UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

Marlon Mafra

Ygor Gasparin

UMA APLICAÇÃO ANDROID PARA O PORTAL MINHA UFSC

FLORIANÓPOLIS
2013

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

Marlon Mafra
Ygor Gasparin

UMA APLICAÇÃO ANDROID PARA O PORTAL MINHA UFSC

Trabalho de Conclusão do Curso de Graduação em Sistemas de Informação, do Centro de Ciências Tecnológicas da Universidade Federal de Santa Catarina, requisito parcial à obtenção do título de Bacharel em Sistemas da Informação. Orientação de: Prof. Dr. Fernando Augusto da Silva Cruz.

FLORIANÓPOLIS
2013

Acadêmicos: Marlon Mafra
Ygor Gasparim

Título: Uma aplicação Android para o portal Minha UFSC.

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação, do Centro Tecnológico da Universidade Federal de Santa Catarina, como requisito parcial à obtenção do título de Bacharel em Sistemas de Informação, aprovado com nota ____.

Florianópolis, _____ de _____ de 2013 .

_____
Dr. Fernando Augusto da Silva Cruz, UFSC
Professor Orientador

_____
Dr. João Bosco Mangueira Sobral, UFSC
Membro da Banca Examinadora

_____
Mestre Roque Bezerra, UFSC
Membro da Banca Examinadora

Ficha Catalográfica elaborada por Renata Duarte de Borja, graduanda em Biblioteconomia da Universidade Federal de Santa Catarina.

M187u
Marlon Mafra, 1985 –

Uma aplicação Android para o portal Minha UFSC./ Marlon Mafra, Ygor Henrique Gasparim. – Florianópolis, 2013.

104f. ; il. color.

Orientador: Fernando Augusto da Silva Cruz, Doutor
Trabalho de Conclusão de Curso (Graduação) – Universidade Federal de Santa Catarina, Centro de Ciências Tecnológicas, Curso de Sistemas de Informação, Florianópolis, 2012.

1. _____ 2. _____ 3. _____ I. Cruz, Fernando Augusto da Silva.
II. Título.

CDU - _____

AGRADECIMENTOS

Marlon Mafra

A minha namorada Renata que se fez presente nos quatro anos de Universidade e em todos os momentos que precisei na fase de elaboração do TCC, me dando força, amor, carinho e incentivo.

Aos meus pais, Luiz Carlos e Nelzi, que sempre estiveram presentes quando precisei e me apoiaram nas decisões que tomei.

A toda a minha família que sempre prestou solidariedade quando me viu necessitado de ajuda, e por sempre estarem torcendo por mim.

Aos amigos de Universidade, pela companhia nestes quatro anos e meio, e pela troca de experiência, que jamais esquecerei.

A todos os meus amigos, que de alguma maneira tiveram alguma participação.

Aos lugares que trabalhei e fiz estágio, sem eles eu não teria onde praticar os meus conhecimentos e aprender ainda mais.

Ao meu amigo de projeto Ygor Gasparin.

Ao Roque Bezerra e todo pessoal do Setic, que foram fundamentais para que esse trabalho acontecesse.

Agradeço também ao professor orientador Fernando Cruz pela sua paciência e dedicação para com esse trabalho.

Por fim, gostaria de agradecer a todos os professores pelo empenho e dedicação na passagem de seus conhecimentos e experiências que tanto contribuíram para o meu crescimento, pessoal e profissional.

E principalmente a Deus, pois sem ele nada seria possível.

AGRADECIMENTOS

Ygor Henrique Gasparin

Primeiramente a minha namorada Maria Cristina por ter me apoiado nos momentos mais difíceis, não me deixando desistir, mesmo nos momentos onde achei que tudo já estava perdido.

Aos meus pais, Ricardo e Milene, que sempre me apoiaram e me incentivaram para que eu trilhasse o meu rumo.

A todos os amigos da Universidade, excelentes profissionais, no qual troquei valiosas experiências e várias garrafas de cerveja.

Aos professores que tiveram papel fundamental em minha formação profissional, e em algumas vezes até mesmo pessoal.

Ao orientador Fernando Cruz, que acreditou em nossas ideias e possibilitou que esse trabalho acontecesse.

Ao Roque Bezerra e Márcio Clemes do SeTIC, que nos auxiliaram na construção deste trabalho.

A todos que de alguma forma ajudaram, agradeço por acreditarem no meu potencial, em minhas ideias e por estarem sempre me incentivando a buscar algo mais.

E por último, e não menos importante, ao meu amigo de projeto, Marlon Mafra.

“O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.”

(José de Alencar)

RESUMO

MAFRA, Marlon; GASPARIN, Ygor Henrique. Uma aplicação Android para o portal Minha UFSC. 2013. 104 p. Trabalho de Conclusão de Curso. (Graduação em Sistemas de Informação) - Universidade Federal de Santa Catarina, Florianópolis, 2013.

Este trabalho de conclusão de curso é um aplicativo para a plataforma Android que disponibiliza as principais funcionalidades utilizadas pelo aluno no dia-a-dia com a Universidade. Sabe-se que o mercado móvel está em grande ascensão, devido à popularização de smarthphones, tablets e da internet, além, é claro, do crescimento de acesso à internet em locais públicos. Devido a esse crescimento, e da necessidade da população acadêmica estar sempre conectada, neste trabalho queremos disponibilizar um novo canal de acesso às informações da Universidade Federal de Santa Catarina (UFSC) para com os alunos, totalmente otimizado para o mercado móvel, disponibilizando as principais funcionalidades do portal Minha UFSC. Neste aplicativo irão ser disponibilizadas as seguintes funcionalidades: perfil do aluno, que conterá os seus dados pessoais; avisos da Universidade para com os universitários; histórico escolar do aluno; sua grade de horários para o semestre atual; estatística do I.A.A, que contém um gráfico comparando o I.A.A. do aluno ao longo do período em que está inserido na Universidade, com o I.A.A. médio do curso para esse mesmo período; cardápio do R.U. (Restaurante Universitário) disponibilizado semanalmente; e informações sobre empréstimos e débitos de livros com a B.U. (Biblioteca Universitária).

Palavras-chave: Aplicações Web. Android. Minha UFSC. Dispositivos móveis.

ABSTRACT

MAFRA, Marlon; GASPARIN, Ygor Henrique. Uma aplicação Android para o portal Minha UFSC. 2013. 104 p. Trabalho de Conclusão de Curso. (Graduação em Sistemas de Informação) - Universidade Federal de Santa Catarina, Florianópolis, 2013.

This work is about an application software for the Android platform that provides the main functionalities used by students on their daily routines at the University. It is known that the mobile market is on the rise, due to the popularity of Smartphones, tablets and the Internet, and due to the higher availability of public Internet access. Because of this increase, and of the need for the academic community to be always connected, we intend with this work to present a new communication channel to information about the Universidade Federal de Santa Catarina (UFSC) to its alumni, completely optimized for the mobile market, by making the main functionalities of the Minha UFSC portal available. The following functionalities will be implemented: student profile, with his or her personal information; University news board; student transcript; student current course schedule; student GPA statistics, with a graph comparing the student's measure with the global program average through time; RU (University Cafeteria) menu, weekly updated; and information about BU (University Library) book loans and debts.

Keywords: Web applications. Android. Minha UFSC. Mobile devices.

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES

ADT - Android Development Tools
API - Application Programming Interface
APK - Android Package File
ARM - Advanced RISC Machine
B.U. - Biblioteca Universitária
GPL - General Public License
JAR - Java Archive
LDPA - Lightweight Directory Access Protocol
MP3 - MPEG Layer 3
RIM - Research in Motion
RU - Restaurante Universitário
SDK - Software Development Kit
SO - Sistema operacional
SOAP - Simple Object Access Protocol
SSO - Single sign-on
UFSC - UNIVERSIDADE FEDERAL DE SANTA CATARINA
URI - Uniform Resource Identifier
URL - Uniform Resource Locator
XML - eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Com o setor tecnológico em constante crescimento há claros sinais de que um novo mercado com grande potencial e ainda pouco explorado está surgindo, e esse mercado são aplicações para dispositivos móveis, que segundo Tozetto (2013):

> Os smartphones, celulares inteligentes que permitem navegar na internet e instalar aplicativos, passaram a representar mais da metade das vendas globais de celulares pela primeira vez, de acordo com novo estudo divulgado pela consultoria IDC. Do total de 418,6 milhões de celulares vendidos pelos fabricantes no primeiro trimestre de 2013, 216,2 milhões eram smartphones ou 51,6% do total.

Sabemos que hoje em dia esse número aumentou muito e a necessidade das pessoas de se manterem conectadas à internet todo o tempo também. E com esse rápido crescimento surge oportunidades no mercado, e um deles é popularização do acesso à internet através de dispositivos móveis. Segundo Xavier (2013):

> Setenta milhões. Essa é a projeção da Anatel para o número de acessos 3G no mercado brasileiro ao final de 2012. Isso representa um crescimento de 75% em relação a 2011 e 250% quando comparado a dezembro de 2010. Mais do que isso, caso essa previsão se confirme, teremos um cenário em que 2 a cada 3 acessos à internet serão feitos por meio de dispositivos móveis e isso muda muita coisa.

Nesse contexto, a utilização de aplicativos no dia a dia torna-se quase que inevitável, no setor bancário, por exemplo, uma transação bancária poderá levar no máximo cinco minutos, enquanto que se fosse feita de forma habitual levaria pelo menos uma hora entre deslocamento e execução. Além da praticidade em executar uma tarefa o aplicativo permite que o usuário possa estar em qualquer lugar, fazendo outra atividade, e poupando seu tempo.

Outro mercado em ascensão é o educacional, que surgiu com uma nova forma de aprendizagem e com inúmeros benefícios, tanto para professores quanto para os alunos.

2 OBJETIVOS

Neste capítulo, apresentam-se os objetivos que enfatizaram a criação do aplicativo para o portal do estudante da Universidade Federal de Santa Catarina.

2.1 OBJETIVO GERAL

Desenvolver um aplicativo, para dispositivos móveis, utilizando a plataforma Android, para o portal Minha UFSC da Universidade Federal de Santa Catarina.

2.2 OBJETIVOS ESPECÍFICOS

- Reunir as principais informações do aluno em um só local – o aplicativo;
- Tornar as informações acessíveis através dos dispositivos móveis;
- Facilitar aos alunos o acesso as informações pertinentes à Universidade.

3 REVISÃO DE LITERATURA

3.1 ECLIPSE

O Eclipse é uma IDE (Integrated Development Enviroment), desenvolvido inicialmente pela IBM e posteriormente doado como software livre para a comunidade, ele é disponibilizado pela licença EPL (Eclipse Public License), que garante o uso, a modificação, a copia e a distribuição dos projetos sob essa licença (ECLIPSE, 2013). Este software começou a ser desenvolvido em 2001 e em 2003/2004 foi fundado a Fundação Eclipse, uma organização sem fins lucrativos:

> Cujos projetos são concentrados na criação de uma plataforma de desenvolvimento aberta composta de estruturas, ferramentas e tempos de execução extensíveis para desenvolvimento, implementação e gerenciamento de software por todo o ciclo de vida. (ECLIPSE, 2013).

Hoje a Fundação Eclipse hospeda mais de 200 projetos, nas mais diversas fazes de desenvolvimento, desde incubação, até projetos maduros como a própria

ferramenta Eclipse. Taurion, gerente de novas tecnologias da IBM - Brasil comenta que:

> Na prática ele (o Eclipse) cria um modelo de Open Innovation Network, onde empresas concorrentes podem criar redes de inovação, cooperando no desenvolvimento de softwares Open Source, que servirão de base para produtos específicos (proprietários), com os quais concorrerão no mercado. O conceito de Open Innovation quebra o paradigma tradicional da P&D feito a portas fechadas, por uma única empresa. Este conceito trata a P&D como um sistema aberto onde tanto ideias externas e internas são debatidas e as melhores alternativas são selecionadas. (IBM, 2007)

O Eclipse é desenvolvido em Java e serve também como ferramenta para desenvolvimento em Java, além disso, é extensível para outras linguagens através da utilização de plug-ins. É possível estender o desenvolvimento para Android, C/C++, PHP, ColdFusion, Flex/ActionScript, Phyton entre outros.

3.2 MAVEN

Maven ou Apache Maven é uma ferramenta de código aberto de gerenciamento de projetos Java distribuído através da licença Apache License 2.0.

Ele fornece facilidades nas mais diversas fases de desenvolvimento, desde a criação de um novo projeto através de arquétipos (projetos pré-configurado e pronto para serem adaptados, eliminando a necessidade de o desenvolvedor ter que configurar um projeto do zero), passando pela utilização de plug-ins durante o desenvolvimento, como plug-ins geradores de métricas que podem ser utilizadas no acompanhamento do desenvolvimento do projeto, até a liberação final de um projeto em um ambiente de produção.

Além dessas funcionalidades o Maven também funciona como gerenciador de dependências de bibliotecas, possibilitando declarar quais são as dependências de determinado projeto e o Maven automaticamente se encarrega de realizar o download desses artefatos utilizando um repositório central, que contém mais de 486 mil artefatos.

O Maven tem como objetivo principal auxiliar na compreensão completa de um projeto, para isso ele trabalha com cinco áreas bases de preocupação:

> 1 Making the build process easy; 2 Providing a uniform build system; 3 Providing quality project information; 4 Providing guidelines for best practices development; 5 Allowing transparent migration to new features. (APACHE MAVEN PROJETC, 2013)

A sua base de configuração é um arquivo XML (eXtensible Markup Language) chamado pom.xml, que é o acrônimo de *Project Object Model* ou literalmente *Projeto Modelo de Objeto.* Nele serão declarado todas as informações básicas do projeto, como por exemplo, dependências de bibliotecas, utilização de plug-ins e demais comportamentos.

3.3 ANDROID

Android é um sistema operacional baseado em Linux, mais precisamente sobre o kernel 2.6. Ele é desenvolvido pela Google juntamente com a Open Handset Alliance (OHA), essa última formado por um conjunto de empresas. Ele foi criado com o objetivo de ser uma plataforma móvel de código aberto que permitisse ao desenvolvedor total liberdade para tirarem o máximo de proveito que os dispositivos têm a oferecer. Por exemplo, uma aplicação pode invocar qualquer uma das principais funcionalidades do dispositivo, como fazer chamadas, enviar mensagens de texto, ou utilizar a câmera, permitindo aos desenvolvedores criar experiências mais ricas e mais coesas para os usuários.

3.3.1 Contexto histórico

Até alguns anos atrás os celulares mais sofisticados tinham apenas pequenos aplicativos, como por exemplo, calculadora e alguns jogos tradicionais. Devido a tecnologia da época e o próprio desinteresse das pessoas, eles ainda não haviam se tornado multiuso ou utilizados como ferramentas pessoais como são utilizados atualmente. Dimarzio comenta que

> desde 1997 os celulares ainda não tinham se tornado multiuso, não tinham ferramentas multifuncionais pessoais. Ninguém ainda via a necessidade de navegação na Internet, tocar MP3 (MPEG Layer 3), ou qualquer uma das muitas funções que estamos acostumados a usar em um celular. No entanto, mesmo se a necessidade estava presente na época, a falta de memória do dispositivo e capacidade de armazenamento era um obstáculo ainda maior para superar (2008, p 4.).

Com o decorrer dos anos, a tecnologia foi evoluindo e os hardwares tornaram-se melhores, e em tamanhos cada vez mais reduzidos. E isso permitiu a evolução dos telefones celulares, que foram ganhando novas funções como

câmeras digitais embutidas, rádio, entre outras inovações para aquela época. Outras funções também foram aprimoradas, os jogos, por exemplo, atingiram um novo nível de sofisticação, chamadas de voz e o próprio visor dos celulares passaram a ganhar mais cores.

> Quando os dispositivos móveis começaram a ganhar outras funções além da chamada por voz e o envio de SMS, foi necessário um avanço nos programas para gerenciar as novas funções. Quanto mais recursos um celular ganhava, mais possibilidades os desenvolvedores enxergavam até que foi necessário repensar completamente os sistemas operacionais. (CINDRAL, 2012).

Nesse contexto surge o SO (Sistema operacional) Symbian, um SO de código aberto, que foi "Adotado pela Nokia, o sistema permitiu que milhares de desenvolvedores no mundo pudessem criar aplicativos baseado nos novos recursos." (CINDRAL, 2012). Além deste outros SOs ganharam espaços no mercado, entre eles estão o IOS da Apple e o BlackberryOS da RIM (Research In Motion).

O IOS é o sistema operacional móvel da Apple, antes chamado de Iphone OS, ele é derivado do Mac OS e roda sobre a plataforma ARM (Advanced RISC Machine). O primeiro dispositivo a receber o sistema foi o Iphone, por isso o nome Iphone OS. O sistema operacional segue o padrão Apple, acabamento impecável, interface muito bem construída e excelente usabilidade, fazendo do IOS um sistema inovador e um exemplo a ser seguido. Porém a Apple mantém um controle rigoroso sobre os desenvolvedores, usuários e aplicações, sendo um grande alvo de críticas e reclamações por grande parte deles.

> Controle sobre os aplicativos lançados, localidade da publicação do aplicativo, plataforma de desenvolvimento proprietária e exclusiva, hardware especifico, design diferenciado e funcionalidades inovadoras entre outros pontos. Foram pontos focais para a liderança da Apple, com IOS. Consequentemente, apresentando um custo superior na construção de aplicativos, para inicio de desenvolvimento, mas com grande perspectiva de vendas. (PACHECO JÚNIOR e CASTRO, 2011, p 4)

BlackberryOS é um sistema operacional para smartphones desenvolvido pela RIM. No começo do século 21 os Blackberry eram os dispositivos mais utilizados no mercado corporativo, seu bom suporte a e-mail, mensagens, e edição de documentos fizeram dele o melhor entre os seus concorrentes. Assim como a IOS é

exclusivo para dispositivos Apple, o BlackberryOS é exclusivo para smartphones fabricados pela RIM, isso significa muito mais estabilidade, consistência e confiabilidade. Contudo com a chegada do IOS e do Android ele perdeu muito espaço no mercado.

> Acredita-se que a perpetuação deste produto neste segmento se deu ao fato dos produtos liberados pela empresa suportar Java ME (Java Micro Edition). Logicamente, atrelado aos fatores que acarretam ao uso desta tecnologia: custo de desenvolvimento de aplicativos baixo, desnecessidade de hardware especifico, simuladores dos aparelhos que a própria empresa disponibiliza e a facilidade de encontrar profissionais que dominem Java ME. Em contra proposta, seus problemas, como alto consumo de memória e limitações de hardware. (MCQUEEN AT. ALL, 2010).

Pode-se concluir apesar da forte concorrência, o BlackBerry ainda possui uma pequena fatia no mercado corporativo.

Em agosto de 2005 a Google adquiriu uma pequena empresa em Palo Alto – Califórnia – USA, que desenvolvia uma plataforma para celulares baseado em Linux totalmente flexível e de código aberto. Em dezembro de 2006 vários rumores apontavam que a gigante Google entraria no mercado de smartphones, esses rumores duraram dois anos até que em dezembro de 2007 ela anunciou oficialmente a plataforma Android para smartphones. Além disso, foi criado o OHA, um conselho com mais de 33 empresas parceiras, dentre elas podemos destacar a LG, Telefônica, Vodafone, Docomo, Intel, Asus, Motorola, Dell entre outras.

Com o lançamento do Android a plataforma Symbian passou a perder espaço no mercado,

> Em 2007, 63,5% dos smartphones vendidos eram Symbian, segundo a Gartner. “Essa porcentagem foi diminuindo drasticamente com a popularização do Android: em 2011, o Symbian era responsável por 18,7% das vendas, enquanto que o Android estava com 46,5%; no terceiro trimestre de 2012, o Symbian tinha só 2,6%, e o Android liderava com 72,4%.”. (HIGA, 2013).

Dado esse contexto, podemos afirmar que “O Android causou um grande impacto quando foi anunciado, atraindo a atenção de muita gente“ (LECHETA, 2010, p 20). E a Google juntamente com a OHA foram os responsáveis por criarem “uma plataforma de código aberto e livre para celulares, justamente para atender a todas

as expectativas e tendências do mercado atual” (LECHETA, 2010, p 20).

A partir da criação da plataforma o mercado cresceu rapidamente, celulares passaram a ganhar funcionalidades muito interessantes como por exemplo aplicativos de bancos, na qual é possível fazer consultas de saldo, pagamento de boletos, entre outras funções. Deve-se destacar também que a popularização da internet contribuiu para o desenvolvimento desse setor, que hoje está a frente do mercado de televisores e de internet.

> Hoje, há 1,5 bilhão de televisores em uso ao redor do mundo. Um bilhão de pessoas estão na Internet. Mas quase três bilhões de pessoas têm um telefone celular, tornando-o um dos produtos de consumo do mundo de maior sucesso. Dessa forma, construir um aparelho celular superior enriqueceria e melhoraria as vidas de inúmeras pessoas (OHA, 2011).

As empresas e os desenvolvedores estão otimistas com a plataforma Android, por ser flexível, robusta, de código aberto, confiável e de fácil desenvolvimento, pois utiliza a linguagem Java na qual está totalmente inserida no mercado. Possui uma boa documentação para iniciantes e uma vasta gama de fóruns, blogs e outras fontes de consultas e aprendizado. Além disso, conta com uma loja de aplicativos que não necessita de alto investimento para disponibilização de aplicações.

