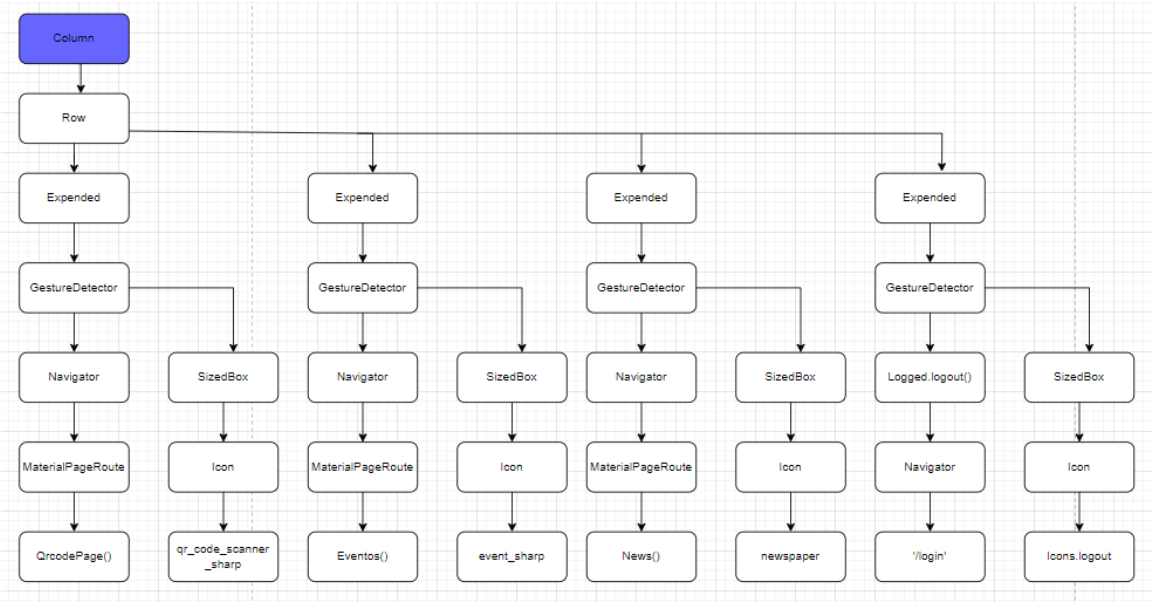

Figura 31 - Arvore de widgets de Card Bloco 1

3.1.2.4 QRcode

Os QR Codes estão a tornar-se cada vez mais presentes no nosso dia-a-dia e são frequentemente usados como um meio de comunicação na vida pessoal, empresarial e académica. Várias empresas de venda no retalho, fabricação, farmácia, distribuição e serviços, entre outras, estão a adotar o uso de QR Codes nas suas operações diárias para

ajudar os utilizadores a terem acesso aos seus sites ou até mesmo a fazer pagamentos por serviços.

O que é um QR code?

Código QR é a abreviatura para "Quick Response Code" (Código de Resposta Rápida). É um código bidimensional, formado por uma imagem quadrada com códigos e formas codificadas, que armazena todo o tipo de informações, desde texto, imagens, números de telefone, e-mails ou hiperligações. Está relacionado com um código de barras, mas assegurando muito mais informação, tanto quanto quantidade como a variedade. Por essa razão, o QR code (ou código QR) é considerado uma evolução relativamente ao código de barras, uma vez que tem a capacidade de armazenar até 100 vezes mais dados.

Este código pode ser lido por qualquer telemóvel, desde que possua uma câmara e uma aplicação que faça a leitura. Atualmente, os smartphones mais avançados não requerem uma aplicação para esse efeito; possuem um leitor próprio integrado na câmara que lê o código de forma instantânea. Quando a leitura é efetuada, o conteúdo armazenado no QR code é exibido no ecrã do smartphone, sendo possível estabelecer uma chamada telefónica direta, abrir uma página web específica, consultar texto e imagens, enviar um email ou preencher um formulário.

Para que serve um código QR?

Algumas das aplicações mais comuns do código QR são, por exemplo, os menus dos restaurantes. Graças a um código QR, o cliente tem acesso ao menu através do seu telemóvel. É também comum encontrar QR codes em jornais e revistas, para que os leitores acedam facilmente a conteúdos adicionais relacionados com o assunto. Nos cartões de visita, facilitam o acesso às informações de contacto, permitindo até que fiquem instantaneamente guardadas no telemóvel.

No Instituto Politécnico da Guarda temos muitos códigos QR dispostos com organização em vários locais e, ao fazer o acesso a eles, conseguimos verificar agendas de aulas e de avaliações ou informações acerca de profissionais e de suas funções, bem como servirem para a criação de convites para eventos, através do agregamento de dados sobre o programa, palestrantes e local, ou mesmo para exposições culturais. Por exemplo, pode-se colocar um QR code como legenda numa obra de arte para que os visitantes possam aceder a mais informação sobre aquela obra e sobre os autores. As possibilidades de aplicação destes códigos são muitas.

Para gerar um código QR, é necessário definir a informação que se pretende armazenar no código QR, seja texto, imagens, ligações, documentos, vídeos ou músicas.

A nossa aplicação StudentCard vai disponibilizar a possibilidade de gerar um código QR, inicialmente contendo o nome e número do estudante, o curso, e a data e hora. Isto pode servir para identificação do aluno de forma eletrónica, permitindo o fim

das listas de chamada assinadas em sala de aula. A ideia é tornar o processo totalmente eletrónico.

Como se pode ver na figura 32, esta é a nossa classe mais simples. Quando o utilizador, na página Card, clica no ícone do qrcode, é encaminhado para esta página com um widget que gera um código QR com os dados fornecidos. Na parte superior, temos o nome do aluno e no corpo do widget temos uma variável denominada 'data' que recebe o número do aluno, que é a informação usada para gerar o qrcode.

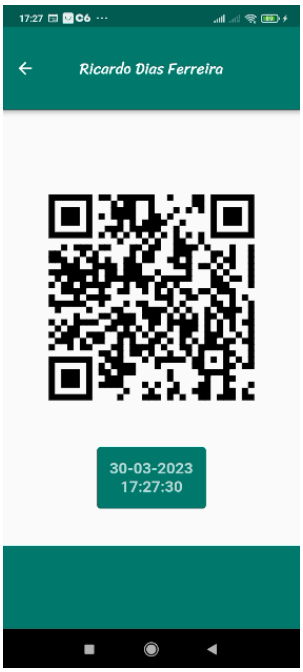

Figura 32 – Qrcode no ecrã

A seguir apresento a árvore de widgets da página Qrcode na figura 33.