O Android pode ser definido como uma pilha de softwares (Software Stack), pois possui um conjunto de aplicações na qual trabalham com um objetivo específico em comum. Em uma estrutura de pilha um elemento depende do outro, e essa ideia de dependência é o que podemos chamar de pilhas de softwares. Essa pilha é composta pelo sistema operacional, middleware e aplicações, onde cada camada possui um conjunto de programas na qual suportam funções específicas do sistema operacional. Segundo Passos,

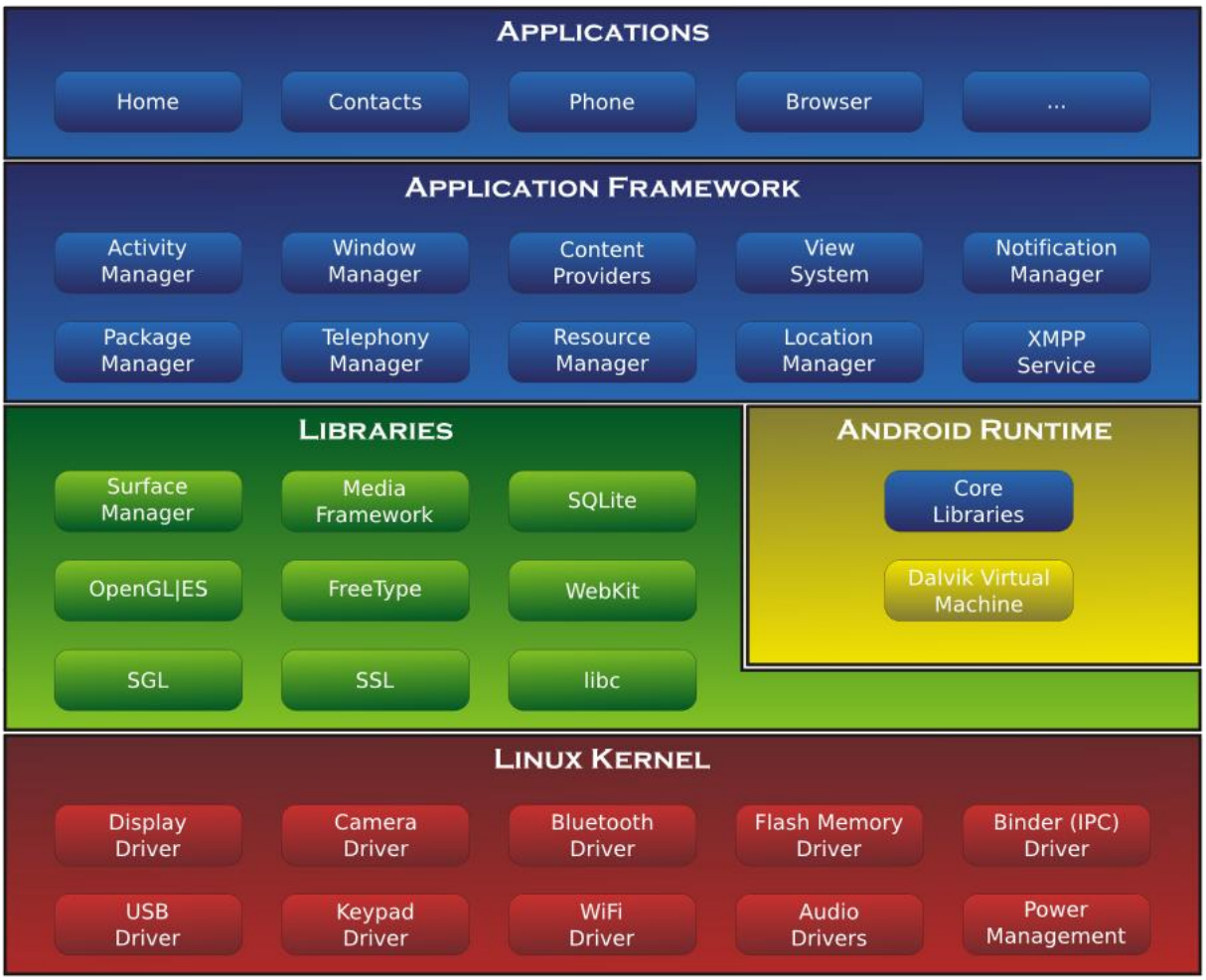
> no Android, além do Sistema Operacional ainda é oferecida uma gama de recursos para programação e desenvolvimento. Esses recursos incluem ferramentas de desenvolvimento (compiladores, emuladores, etc), classes reescritas, bibliotecas, APIs (Application Programming Interface) e Frameworks para tornar possível a confecção de softwares para o Android. (2009, p 4)

Esses recursos são disponibilizados para os desenvolvedores utilizando a linguagem de programação Java.

3.3.2 Arquitetura Android

É fundamental que o desenvolvedor conheça de forma satisfatória a arquitetura do Android. E a partir desse conhecimento possa colocar em prática no momento do desenvolvimento da aplicação, pois o bom entendimento de todas as camadas da arquitetura permitirá tirar o maior proveito dos recursos que o sistema operacional tem a oferecer, resultando em aplicações mais robustas e flexíveis. A figura abaixo detalha cada uma das camadas dessa arquitetura.

Figura 1 - Arquitetura da plataforma Android.



Fonte: Wikipedia (http://pt.wikipedia.org/wiki/Ficheiro:Android-System-Architecture.svg)

3.3.2.1 Kernel Linux

Na base da pilha temos o Kernel do Linux, a versão utilizada no Android é a 2.6, a mesma utilizada em sistemas operacionais Linux atual. "O Android utiliza o Kernel Linux como camada de abstração de hardware. Em outras palavras, o Kernel

Linux constitui o núcleo do Android, realizando todas as funções cabíveis à um sistema operacional*”* (PASSOS, 2009, p 4). Mesmo sendo utilizada a mesma versão do kernel dos sistemas operacionais para desktops, pode–se dizer que o Android não é um sistema Linux. Muitos recursos do Kernel foram eliminados, como por exemplo, o sistema nativo de janelas e outros nos quais não seriam utilizados. No entanto alguns recursos precisaram ser modificados e otimizados para um melhor desempenho em dispositivos móveis. Passos comenta que

> o Linux tem se mostrado um sistema operacional bastante robusto e estável, inclusive sendo usado em servidores e, cada vez mais, em empresas. Se, no seu surgimento, o Linux era usado praticamente só em universidades e apenas por especialistas na área (geralmente, no meio acadêmico), cada vez mais o Linux tem ocupado um lugar de destaque em outras áreas, seja em ambientes comerciais ou para uso pessoal. (2009, p 8)

Outro fator importante é que o Linux possui um bom gerenciamento de memória, modelo de segurança baseado em permissões, e suporte para bibliotecas compartilhadas, “o Linux tem um modelo de driver que permitiu aos desenvolvedores da plataforma criar uma abstração entre os dispositivos de hardware (e.g. LCDs, TouchScreen, Keyboard, etc) e os softwares que eles utilizam” (PASSOS, 2009, p 4). Outra característica importante é o fato de ele ser de código aberto, sabe-se que a comunidade que mantém o Linux é bem numerosa e tem contribuído bastante para tornar o Linux um sistema operacional com a qualidade que se tem hoje.

Alguns programas foram acrescentados ao Kernel para que ele pudesse “se adequar” aos dispositivos móveis. Dentre os principais programas, podemos destacar os seguintes:

- Alarm – Driver para prover temporizadores e funções básicas de alarmes (e.g., ligar ou desligar o aparelho).
- Ashmem (Android shared memory) – Permite que as aplicações compartilhem memória.
- Binder – Gerencia a intercomunicação entre processos. No Android todas as aplicações rodam em processos diferentes e as vezes é necessário que haja comunicação entre processos para compartilhar dados. Tipicamente, qualquer comunicação entre processos (IPC) pode causar um overhead significativo e ainda, falhas de segurança. O Binder foi criado para resolver estes problemas.
- Power Management – Gerenciador de energia - Baseado no projeto OpenBinder, entre outros. (PASSOS, 2009, p 9).

3.3.2.2 Bibliotecas

Camada composta por um conjunto de bibliotecas em C e C++ e que são utilizadas pelo sistema.

3.3.2.2.1 Bionic Libc

Uma das Bibliotecas dessa camada é a Bionic Libc, segundo Passos

> é uma implementação específica da biblioteca Libc, otimizada para uso embarcado. Novamente, que fique claro que esta biblioteca é diferente dos ambientes de programação em C encontrados em sistemas Linux, como Ubuntu ou Fedora. (2009, p 11).

Três fatores fundamentais contribuíram para que fosse decidido criar uma nova biblioteca Libc para o uso em Android.

A biblioteca Libc é mantida sob a licença GPL (General Public License). Ela é uma licença de software livre que obriga que trabalhos derivados dessa licença sejam licenciados sob a mesma. Em contrapartida a biblioteca Bionic Libc criada para Android é distribuída sob a licença BSD na qual não possui essa restrição. Os outros dois fatores foram citados por Passos:

> Tamanho – Para um desktop com GBs de memória RAM pouco importa o tamanho de aplicativos em C, mas para um dispositivo móvel é diferente. A biblioteca Bionic Libc é realmente muito pequena, tem 200K, aproximadamente metade do tamanho da biblioteca GLibc.
> Rapidez – Para aumentar a velocidade das aplicações que utilizam threads (praticamente todas hoje em dia) foi feita a re-implementação da biblioteca pthread. Enquanto a biblioteca pthread convencional tem tamanho de cerca 12 KB a biblioteca reescrita tem apenas 4 KB. (2009, p 11).

Outras funções ainda foram implementadas para uso específico no Android, como por exemplo, a capacidade de criar Logs, além da possibilidade de realizar debug em aplicações nativas.

3.3.2.2.2 Bibliotecas de funções

O Android possui algumas bibliotecas de funções, dentre elas podemos destacar três tipos, WebKit, Media Framework, SQLite, descritas por Passos:

> o WebKit é um engine (i.e., motor) de browsers. Ele é o responsável por analisar o código das páginas Web e criar a visualização que os

> navegadores exibem (processo conhecido por renderização). Em outras palavras o WebKit é o motor que faz um browser funcionar.
> Media Framework é responsável por fornecer suporte a todo tipo de arquivos de mídia. Ela é baseada no pacote de vídeos OpenCore. As bibliotecas oferecem suporte para visualizar e gravar em diversos formatos de áudio, vídeo e também imagens.
> O SQLite é um banco de dados otimizado para dispositivos móveis. Como sugere o nome, é um banco de dados mais "leve", apropriado para ambientes com menos recursos. (2009, p 13).

3.3.2.2.3 Servidores nativos

Servidores nativos, para Passos "são processos na qual rodam em modo background e realizam tarefas que controlam a entrada e saída de dispositivos." (PASSOS, 2009, p 15). Essas tarefas são:

- Gerenciador de superfícies – É responsável pelo gerenciamento da interface em geral.
- OpenGL | ES e SGL – "Bibliotecas para gráficos 2D e 3D, que podem ser combinados numa mesma aplicação. O núcleo (core) dos gráficos é formado por essas duas bibliotecas, ambas livres*".* (PASSOS, 2009, p 15).
- Gerenciador de áudio – É responsável por toda entrada e saída de áudio do dispositivo.

3.3.2.2.4 Bibliotecas de abstração de hardware

Essas bibliotecas estão para o hardware, assim como o Android SDK (Software Development Kit) está para o software. Passos descreve que

> entre o hardware e o níveis superiores da arquitetura do Android se encontra a camada de abstração de hardware. Acima da camada do Kernel Linux há vários fabricantes de hardware que irão fazer drivers para se comunicar com o software rodando no Android. (2009, p 17).

3.3.2.3 Android runtime (ambiente de execução)

Esse componente foi desenvolvido exclusivamente para Android para permitir que os programas sejam executados da melhor maneira possível e em um ambiente com memória, processamento e bateria limitados, ou seja em um ambiente para

dispositivos móveis. Nessa camada estão inseridos a Máquina Virtual Dalvik e as bibliotecas que foram desenvolvidas exclusivamente para ela. Para Lecheta

> ao desenvolver as aplicações para o Android você vai utilizar a linguagem Java e todos os seus recursos normalmente, mas depois que o bytecode (.class) é compilado ele é convertido para o formato .dex(Dalvik Executable), que representa a aplicação do Android compilada. Depois disso, os arquivos .dex e outros recursos como imagens são compactados em um único arquivo com a extensão .apk (Android Package File), que representa a aplicação final, pronta para ser distribuída e instalada. (2010, p 24).

A execução da Maquina Virtual Dalvik é feita da seguinte forma: cada aplicação Android executa uma instância da maquina virtual, isso significa que o sistema operacional é capaz de gerenciar múltiplas instâncias dela, cada qual rodando sua própria aplicação, ou seja, em cada processo.

Nessa mesma camada encontram-se as *Core Libraries*, ou as bibliotecas na qual foram desenvolvidas para serem utilizadas na Máquina de Dalvik. Essas classes são basicamente classes Java que foram reescritas para serem utilizadas pela Máquina de Dalvik, facilitando assim o rápido aprendizado do desenvolvedor com que já possui alguma intimidade com a linguagem. Passos ressalta que

> todas as classes em Java foram reescritas para uso específico pela Máquina Virtual Dalvik. Apesar de algumas classes serem bem parecidas (ou idênticas) com as classes Java, elas foram reescritas. Portanto, o Android não usa as mesmas classes Java. (2009, p 17).

Dentre essas classes podem-se citar algumas:

- java.io
- java.math
- java.net
- java.security
- java.sql
- java.util

3.3.2.4 Framework de aplicação

Nessa camada está localizado o Framework da plataforma, totalmente escrito em linguagem Java. A ideia principal do Framework é o reaproveitamento de código,

aumentando a produtividade e por consequência diminuindo o tempo de desenvolvimento e manutenção. Ele possui um conjunto de APIs que são utilizadas por aplicativos que rodam sobre a plataforma, isso significa que tanto os aplicativos nativos, quanto os desenvolvidos por terceiros utilizam esse mesmo Framework e as mesmas APIs. Para Passos esses frameworks são

> uma caixa de ferramentas (comumente chamados de toolkit) que todas as aplicações utilizam. Estas aplicações incluem implementações do celular (como por exemplo, as aplicações Home e Phone), aplicações escritas pela equipe do Android (Google) e aplicações escritas por nós, programadores. Todas as aplicações usam o mesmo framework e as mesmas APIs. (2009, p. 18)

3.3.2.5 Aplicações

É a camada de mais alto nível da arquitetura, nela estão localizadas todas as aplicações que rodam sobre o sistema operacional, isso inclui aplicações nativas e de terceiros.

3.3.2.5.1 Componentes da aplicação

Uma característica importante em aplicações para Android é o fato de elas serem capazes de referenciar componentes de outras aplicações e utilizar elementos da mesma, dessa maneira se torna muito mais fácil e prático reutilizar códigos já existentes, deixando a aplicação mais elegante e menos custosa. Dentre esses componentes podemos destacar:

- Activity (Atividades) – Uma Activity pode ser definida como uma classe responsável por desenhar a interface de um layout, assim como tratar eventos da mesma.
- Broadcast Receivers – Pode ser definido como um receptor de eventos, que é utilizado para notificar o sistema sobre alguma mudança, como, por exemplo, quando um download terminou, ou se não há uma conexão com a internet.

3.4 PLUGIN ADT

O ADT (Android Development Tools) é um plug-in para o Eclipse que foi desenvolvido para ser um ambiente de desenvolvimento integrado e poderoso para a criação de aplicações Android, ele estende as funcionalidades do Eclipse e

possibilita a criação rápida de um novo projeto, a criação de interfaces com o usuário, a criação de pacotes baseados na API do Android, o debug de aplicações e também a sua distribuição assinada digitalmente ou não.

Este plug-in fornece as suas ferramentas de maneira integrada ao Eclipse, utilizando menus, perspectivas ou processos em backgrounds. Além de fornecer as mesmas facilidades existentes nos editores do Eclipse para ambiente Java, tais como checagem de sintaxe, auto completar e integração de documentação com a API do Android. Também existem editores específicos para a criação dos arquivos de configuração do Android, sendo possível a construção desses arquivos através da utilização de editores XML ou através de uma interface gráfica utilizando arrastar e soltar, o que facilita e muito a sua criação. Segundo o Google:

> Desenvolver em Eclipse com ADT é altamente recomendável, é o caminho mais rápido para começar. Com guia de configuração de projeto, bem como ferramentas de integração e editores de XML, painel de debug. O ADT fornece um grande impulso para o desenvolvimento de aplicações Android. (ANDROID, Tradução nossa)

Além disso, o plug-in se integra totalmente ao SDK do Android, permitindo a criação de emuladores de dispositivos para a execução de testes, e também para o gerenciamento e download de novas versões da API e extras.

3.5 KSOAP2 - ANDROID

O projeto KSOAP2-Android é uma biblioteca cliente para o protocolo SOAP (Simple Object Access Protocol) na plataforma Android, foi criado inicialmente através de um fork (Software desenvolvido com base em outro, já existente, sem a descontinuidade deste último) do projeto KSOAP2, este no qual foi mantido até 2006. O projeto KSOAP2-Android praticamente emergiu de um fórum da Google Groups voltado para desenvolvimento Android, o desenvolvimento do projeto começou com o post de um usuário dizendo que muitas pessoas perguntavam sobre bibliotecas de cliente SOAP para Android, mas não encontravam nada satisfatório, portanto ele resolveu disponibilizar a sua biblioteca para a comunidade, este post data de 2007.

O seu desenvolvimento vem sendo mantido até os dias atuais, contando com atualizações constantes feitas pela comunidade de forma colaborativa, sendo o seu último (considerando a data em que esse projeto foi escrito) release em 03/05/2013.

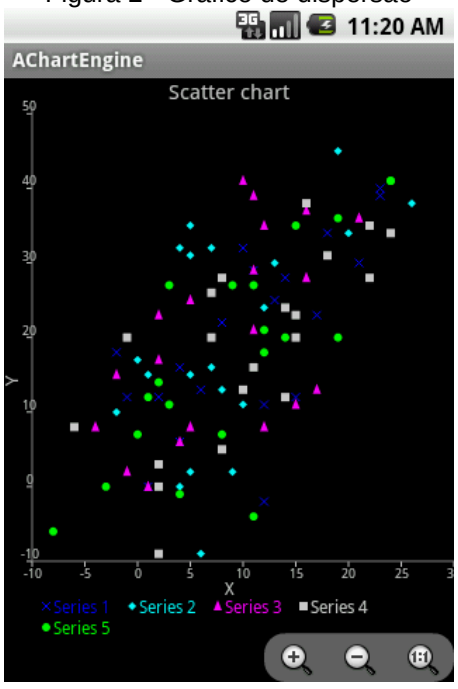
A biblioteca possui integração com o Maven, o que facilita e muito a sua inclusão no projeto, bastando apenas apontar a sua dependência no arquivo de configuração do Maven do projeto.

3.6 ACHARTENGINE

AchartEngine é uma biblioteca que permite a geração de gráficos em aplicações Android. Em seu site é possível encontrar alguns exemplos e documentações para simplificar o uso dessa biblioteca. O AchartEngine oferece suporte para gráficos de linhas, gráficos em barra, gráficos de dispersão, gráfico de pizza e muitos outros.

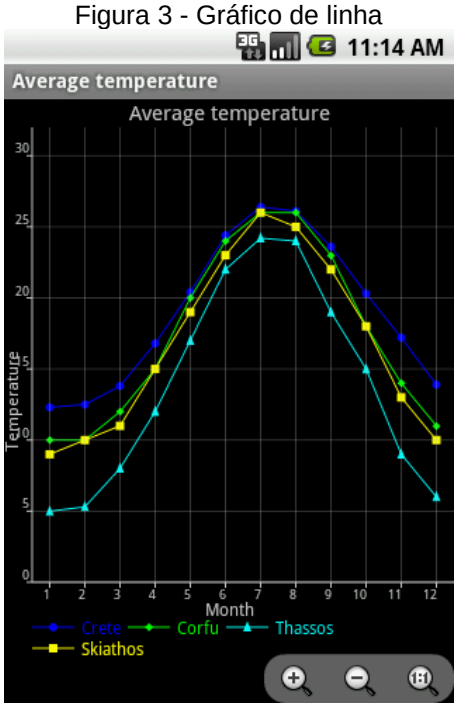
“Todos os tipos de gráficos suportados podem conter múltiplas séries, podem ser exibidas com o eixo X horizontal (padrão) ou vertical, e ainda disponibiliza muitas outras funções personalizadas.” (ACHARTENGINE , Tradução nossa).

Figura 2 - Gráfico de dispersão



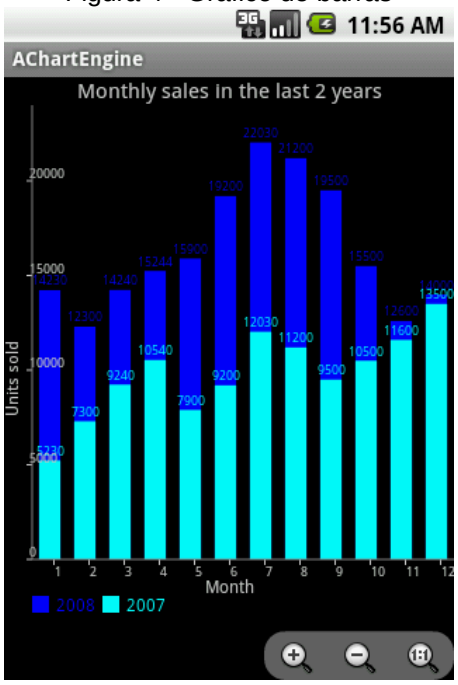
Fonte: AchartEngine (http://www.achartengine.org/).