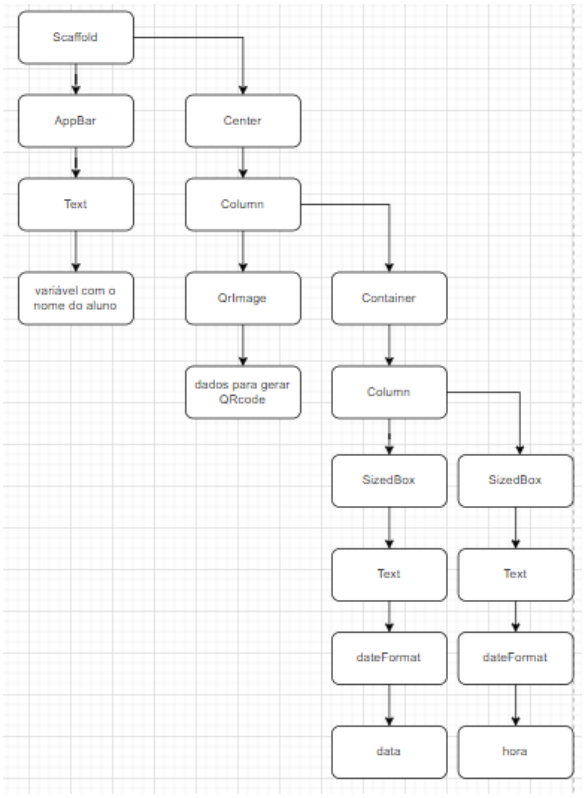

Figura 33 – arvore de widget do Qrcode

3.1.2.5 Events

Esta página mostra os eventos relacionados com o IPG que se encontram agendados, incluindo as suas datas e horas de início e de fim, assim como uma imagem e um link que, quando pressionado, leva o utilizador para a página do evento em causa, onde lhe são dadas todas as informações necessárias. O meu supervisor, Carlos Fonseca, responsável pela parte de back-end do projeto, criou uma API (https://portal.ipg.pt/webapi/XXXXXXXXXXXXXX) que transporta os dados do site do IPG na parte referente aos eventos. Os dados fornecidos pela API são consumidos e apresentados na página de Eventos. Segue-se uma descrição da mesma.

Começamos a página como de costume importando as bibliotecas que utilizaremos, foi criada a classe Eventos que herda de StatefulWidget, por isso, pode sofrer alteração de estado. Criamos um método assíncrono que utiliza o pacote de url_launcher para abrir um link externo. Em seguida continuamos com a criação do corpo da página, essa página tem um AppBar com o título da página, recebe a mesma cor de fundo que estamos usando em toda aplicação, e em seguida usamos um FutureBuilder do tipo List, pois vamos mostrar uma lista de eventos. Chamamos em seguida a função Api.getEvents(), essa função tem a responsabilidade de fazer a requisição http para trazer os dados da página mencionada acima para a nossa aplicação. De seguida fazemos uma verificação se houve algum erro da API ao obter os dados,

caso não exista exibimos esses dados em um ListView. Fazemos a exibição de um indicador de progresso enquanto a chamada da API é processada. Após o processamento, retornamos um ListTile com os dados de cada evento:

-Na posição leading temos um CircleAvatar que é onde mostramos a imagem, é o que fica mais a esquerda do ecrã do utilizador

-Na posição Title mostramos o nome do evento

-Na posição subtitle mostramos a data e o horário do evento

-E por último na posição trailing colocamos o link para acesso do portal do IPG, essa parte fica mais a direita do ecrã do utilizador.

A seguir mostro na figura 34 como ficou a página de eventos como uma lista de vizualizações e ainda temos uma segunda opção de apresentação dessa página, onde pegamos cada valor que recebemos da api, carregamo-las em variáveis e a colocamos no ecrã de forma bem organizada como pode ser observada na figura 35.

Figura 35 – Event

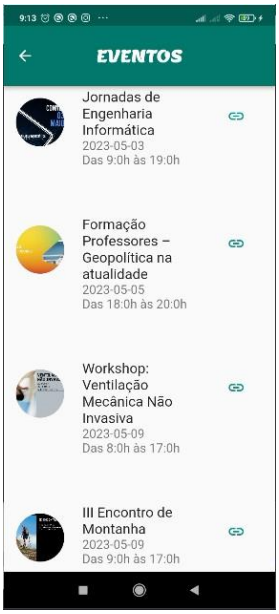

Figura 34 – EventLidt

Termino este capítulo com a árvore de widget da página, como se mostra na Figura 36.

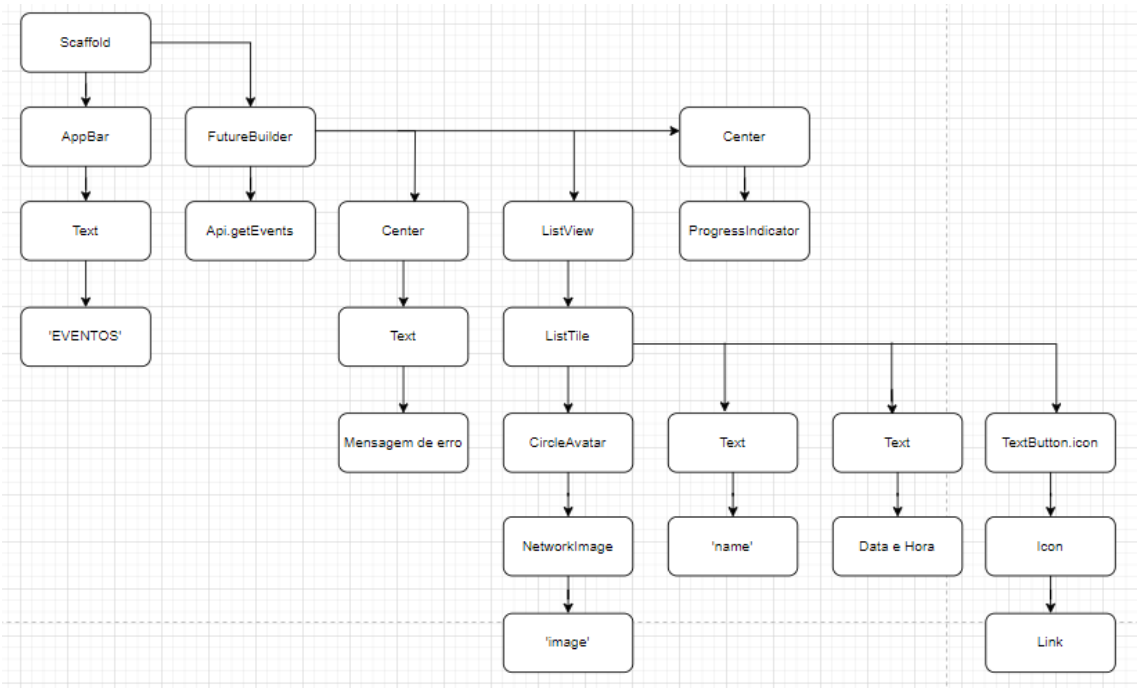

Figura 36 - Arvore de widgets da página Eventos

3.1.2.6 News

Esta página é muito similar à anterior, a diferença é que, em vez de mostrar os eventos, ela mostra as notícias do IPG, publicadas no portal do IPG, que estão por ocorrer, com as suas datas, juntamente com uma imagem da notícia e um link ao qual se pode clicar para ser direcionado para a respetiva página da notícia, na qual estarão presentes as informações completas. Carlos Fonseca, supervisor do projeto, criou uma API que traz os dados do site (https://portal.ipg.pt/webapi/XXXXXXXXXXXXX), que pertence ao IPG na parte que mostra as notícias. Consumimos os dados resultantes desta API e apresentamo-los na página News. A seguir faço uma descrição da página.