Figura 3 - Gráfico de linha



Fonte: AchartEngine (http://www.achartengine.org/).

Figura 4 - Gráfico de barras



Fonte: AchartEngine (http://www.achartengine.org/).

3.7 MINHA UFSC

Minha UFSC é um portal web, ainda em desenvolvimento, que tem como principal objetivo a centralização das principais informações do aluno com a

Universidade, tais como empréstimos de livros da biblioteca, cardápio do dia do R.U., índice de desempenho do aluno, grade de horários, acesso ao Moodle, notícias da Universidade entre outros. Muitos desses serviços já são acessados através de *Web services*, porém outros ainda precisam ser criados para ser acessados desta forma. Outro ponto a ser analisado é a autenticação, pois como quase todos os serviços necessitam de um usuário e senha válidos, não seria prático e nem correto solicitar uma autenticação a cada serviço. Para resolver esse problema foi utilizado um processo de autenticação de usuário chamado Single sign-on (SSO). Definido por Chicout como

> à técnica de criação de um servidor de logins, de maneira que todas as aplicações que precisem de autenticação por login e senha dentro de uma organização busquem sempre o login de um só lugar. Isso evita que existam diversos pares login/senha para cada usuário dentro da organização. (2011)

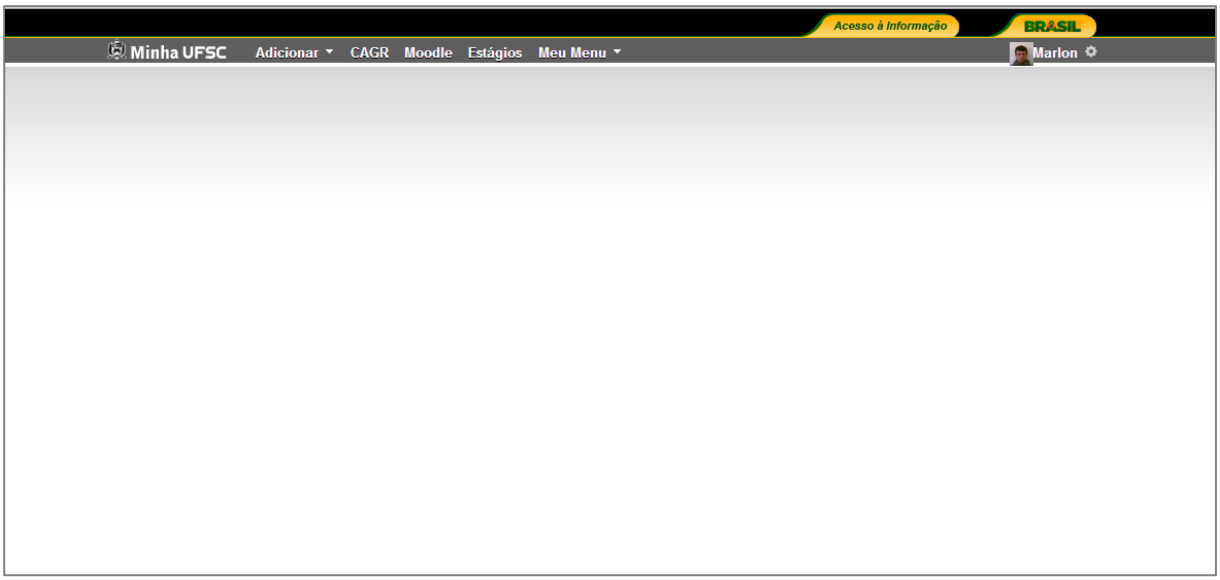
Existem dois softwares que possibilitam o uso do SSO em uma rede: OpenLDPA e Active Directory. Por trás desses softwares é utilizado um padrão chamado Lightweight Directory Access Protocol (LDPA). Esse protocolo "oferece funcionalidade de serviços de diretório, que são bases de dados de sistema, (geralmente com conteúdo sobre serviços, máquinas, usuários e permissões na rede) e SSO é uma feature que se utiliza dos dados do diretório". (CHICOUT, 2011).

Figura 5 - Tela de login do Portal Minha UFSC

Fonte: Minha UFSC (http://minhaufsc.homologacao.ufsc.br).

A interface do website será baseada em componentes. Na primeira vez em que o usuário se autenticar ele será redirecionado para uma página aparentemente vazia, haverá somente uma barra superior com menus e links.

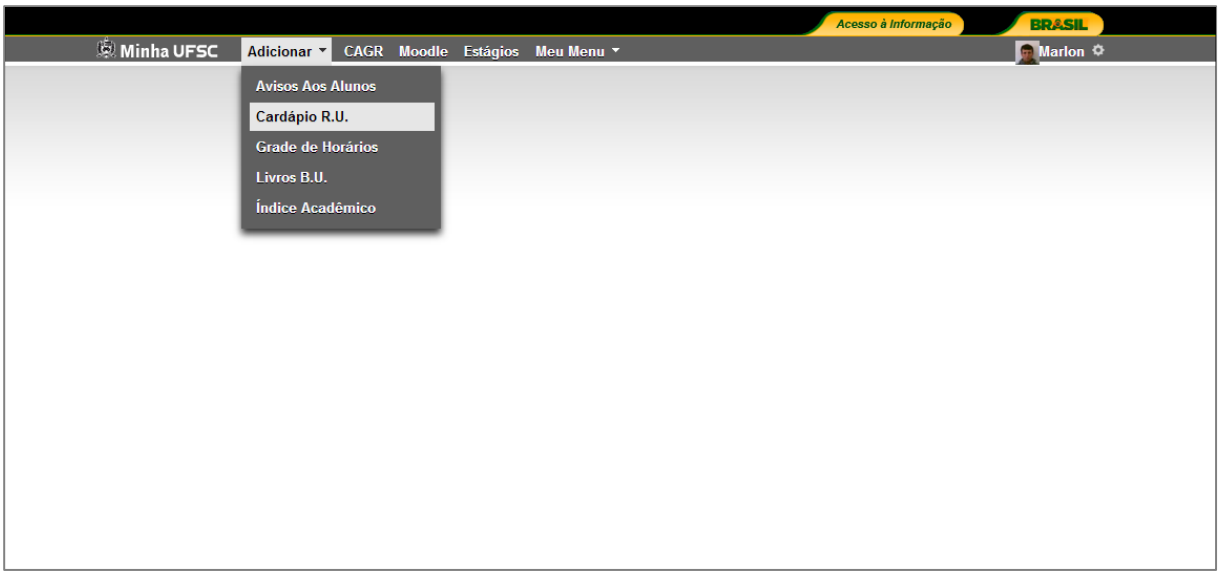
Figura 6 - Área de trabalho inicial do Portal Minha UFSC



Fonte: Minha UFSC (http://minhaufsc.homologacao.ufsc.br).

Cada usuário poderá personalizar a sua página, é nesse momento que o conceito de componentes passa a fazer sentido. Através do menu Adicionar ele poderá escolher dentre os serviços disponíveis qual deles gostaria de adicionar a área de trabalho.

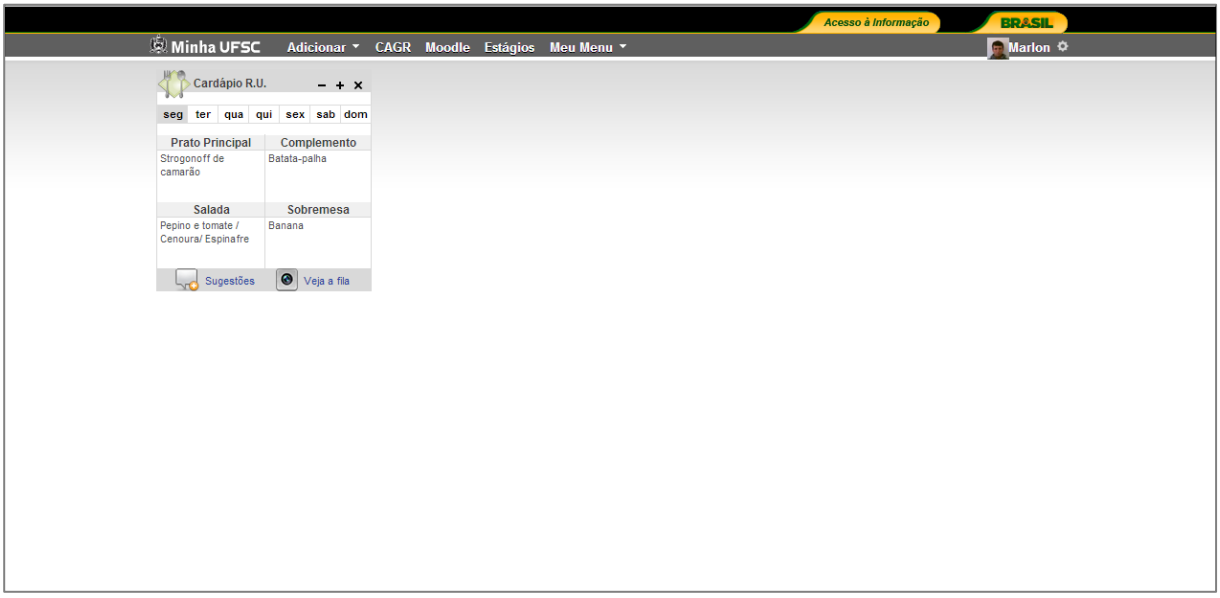
Figura 7 - Adicionando um componente no Portal Minha UFSC



Fonte: Minha UFSC (http://minhaufsc.homologacao.ufsc.br).

Também será possível remover, clicando sobre o X no canto superior direito do componente naquele em que o aluno não desejar mais utilizar, conforme pode ser visto na imagem a seguir.

Figura 8 - Componente adicionado ao Portal Minha UFSC



Fonte: Minha UFSC (http://minhaufsc.homologacao.ufsc.br).

3.8 INTERFACE DE USUÁRIO

A concepção de uma interface de usuário para sistemas móveis segue um paradigma de desenvolvimento totalmente diferente ao que se está acostumado nos outros tipos de aplicações. Ao começar pelas diferenças físicas dos dispositivos, quando estamos pensando em idealizar uma tela para uma aplicação móvel, temos que ter em mente que a área disponível para exibição da informação é bastante limitada, e os componentes de interação com o usuário devem ser escolhidos muito bem para que a experiência do usuário não acabe ficando comprometida. A maneira com que o usuário interage com a aplicação também é diferente, em aplicações móveis podemos utilizar os dedos como se fossem um mouse, podemos utilizar a câmera para ler um QR Code (código de barras bidimensional que pode ser interpretado pela maioria dos celulares através de sua câmera), ou para utilização em realidade aumentada, ou ainda podemos utilizar os mais diversos tipos de sensores, como exemplo podemos citar o smartphone Samsung Galaxy S4, anunciado em 14 de março de 2013, este dispositivo possui 9 sensores, desde sensor de luminosidade até sensor de pressão e de campo magnético. Todas essas funcionalidades e sensores podem ser utilizados para enriquecer uma aplicação, e tornar a experiência com o usuário muito mais imersível e agradável.

O Google fornece uma série de guias de desenvolvimento para auxiliar os desenvolvedores a construir aplicações excepcionais. Embora a aplicação Minha UFSC – Android não faça a utilização de nenhum sensor especifico, neste projeto iremos utilizar todos esses conceitos criados e disponibilizados pela Google para auxiliar na concepção de interfaces. Este guia segue alguns princípios básicos:

- Surpreenda o usuário, um design bonito, uma animação cuidadosamente pensada. Efeitos sutis contribuem para que o usuário tenha uma sensação agradável ao interagir com a aplicação;
- Objetos reais são mais divertidos do que os botões e menus, pois reduzem o esforço cognitivo para realizar uma tarefa;
- Possibilitar pequenas customizações, as pessoas gostam de adicionar toques pessoais porque os ajuda a se sentir em casa e no controle;
- Conheça o usuário e saiba as suas preferências ao longo do tempo. Ao invés de solicitar para fazer as mesmas escolhas varias vezes;

- Frases curtas com palavras simples, sentenças muito longas tendem a ser ignoradas;
- Considere a utilização de imagens para explicar ideias, elas chamam mais a atenção e pode ser muito mais eficiente do que as palavras;
- Decida pelo usuário, mas deixe-o ter a palavra final;
- Não sobrecarregar o usuário de informações e mostrar apenas o necessário;
- O usuário deve reconhecer as telas e deve facilmente saber onde está;
- Se algo tem a mesma aparência, portanto deve agir da mesma forma. Deve-se evitar a criação de componentes e telas que tenham a mesma aparência, porém com comportamento diferente.

Um aplicativo fácil de usar não é o suficiente, é necessário possibilitar que os usuários experimentem coisas novas e usem os aplicativos de diversas maneiras criativas. É importante permitir que as pessoas combinem aplicações em novos fluxos de trabalho. Ao mesmo tempo, o usuário deve sentir a aplicação pessoal, dando a eles acesso a tecnologia.

4. DOCUMENTO DE REQUISITOS

A partir deste capitulo será enfatizado a análise da aplicação “Minha UFSC”, que contemplará documentações de requisitos, regras de negócio, assim como uma descrição do sistema e um aprofundamento em seus requisitos, tanto funcionais como não funcionais.

4.1 DESCRIÇÕES DO SISTEMA

A aplicação “Minha UFSC” é um software de celular, para o sistema operacional Android, que tem como foco principal ser um novo canal de comunicação e interação do aluno de graduação com a Universidade. Nesta aplicação estarão disponíveis algumas das principais funcionalidades e serviços utilizados pelo aluno. As funcionalidades inseridas na aplicação serão voltadas para a mobilidade, para que se encaixe melhor no perfil de utilização de um celular, isso significa que a informação exibida deve ser objetiva e de fácil acesso, para que o usuário possa utilizar a aplicação até mesmo dentro do ônibus até enquanto caminha em direção à sala de aula.

O publico alvo desta aplicação é o usuário de graduação da UFSC, seja ele a distância ou não, de qualquer curso e que possua um celular smartphone com Android.

As informações devem ser obtidas através dos servidores da UFSC, por interfaces Web service, disponibilizadas pelos mesmos. Com isso é possível garantir o sincronismo de informações, fazendo com que a informação que o usuário vê na aplicação é a mesma exibida no site da UFSC, no CAGR (Sistema de Controle Acadêmico da Graduação), no site da B.U. ou até mesmo no site do R.U..

Com essas informações o usuário poderá fazer todo o gerenciamento da sua grade de horário, sabendo quais são as aulas da semana, quais são os professores da disciplina e qual é a sala. Hoje a única maneira de obter essa informação é através do CAGR. Como o mesmo não possui interface móvel, é pouco usual ter que ficar acessando este portal sempre que precisar consultar a grade de horario, o que geralmente necessita de muito tempo. Para contornar estes problemas os alunos geralmente “repessam” as informações da grade de horário para outro sistema, geralmente acessível pelo celular, como o Google Agenda, ou até mesmo realizam a sua impressão. O problema dessas medidas, é que a cada inicio de semestre a atualização da grade de horário deve ser refeita. Com essa funcionalidade implementada na aplicação “Minha UFSC”, e tendo a sua informação totalmente sincronzada com os servidores da UFSC é possível resolver esse problema.

A atividade principal do R.U. da UFSC é o atendimento aos alunos da Universidade fornecendo alimentação durante o horário das principais refeições.

> Ele favorece a manutenção da saúde de seus usuários através do fornecimento de uma alimentação balanceada e diversificada, produzida dentro de um padrão de controle qualidade, preocupando-se com a heterogeneidade de hábitos alimentares presentes em nosso estado. Contribui também na promoção da qualidade de Ensino, Pesquisa e Extensão, através da abertura de campos de estágio para as mais diversas disciplinas. (PRAE, 2013)

O que o caracteriza como um importante serviço. Portanto é de extremo valor que as suas informações estejam disponibilizadas na aplicação “Minha UFSC”. Essas informações contemplaram todo o cardápio da semana, podendo que o usuário consulte o cardápio atualizado direto pelo celular.

Outro serviço importante da Universade é a Bilioteca Universitaria, também conhecido como Sistema de Biliotecas da UFSC, SiBi. Este sistema é composto por noves bibliotecas, de diversas cidades.

> O SiBi conta atualmente com 48.265 usuários cadastrados, sendo que 85,28% são alunos, destes 47,82% são alunos de graduação presencial. Os alunos na modalidade à distância também são atendidos pelo SiBi, seja pelo empréstimo de acervo físico ou pelo acesso ao acervo virtual. (PORTAL BU, 2013)

O que significa aproximadamente 25mil usuários em potencial à aplicação “Minha UFSC”, e que utilizam os serviços da B.U.. Levando em consideração esses usuários a aplicação disponibilizará um sistema de consulta de empréstimo, onde o usuário acessará as informações relacionadas ao empréstimo realizado tais como nome do livro, data do empréstimo, data de devolução etc.

A segurança também é um requisito muito importante para aplicação, se formos levar em consideração que os dados expostos são dados pessoais da vida acadêmica do usuário, é imprescindível adotar um critério rigoroso de privacidade de dados. Por se tratar de uma aplicação mobile, instalada no smartphone do usuário, que se comunica através de *Web services*, diversos pontos devem ser considerados para se atingir um nível mínimo de segurança aceitavel, pois por mais que o programador utilize boas práticas de desenvolvimento, algumas informações importantes sempre acabam sendo armazenadas no código fonte.

Dentre essas informações podem ser senhas, endereços ou qualquer outro tipo de informação que ofereça uma porta de entrada para um possível usuário mal intencionado. Por mais que elas estejam compiladas, e no caso do Android, dentro de um arquivo.dex compactado em um arquivo .apk, elas podem facilmente serem descompiladas através de um processo chamado engenharia reversa. Afim de dificultar esse tipo de técnica será feito o ofuscamenteo de código, que visa embaralhar todo o código sem alterar o seu comportamento para dificultar a sua compreensão.

O principal ponto que deve ser levado em consideração em relação a segurança é a da comunicação entre a aplicação “Minha UFSC” e os servidores da UFSC. Como essa comunicação é feita através da internet ela está sempre sujeita a interceptações. Portanto, de nada adianta possuir o código ofuscado se toda comunicação pode ser interceptada e o usuário pode ter os seus dados roubados. Por isso toda comunicação deve ser feita através do protocolo seguro https, este protocolo segue o princípio de chave pública e chave privada, que garante que só o

destinatário da mensagem consiga ler as informações que estão sendo enviadas. Com isso tem-se a garantia de sigilo e de privacidade.

Outro ponto importante a ser considerado no quesito de segurança é o mecanismo de autenticação, tanto do usuário na aplicação quanto a aplicação nos serviços da UFSC. A aplicação, para se comunicar com os serviços da UFSC, possui um usuário específico. Esse usuário possui permissão para métodos específicos dentro de cada serviço, sendo que são esses métodos que irão compor todas as funcionalidades existentes nesta aplicação. Se a aplicação se autentica diretamente nos serviços da UFSC, existe um grande problema, que é o armazenamento deste acesso direto no código fonte, por mais que ele seja ofuscado ou até mesmo criptografado, ele ainda é passivel de ser capturado.

Um usuário malicioso que consiga capturar essas informações pode ter acesso a todos os dados de qualquer aluno de graduação da UFSC. Para resolver este problema, optamos por criar um serviço *middleware*, chamado de “MinhaUFSC-Auth”, no qual será responsável por fazer a autenticação nos serviços da UFSC, além de validar a autenticação do usuário na aplicação. Ele funcionará como um proxy, recebendo toda a requisição da aplicação e redirecionando para serviços da UFSC. A vantagem de sua utilização é que o usuário não terá acesso ao código fonte que se comunica com a Universidade, pois este código está hospedado em um servidor na web, como um serviço intermediario. Assim é garantida a segurança do mecanismo de autenticação.

4.2 REQUISITOS FUNCIONAIS

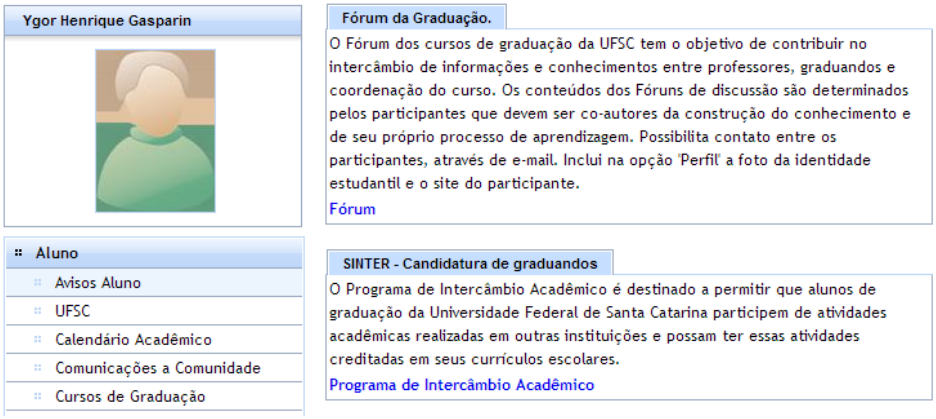
A aplicação “Minha UFSC” contemplará sete funcionalidades básicas, que visam facilitar o dia-a-dia de um aluno de graduação. Essas funcionalidades foram escolhidas pelos autores, criadores da aplicação, baseando-se em suas proprias experiencias acadêmicas. Todas essas especificações foram criadas pensando em todas as dificuldades que foram encontradas durante a Universidade. O escopo dessa aplicação é de sete funcionalidades as quais os autores julgam ser as principais, que são:

- Perfil: Nesta funcionalidade o usuário terá acesso as suas informações que estão cadastradas na UFSC. Informações tais como matrícula, curso,

situação, ano de ingresso, semestres cursados, provável data de formatura, localização, foto e sexo. Essas informações não são necessariamente informações que precisam ser acessadas com frequência, porém a maioria das aplicações hoje em dia, seguem esse princípio de sempre ter uma tela exclusiva para visualização do perfil. Nesta tela o usuário não podera editar as informações, apenas visualizar. A ideia de exibir a foto é para trazer uma proximidade com as redes sociais, por isso ela também esta inclusa nessa funcionalidade.