Começamos a página como de costume importando as bibliotecas que utilizaremos, criamos a classe Eventos que herda de StatefulWidget, por isso, pode sofrer alteração de estado. Criamos um método assíncrono que utiliza o pacote de url_launcher para abrir um link externo. Em seguida continuamos com a criação do corpo da página, essa página tem um AppBar com o título da página, recebe a mesma cor de fundo que estamos usando em toda aplicação, e em seguida usamos um FutureBuilder do tipo List, pois vamos mostrar uma lista de eventos. Chamamos em seguida a função Api.getNews(), essa função tem a responsabilidade de fazer a requisição http para trazer os dados da página mencionada acima para nossa aplicação. Após isso fazemos uma verificação se houve algum erro da API ao obter os dados. Se houver algum erro exibe-se uma mensagem de erro, caso contrário exibimos esses dados em um ListView. Fazemos a exibição de um indicador de progresso enquanto a

chamada da API é processada. Após o processamento, retornamos um ListTile com os dados de cada evento:

-Na posição leading temos um CircleAvatar que é onde mostramos a imagem, é o que fica mais a esquerda do ecrã do utilizador

-Na posição Title mostramos a data da notícia

-Na posição subtitle mostramos o título da notíca

-E por último na posição trailing colocamos o link para acesso do portal do IPG, essa parte fica mais a direita do ecrã do utilizador.

Na imagem 38 podemos ver a página de notícias como uma lista de visualizações, e na figura 37 temos uma segunda opção de apresentação dessa página, onde todos os valores recebidos da API são colocados em variáveis, e a informação é mostrada no ecrã de forma organizada.

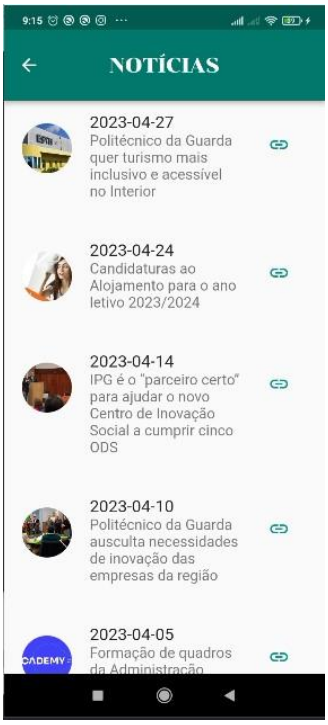

Figura 38 - noticiasList

Figura 37 - noticias

Encerro esse capítulo com o diagrama de árvore dessa página, figura 39.

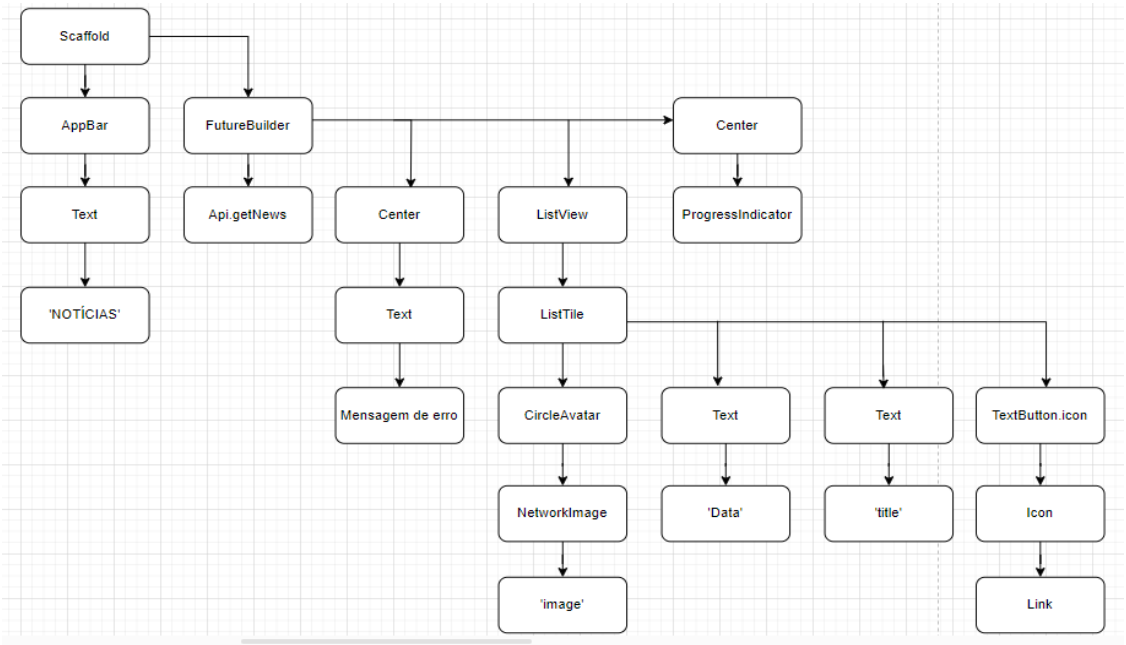

Figura 39 - Arvore de widgets da página News

3.1.3 Controller

A respeito dos controladores de nossa aplicação, temos apenas o controle de login.

3.1.3.1 Login Controller

Neste ficheiro temos a classe LoginController que é uma das mais importantes da nossa aplicação, pois aqui, como o nome indica, controlamos o acesso do estudante.

No início temos a declaração de uma variável do tipo booleano, que gere o carregamento enquanto a aplicação se encontra a validar o acesso do estudante. Seguidamente, declaramos duas variáveis que vão receber o número de estudante e a sua senha. Seguidamente temos a criação de uma função futura (denominada auth), é esta que é chamada quando o utilizador carrega no botão de início de sessão, depois de inserir o número de estudante e senha, os quais são enviados como parâmetros na chamada desta função. A função auth chama outra função, da classe LoginApi, chamada login. Se os dados do aluno são confirmados, o retorno da função é verdadeiro e é chamada uma função chamada _navegaCardPage. Esta novamente como o seu nome sugere, faz a navegação da aplicação para a página Card. Se os dados do aluno não são confirmados, então a função chamada é a Alert, esta faz a navegação do utilizador de volta para a página de login.

3.1.4 Services

Nossa aplicação tem quatro ficheiros de serviços.

3.1.4.1 ApiService

Nesse ficheiro temos a classe LoginApi e esta tem três funções: login, getPublickey e fetchData.

Esta classe tem duas variáveis que irão receber os parâmetros de número de aluno e senha. Na classe que anteriormente mencionámos, LoginController, temos a função auth que chama a função login e envia os dados do aluno, esta, por sua vez, recebe os dados e guarda-os nas variáveis criadas na classe; estas variáveis são globais, ou seja, podem ser usadas em todas as funções da classe. Além disso, a função login chama a função fetchData, esta é a função que irá buscar os dados do aluno. Ela começa chamando a função getPublicKey, esta faz a primeira requisição HTTP, a fim de obter a chave pública; se tudo correr bem, ela recebe um ficheiro do tipo JSON, decodifica-o e salva a string em uma variável chamada publiKey e a retorna.