- Avisos: As informações de avisos são as primeiras a serem exibidas ao fazer login no serviço. Nesse site podem ser exibidas informações relacionadas a estágios, classificados, novidades e eventos da UFSC, novos serviços, resposta à dúvidas, rematrícula e etc. Como pode ser visto na imagem a seguir.

Figura 9 - Perfil do aluno na página do CAGR



Fonte: CAGR/UFSC (http://cagr.sistemas.ufsc.br/)

Essas informações também deverão ser acessíveis através do celular, pois o objetivo é para o aluno estar sempre informado, o que está acontecendo com a Universidade e o seu curso, e como essas informações disponíveis através dos dispositvos móveis, de maneira rápida e f'ácil, contribuirá para que isso ocorra.

- Histórico escolar: O objetivo dessa funcionalidade é possibilitar o acompanhamento semestre a semestre do desempenho escolar. O usuário

deve saber quais disciplinas foram cursadas em cada semestre, quais foram as notas de cada disciplina e qual o Índice de Aproveitamento e o Índice de Aproveitamento Acumulado do semestre. O objetivo é incentivar o acompanhamento de notas e fazer com que o aluno se preocupe com o seu desempenho escolar. Manter um I.A elevado é muito importante, não só pelo fato de indicar que o aluno realmente está tendo algum proveito na Universidade, mas também pelo fato de que algumas oportunidades só estão acessíveis para quem possui um I.A elevado, por exemplo, hoje na Universidade, estágios só podem ser realizados com alunos de I.A maior do que 6 (seis), e em alguns casos somente a quem possuir I.A maior do que 7 (sete). Muitas empresas consideram essa informação no momento da contratação, então possuir um número elevado nesse indicador pode contar bastante no curriculum. Muitos alunos não tem o costume de acompanhar o seu I.A, e em algumas vezes nem sabem em que nível estão. Possibilitando essa informação ser acessada de maneira rápida e simples, esperamos diminuir cada vez mais esse número de alunos que não acompanham o seu desempenho.

- Grade de horários: A cada mudança de semestre uma nova grade de horário é disponibilizada, fazendo-se necessário alterar o que se havia feito no semestre anterior. Hoje muitas pessoas imprimem a grade de horário, escrevem no caderno, cadastram as diciplinas em agendas fisicas ou digitais e em alguns casos até mesmo tiram foto da grade de horário disponibilizada no serviço CAGR .

Figura 10 - Grade de horários do aluno no CAGR

	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
07:30						INE5632-09238B CTC/CTC111
08:20						
09:10						
10:10						
11:00 11:50						
13:30						
14:20						
15:10						
16:20						
17:10 18:00						
18:30	EGC5008-09238 CTC/CTC306	INE5642-09238 CTC/CTC204	EGC5008-09238 CTC/CTC303			
19:20	EGC5008-09238 CTC/CTC306	INE5642-09238 CTC/CTC204	EGC5008-09238 CTC/CTC303			
20:20						
21:10 22:00						

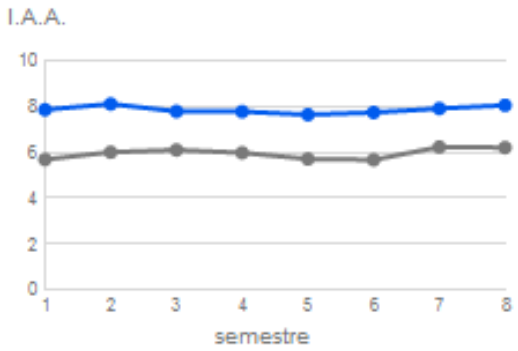
DISCIP.	TURMA	NOME DA DISCIPLINA	PROFESSOR
EGC5008	09238	Qualidade da Informação	ROGERIO CID BASTOS
INE5632	09238B	Projetos II	RENATO CISLAGHI
INE5642	09238	Gerência de Redes de Telecomunicações	CARLOS BECKER WESTPHALL

Fonte: CAGR/UFSC (http://cagr.sistemas.ufsc.br/)

Com essa informação centralizada na aplicação o usuário não precisa mais fazer o seu gerenciamento. A grade de horários é obtida automaticamente pela aplicação de acordo com o usuário logado, e estará sempre sincronizada com as disciplinas do semestre atual. Devem ser exibidas informações como nome da disciplina, horário e dia da disciplina, nome dos professores e localização. Todas as informações necessárias para o usuário saber qual é a proxima discíplina, horário que inicia e termina, em que lugar vai ser a aula e quem vai ser o professor.

- Estatistica I.A.A: Não existe uma funcionalidade semelhante a essa nos serviços atuais da UFSC, ela está sendo introduzida com o novo portal *MinhaUFSC* e com o aplicativo “Minha UFSC”. O Objetivo desta funcionalidade é possibilitar um acompanhamento visual do Índice de Aproveitamento Acumulado, além de poder fazer uma comparação com o I.A.A médio do curso . Com isso o aluno poderá saber como está as suas notas em relação aos demais alunos do seu curso.

Figura 11 - Estatística do I.A.A. do portal Minha UFSC



Fonte: Minha UFSC (http://minhaufsc.homologacao.ufsc.br).

- Cardápio do Restaurante Universitário: “O restaurante universitário serve aproxidamente 10 mil refeições por dia.” (PRAE, 2013), grande parte dessas refeições é destinadas a alunos de graduação, que portanto representam usuários em potenciais desta aplicação. O objetivo desta funcionalidade é possibilitar que esses usuários acessem o cardápio do restaurante de qualquer lugar, de maneira rápida e descomplicada. Ela deve fornecer informações como prato principal, complemento, acompanhamento, salada e sobremesa, de todos os dias da semana atual, tendo em vista que a validade do cardápio é de uma semana. Uma vantagem proporcionada por esta funcionalidade é que hoje o cardapio só é acessivel pelo site do R.U.. Muitas vezes ele não está atualizado e acaba gerando um certo desconforto para quem o utiliza e ao chegar no restaurante vê que o cardapio é outro. Com esta funcionalidade esse problema deixará de ocorrer, pois essas informações são obtidas direto do sistema da UFSC, o que garantem a veracidade e a atualidade da informação.

Figura 12 - Cardápio do R.U.

Cardápio de 21/10 /2013 a 27/10/2013

Cardápio sujeito a alterações

Link da Câmera

	Acompanhamento		Complemento	Prato principal	Salada	Sobremesa
Segunda-feira	Arroz Branco e Integral	Feijão / Lentilha	Polenta	Coxa e sobrecoxa assada com ervas	Alface/ Pepino	Iogurte
Terça-feira						
Quarta-feira						
Quinta-feira						
Sexta-feira	Arroz Branco e Integral	Feijão / Lentilha	Batata palha	Picadinho de carne e linguiça	Beterraba orgânica/ Acelga	Laranja
Sábado	Arroz Branco e Integral	Feijão / Lentilha	Creme de milho	Filé de peixe in natura ao molho de ervas	Agrião/ Rabanete	Morango
Domingo	Arroz Branco e Integral	Feijão / Lentilha	Farofa	Carreteiro	Vagem	Gelatina

Fonte: Restaurante Universitário (http://ru.ufsc.br/ru/).

- Biblioteca Universitária: Com esta funcionalidade o usuário conhece quais são os livros que ele solicitou empréstimo, e quais sãos os débitos (empréstimos vencidos) com a biblioteca. O objetivo é evitar que o aluno acabe esquecendo que realizou algum empréstimo, e que acabe sendo prejudicado e até mesmo prejudicando outros alunos que tenham interesse neste mesmo livro.

4.3 REQUISITOS NÃO FUNCIONAIS

- Autenticação: Para ter acesso as informações da aplicação o usuário deve obrigatóriamente estar autenticado. Essa autenticação deve ser feita através da validação do seu número de matrícula e senha, na qual são necessários devido ao tipo de informação exposta. Algumas informações até são de domínio publico, como cardápio do R.U. Porém, visando a facilidade do desenvolvimento, essas informações também só estarão disponiveis mediante autenticação.

- Confidencialidade: A matricula e senha serão armazenadas apenas na memória e estarão disponíveis durante todo o ciclo de vida da aplicação. Esssas informações serão codificadas em base64, e o próprio isolamento de

processos do sistema operacional garantirá que elas não sejam roubadas por qualquer outra aplicação. Para garantir a segurança a nível de comunicação, deve ser utilizado protocolo https, além da criação do “MinhaUFSC-Auth”, descritor anteriormente.

- Ofuscar código: O código deve ser ofuscado, a fim de dificultar uma possível ação de engenharia reversa.

- Portabilidade: A aplicação pode ser executada tanto em smartphone quanto em um tablet, com processamento mínimo de 600MHz e com mínimo de 250MB de memória ram. O sistema operacional é Android 2.3.3 ou superior.

- Usabilidade: O sistema deve ser fácil de usar, e todas as suas funcionalidades principais devem ser acessíveis em no máximo dois toques.

5 VISÃO DE ANÁLISE

Após as definições do projeto, como requisitos e regra de negócios, é possível iniciar a análise e modelagem do sistema, para isso será utilizado a linguagem de modelagem chamada UML.

5.1 DIAGRAMA DE CASOS DE USO

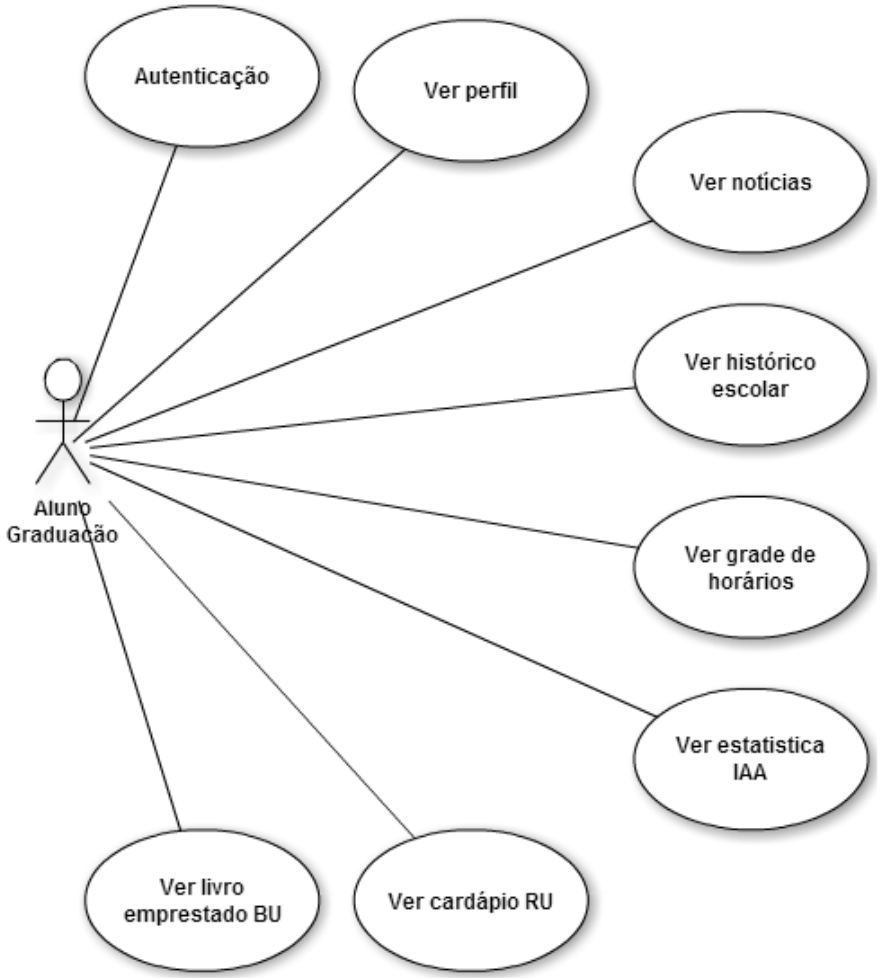
O diagrama de caso de uso, “é um diagrama de modelagem dinâmica de sistema em alto nível de abstração, que modela as funcionalidades do software (casos de uso) com os elementos externos que interagem com o software (atores).” (SILVA, 2009, p 35).

Todas as funcionalidades descritas no capítulo anterior podem ser interpretadas como um caso de uso diferente, pois elas representam as principais funcionalidades do software.

Analisando os requisitos podemos identificar que o agente externo que interage com o software é o próprio usuário aluno de graduação. Portanto, ele será o único ator do diagrama de casos. Cada balão do diagrama representa um caso de

uso, o personagem representa um ator, e as linhas representam a interação, como pode ser visto na imagem a seguir.

Figura 13 - Diagrama de caso de uso



Fonte: elaborado pelos autores (2013).

No diagrama de casos de uso da aplicação “Minha UFSC” podemos perceber que a *Autenticação,* foi descrita como um requisito não funcional, ela aparece no diagrama como um caso de uso. Isso se da pelo fato da grande complexidade em desenvolver o mecanismo de autenticação, levando em conta os requisitos de segurança propostos no capítulo anterior, e pelo fato de que a sua implementação será muito parecida com os demais casos de uso. Por isso optamos por modelá-la desta forma.

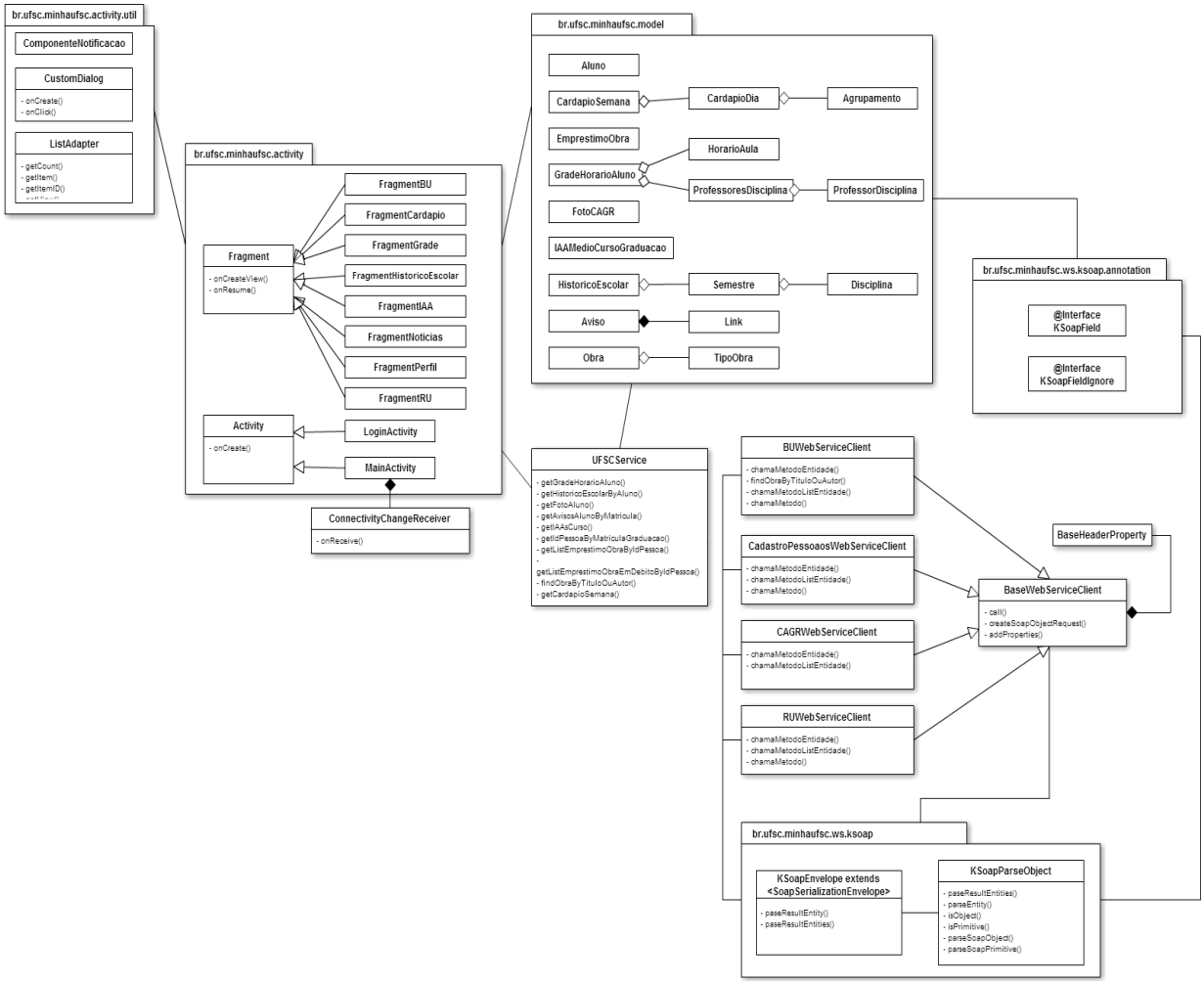
5.2 DIAGRAMAS DE CLASSES

Segundo Silva, “o diagrama de classes representa a estrutura do código e o relacionamento entre classes de forma mais explicita” (2009, p 245). Com esse diagrama é possível ter uma visão de como os requisitos irão funcionar, e de que maneira isso se reflete em estrutura de código e com isso fornecer aos desenvolvedores uma base para projetar o sistema.

O diagrama deve apresentar todas as classes do sistema, junto com seus métodos e atributos, além de apresentar a relação entre classes e pacotes.

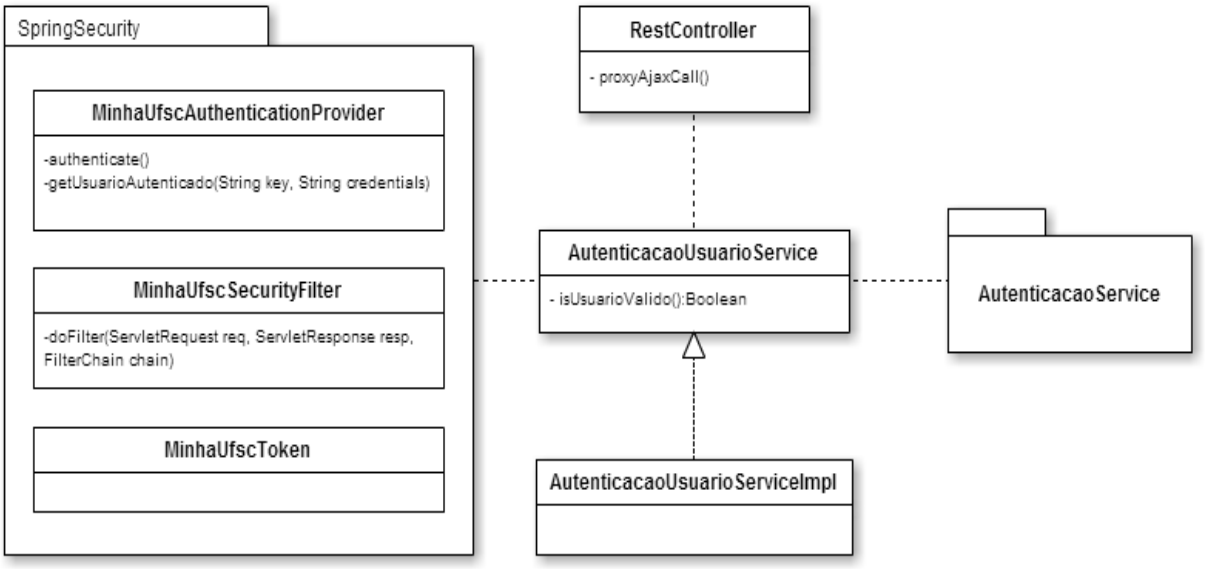
A figura abaixo apresenta o diagrama de classes da aplicação “Minha UFSC”. É possível notar que neste diagrama foram omitidos alguns métodos e atributos de algumas classes, pois não agregavam em nada na compreensão do sistema e apenas dificultavam a visualização do diagrama.

Figura 14 - Diagrama de classes



Fonte: elaborado pelos autores (2013).

Figura 15 - Diagrama da aplicação MinhaUFSC-Auth



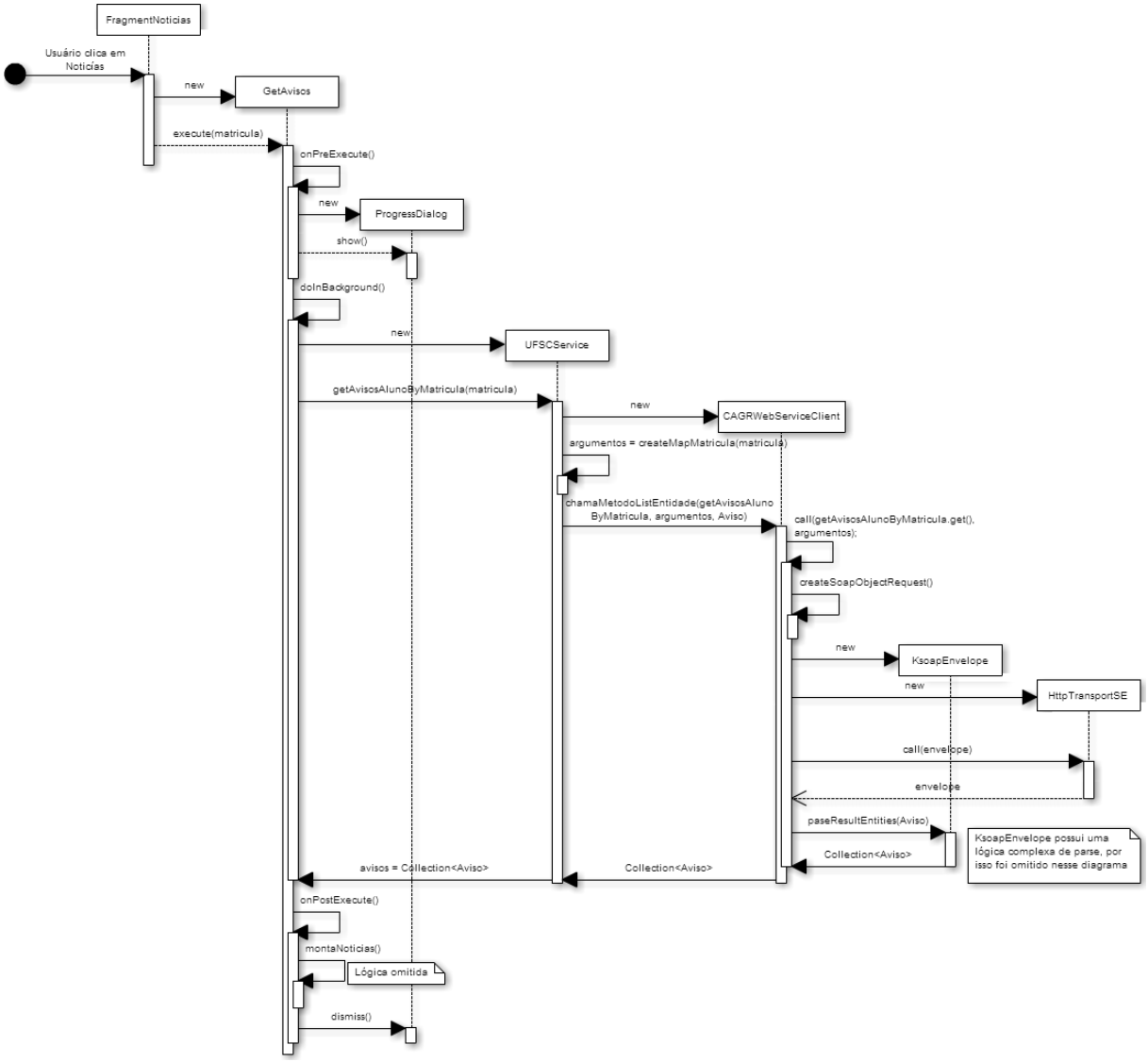
Fonte: elaborado pelos autores (2013).

5.3 DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência é um diagrama temporal que representa a ordem em que as mensagens são passadas de objeto a objeto. Ele visa facilitar a compreensão de um determinado comportamento do sistema, tendo em vista que este comportamento pode estar distribuido em vários métodos e objetos ao longo do sistema.

Por ele representar um comportamento, ou em outras palavras, uma funcionalidade, ele é sempre construido a partir de um caso de uso do diagrama de *Casos de Uso,* a figura a seguir apresenta o diagrama de sequência do caso de uso ver notícias.

Figura 16 - Diagrama de sequencia para funcionalidade de avisos



Fonte: elaborado pelos autores (2013).

6 AMBIENTE DE DESENVOLVIMENTO

Neste capítulo será apresentada a metodologia utilizada para o desenvolvimento do aplicativo “Minha UFSC”, e uma breve orientação para instalação e configuração do ambiente de desenvolvimento.

6.1 JAVA (JDK)

A linguagem de programação utilizada no desenvolvimento do aplicativo foi *Java,* pois é uma linguagem gratuita e como foi utilizada ao longo do curso de

Sistemas de Informação da UFSC, há certo domínio da linguagem pelos alunos que desenvolveram o aplicativo. Além disso, ela possui uma API voltada a plataforma *Android*.

Independente da plataforma, para o desenvolvimento de projetos em *Java* é necessário a instalação do JDK (Java Development Kit), que está disponível em http://www.oracle.com/technetwork/java/javase/downloads/index.html Acesso em: 09/10/2013.

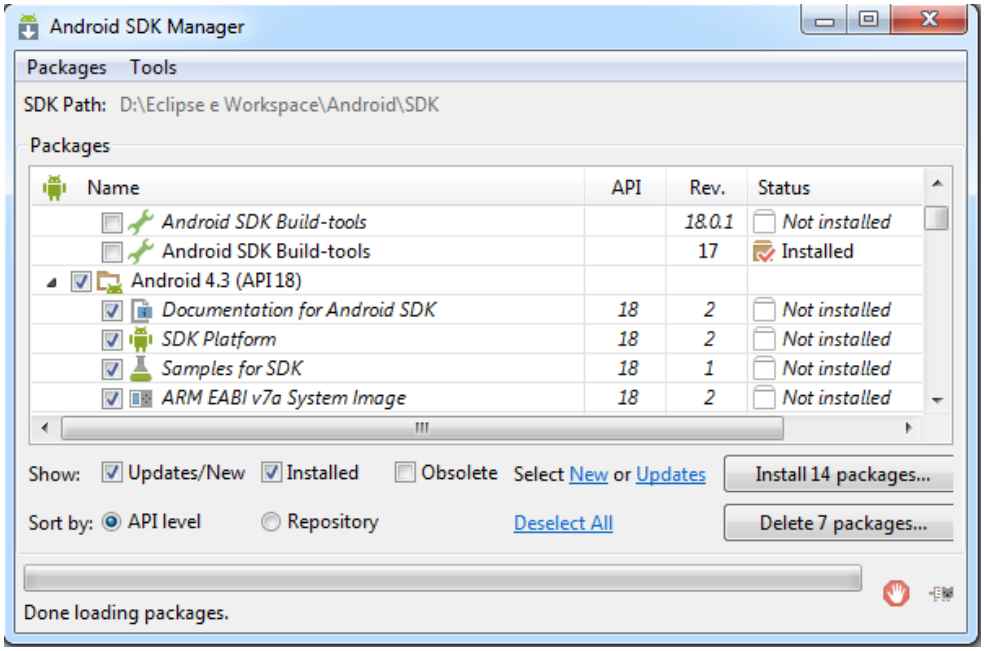
Também foi definida uma variável de ambiente para o JDK. Instruções sobre a definição estão disponíveis em: <http://devsubdev.wordpress.com/2010/06/04/dica-configurando-as-variaveis-de-ambiente-do-jdk-no-windows/> Acesso em: 09/10/2013.

6.2 SDK ANDROID

Para o desenvolvimento de aplicativos para a plataforma Android é necessária a presença do SDK do Android. Esse pacote está disponível em: <http://developer.android.com/sdk/index.html> Acesso em: 09/10/2013.

Esse pacote é apenas um arquivo compactado que deverá ser extraído para algum diretório do computador. No interior dessa pasta ainda não contém nenhuma das APIs, para obtê-las é necessário executar o arquivo “SDK Manager.exe”. É também através da execução dele que são obtidas atualizações das versões lançadas.

Figura 17 - SDK Manager

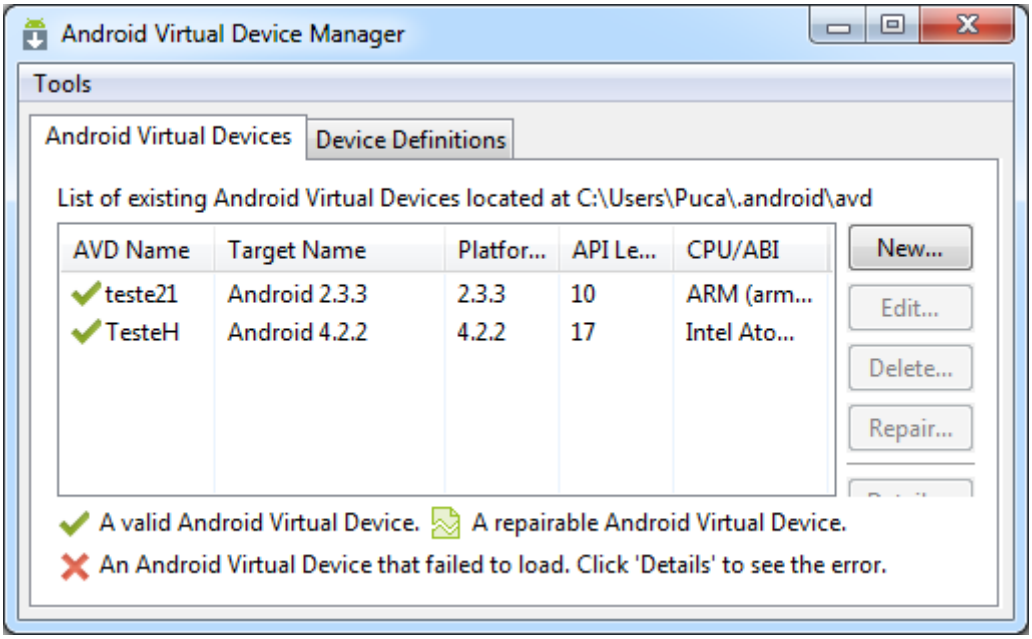


Fonte: elaborado pelos autores (2013).

6.3 DISPOSITIVO VIRTUAL (AVD)

Uma utilidade interessante que está presente no SDK é a possibilidade de simular um dispositivo móvel no seu computador. Para criar um AVD (Android Device Virtual), é preciso executar o arquivo “AVD Manager.exe”. Ao executá-lo tem-se a possibilidade de criar dispositivos com diferentes configurações, desde a versão do Android até a quantidade de espaço interno que o dispositivo terá, ou ainda editar um existente.

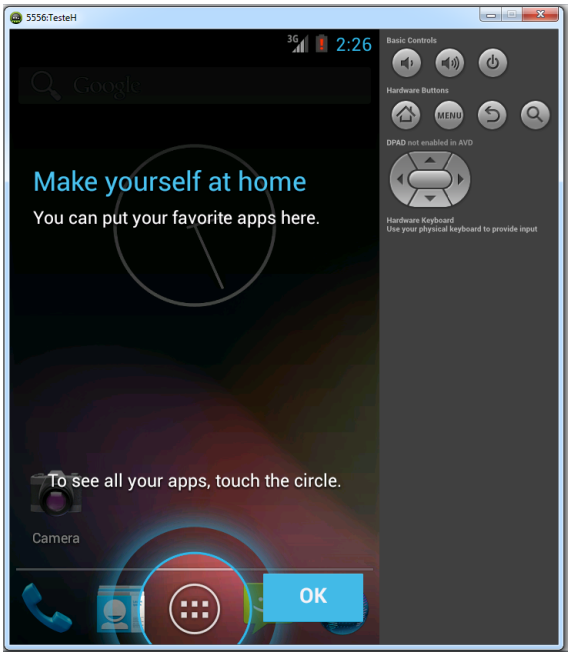
Figura 18 - Criação ou edição de um AVD



Fonte: elaborado pelos autores (2013).

Após configurar um novo dispositivo, basta selecioná-lo na lista e clicar no botão “Start”.

Figura 19 - AVD após iniciá-lo



Fonte: elaborado pelos autores (2013).

6.4 MAVEN

A ferramenta Maven foi escolhida por ter um bom gerenciamento de dependências, e isso faz uma grande diferença no dia-a-dia do desenvolvimento. Além disso, ela irá trabalhar em conjunto com o arquétipo que será utilizado como base para o desenvolvimento do aplicativo, permitindo que seja gerado um arquivo .apk, utilizando plug-ins do Maven para ofuscar, assinar e compactar o código do aplicativo, oferecendo uma maior segurança em um espaço menor, tornando o aplicativo mais leve e seguro.

O arquétipo utilizado no “Minha UFSC” está disponível em: <https://github.com/akquinet/android-archetypes> Acesso em: 09/10/2013.

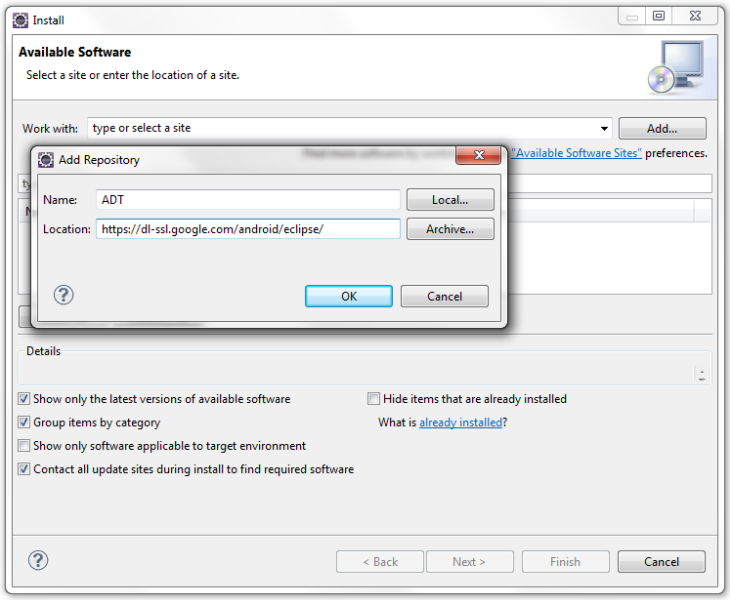
6.5 IDE ECLIPSE

A IDE escolhida para o desenvolvimento do aplicativo foi o Eclipse, por se tratar de um software gratuito, estável e suportar as tecnologias nas quais irão ser utilizadas no desenvolvimento do aplicativo. Outro fator importante é que o Eclipse foi utilizado para desenvolvimento ao longo de toda a graduação, facilitando a ambientação com a mesma. A versão utilizada foi o Eclipse Juno 64bits, seu instalador está disponível em <http://www.eclipse.org/> Acesso em: 09/10/2013.

6.5.1 Plug-in Android para Eclipse (ADT)

O plug-in ADT está disponível em: <https://dl-ssl.google.com/android/eclipse/> Acesso em: 09/10/2013. Ele deve ser instalado através da opção “Install New Software” no menu “Help” adicionando o endereço no campo “Location".

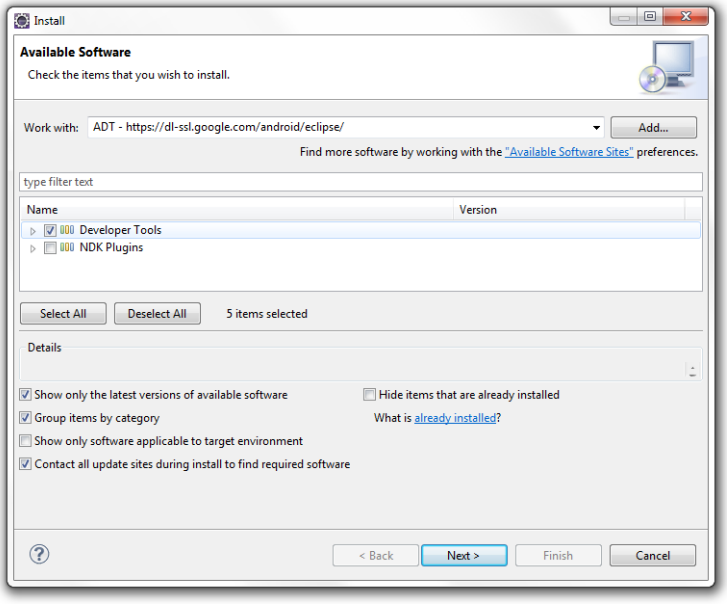
Figura 20 - Adicionando a endereço do ADT



Fonte: elaborado pelos autores (2013).

Dentre os resultados retornados da consulta, deve ser selecionado o plug-in “Developer Tools”.

Figura 21 - Resultado da busca do ADT



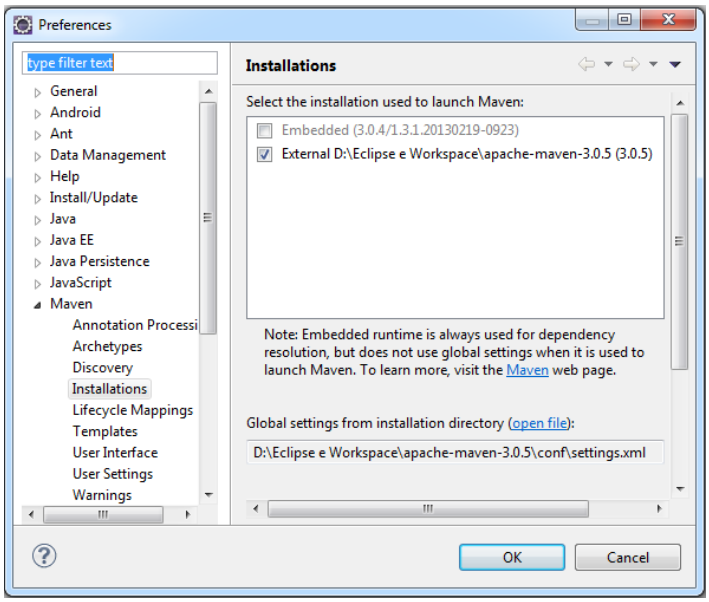
Fonte: elaborado pelos autores (2013).

Após a instalação do plug-in é necessário referenciar onde está instalado o SDK do Android. Navegue até o menu “Windows/Preferences” no item “Android” no menu do lado esquerdo e referencie o diretório no campo “SDK Location”.

6.5.2 Plug-in do Maven

Para que o Eclipse e o Android executem de forma correta com o Maven, foi necessário utilizar o plug-in “Android Configurator for M2E”, que está disponível através da “Marketplace”, que pode ser definido como uma “loja” de plug-ins. Por fim é necessário referenciar a pasta onde o Maven foi instalado, para isso é indicado o local da instalação no item “Maven/installations” em “Preferences”.

Figura 22 - Adicionado o diretório do Maven



Fonte: elaborado pelos autores (2013).

7 DESENVOLVIMENTO

Neste capítulo serão descritos os passos e códigos para o desenvolvimento do aplicativo “Minha UFSC”. Todo o projeto está hospedado em: <https://www.assembla.com/code/minha-ufsc/subversion/nodes> Acesso em: 24/10/2013, que permite o controle de versões de aplicações gratuitamente.

Durante esse capítulo, serão feitas referências aos arquivos hospedados nesse site.

7.1 API ANDROID – CONCEITOS BÁSICOS DE COMPONENTES

Nas próximas seções serão apresentados os conceitos dos componentes utilizado no desenvolvimento do aplicativo “Minha UFSC”, apresentando a teoria e prática dos mesmos.

7.1.1 Activity

O primeiro componente importante de API é a classe *Activity* (android.app.Activity). Nela é definida qual tela de aplicação será apresentada ao aluno, bem como gerenciar eventos, passar e receber parâmetros dela para com o arquivo de layout.

Uma tela é composta por diversos componentes visuais, como caixas de edição, texto para definição de rótulos, botões, contêiner de layout, e outros. E cada componente é representado pela classe *View* (android.view.View).

Após definir o layout de uma tela, para que seja possível apresentá-la para o aluno é necessário criar uma classe, estender *Activity* e referenciar o layout dentro dessa classe. Por padrão esse procedimento é feito dentro do método *onCreate()* da *Activity*, invocado no momento da criação da tela. Dentro desse método o layout é referenciado invocando o método *setContentView()* passando como parâmetro o ID do arquivo de layout, com pode ser visto na classe *ActivityLogin.*

Figura 23 - Classe ActivityLogin

```
21
22 public class ActivityLogin extends Activity implements OnClickListener {
23
24     private Button btn_entrar;
25     private EditText et_matricula;
26     private EditText et_senha;
27     String matricula;
28     String senha;
29     private static Activity activity;
30
31     public void onCreate(Bundle savedInstanceState) {
32         super.onCreate(savedInstanceState);
33
34         setContentView(R.layout.login);
35
```

Fonte: elaborado pelos autores (2013).

Todas as Activitys do projeto “Minha UFSC” podem ser encontradas dentro do pacote “br.ufsc.minhaufsc.activity” e nelas são realizados os procedimentos descritos acima.

7.1.1.1 Activitys do aplicativo

Para o projeto “Minha UFSC” foi criada duas *Activitys*:

ActivityLogin, usada para criar a tela de login do aplicativo. Nessa classe a matrícula e senha digitadas pelo aluno são recuperados e verificados junto ao *Web Service* da Universidade. Caso a autenticação for realizada com sucesso, os dados do usuário são salvos dentro do contexto da Activity no formato Base64, para posteriormente ser utilizados na consulta das informações do aluno quando uma funcionalidade for selecionada.