Continuando com a função fetchData, depois de receber a chave pública, ela é usada para cifrar a password e o número de aluno do estudante. Em seguida uma nova solicitação http é feita, desta vez enviando o número do aluno e a password cifrada em base 64, de forma a poder ser enviada através da web. Quando a mensagem é recebida, a password é decifrada com a chave privada e é feita a verificação dos dados enviados. Novamente na resposta virá um ficheiro do tipo json, neste ficheiro existirá uma chave com o nome 'result', se o valor desta chave for '0', significa que os dados estão incorretos e a autenticação não é feita, mas se o valor for '1', significa que os dados estão corretos e esse ficheiro json terá outras chaves e valores que são os dados do aluno. De seguida esse ficheiro é decodificado e os dados são enviados como retorno. A função login recebe os dados e verifica a chave ‘result’, sendo igual a ‘1’ retorna os dados, sendo igual a ‘0’ retorna null.

Criptografia

Estou a utilizar a biblioteca encrypt para criptografia. Esta é uma biblioteca Dart para criptografar e descodificar dados. Para os seus algoritmos criptográficos, a biblioteca fornece suporte para AES, RSA e mais. Em geral, a encrypt é uma biblioteca muito útil para trabalhar com criptografia através de Dart, oferecendo aplicações variadas para que possa garantir a segurança dos dados.

Segundo o site NovaGeo Solutions, 2022 [3], O RSA é um dos primeiros sistemas de criptografia de chave pública e é amplamente utilizado para a transmissão segura de dados. Neste sistema de criptografia, a chave de encriptação é de acesso

público, sendo diferente da chave de decriptação, que é de acesso privado. O algoritmo RSA foi descrito no final da década de 70 e o acrónimo RSA é composto pelas iniciais dos sobrenomes dos seus criadores Ron Rivest, Adi Shamir e Leonard Adleman.

Este tipo de algoritmo é chamado de criptografia assimétrica ou de chave pública, pois existem duas chaves que são utilizadas. A primeira chave é a chave pública, usada para cifrar ou encriptar os dados que são enviados, por exemplo: Logins e senhas. A segunda chave é responsável por desencriptar a informação, é conhecida como chave privada, somente ela consegue converter o texto cifrado em texto legível. Esta estrutura, que usa duas chaves, difere da criptografia simétrica, na qual é utilizada apenas uma para cifrar e decifrar o texto. Esta forma torna-se inadequada para a internet, já que a chave teria de ser enviada com a mensagem cifrada, com o risco de ser detectada por um atacante, permitindo-lhe ler os dados enviados.

Base64

O método base64 é utilizado para converter dados em bytes para uma representação em texto que pode ser facilmente transmitida ou armazenada em diferentes sistemas. O formato resultante é composto apenas por caracteres ASCII e consiste em um conjunto de 64 caracteres diferentes, incluindo letras maiúsculas e minúsculas, números e símbolos. É uma forma comum de codificação de dados em sistemas de rede e na Web. A conversão para base64 não altera os dados originais, apenas os codifica para uma representação em texto. A conversão reversa também pode ser feita usando o método base64Decode.

Shared_Preferences

O recurso SharedPreferences é usado para armazenar pequenas quantidades de dados como dados primitivos em formato chave-valor no Android e iOS. o armazenamento persistente específico da plataforma é para dados simples (NSUserDefaults no iOS e macOS, SharedPreferences no Android, etc.).

Esses dados geralmente estão associados à aplicação; portanto, quando o utilizador desinstala a aplicação, os dados também são eliminados. Assim, em vez de usar uma base de dados para armazenar pequenas quantidades de informação, como nome de utilizador, variáveis int, double, bool e string, podemos usar o SharedPreferences.

Na nossa aplicação, estamos a usar este plug-in para guardar os dados após o consumo da api. Esses dados serão mostrados em outras páginas da nossa aplicação.

Para apagar os dados armazenados, podemos utilizar o método remove(), passando como parâmetro a chave do valor que se pretende apagar, ou o método clear(), o qual elimina todos os dados que estão guardados.

3.1.4.2 UserData

O nosso objetivo nessa classe é desserializar o json que recebemos da requisição http que fazemos na função fetchData.

JSON significa Java Script Object Notation e é um formato para intercâmbio de dados aberto e baseado em texto fácil de ler e de escrever. A linguagem Dart oferece suporte para analisar ficheiros JSON, e, usando a biblioteca convert.dart para converter o Json (se for o json válido) em um Map com chaves de string e objetos dinâmicos. Na linguagem Dart, a classe Map é uma coleção de pares chave/valor, a partir dos quais é possível recuperar um valor usando sua chave associada.

3.1.4.3 Constants

É comum e recomendável que num projeto Flutter haja um ficheiro com constantes para guardar partes de código que são utilizados em várias partes do projeto. Esta é a função desse ficheiro, onde armazeno algumas constantes.

É uma convenção no flutter que todas as constantes deveriam começar com a letra ‘k’, também estou seguindo essa convenção nesse projeto.

3.1.4.4 Alert

Neste ficheiro temos a função *Alert* que como seu nome sugere, é responsável por gerar um alerta. Essa função é chamada pela função *LoginController* se algo ocorrer errado na autenticação. Nesse caso ela retorna uma caixa de diálogo com uma mensagem informando que houve um erro no login e dando como opção um botão no qual está escrito 'OK'. Caso o botão 'OK' seja clicado, o utilizador regressa à página de login, podendo então verificar o que foi digitado incorretamente.

3.2 Desenvolvimento da Aplicação IpgService

A nossa aplicação StudentCard conta com tecnologia para geração de um QRcode, com o nome e número do estudante, o curso, e a data e hora como informação do código. A princípio o alvo era a possibilidade de acabar com as listas de chamada assinadas em sala de aula, esse processo seria eletrónico, seria então desenvolvido uma

aplicação para os professores com a qual poderiam marcar a presença do aluno lendo o código gerado a partir da da aplicação StudentCard do aluno, isso vai ficar para o futuro, o que fizemos para já foi uma outra aplicação que pode ser utilizada nos seminários e eventos do instituto, chamamos essa aplicação de IpgService.

A aplicação IpgService conta com tecnologia para leitura de QRcode, as informações são enviadas e armazenadas na base de dados do Instituto Politécnico da Guarda, dessa forma pode ser utilizada para fazer registo de presença dos alunos em eventos (exemplo, seminários), e é possível gerar um pequeno relatório com gráficos e informações como o número de participantes total no evento, a média de participação por turma, qual a turma com maior e menor número de participantes.

Nessa parte do estágio trabalhámos bastante com solicitações http, encriptação, consumo de api e geração de diferentes gráficos com bibliotecas que importamos para trabalhar no flutter. A seguir descrevo as funções e o que acontece em cada parte da aplicação. A seguir a imagem 40 mostra em forma de diagrama como a nossa aplicação está organizada.

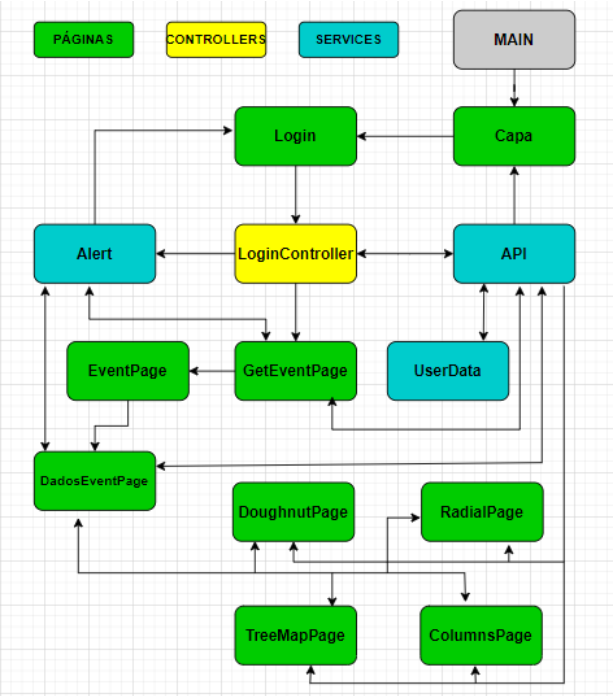

Figura 40 – Diagrama da aplicação IpgService

3.2.1 Main

A função MAIN dessa aplicação é idêntica a da aplicação anterior e define o ponto de entrada principal da aplicação, chama a execução de toda aplicação, configura as rotas de navegação e gerência o estado da mesma.