ActivityPrincipal, usada para criar o layout base que será utilizado pelos fragmentos de cada funcionalidade do “Minha UFSC”. É também nessa classe que é criado uma lista de funcionalidades que irá funcionar como um menu para que uma funcionalidade possa ser selecionada.

7.1.2 Android Manifest

O arquivo “AndroidManifest.xml” que é criado automaticamente pelo Eclipse na raiz do projeto é um elemento importante no desenvolvimento de um projeto Android. Nele devem ser declaradas todas as *Activitys* presente no aplicativo. Além disso, são declaradas algumas informações importantes, como a versão mínima do Android que é necessário para rodar a aplicação, permissões para utilização dos recursos, nome do aplicativo, tema e *Receivers*.

Figura 24 - Arquivo AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="br.ufsc.minhaufsc"
4      android:versionCode="1"
5      android:versionName="1.0-SNAPSHOT" >
6
7      <uses-sdk
8          android:minSdkVersion="10"
9          android:targetSdkVersion="15" />
10
11     <uses-permission android:name="android.permission.INTERNET" />
12     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
13
14     <application
15         android:icon="@drawable/ic_minha_ufsc"
16         android:label="@string/app_name" >
17         <activity
18             android:name="br.ufsc.minhaufsc.activity.ActivityLogin"
19             android:configChanges="orientation|screenSize"
20             android:label="@string/app_name"
21             android:theme="@style/Theme.Base" >
22             <intent-filter>
23                 <action android:name="android.intent.action.MAIN" />
24                 <category android:name="android.intent.category.LAUNCHER" />
25             </intent-filter>
26         </activity>
27         (...)
```

Fonte: elaborado pelos autores (2013).

7.1.3 Intent

Uma *Intent* (android.content.Intent) pode ser definida como uma intenção do aplicativo em realizar alguma tarefa. Através de uma *Intent* é possível abrir uma localização no *Google Maps*, realizar uma chamada telefônica, abrir um aplicativo para realizar a leitura de um QR Code, entre outros. *Intents* são normalmente utilizadas para navegar entre telas, porém pode ser utilizada para realizar determinada ação através da passagem de parâmetros na Intent, essa ação é realizada na *Activity* que foi chamada, e o resultado é devolvido para a *Activity* que a instanciou.

No “Minha UFSC” uma *Intent* é utilizada na classe *ActivityLogin* para chamar a classe *ActivityPrincipal* caso os dados digitados pelo aluno na tela de login sejam validados no *Web Service*. Para navegar para *ActivityPrincipal* uma *Intent* é instanciada passando dois parâmetros no construtor: a *Activity* que está realizando a chamada, e a *Activity* para qual dever ser redirecionada, e invocar o método *startActivity()* passando como parâmetro a *Intent* criada.

Figura 25 - Declaração e invocação de uma Intent

```
101             Intent intent = new Intent(ActivityLogin.this, ActivityPrincipal.class);
102             String ALUNO_SENHA = matricula + ":" + senha;
103             SharedPreferences settings = getSharedPreferences("aluno", Activity.MODE_PRIVATE);
104             SharedPreferences.Editor editor = settings.edit();
105             editor.putString("matricula", Base64.encode(matricula.getBytes()));
106             editor.putString("aluno_senha", Base64.encode(ALUNO_SENHA.getBytes()));
107             editor.commit();
108
109             startActivity(intent);
```

Fonte: elaborado pelos autores (2013).

7.1.4 – Adapter

A classe *Adapter* (android.widget.Adapter) pode ser definida como uma ponte entre o componente *ListView* e os dados que serão utilizados para popular uma lista de itens nesse componente. O *Adapter* gerencia o acesso aos itens dessa lista, e também é responsável pela exibição desses itens em uma tela. O uso do *Adapter* é indicado quando é preciso apresentar o item dessa lista de forma personalizada, com um layout previamente desenvolvido para essa lista.

Para criar um *Adapter* é preciso criar uma classe, estender *BaseAdapter* e implementar os métodos *getCount()*, *getItem()*, *getItemId()* e *getView()*, além disso é

preciso criar os atributos de classe e o construtor para que esses atributos sejam instanciados. Um desses atributos deve ser um *Array* de objetos que será utilizado nos métodos implementados. O outro, um *LayoutInflater* (android.view.LayoutInflater), que será utilizado para recuperar o layout que apresentará cada item desse *Array* de objetos.

No método *getCount()* é retornado a quantidade de itens dessa lista; *getItem()* deverá retornar o objeto da lista na posição recebida por parâmetro; *getItemId()* deverá retornar o ID do objeto na posição recebida por parâmetro; e o *getView()* que deverá retornar uma *View* que será utilizado na montagem do layout do item da lista na posição recebida por parâmetro.

7.1.4.1 – Adapter de funcionalidades

Para o “Minha UFSC” foi criado um *Adapter* para ser utilizado no *ListView* que abrigará a lista de funcionalidades. Foi declarado como atributo de classe nesse *Adapter* um Array de *String* para armazenar o nome da funcionalidade; Um *Array* de inteiro para armazenar o ID da imagem vinculada a cada funcionalidade; e um *LayoutInflater* para recuperar o layout criado para apresentar os itens dessa lista.

Figura 26 - Trecho de código da classe FuncionalidadesAdapter

```
12 public class FuncionalidadesAdapter extends BaseAdapter {
13
14     private String[] array_titulos;
15     private int[] array_icones;
16     private LayoutInflater inflater;
17
18     public FuncionalidadesAdapter(Context context, String[] array_titulos, int[] array_icones) {
19         this.array_titulos = array_titulos;
20         this.array_icones = array_icones;
21         this.inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
22     }
```

Fonte: elaborado pelos autores (2013).

No método *getView()* o layout funcionalidade_item_lista é recuperado juntamente com os componentes *TextView* e *ImageView*. É atribuído a esses componentes os valores de acordo com a posição recebida por parâmetro de seus respectivos Arrays e uma *View* é retornada para o item da posição, repetindo esse processo até que todos os itens do *Array* sejam percorridos.

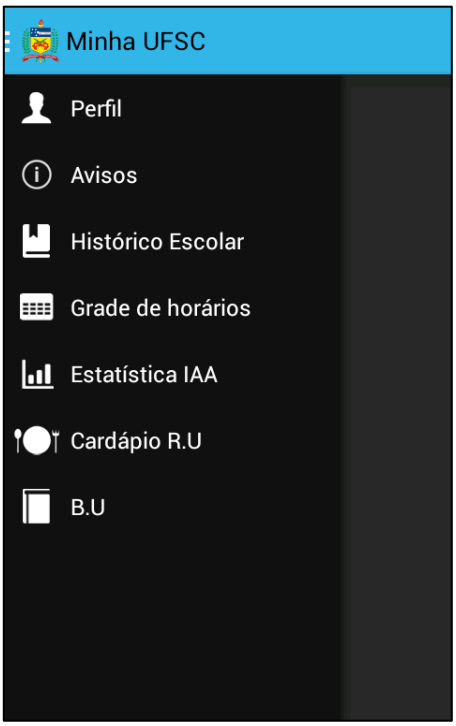
Figura 27 - Método getView() da classe FuncionalidadesAdapter

```
43    public View getView(int position, View convertView, ViewGroup parent) {
44
45        convertView = inflater.inflate(R.layout.funcionalidade_item_lista, parent, false);
46
47        TextView tv_titulo = (TextView) convertView.findViewById(R.id.tv_titulo);
48        ImageView img_icone = (ImageView) convertView.findViewById(R.id.img_icone);
49
50        tv_titulo.setText(array_titulos[position]);
51        img_icone.setImageResource(array_icones[position]);
52
53        return convertView;
54    }
```

Fonte: elaborado pelos autores (2013).

Após recuperar o layout de cada item dessa lista e retornar para a classe responsável por apresentá-la o resultado poderá ser visto na imagem a seguir

Figura 28 - Menu de funcionalidades



Fonte: elaborado pelos autores (2013).

7.1.4.2 – Adapter dias da semana

Também foi criado para o “Minha UFSC” um *Adapter* para ser utilizado no *ListView* que abrigará os dias da semana para o cardápio do R.U.. Foi declarado como atributo de classe nesse *Adapter* um *Array* de *String* para armazenar os dias da semana; e um *LayoutInflater* para recuperar o layout criado para apresentar os itens dessa lista.

Figura 29 - Classe DiasDaSemanaAdapter

```
11 public class DiasDaSemanaAdapter extends BaseAdapter {
12
13     private String[] dias_da_semana;
14     private LayoutInflater inflater;
15
16     public DiasDaSemanaAdapter(Context context, String[] lista_disciplinas) {
17         this.dias_da_semana = lista_disciplinas;
18         this.inflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
19
20     }
```

Fonte: elaborado pelos autores (2013).

Assim como no *Adapter* de funcionalidades o layout criado para o *ListView* de dias da semana é recuperado e é adicionado o valor referente a posição recebida por parâmetro.

Figura 30 - Método getVew() da classe DiasDaSemanaAdapter

```
37     @Override
38     public View getView(int position, View convertView, ViewGroup parent) {
39
40         convertView = this.inflater.inflate(R.layout.dia_semana_item_lista, parent, false);
41
42         TextView tv_nome = (TextView) convertView.findViewById(R.id.tv_nome);
43         tv_nome.setText(this.dias_da_semana[position]);
44
45         return convertView;
46     }
```

Fonte: elaborado pelos autores (2013).

Figura 31 - Menu do cardápio

Fonte: elaborado pelos autores (2013).

7.1.5 Layout

As definições de layout da tela são realizadas em arquivos XML, onde os componentes que as compõe são definidos através de TAGs. Todos os arquivos de layouts são salvos no diretório “res/layout” do projeto.

Para vincular um layout a uma *Activity* deverá ser invocado o método *setContentView()* passando como parâmetro através da classe “R.layout.nome_layout”. Na imagem abaixo é possível visualizar o layout da tela de login sendo vinculado a *ActivityLogin*.

Figura 32 - Menu do cardápio

```
25
26      public void onCreate(Bundle savedInstanceState) {
27          super.onCreate(savedInstanceState);
28          setContentView(R.layout.login);
29      }
30
```

Fonte: elaborado pelos autores (2013).

7.1.6 View – componentes de layout

Os componentes de layout das aplicações Android podem ser definidos no arquivo XML, ou ainda programaticamente dentro das classes Java. No “Minha UFSC” grande parte dos componentes de layout são sob demanda, ou seja, serão definidos de acordo com as informações recuperadas no *Web Service*.

Figura 33 - Componente EditText declarado no arquivo de layout login.xml

```
74          <EditText
75              android:id="@+id/et_matricula"
76              android:layout_width="fill_parent"
77              android:layout_height="wrap_content"
78              android:background="@drawable/estilo_text_view"
79              android:ems="10"
80              android:hint="@string/matricula" >
81
82              <requestFocus />
83          </EditText>
```

Fonte: elaborado pelos autores (2013).

Quando definidos em arquivo XML é necessário recuperar esses componentes através do método *findViewById()*, passando como parâmetro a identificação desse componente no arquivo XML

Figura 34 - Recuperação do EditText do layout login.xml

```
40
41          et_matricula=(EditText) findViewById(R.id.et_matricula);
42          et_senha=(EditText) findViewById(R.id.et_senha);
43
```

Fonte: elaborado pelos autores (2013).

A seguir serão apresentados os componentes utilizados no desenvolvimento do projeto “Minha UFSC” bem como suas características.

7.1.6.1 TextView

O componente *TextView* apresenta um simples campo de texto, muito utilizado como rótulo nas aplicações. No “Minha UFSC” ele é visto na tela de login abaixo do logo da Universidade com o texto “Minha UFSC”.

Figura 35 - Componente TextView declarado no arquivo login.xml

```
39          <TextView
40              android:id="@+id/tv_minha"
41              android:layout_width="wrap_content"
42              android:layout_height="wrap_content"
43              android:layout_gravity="center_horizontal"
44              android:layout_marginLeft="15dp"
45              android:text="@string/minha"
46              android:textColor="@color/cor_fonte_conteudo_mais_escuro"
47              android:textSize="@dimen/fonte_minha_ufsc" />
```

Fonte: elaborado pelos autores (2013).

Para definir um texto a um *TextView* de dentro de uma classe *Java* usa-se o método *setText(),* e em um arquivo .xml é atribuído o texto através da propriedade “android:text”.

7.1.6.2 EditText

O componente *EditText* pode ser definido como uma caixa de diálogo para recuperar informações digitadas pelo aluno na tela do aplicativo e realizar alguma tarefa ou rotina.

Figura 36 - Componente EditText declarado no arquivo de layout login.xml

```
87      <EditText
88          android:id="@+id/et_senha"
89          android:layout_width="fill_parent"
90          android:layout_height="wrap_content"
91          android:layout_marginTop="15dp"
92          android:background="@drawable/estilo_text_view"
93          android:hint="@string/senha"
94          android:inputType="numberPassword"
95          android:textColor="@color/cor_fonte_conteudo_mais_escuro" />
```

Fonte: elaborado pelos autores (2013).

Para definir um texto usa-se o método *setText()*. Para recuperar as informações digitadas pelo aluno usa-se o método *getText().toString()*.

7.1.6.3 Button

O componente *Button* disponibiliza um botão na tela do aplicativo para realizar uma ação no momento em que o aluno pressionar o botão na tela.

Figura 37 - Componente Button declarado no arquivo de layout login.xml

```
94      <Button
95          android:id="@+id/btn_entrar"
96          android:layout_width="fill_parent"
97          android:layout_height="wrap_content"
98          android:layout_marginTop="20dp"
99          android:background="@drawable/estilo_botao_login"
100         android:text="@string/botao_entrar"
101         android:textColor="@color/branco"
102         android:textSize="20sp"
103         android:textStyle="bold" />
```

Fonte: elaborado pelos autores (2013).

Para que uma ação seja executada quando pressionado o botão é necessário que adicione um evento através do método *setOnClickListener()* passando como parâmetro o contexto da *Activity*.

Figura 38 - Recuperação do componente Button login.xml

```
43      btn_entrar=(Button) findViewById(R.id.btn_entrar);
44      btn_entrar.setOnClickListener(this);
```

Fonte: elaborado pelos autores (2013).

Ao adicionar essa notação será preciso que a classe implemente a interface *OnClickListener* e o método *onClick(),* e no corpo desse método é que deverá ser realizado a ação.

Figura 39 - Método onClick() referente ao botão entrar do login.xml

```
52    @Override
53    public void onClick(View v) {
54        matricula = et_matricula.getText().toString();
55        senha = et_senha.getText().toString();
56
57        if (!(matricula.equals("")) && !(senha.equals(""))) {
58            if (verificaConexao()) {
59                new verificaAutenticacao().execute(matricula, senha);
60            } else {
61                mostrarAlerta("Verifique a conexão com a internet");
62            }
63        } else {
64            int duracao = Toast.LENGTH_SHORT;
65            String mensagem = getResources().getString(R.string.dados_em_branco);
66            Toast.makeText(this, mensagem, duracao).show();
67        }
68    }
```

Fonte: elaborado pelos autores (2013).(colocar a ação do login)

7.1.6.4 ImageView

O componente *ImageView* como o próprio nome sugere permite carregar uma imagem. No “Minha UFSC” ele é utilizado na tela de perfil para carregar a foto do aluno e o ícone para identificar o curso em que ele está matriculado. Ele também pode ser visto na tela em que é apresentada a lista de dias da semana do cardápio do R.U..

Figura 40 - Componente ImageView declarado no dia_semana_item_lista.xml

```
22    <ImageView
23        android:layout_width="wrap_content"
24        android:layout_height="wrap_content"
25        android:layout_alignParentRight="true"
26        android:layout_centerVertical="true"
27        android:src="@drawable/seta_item_lista"/>
```

Fonte: elaborado pelos autores (2013).

7.1.6.5 ListView

O componente *ListView* permite que sejam exibidos uma coleção de objetos na tela do aplicativo. Para gerenciar o acesso a esses itens é utilizado a classe *Adapter* conforme explicado no item 7.1.4.1.. Esse componente é utilizado para criar

uma lista de funcionalidades do “Minha UFSC”, no entanto ele também é utilizado para criar a lista de dias da semana do cardápio do R.U..

Figura 41 - Componente ListView declarado no layout principal.xml

```
14      <ListView
15          android:id="@+id/lista_dias_semana"
16          android:layout_width="fill_parent"
17          android:layout_height="fill_parent"
18          android:divider="@color/cor_contorno_padrao"
19          android:dividerHeight="1px"
20          android:listSelector="@drawable/lista_dia_semana_selector">
21      </ListView>
```

Fonte: elaborado pelos autores (2013).

Para que uma ação seja executada quando um item dessa lista for selecionado é necessário que adicione a ele um evento através do método *setOnItemClickListener()* passando como parâmetro o contexto da *Activity*. Ao adicionar essa notação será preciso que a classe implemente a interface *OnItemClickListener* e o método *onItemClick(),* e no corpo desse método é que deverá ser realizado a ação.

7.1.6.6 Contêiner

Os contêineres são utilizados para organizar de alguma maneira outros componentes presentes em um layout, ou seja, o contêiner irá definir a forma com que esses componentes serão apresentados na tela.

Dentre os contêineres disponíveis na API, foram utilizados no desenvolvimento do projeto “Minha UFSC” o *LinearLayout*, *RelativeLayout*, *TableLayout*, *FrameLayout*, *DrawerLayout* e *ScrollView*.

- LinearLayout

Esse contêiner permite que os componentes sejam organizados de forma linear, podendo ser na horizontal ou na vertical. A propriedade “android:orientation” define a orientação que será utilizada na organização dos componentes.

- RelativeLayout

Esse contêiner permite que os componentes sejam organizados sempre com base em algum outro componente, por exemplo, é possível informar ao contêiner que um componente x deverá ser colocado abaixo, acima, do lado esquerdo ou direito de um componente y.

Figura 42 - RelativeLayout declarado no arquivo funcionalidade_item_lista.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      style="?attr/spinnerDropDownItemStyle"
4      android:layout_width="match_parent"
5      android:layout_height="?attr/dropdownListPreferredItemHeight" >
6
7      <ImageView
8          android:id="@+id/img_icone"
9          android:layout_width="wrap_content"
10         android:layout_height="match_parent"
11         android:adjustViewBounds="true" />
12
13     <TextView
14         android:id="@+id/tv_titulo"
15         style="?attr/spinnerDropDownItemStyle"
16         android:layout_width="wrap_content"
17         android:layout_height="match_parent"
18         android:layout_toRightOf="@+id/img_icone"
19         android:ellipsize="end"
20         android:singleLine="true"
21         android:textColor="@color/branco" />
22 </RelativeLayout>
23
```

Fonte: elaborado pelos autores (2013).

- TableLayout

Esse contêiner permite que os componentes sejam organizados como uma tabela. Dentro dele são definidas linhas, e dentro dessas linhas são colocados os componentes que se deseja organizar. Esse contêiner é utilizado com frequência no “Minha UFSC” por ser a melhor forma de apresentar dados de uma lista de itens.

Figura 43 - TableLayout da arquivo de layout fragmento_bu.xml

```
39              <TableLayout
40                  android:id="@+id/tl_livros_emprestado"
41                  android:layout_width="fill_parent"
42                  android:layout_height="wrap_content"
43                  android:layout_below="@+id/linha"
44                  android:layout_marginTop="1px"
45                  android:background="@color/cor_contorno_padrao"
46                  android:stretchColumns="1" >
47
48                  <TableRow
49                      android:id="@+id/linha_emprestimos"
50                      android:layout_width="match_parent"
51                      android:layout_height="wrap_content"
52                      android:background="@color/cor_fundo_padrao"
53                      android:visibility="gone" >
54
55                      <TextView
56                          android:id="@+id/tv_sem_emprestimos"
57                          android:layout_width="wrap_content"
58                          android:layout_height="wrap_content"
59                          android:layout_marginBottom="10px"
60                          android:layout_marginLeft="10px"
61                          android:layout_marginTop="10px"
62                          android:text="@string/sem_emprestimos"
63                          android:textColor="@color/cor_fonte_titulo_padrao" />
64                  </TableRow>
65              </TableLayout>
```

Fonte: elaborado pelos autores (2013).

- FrameLayout

Esse contêiner é utilizado quando é preciso criar uma sobreposição de componentes, isto é, a possibilidade de posicionar um componente sobre outro. Uma situação em que é muito utilizado esse contêiner é quando se quer trabalhar com *Fragment (android.support.v4.app.Fragment)*, pois com esse contêiner é possível substituir o conteúdo dele pelo conteúdo do *Fragment*.

Figura 44 - Componente declarado no arquivo de layout principal.xml

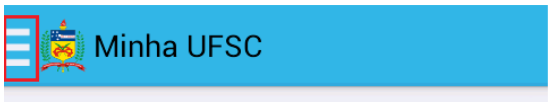
```
7       <FrameLayout
8           android:id="@+id/conteudo_fragmento"
9           android:layout_width="match_parent"
10          android:layout_height="match_parent"/>
```

Fonte: elaborado pelos autores (2013).

- DrawerLayout Ele é um contêiner na qual permite que componentes sejam deslizados da borda da tela para o centro, normalmente utilizados para abrigar um menu deslizante no aplicativo.

No “Minha UFSC” ele é utilizado para abrigar o menu de funcionalidades, que pode ser acionado de duas maneira diferentes. A primeira opção seria pressionando o ícone localizado na parte superior do canto esquerdo do aplicativo, conforme pode ser visto na imagem a seguir.

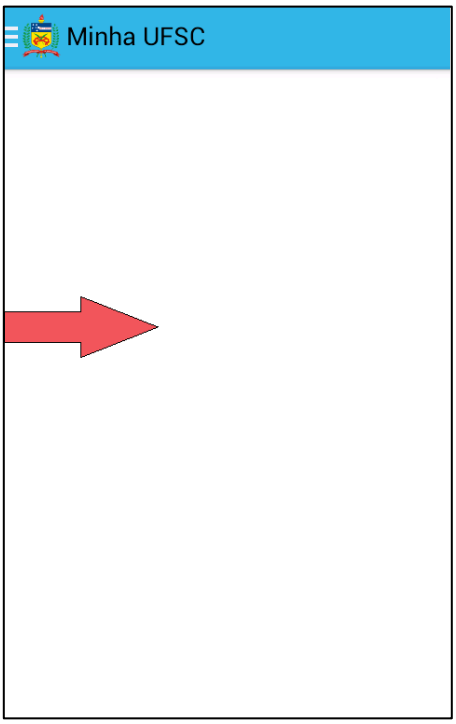
Figura 45 - ActionBar do aplicativo



Fonte: elaborado pelos autores (2013).

A segunda opção seria realizando o movimento com o dedo deslizando da borda esquerda da tela do dispositivo para o centro, como mostra a imagem a seguir.

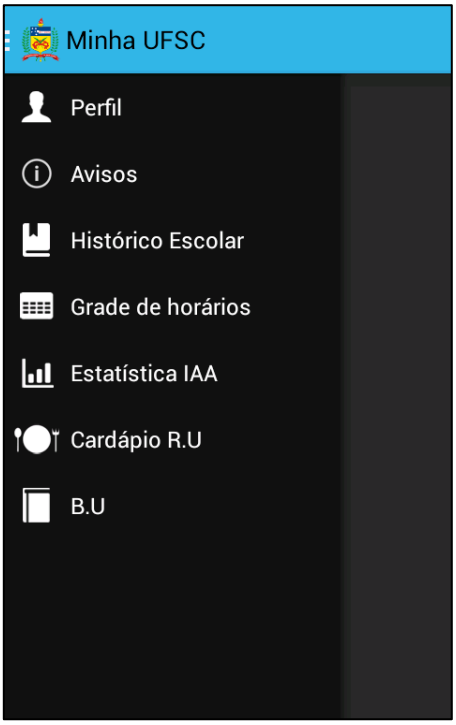
Figura 46 - Tela do aplicativo ainda sem funcionalidade implementada



Fonte: elaborado pelos autores (2013).

Na imagem abaixo pode ser visto o resultado do menu após ser acionado utilizando um dos método descritos acima.

Figura 47 - Menu de funcionalidades acionado



Fonte: elaborado pelos autores (2013).

- ScrollView

Esse contêiner é utilizado quando não há espaço suficiente para que seja exibido todo o conteúdo de uma tela. Quando uma situação dessas acontecer basta o aluno deslizar o dedo de baixo para cima para que os itens ocultos sejam apresentados. Esse contêiner pode ser visto realizando essas tarefas descritas em algumas das telas do “Minha UFSC”.

7.2 COMPONENTES PERSONALIZADOS

Componentes personalizados foram utilizados no “Minha UFSC” para realizar algumas funções na qual a API do Android não oferece uma solução pronta.

7.2.1 Componente Notificação

Esse componente foi desenvolvido para permitir que a aplicação notificasse o aluno de forma automática quando não houvesse uma conexão com a internet, e que a mesma deveria desaparecer caso a conexão passasse a existir. Na imagem

abaixo é possível visualizar o “Minha UFSC” notificando o aluno de que não há conexão com a internet.

Figura 48 - Componente de notificação ativado

Fonte: elaborado pelos autores (2013).

Para utilizar essa notificação foi declarado o componente em todos os arquivos de layout dos fragmentos e na tela principal, como pode ser visto na imagem a seguir.

Figura 49 - Declaração do componente de notificação

```
42          <br.ufsc.minhaufsc.activity.util.ComponenteNotificacao
43              android:id="@+id/notificacao"
44              android:layout_width="fill_parent"
45              android:layout_height="wrap_content"
46              android:visibility="gone" />
```

Fonte: elaborado pelos autores (2013).

Além disso, foi preciso criar uma classe chamada *VerificaConexao que* estende a classe *BroadcastReceiver.* Ao estender essa classe será preciso implementar o método *onReceive()*, que é invocado quando há uma mudança no estado da conexão. No corpo desse método quando uma conexão é perdida o método *mostrarNotificacao()* da classe *ActivityPrincipal* é invocado. Do mesmo modo quando uma conexão passa a existir o método *fecharNotificacao()* é invocado.

Figura 50 - Método OnReceive() da classe verificaConexao

```
16  @Override
17  public void onReceive(Context context, Intent intent) {
18       boolean conexao = intent.getBooleanExtra(ConnectivityManager.EXTRA_NO_CONNECTIVITY, false);
19      if (conexao) {
20          ActivityPrincipal.mostrarNotificacao();
21      } else {
22          ActivityPrincipal.fecharNotificacao();
23      }
24  }
```

Fonte: elaborado pelos autores (2013).

Por fim é preciso registrar *Receiver* no arquivo *AndroidManifest.xml,* bem como a permissão para acessar o estado da conexão.

Figura 51 - Receiver declarado no AndroidManifest.xml

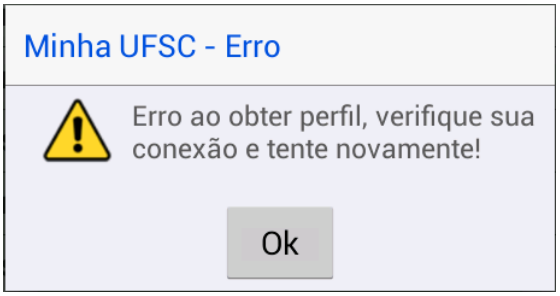
```
38          <receiver android:name=".activity.util.VerificaConexao" >
39              <intent-filter>
40                  <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
41              </intent-filter>
42          </receiver>
```

Fonte: elaborado pelos autores (2013).

7.2.2 Componente de Alerta

Outro componente que precisou ser personalizado foi o de alerta, esse componente tem como objetivo emitir um alerta ao aluno quando houver alguma falha consulta das informações no *Web Service*. Esse componente na verdade utiliza como base o *AlertDialog*, que pode ser definido como uma componente responsável por emitir um alerta na aplicação, porém esse componente não permite muitas modificações quanto as cores e temas.

Figura 52 - Componente de alerta



Fonte: elaborado pelos autores (2013).

7.3 MENU

O projeto “Minha UFSC” dispõe de um menu localizado no canto direito da ActionBar. O menu também é um arquivo XML e fica armazenado em “res/menu”, na imagem abaixo tem-se o layout do menu do “Minha UFSC”.

Figura 53 - Arquivo menu.xml

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/sobre"
        android:icon="@drawable/ic_noticias"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/sobre"/>
    <item
        android:id="@+id/sair"
        android:orderInCategory="100"
        android:showAsAction="never"
        android:title="@string/sair"/>

</menu>
```

Fonte: elaborado pelos autores (2013).

O menu é carregado através da invocação do método *onCreateOptionsMenu()* passando como parâmetro um objeto da classe *Menu* (android.view.Menu) na classe ActivityPrincipal. Esse método é invocado automaticamente pela Activity na qual o menu está vinculado.

Figura 54 - Arquivo menu.xml

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true;
}
```

Fonte: elaborado pelos autores (2013).

Nesse menu há duas opções: a primeira é o item “Sobre”, na qual contém as informações desse projeto assim como a de seus autores; e o outro item é o “Sair”, na qual leva o usuário para a tela de login, permitindo que um outro usuário utilize a aplicação.

7.4 CLASSE ASYNCTASK

É uma classe auxiliar utilizada para realizar tarefas que demande um certo tempo e que poderiam fazer com que o processo principal permanecesse esperando o retorno dessa chamada, resultando no encerramento inesperado do aplicativo em caso de falha, para corrigir esse problema a API fornece a classe *AsyncTask*.

Para utilizar AsyncTask uma classe deverá ser criada estendendo *AsyncTask,* como resultado três métodos deverão ser implementados: o *onPreExecute()*, recomendado para quando é necessário realizar alguma pré-configuração antes da chamada. O *doInBackground()* que deve ser utilizado para realizar a chamada propriamente dita. E o *onPostExecute()* onde é feito a validação e uso do retorno. Esses métodos são executados de forma sequencial e na ordem em que foram descritos no momento em que a classe for executada.

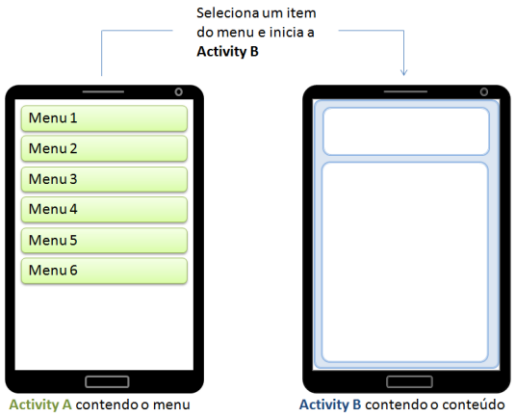
No “Minha UFSC” para cada *Fragment* foi criado uma classe auxiliar estendendo *AsyncTask*. No método *onPreExecute() e* uma barra de progresso foi instanciada e iniciada. No método *doInBackground()* uma chamada para o *Web service* é feita de acordo com o *Fragment*. No método *onPostExecute()* é verificado o retorno dessa chamada, caso o retorno seja diferente de nulo é invocado o método *montarLayout()*, que é responsável por construir a tela do *Fragment* com as informações recuperadas. Caso contrário um alerta é mostrado. Ainda nesse método a barra de progresso é finalizada.

7.5 FRAGMENT

O componente *Fragment* pode ser definido com uma interface ou parte dela. Eles são totalmente flexíveis e podem ser adicionados ou removidos com facilidade de uma estrutura de layout. Normalmente são utilizados em situações que se espera um comportamento diferente quando o aplicativo é executado em um smartphone e em um tablet.

No smartphone por possuir uma área de apresentação menor comparado ao Tablet, o seguinte layout é construído:

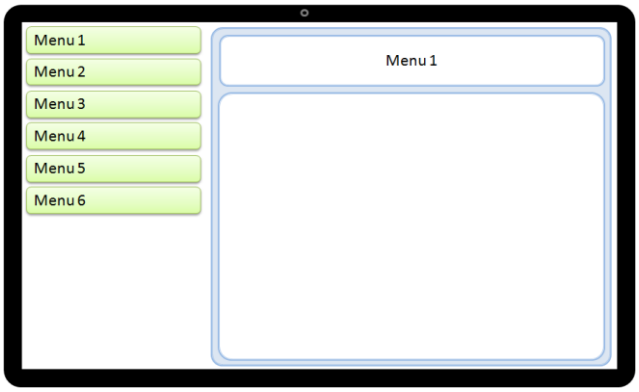
Figura 55 - Fragment em Smartphone



Fonte: Suelen Goularte Carvalho's Blog (http://suelengc.com.br/blog/?cat=18)

No entanto no tablet a seguinte situação acontece:

Figura 56 - Fragment em Tablet



Fonte: Suelen Goularte Carvalho's Blog (http://suelengc.com.br/blog/?cat=18)

É importante ressaltar que é preciso que um *Fragment* deve sempre estar vinculado a uma *Activity*.

7.5.1 Fragments do Aplicativo

No projeto "Minha UFSC" cada funcionalidade é representada por um *Fragment*. No layout da tela principal foi adicionado o componente *FrameLayout* para que possa ser feito a substituição pelo *Fragment* da funcionalidade selecionada.

Figura 57 - Componente declarado no arquivo de layout principal.xml

```
7     <FrameLayout
8         android:id="@+id/conteudo_fragmento"
9         android:layout_width="match_parent"
10        android:layout_height="match_parent"/>
```

Fonte: elaborado pelos autores (2013).

Quando uma funcionalidade é selecionada o método *onItemClick*() é invocado passando como parâmetro a posição do item selecionado no *Array* de funcionalidades. No corpo desse método é verificado qual funcionalidade foi selecionada para que possa ser feito a substituição do *Fragment* atual pelo selecionado. Essa substituição é feita utilizando a classe *FragmentTransaction* (android.support.v4.app.FragmentTransaction). Primeiramente é recuperado o *FragmentTransaction* da *Activity* principal e instanciado um objeto. A substituição é feita através da invocação do método *replace()* passando como parâmetro o ID do *Fragment,* nesse caso "conteudo_fragmento", e o objeto do *Fragment* da funcionalidade selecionada, seguido da invocação do método *commit().*

Figura 58 - Substituindo o conteúdo do Framelayout pelo do fragmento selecionado

```
163         FragmentTransaction fragment_transaction = getSupportFragmentManager().beginTransaction();
164         switch (posicao) {
165         case 0:
166             fragment_transaction.replace(R.id.conteudo_fragmento, fragmento_perfil);
167             fragment_transaction.addToBackStack("perfil");
168             break;
169         case 1:
170             fragment_transaction.replace(R.id.conteudo_fragmento, fragmento_avisos);
171             fragment_transaction.addToBackStack("avisos");
172             break;
173         case 2:
174             fragment_transaction.replace(R.id.conteudo_fragmento, fragmento_historico_escolar);
175             fragment_transaction.addToBackStack("historico");
176             break;
```

Fonte: elaborado pelos autores (2013).

7.5.1.1 Fragment - perfil

A funcionalidade perfil como o próprio nome sugere visa recuperar o perfil do aluno para com a Universidade. Nessa tela além das informações básicas do aluno como nome do aluno, matrícula, curso em que está matriculado, ano de ingresso no curso, semestres cursados entre outros. Foi decidido que seria apropriado mostrar a foto do aluno no perfil, pois traria uma maior elegância na interface, além de torná-la um padrão muito utilizado atualmente, como em redes sociais por exemplo.

O retorno da consulta da foto do aluno em alguns casos poderá vir nulo, pois existem alunos que não possuem foto cadastrada, nesse caso uma imagem padrão é carregada, pois não se trata de um erro de consulta e sim de que o aluno não possui foto cadastrada no sistema. Dentro do método *montarLayout()* as informações do aluno recuperados junto ao *Web Service* são preenchidas nos respectivos *TextView*, que são responsáveis por apresentar cada informação do aluno na tela.

Figura 59 - Tela de perfil

Fonte: elaborado pelos autores (2013).

7.5.1.2 Fragment - avisos

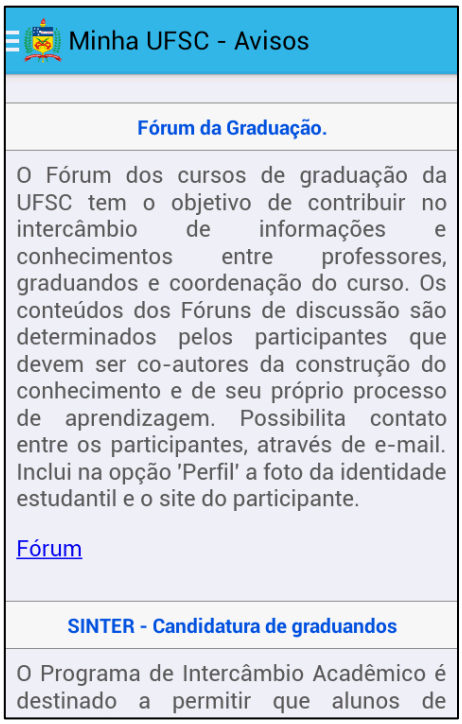
A funcionalidade de avisos tem como objetivo recuperar os principais avisos da Universidade para com os alunos, como por exemplo, quando inicia-se o período de matrícula em cada semestre, informações de estágios, Moodle, entre outros.

Dentro do método *montarLayout()* a lista de objetos da classe *Aviso* recebida por parâmetro é percorrida, uma linha da tabela é declarada, um *TextView* com o título do aviso é adicionado a essa linha, que por sua vez é adicionada ao *TableLayout* responsável por abrigar esses dados.

Outra linha é criada, desta vez para adicionar a mensagem contida dentro do aviso, mas ao invés de utilizar um *TextVie*w para abrigar a mensagem é utilizado um *WebView* pois em algumas mensagens pode conter links e esse componente

permite que se utilize TAGs html para declará-los. E esse processo se repete até que toda a lista de avisos seja percorrida.

Figura 60 - Tela de avisos da Universidade



Fonte: elaborado pelos autores (2013).

7.5.1.3 Fragment - histórico escolar

A funcionalidade do histórico escolar tem como objetivo recuperar a relação de notas, índices I.A. e I.A.A. do aluno em cada disciplina cursada por ele ao longo de cada semestre do curso em que está matriculado.

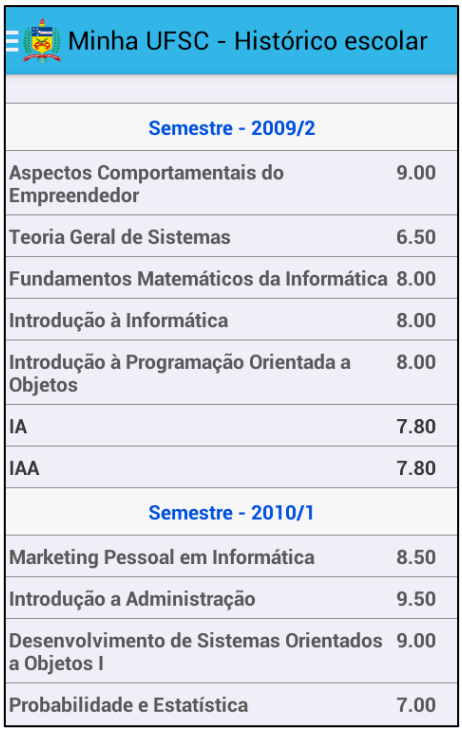
Dentro do método *montarLayout() a* lista de objetos da classe *Semestre* recebida por parâmetro é percorrida, uma linha da tabela é criada e um *TextView* com o identificador do semestre é adicionado a essa linha, que por sua vez é adicionada ao *TableLayout*.

Após adicionar o titulo, a lista de objetos da classe *Disciplina* é percorrida para que se possam recuperar as disciplinas juntamente com as notas que o aluno obteve no semestre da atual iteração. A cada iteração na lista de objetos da classe *Disciplina* é criado uma linha na tabela. Nessa linha é adicionado um *LinearLayout* que fará o papel de uma coluna com 90% da área total dela, e nele é adicionado um *TextView* com o título da disciplina. Nessa mesma linha é adicionado outro

LinearLayout que ocupará os 10% restantes dela, e nele é adicionado um *TextView* com a nota obtida pelo aluno na disciplina, e por fim essa linha é adicionada a *TableLayout*. Os dois *LinearLayout* são utilizados para que o nome da disciplina e a nota sejam adicionados lado a lado, e que o layout seja apresentado de forma correta mesmo em casos onde o nome da disciplina ocupe mais de uma linha da tabela. E esse processo se repete até que todas as disciplinas sejam percorridas.

Abaixo das disciplinas foram adicionados os índices de I.A e I.A.A. de cada semestre cursado pelo aluno, na qual foi utilizado o mesmo processo utilizado para adicionar as disciplinas e as notas obtidas na mesma. E esses processos são feitos até que todos os semestres sejam percorridos e as informações presentes nessa classe sejam recuperadas.

Figura 61 - Tela de histórico escolar



Fonte: elaborado pelos autores (2013).

7.5.1.4 Fragment - grade de horários

A funcionalidade de grade de horários tem como objetivo recuperar os horários das aulas durante o semestre em que o aluno esta cursando. Nessa grade além indicar quais disciplinas o aluno terá em determinado dia da semana para cada período, também indicará o número da sala e localização da mesma para cada

horário. Também será possível verificar o nome do professor que leciona cada disciplina.

Dentro do método *montarLayout()* a lista de objeto da classe *HorarioAula* que está presente dentro da classe *GradeHorarioAluno* é percorrida. A cada iteração é verificado qual o horário da aula do item atual, caso o horário for menor do que "*1330*" significa que esse item pertence ao turno matutino e o método *preencherMatutino()* é invocado. Caso ele for maior que "*1330*" e menor que "*1830*" ele pertence ao turno vespertino e o método *preencherVespertino()* é invocado. Quando ele não pertencer a nenhum dos dois casos descritos acima o método *preencherNoturno()* é invocado.

Após o método correspondente ao turno ser invocado o contêiner *RelativeLayout* corresponde ao mesmo é recuperado e atribuído a ele a notação para que ele torne a ser visível, ou seja, se o aluno possui uma disciplina cadastrada nesse turno a tabela referente ao mesmo deve ficar visível.

Nesses métodos, que possui a mesma lógica em todos eles, consiste em verificar a qual célula na tabela do turno correspondente esse item pertence. Dentro de cada método é verificado a qual dia da semana e horário o objeto *HorarioAula* vindo por parâmetro nesse método ele pertence. Após identificar a célula corresponde, o componente *TextView* referente a ela é carregado e atribuído o valor do código da disciplina e a localização da sala. Esses processos se repetem até que todos os itens da lista de horários sejam percorridos.