Toda aplicação flutter tem uma função principal. Cada aplicação deve ter uma função *main*, que serve como ponto de entrada para a aplicação, é através dela que a aplicação é inicializado e a interface gráfica é construída no ecrã do dispositivo.

Usamos na construção dessa função, no caso da nossa aplicação, uma propriedade chamada *routes*, ela define as rotas nomeadas para a aplicação e as mapeia para as páginas correspondentes. A *initialRoute* propriedade define a primeira rota a ser exibida quando a aplicação é iniciada, nesse caso a primeira rota será a página da capa.

A função *main* é extremamente importante numa aplicação Flutter, pois é através dela que se dá a inicialização da aplicação e a configuração inicial da interface gráfica e das funcionalidades básicas da mesma.

3.2.2 Páginas da aplicação IpgService

Essa aplicação conta com 9 páginas, vamos falar sobre cada uma delas a seguir.

3.2.2.1 CapaService

Faço a seguir a descrição da constituição e funcionamento dos componentes desta página. Para essa primeira página temos também algo bem parecido com a aplicação anterior. Eu começo importando o pacote de material do flutter e o ficheiro de login_page.dart.; a função do pacote de material do flutter é trazer os widgets que serão utilizados na criação da página, e o ficheiro login_page.dart está presente aqui porque ele será chamado após três segundos. A aplicação começa com uma capa, e após três segundos passa para a página de login, da qual vamos falar mais adiante.

Voltando para a página da capa temos a definição de uma classe chamada CapaService, quando a aplicação inicializa é o que primeiro aparece no telemóvel do utilizador. Temos um método *initState()* em que nele é definido um atraso de 3 segundos para apertar a rota para a página de login. Como corpo dessa classe temo um retorno de uma *materialApp*, com o conteúdo do widget principal envolvido em um *Scaffold*. No corpo do *Scaffold* é colocado um widget do tipo *image* onde há a logo do IPG e na parte de baixo um outro widget do tipo *Text* com o texto “IpgService”. É possível visualizar como ficou a mesma na figura 41.

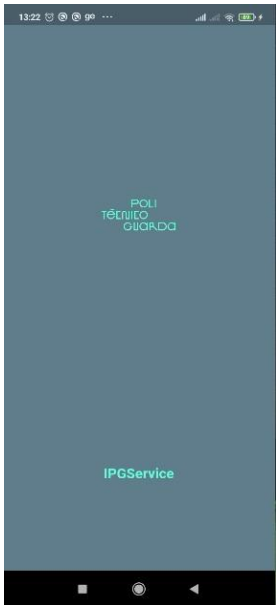

Figura 41 - CapaIpgService

3.2.2.2 Login

A página de login inclui um formulário com dois campos de entrada para as credenciais de utilizador e senha. O formulário também inclui um botão de envio para iniciar o processo de login. Faço a seguir a descrição da constituição e funcionamento dos componentes desta página.

O widget importa duas dependências, como *Material.dart* que fornece vários widgets que são utilizados na aplicação, também o *login_controllers.dart* para fazer chamadas de API.

Defini uma variável *_formkey* que é usada para acompanhar o estado do formulário e validar o formulário. Outras duas variáveis são *utilizadorController* e *senhaController*, elas são usadas para controlar os campos de entrada para os dados de utilizador e senha, respetivamente.

Quando o botão login é pressionado, é chamada a função *LoginController e* caso tudo corra bem, ou seja, o processo de autenticação proceda com sucesso, o aluno é enviado para a página principal da aplicação, caso o processo de autenticação der erro, recebe uma mensagem de alerta e a aplicação retorna para a página de login.

A seguir temos a figura 42 onde é possível conferir como ficou o nosso ecrã de login.

Figura 42 - LoginIpgService

3.2.2.3 GetEventPage

Quando tudo ocorre bem na página de login, a aplicação vem para esta página, a página GetEvent (figura 43). O alvo dessa página é verificar os eventos que o utilizador logado tem acesso.

Faço a seguir a descrição da constituição e funcionamento dos componentes desta página. Começamos com as importações de pacotes e ficheiros necessários para o funcionamento do código. O pacote *flutter/material.dart* é importado para utilizar os widgets do Flutter e construir a interface gráfica. Os ficheiros *alert.dart* e *api.dart* são importados a partir de diretórios superiores e contêm a implementação de funções relacionadas a alertas e ao consumo de API, respetivamente.

Em seguida definimos uma classe chamada *GetEventPage* que herda do widget *StatefulWidget*. Esse widget representa uma página que possui um estado mutável.

Uma variável booleana chamada *heveEvent* é declarada como nula (bool?). Ela será usada para armazenar o resultado retornado pela função *Api.getEvent*(), que indica se há um evento disponível ou não.

Como próximo passo criamos um método assíncrono chamado *whichEvent*, que recebe um parâmetro *context*. Ele faz uma chamada assíncrona para *Api.getEvent*() e armazena o resultado em *heveEvent*. Se *heveEvent* for falso, é exibido um alerta utilizando a função *AlertNotEvent* definida no ficheiro *alert.dart*. Caso contrário, é feita uma navegação para a página de Eventos utilizando o *Navigator* do Flutter.

Por último temos o método *build*, que constrói a interface da página. É retornado um widget *Scaffold* que fornece uma estrutura básica para a página. O conteúdo da página é definido dentro do widget *SafeArea*, que garante que o conteúdo seja exibido em uma área segura, evitando a sobreposição com elementos como a barra de status do dispositivo. Dentro do *SafeArea*, há um *Column* que organiza os elementos verticalmente no centro da página.

De forma resumida esta página apresenta um botão onde está escrito 'Buscar eventos', como de pode visualizar na figura 40 ao clicar no mesmo, se houver eventos para o utilizador ligado à aplicação, ocorre a navegação para a página de eventos.

Figura 43 - GetEvent

3.2.2.4 EventPage

Faço a seguir a descrição da constituição e funcionamento dos componentes desta página. Começamos com a importação do pacote *flutter/material.dart*, que contém os widgets do Flutter e é necessário para construir a interface gráfica.

Em seguia definimos uma classe chamada EventPage que herda do widget *StatefulWidget*. Esse widget representa uma página que possui um estado mutável.

De seguida temos o método *build*, responsável por construir a interface da página. É retornado um widget Scaffold que fornece uma estrutura básica para a página. O conteúdo da página é definido dentro do widget SafeArea, que garante que o conteúdo seja exibido em uma área segura, evitando a sobreposição com elementos como a barra de status do dispositivo.

Dentro do SafeArea, há um *SingleChildScrollView* que permite rolar o ecrã caso o conteúdo seja maior do que a área visível. Ele tem a propriedade reverse definida como *true*, o que faz com que o conteúdo role para cima. Dentro do

SingleChildScrollView, há um *SizedBox* que ocupa toda a largura do ecrã. Dentro desse *SizedBox*, há um *Column* que organiza os elementos verticalmente e centralizados no eixo principal (verticalmente). Dentro do *Column*, há vários *ElevatedButton.icon*, que são botões elevados com um ícone e um rótulo de texto. Cada botão tem um ícone representado pelo widget *Icon*, e o rótulo de texto é definido pelo widget *Text*. Os botões têm uma função *onPressed* atribuída, que define a ação a ser executada quando o botão é pressionado. Nesse caso, todos os botões executam a função de navegação específica daquele evento.

Em geral, esse código constrói uma página com vários botões que levam a uma rota específica quando pressionados. Os botões são organizados verticalmente e centralizados na página como é possível visualizar na figura 44.

Figura 44 - EventPage

3.2.2.5 DadosEventPage

Faço a seguir a descrição da constituição e funcionamento dos componentes desta página. As primeiras linhas importam os pacotes necessários para o funcionamento do código. *flutter/material.dart* é o pacote do Flutter para a construção da interface gráfica. *flutter_barcode_scanner.dart* fornece recursos para ler códigos de barras e QR codes. *shared_preferences.dart* permite armazenar dados localmente para uso posterior. Os ficheiros importados a partir de *../service/* conteem implementações personalizadas para o código.

Em seguida temos definida uma classe chamada DadosPage que herda do widget StatefulWidget. Esse widget representa uma página que possui um estado mutável.

As próximas linhas declaram variáveis usadas no estado da página. *inLoader* é um ValueNotifier que armazena um valor booleano indicando se um carregamento está em andamento. *UserData* é uma classe personalizada que contém dados do utilizador. *event* é um booleano que indica se há um evento. *eventId* armazena o ID do evento. *data*

armazena os dados lidos de um QR code. *nome*, *numero*, *curso*, *dateTime* e *status* são variáveis que armazenam diferentes informações sobre o utilizador.

A seguir temos o método assíncrono *authEvent* que verifica se há participantes inscritos em um evento. Ele obtém o ID do evento a partir do armazenamento local usando SharedPreferences. Em seguida, chama a função *countPresence* da classe Api para verificar se há participantes. Se não houver participantes, exibe um alerta. Caso contrário, navega para a rota '/*Doughnut*'. O método retorna um valor booleano indicando se há participantes. Esse método chama a função *insertPresence* da classe Api, passando as variáveis *nome*, *curso* e *status* como argumentos. A função retorna um objeto que é armazenado na variável userData. Em seguida, ele verifica se o atributo result do objeto userData é igual a '1'. Se for, retorna true, indicando que a inserção foi bem-sucedida. Caso contrário, retorna false.

O próximo método *readQRCode* é chamado quando o botão "Validar" é pressionado. Ele utiliza a biblioteca *flutter_barcode_scanner* para ler um QR code. O código lido é armazenado na variável code. Em seguida, o código é verificado para determinar se foi lido corretamente ou não. Se o código foi lido corretamente, ele é atribuído à variável *data*, caso contrário, a string "Não validado" é atribuída. A variável data é atualizada usando *setState* para refletir a alteração na interface. Se o código foi lido corretamente, ele é dividido em partes separadas com base no caractere ';' e as partes são atribuídas às variáveis *nome*, *curso* e *status*. Em seguida, o valor de inLoader é definido como true, indicando que um carregamento está em andamento.

Em seguida temos o método *authInsert* que é responsável por inserir a presença de um utilizador em um evento. Ele recebe o parâmetro *context* que representa o contexto da aplicação. Dentro do método, é feita uma chamada assíncrona para o método *Api.insertPresence*, que recebe os parâmetros *nome*, *curso* e *status*. Esses parâmetros são usados para inserir a presença do utilizador no evento por meio de uma API. A variável *userData* é usada para armazenar a resposta da API. Após aguardar a conclusão da chamada assíncrona, o código verifica se o resultado retornado em *userData.result* é igual a '1', indicando que a presença foi inserida com sucesso. Se o resultado for igual a '1', o método retorna true, indicando que a inserção da presença foi bem-sucedida. Caso contrário, retorna false, indicando que ocorreu algum erro durante a inserção da presença. Esse método é utilizado no contexto da leitura de um código QR, onde as informações extraídas do código são utilizadas para inserir a presença do utilizador no evento.

No método build, é construído o layout da página. O *Scaffold* é o widget principal que contém a estrutura básica da página. Ele possui um *body* que consiste em um *SafeArea* para evitar que o conteúdo fique atrás das áreas seguras do dispositivo (como a barra de status) e um *SingleChildScrollView* para permitir a rolagem do conteúdo. Dentro do *SingleChildScrollView*, temos um *SizedBox* que define a largura com base na largura disponível no ecrã. Dentro desse *SizedBox*, há um *Column* que

organiza os elementos verticalmente. Dentro do *Column*, temos uma série de *ElevatedButton.icon*, que são botões com ícones e rótulos. Cada botão tem uma função associada a ele, como chamar o método *readQRCode*, navegar para uma determinada rota ou chamar o método *authEvent*. Cada botão também tem um espaço vazio (SizedBox) abaixo dele para criar o espaçamento entre os botões.

Essa é a estrutura e o funcionamento geral da página DadosPage com funcionalidades como leitura de códigos QR, autenticação de eventos e exibição de botões para diferentes ações. Na figura 45 vemos como ficou a página.

Figura 45 - DadosPage

3.2.2.6 DoughnutPage

Começamos com os nossos importes e com a definição uma classe Data com as propriedades cursoValue e count, que vão receber mais a frente o nome do curso e a contagem de quantos inscritos no evento daquele específico curso.

A classe DoughnutPage é um StatefulWidget que representa uma página que exibe um gráfico de rosca (doughnut chart).

O método assíncrono *getData* é responsável por obter os valores das preferências compartilhadas usando a biblioteca *shared_preferences*. Ele busca o valor da chave *'curso'* e *'count'* e os atribui às variáveis *cursoValue* e *count*, respetivamente.

O método *initState* é executado quando o estado da página é inicializado. Nele, é definido o comportamento da *tooltip* do gráfico e chamado o método *getData* para obter os valores das preferências compartilhadas.

Temos no código a definição de algumas variáveis para representar os valores de cursos e suas respetivas contagens. As variáveis *_curso1* e *_qnt1* são obtidas das preferências compartilhadas, enquanto as outras variáveis são definidas com valores literais pois as estamos utilizando apenas para exemplificar com os gráficos. As variáveis *minc* e *maxc* serão utilizadas para armazenar os nomes dos cursos com a menor e maior participação, respetivamente.

Em seguida temos um trecho do código, onde é realizado o cálculo de algumas estatísticas com base nos valores dos cursos e suas contagens. Defini uma lista que contém os valores das contagens dos cursos. O número de elementos na lista é armazenado na variável chamada *divisor*. A soma total das contagens é calculada e armazenada na variável *total*. A média das contagens é calculada como a divisão de total por divisor, e o valor resultante é arredondado para o tipo *double* usando *truncateToDouble*() e armazenado na variável *media*. A variável min recebe o valor mínimo da lista usando o método *reduce*. Em seguida, são feitas verificações condicionais para determinar qual curso tem a menor participação (minc), comparando os valores. Da mesma forma, a variável max recebe o valor máximo da lista usando o método *reduce*. Em seguida, são feitas verificações condicionais para determinar qual curso tem a maior participação (maxc), comparando os valores.

A lista chartData é criada para armazenar os dados que serão exibidos no gráfico de rosca. Ela é inicializada com instâncias da classe ChartData, onde cada instância representa um par de valores (curso e contagem).

Em seguida temos a construção da estrutura do widget da página. O widget *Scaffold* é utilizado como base, contendo um *AppBar* com um título, um *SfCircularChart* para exibir o gráfico de rosca e um *Column* para organizar os demais elementos da página. Dentro do *SfCircularChart*, é utilizado o *DoughnutSeries* para renderizar o gráfico de rosca. Os dados são fornecidos através da propriedade *dataSource*, que recebe a lista *chartData*. O *xValueMapper* e o *yValueMapper* são usados para mapear os valores do curso e da contagem, respetivamente.

No fim temos ainda a classe chartData que é usada para representar os dados do gráfico de rosca. Ela possui três propriedades: x, y e color. A propriedade x armazena o valor do curso, a propriedade y armazena a contagem associada ao curso e a propriedade color armazena a cor associada ao curso (opcional). Na figura 46 vemos como ficou a página.

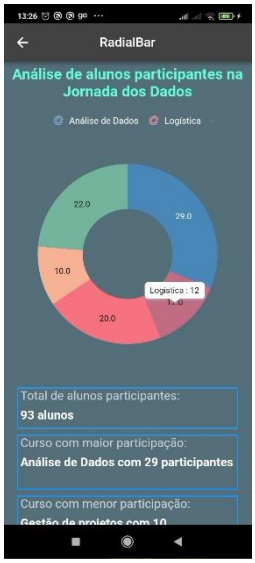

Figura 46 - DoughnutPage

3.2.2.7 RadialPage, ColumnsSyncPage e TreeMapPage

Essas três páginas têm praticamente o mesmo comportamento da página anterior, gerando um gráfico com as informações relativas a inscrição de alunos num determinado evento. No entanto são gráficos que condizem com o título de cada página, como se pode ver nas figuras 47, 48 e 49.

Figura 47 - TreeMap

Figura 48 - Radios

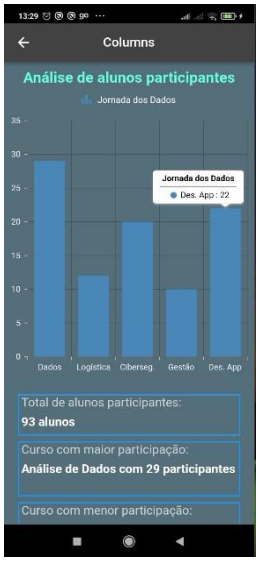

Figura 49 - Columns

3.3.3 Controllers

3.2.3.1 Login Controller

Nesse diretório temos a classe LoginController que como o nome sugere, é responsável por controlar a lógica do login, ou seja, fazemos o controle de acesso do utilizador.

Começamos declarando uma variável chamada inLoader do tipo boleana, esta é na verdade um objeto ValueNotifier que controla o estado do carregamento. Ele é usado para atualizar a interface do utilizador com base no estado do carregamento.

Temos também a declaração de duas variáveis (_number e _pass), que são variáveis estáticas que armazenam o username e a senha do utilizador, respetivamente.

Contamos ainda com o método *auth* que realiza a autenticação do utilizador. Ele recebe o contexto como parâmetro. Dentro do método, o valor de inLoader é definido como true, indicando que o carregamento está em andamento. Em seguida, é feita uma chamada à API para realizar o login usando os valores de _number e _pass. Se a resposta da API for diferente de nulo, é chamado o método *_navegaGetEventPage* para navegar para a página *GetEvent*, e o valor de inLoader é definido como false. Caso contrário, é exibido um alerta informando que o login é inválido.

O método estático *_navegaGetEventPage* recebe o contexto como parâmetro e navega para a página /*GetEvent* usando o *Navigator*.

3.2.4 Services

3.2.4.1 ApiService

Nesse diretório temos a classe Api e esta tem algumas funções, vou fazer uma rápida descrição de cada uma delas, esta classe é responsável por realizar chamadas à API e lidar com os dados relacionados à autenticação e interação com eventos.

Comecei essa classe criando duas variáveis que vão receber os parâmetros número do aluno e senha. Na classe que falamos anteriormente, LoginController, temos a função *auth* que chama a função login e envia os dados do aluno, esta por sua vez recebe os dados e salva-os nas variáveis criadas na classe, essas variáveis são globais, ou seja, podem ser usadas em todas as funções da classe. Além disso a função login chama a função *fetchData*, esta é a função que vai buscar os dados do utilizador; ela começa chamando a função *getPublickey*, esta faz nossa primeira requisição http para pegar a chave pública, se tudo correr bem ela recebe um ficheiro do tipo *json*, decodifica-o e guarda a *string* em uma variável chamada publiKey e a retorna.

Continuando com a função *fetchData*, após receber a chave púbica, a utiliza para fazer a encriptação da senha e do username do utilizador. Em seguida uma nova requisição http é feita dessa vez enviando a o username e a senha criptografada e em base 64 de modo a que a mesma pode ser enviada pela web. Quando a mensagem é recebida, a senha é decriptada com a chave privada e é feita a verificação dos dados enviados. Novamente na resposta virá um ficheiro do tipo *json*, nesse ficheiro haverá uma chave com o nome 'result', se o valor dessa chave for '0', significa que os dados estão incorretos e a autenticação não é feita, mas se o valor for '1', significa que os dados estão corretos e esse ficheiro *json* terá outras chaves e valores. De seguida esse ficheiro é decodificado e os dados enviados como retorno. A função login recebe os dados e verifica a chave 'result', sendo igual a '1' retorna os dados, sendo igual a '0' retorna *null*.

O método estático *insertPresence* recebe o nome, curso e status como parâmetros e realiza uma solicitação PUT para registrar a presença do aluno em um evento específico. Ele também realiza etapas semelhantes às descritas anteriormente, como a obtenção da chave pública, criptografia dos dados do utilizador e envio da solicitação PUT com os dados criptografados.

O método estático *countPresence* recebe o ID do evento como parâmetro e realiza uma solicitação GET para obter o número de participantes registrados para o evento. Ele também armazena o número de participantes e outros dados relevantes nas preferências compartilhadas (SharedPreferences) da aplicação.

O método estático *getEvent* realiza uma solicitação GET para obter os detalhes do evento associados ao utilizador autenticado. Ele também armazena informações relevantes do evento nas preferências partilhadas.

Essa é uma visão geral das funcionalidades da classe Api e seus métodos. Eles são responsáveis por lidar com a autenticação do utilizador, interação com a API e manipulação dos dados retornados.

3.2.4.2 Alerts

Nesse diretório temos a apresentação de uma série de funções para exibir caixas de diálogo (alertas) personalizadas em uma aplicação Flutter. Segue uma descrição geral dessas funções:

A função Alert exibe um alerta genérico, mostrando um título fixo ("Login") e uma mensagem de texto fornecida como parâmetro. O alerta exibe um botão "Ok" e, quando pressionado, fecha o alerta.

A função AlertSuccess exibe um alerta de sucesso, mostrando um título fixo ("Bem-vindo!!!") e uma mensagem de texto fornecida como parâmetro. O alerta também inclui um botão "Ok" que, quando pressionado, fecha o alerta.

A função AlertError exibe um alerta de erro, mostrando um título fixo ("Não inserido") e uma mensagem de texto fornecida como parâmetro. Assim como nos alertas anteriores, o botão "Ok" fecha o alerta ao ser pressionado.

A função AlertNotEvent exibe um alerta informando que não há eventos disponíveis. Ele mostra um título fixo ("Não há evento") e uma mensagem de texto fornecida como parâmetro. O botão "Ok" é utilizado para fechar o alerta.

A função AlertEvents exibe um alerta relacionado a eventos. Ele mostra um título fixo ("Evento") e exibe uma mensagem de texto fornecida como parâmetro. O alerta inclui um botão "Ok" que fecha o alerta quando pressionado.

Essas funções são usadas para fornecer feedback visual ao utilizador em diferentes cenários na aplicação, permitindo exibir mensagens de sucesso, erros ou informações relevantes por meio de caixas de diálogo personalizadas.

Por fim temos ainda um ficheiro chamado UserData que faz o mesmo na aplicação anterior.

4. Conclusão

Ao concluir este estágio, pude vivenciar na prática parte do conhecimento adquirido durante o curso de análise de dados, e isso proporcionou uma grande aprendizagem e contribuiu significativamente para o desenvolvimento das minhas habilidades técnicas e interpessoais. Durante o período de estágio, pude aplicar conhecimentos teóricos em situações reais de trabalho e também tive a oportunidade de trabalhar com pessoas experimentadas na área de tecnologia, o que me proporcionou um ambiente de trabalho desafiador e ao mesmo tempo muito enriquecedor. Além disso, pude vivenciar a dinâmica do trabalho em equipe e perceber um pouco em que consiste e o que significa fora da sala de aula.

No desenvolvimento das aplicações surgiram algumas dificuldades que consegui superar com esforço, dedicação, empenho, com a ajuda dos professores, com muito trabalho e pesquisa. A aplicação fica concluída naquilo que me foi proposto a fazer, mas ainda não está em produção, certamente receberá alguns contributos, mas já sinto-me feliz por a ter desenvolvido.

Nesse momento de final do estágio, sinto-me muito mais preparado para enfrentar os desafios do mercado de trabalho na área de tecnologia. Aprendi a lidar com diferentes tecnologias e ferramentas, com prazos e expectativas, desenvolver uma postura proativa e de solucionador de problemas, aprimorei minhas habilidades de programação e desenvolvi a capacidade de trabalhar em equipe.

Por fim, gostaria de agradecer ao professor Paulo Vieira pela oportunidade de aprendizagem e também ao professor e meu supervisor no estágio Carlos Fonseca pelas instruções, esclarecimentos e orientações sempre que foram necessários durante esse processo. Estou confiante de que os conhecimentos e habilidades adquiridos nesse projeto serão fundamentais para a minha trajetória profissional futura, e por isso estou muito grato pela oportunidade e por tudo o que aprendi durante este estágio.

5. Webgrafia

Alura: https://www.alura.com.br/artigos/figma

Acervo de tutoriais e referências, ACERVO LIMA (2023): https://acervolima.com/

Baixar arte vetorial, vecteezy (2023): https://pt.vecteezy.com/

Curso de Darts, Daniel Ciolfi, Startto.dev (2023): https://www.youtube.com/watch?v=V9PL8S-ihfk&list=PLR5GUTqrcwXhVV-jNR38vfAZabkmGGKfO

[2] Dart Apprentice: Fundamentals (2022): https://www.kodeco.com/books/dart-apprentice-fundamentals/v1.0

Dart (2023): https://dart.dev/

Do figma pro flutter, Alura, (2023): https://www.youtube.com/watch?v=c-ZfdDqKbRs

Emanuel Severino (2022): https://www.youtube.com/@emanuelseverino8721/featured

Everton, EvertonDev (2022): https://www.youtube.com/@evertondev

Figma (2023): https://www.figma.com/

Flutter (2023): https://flutter.dev/

flutter_barcode_scanner, pub.dev, (2021): https://pub.dev/packages/flutter_barcode_scanner

Flutter para iniciantes (2023): https://flutterparainiciantes.com.br/

Fontes e ícones Google, (2023): https://fonts.google.com/

Gerador de ícone de aplicação, (2023): https://www.appicon.co/

Gerador de ícone de aplicação (2023): https://www.appicon.co/

Guilherme Dougaich de Oliveira, linkedin (2021): https://www.linkedin.com/pulse/meu-aprendizado-de-git-e-github-guilherme-dugaich-de-oliveira/?originalSubdomain=pt

Introdução ao GitHub Desktop, GitHub Docs (2023): https://docs.github.com/pt/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop

Json Placeholder: https://jsonplaceholder.typicode.com/...

Json to Dart: https://javiercbk.github.io/json_to_d...

Livro de receitas, Flytter (2023): https://docs.flutter.dev/cookbook

Lucas Alves, Medium (2019): https://medium.com/flutter-comunidade-br/12-bibliotecas-%C3%BAteis-para-auxiliar-no-desenvolvimento-utilizando-flutter-5982fdf01c80

Lucas Fernando, Youtube (2023): https://www.youtube.com/@lucasfernando7578

[1] Meu aprendizado de Git e GitHub (2021): https://www.linkedin.com/pulse/meu-aprendizado-de-git-e-github-guilherme-dugaich-de-oliveira/?originalSubdomain=pt

Macoratti (2023): https://www.macoratti.net/

[3] Novageo, Criptografia Assimétrica RSA, (2022): https://www.novageo.pt/novageo/displayArticles?numero=38346&_que__criptografia_assimetrica_rsa_algoritmo_chave_assimetrica_

Paula Simionato traduziu Ilhechikara Vincent Abba, freeCodeCamp (2022): https://www.freecodecamp.org/portuguese/news/tutorial-de-git-e-github-controle-de-versao-para-iniciantes/

Pedro Massango, Gerando Código QR com Qr.Flutter, (2020): https://www.youtube.com/watch?v=-M4Pb7HxFAg

Pinkesh Darji, Flutter Beads (2023): https://www.flutterbeads.com/about-me/

Professor Carlos Duarte, Polimorfismo (2021): https://www.youtube.com/@CanalPolimorfismo

Professor Diego Antunes, Curso flutter na prática Youtube (2023): https://www.youtube.com/@drantunes

Rafael Terra, talentnetwork, (2021): https://rockcontent.com/br/talent-blog/hacks-de-figma/

Rafael Outeiro, Comparaja.pt (2022): https://www.comparaja.pt/blog/qr-code

RTP (2021): https://www.rtp.pt/noticias/mundo/estudo-como-os-smartphones-se-transformaram-no-lugar-onde-agora-vivemos_n1318827

Site para criação de fluxogramas – draw.io

Vijaycreations, Flutter TextFormField Stylings (2021): https://www.youtube.com/watch?v=pyVlrcci6nw