Ainda dentro do método *montarLayout()* após percorrer toda a lista de *HorarioAula* o método *preencherProfessorDisciplina()* é invocado para que seja montado o layout com os professor(es) de cada disciplina. Nesse método é criado um linha, e nela é são adicionados dois *TextView* ocupando 50% da linha cada um. Um deles é atribuído o rótulo "*Disciplina*", e o outro o rótulo "*Professores*". E essa linha é adicionado ao *TableLayout* correspondente aos professores da disciplina.

Após adicionar os títulos, a lista de *ProfessoresDisciplina* é percorrida e a cada iteração é adicionado uma linha. Nessa linha é adicionado um *LinearLayout* ocupando 50% dela, e nele é adicionado um *TextView* contendo o nome da disciplina. Ainda nessa linha é adicionado outro *LinearLayout* ocupando os outros 50%, e nele é adicionado um *TextView* contendo o nome dos professores da disciplina separados por vírgula quando houver mais de um.

E por fim essa linha é adicionado ao *TableLayout*, e esse processo se repete até que toda a lista de *ProfessoresDisciplina* seja percorrida.

Figura 62 - Tela de grade de horários

Fonte: elaborado pelos autores (2013).

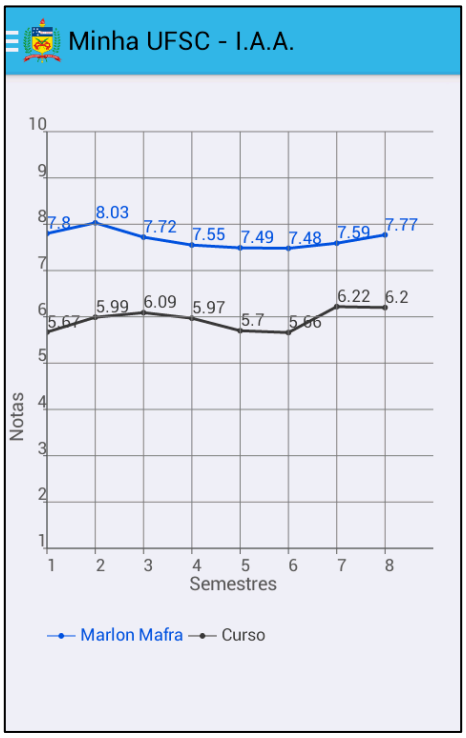
7.5.1.5 Fragment de estatísticas I.A.A.

A funcionalidade de estatística do I.A.A. consiste em apresentar um gráfico em linha, comparando os índices I.A.A. do aluno obtidos ao longo da graduação, com os índices I.A.A médio do curso para cada semestre na qual ele esteve matriculado.

Dentro do método *montarLayout()* foram criados dois objetos do tipo *TimeSeries*. Ele é oriundo da biblioteca *GraphicalView* e podem serem definidos como uma lista de valores utilizada para popular o gráfico. Nelas são adicionados os valores I.A.A. do aluno e do curso nos seus respectivos *TimeSeries* utilizando os dados recebidos por parâmetro. Essas listas de valores foram colocadas em um objeto do tipo *XYMultipleSeriesDataset*, e esse por sua vez foi adicionado ao objeto do tipo *GraphicalView*, que por fim foi adicionado ao componente *RelativeLayout* recuperado no arquivo de layout.

Para que o aplicativo esteja apto a executar esse funcionalidade é preciso declarar a *Activity org.achartengine.GraphicalActivity* no arquivo *AndroidManifest.*

Figura 63 - Tela de estatística de I.A.A.



Fonte: elaborado pelos autores (2013).

7.5.1.6 Fragment de cardápio do R.U

A funcionalidade do cardápio do R.U. tem com o objetivo recuperar o menu para cada dia da semana do restaurante universitário. Para que essa funcionalidade fosse apresentada ao aluno de uma forma simples e totalmente adaptada a dispositivos móveis foi decidido que seria utilizado dois layouts. O primeiro layout será exibido uma lista com os dias da semana, e o segundo será exibido o cardápio do dia quando selecionado.

Dentro do método *montarLayout()* a lista de objetos *CardapioDia* é percorrida para que possa ser criado um *ArrayList* contento o título de cada dia da semana, que por fim será adicionado ao *ListaDiaSemanaAdapter* para montar uma lista de seleção. Quando um item é selecionado o método *selecionarItem()* é invocando recebendo como parâmetro a posição do item selecionado na lista de dias da semana, e o objeto *CardapioDia* é recuperado utilizando a posição recebida por parâmetro.

O objeto *FragmentoCardapio* é declarado para que possa ser feito a substituição do fragmento atual pelo recém criado. Além disso, foi criado um objeto da classe *Bundle* e nele é colocado o objeto *CardapioDia* recuperado. Quando a função de substituir o Fragment for chamada esse objeto irá junto para que as informações do dia selecionado estejam disponíveis na segunda tela conforme é demonstrado na imagem a seguir.

Figura 64 - Método selecionar item da classe FragmentoCardapioRU

```
99    public void selecionarItem(int posicao) {
100
101       //Cardápio do dia recuperado
102       CardapioDia cardapio_dia = cardapio_semana.getCardapiosDia().get(posicao);
103       //Coloca o objeto recuperado na classe Bundle
104       Bundle argumentos = new Bundle();
105       argumentos.putParcelable(CardapioDia.class.getName(), cardapio_dia);
106       //O Bundle é passado como argument para que possa ser recuperado no fragmento.
107       FragmentoCardapio fragmento_cardapio = new FragmentoCardapio();
108       fragmento_cardapio.setArguments(argumentos);
```

Fonte: elaborado pelo autor (2013)

Na imagem abaixo é possível verificar como a tela deverá ser apresentada após essa implementação.

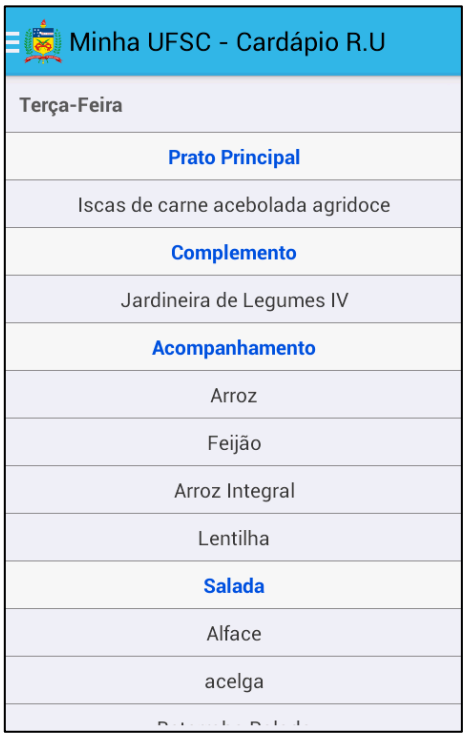
Figura 65 - Tela com o cardápio da semana

Fonte: elaborado pelos autores (2013).

Na segunda parte da implementação dessa funcionalidade foi criado a classe *FragmentCardapio* para exibir a tela com o cardápio do dia selecionado. No método *onResume()* o objeto da classe *CardapioDia* é recuperado, e o método *montarLayout()* é invocado passando como parâmetro esse objeto.

Dentro desse método *o TextView* responsável pela identificação do dia da semana é recuperado e atribuído a ele a identificação do dia da semana selecionado. A lista de *Argumentos* contida dentro do objeto *CardapioDia* é percorrida e a cada iteração uma nova linha na tabela é criada. Nessa linha é adicionado um *TextView* contendo a identificação desse argumento, como por exemplo, prato principal, acompanhamento, salada entre outros. E por fim essa linha é adicionada na tabela. Um *Argumento* pode conter um ou mais itens da classe *Alimento*, sendo assim também é necessário percorrer essa lista. A cada iteração na lista de alimentos uma nova linha é criada e nela é adicionado um *TextView* contendo o nome do alimento, e essa linha é adicionada a tabela. Esse processo se repete até que todos os itens da lista de *Alimento* seja percorrida para cada item da lista de objetos A*rgumento*.

Figura 66 - Tela com o cardápio do dia



Fonte: elaborado pelos autores (2013).

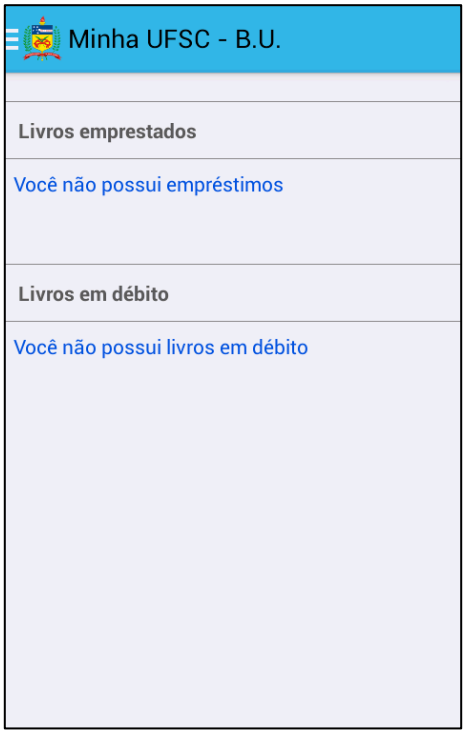
7.5.1.7 Fragment de informações da B.U.

A funcionalidade de informações da B.U., com o objetivo de verificar se o aluno possui pendências de empréstimos e débitos de livros com a Universidade. Para recuperar essas informações são feitas três consultas no *Web Service* da Universidade: a primeira consulta é feita utilizando o método *getIdPessoaByMatriculaGraduacao()* para recuperar o ID do aluno, pois o sistema da biblioteca utiliza um identificador ao invés da matrícula, e esse método é utilizado para realizar essa conversão. As outras duas consultas são realizadas utilizando os métodos *getListEmprestimoObraByIdPessoa()* e *getListEmprestimoObraEmDebitoByIdPessoa()*, utilizando o ID recuperado como parâmetro.

Cada método irá retornar uma lista de objetos da classe *EmprestimoObra* que serão utilizados para montar a tela de informações da B.U utilizando os métodos *montarLayoutDebitos()* e *montarLayoutEmpréstimos()*

Dentro desses métodos os componentes relacionados a cada layout são recuperados. Caso a lista esteja vazia a linha que possui a mensagem “Você não possui empréstimo” ou “Você não possui livros em débito” é recuperada e atribuída como visível e a rotina termina. Por outro lado caso a lista possua ao menos um item o seguinte processo é feito para ambas às listas. Primeiramente é criado uma linha, e adicionado a ela um *TextView* contendo o título do livro, que por sua vez é adicionado ao *TableLayout* correspondente. Seguindo no processo é criado uma nova linha, desta vez para adicionar os dados do livro. Dentro dela é adicionado um *LinearLayout* contento dois *TextView*, um para adicionar o rótulo de uma característica do livro, como exemplar, data do empréstimo, data da devolução prevista, data de devolução efetiva e biblioteca, que se refere ao nome da biblioteca na qual foi realizado o empréstimo. E o outro para adicionar ao valor referente a cada rótulo. Essa linha é adicionada a seu respectivo *TableLayout* e esse processo se repete até que todas as características dos livros sejam preenchidas nos *TableLayout* correspondente.

Figura 67 - Tela com as informações de empréstimos e débitos



Fonte: elaborado pelos autores (2013).

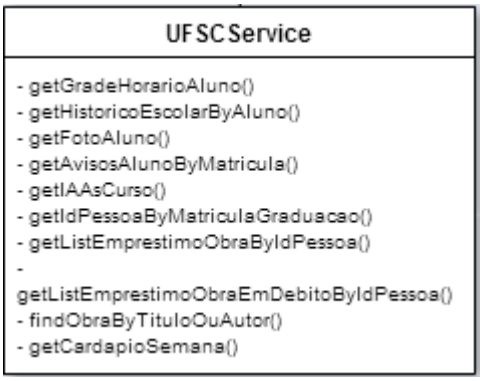
7.5.2 Service

A camada de serviço na aplicação “Minha UFSC” representa a camada de comunicação com os serviços da UFSC. Ela é responsável por montar toda a requisição, fazer o envio da solicitação para o “MinhaUFSC-Auth” (que redireciona para os serviços da UFSC), faz o tratamento de retorno, transformando em objetos para que possam facilmente serem consumidos e apresentados pelas telas do sistema.

Nesta camada é utilizada a biblioteca chamada KSOAP2-Android, ela fornece uma relativa abstração na montagem da comunicação com o protocolo SOAP, e algumas facilidades no tratamento de retorno.

A interface de serviço se chama *UFSCService*, e a partir dela que toda a requisição para UFSC é iniciada. Na imagem abaixo é possível visualizar os métodos presente nessa classe.

Figura 68 - Métodos da classe UFSCService

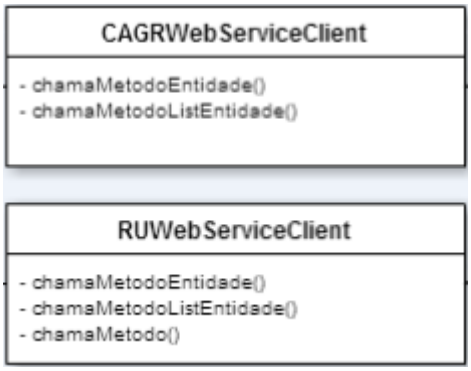


Fonte: elaborado pelos autores (2013).

Como a UFSC possui vários servidores de serviços web, a classe *UFSCService* deve ter a lógica que entende a requisição que está sendo feita, e com isso deve redirecioná-la para o serviço correto. Isso significa que ela deve conhecer todos servidores da UFSC, para que assim consiga fazer o seu devido redirecionamento. Cada um desses servidores é representado por uma classe diferente, e um enumerado que armazena todos os métodos que podem ser acessados por determinado serviço.

O serviço do servidor do CAGR, por exemplo, existe a classe *CAGRWebServiceClient*, que o representa, isso significa que toda a lógica relacionada a este serviço deve estar armazenada nesta classe, seja essa lógica de comunicação, de consumo ou de tratamento de informações. Na imagem a seguir pode-se observar as classes responsável pelo consumo dos métodos do CAGR e do R.U..

Figura 69 - Métodos das classes CAGRWebServiceClient e RUWebServiceCliente



Fonte: elaborado pelos autores (2013).

Os métodos do CAGR que estão disponíveis para consumo pela aplicação estão representados no enumerado *CAGRMetodos* , que foi criado com o intuito de facilitar o desenvolvimento da aplicação, pois ele cria uma maneira organizada e centralizada de armazenar os métodos que podem ser consumidos pela aplicação.

Figura 70 - Classe do tipo Enumerador para armazenar o nome dos métodos

```
10 public enum CAGRMetodos {
11
12     getGradeHorarioAluno,
13     getHistoricoEscolarByAluno,
14     getFotoAluno,
15     getAvisosAlunoByMatricula,
16     getIAAMedioCursoSemestres;
17
```

Fonte: elaborado pelos autores (2013).

A aplicação consome de cinco servidores de serviço diferentes: *AutenticacaoWebServiceClient, BUWebServiceClient, CadastroPessoaosWebServiceClient, CAGRWebServiceClient e RUWebServiceClient. C*ada um deles possui a sua relação de métodos que podem ser acessados para cada serviço, e essa estrutura representa a maneira com que a informação é obtida.

Por mais que existam cinco servidores e cada um deles possua uma lista de métodos distintos, existem semelhanças no processo de montagem da requisição permitindo que seja aplicado o principio de refatoração e reutilização de código. Portanto, foi criada uma classe base, a fim de criar uma hierarquia de herança chamada *BaseWebServiceClient.* Essa classe utiliza o Ksoap de fato, montando o envelope e fazendo a requisição das informações.

As outras classes, são apenas especializações e pequenas implementações especificas para cada serviço.

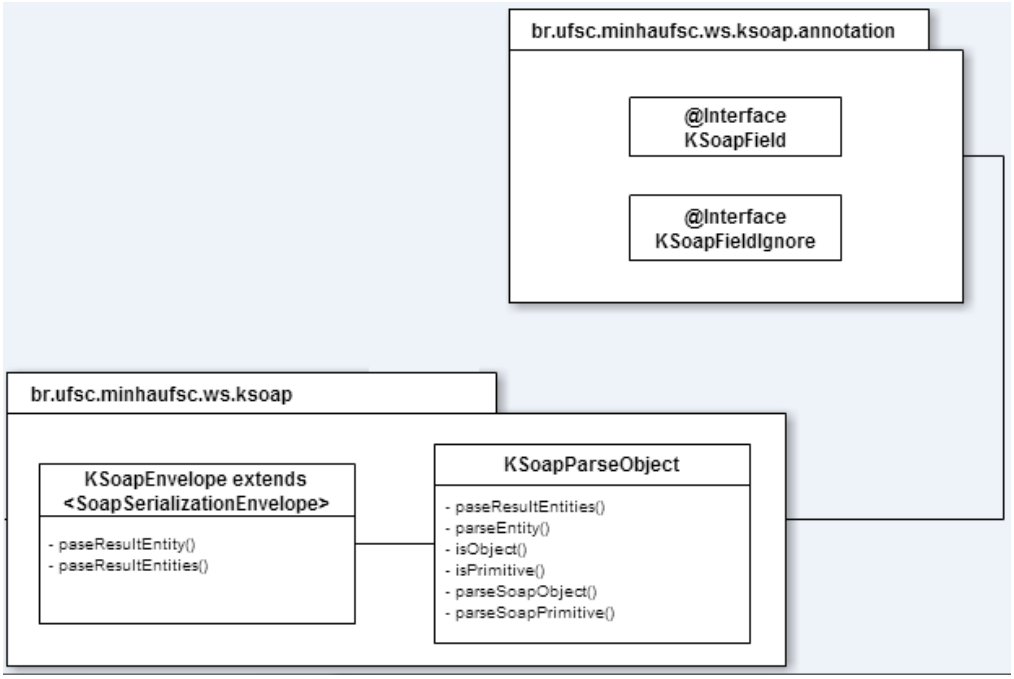
7.5.3 Ksoap

O retorno de uma requisição do *Ksoap* é sempre um objeto da classe *SoapObject*. Este objeto tem uma estrutura semelhante à de um objeto java.util.Map, fazendo o armazenamento de propriedades de forma dinâmica. É muito útil quando se deseja flexibilidade, porém, quando se quer manipular essas informações para

serem exibidas em uma tela, quanto mais concreto for esse objeto, mais fácil e mais legível será esta implementação.

A partir dessa análise que foi criada a classe *KSoapParseObject*, que faz a conversão de um objeto do tipo *SoapObject*, para qualquer entidade escrita em Java através da API de reflection. Para esse processo funcionar, é necessário que exista uma entidade correspondente com as propriedades armazenadas no *SoapObject* que se deseja converter. Para se flexibilizar essa conversão, foram criadas as classes *KSoapField*, *KSoapFieldIgnore*, *KSoapEnvelope*.

Figura 71 - Diagrama de pacotes das classes responsável pela conversão do resultado



Fonte: elaborado pelos autores (2013).

7.5.3.1 KSoapParseObject

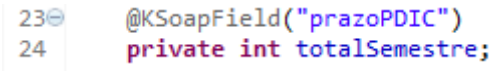
Responsável por fazer a conversão de um *SoapObject* para qualquer tipo entidade, utilizando a API de reflection. Para fazer a conversão é necessário percorrer todas as propriedades existentes na entidade para qual deseja se converter, e verificar se existe uma propriedade correspondente no objeto *SoapObject*. Em caso positivo, a entidade tem o conteúdo da sua propriedade alterado para o conteúdo da propriedade correspondente do *SoapObject*, caso negativo, o conteúdo da propriedade da entidade é alterado para *“null”.*

Para flexibilizar o processo de conversão, foram criadas duas anotações *KSoapField* e *KSoapFieldIgnore*, que funcionam como parâmetro de configuração.

7.5.3.2 KSoapField

O critério de identificação de propriedades é através de propriedades com nomes exatamente iguais. O objetivo desta anotação é facilitar essa identificação, como pode ser visto na imagem abaixo.

Figura 72 - Notação que permite marcar atributo com nome diferente do esperado



Fonte: elaborado pelos autores (2013).

Com esta anotação, o processo de conversão irá utilizar o seu valor para identificar a propriedade armazenada no *SoapObject*, com isso, pode-se ter uma propriedade X na entidade, que está mapeada para a propriedade Y no *SoapObject*.

7.5.3.3 KSoapFieldIgnore

O comportamento padrão do conversor é olhar para todas as propriedades existentes na entidade para qual será convertida. Caso existe alguma propriedade que deva ser ignorada, ela deve ser anotada com *KSoapFieldIgnore*.

7.6 MinhaUFSC-Auth

Um dos principais objetivos deste trabalho é poder criar uma ferramenta que possa ser disponibilizada para a comunidade acadêmica. Para isso é importante levar em consideração todas as questões relativas à disponibilização de dados pessoais de um usuário para que seja possível fornecer um nível adequado de confiança e segurança para o usuário, tendo em vista que através dessa ferramenta ele acessará todos os seus dados relativos à sua vida acadêmica. Do ponto de vista do usuário, tudo será realizado através de uma autenticação de matrícula e senha. Este é o ponto de partida para acesso aos dados. Do ponto de vista da implementação e da comunicação com o *Web service* muitas coisas devem ser

levadas em consideração, como os mecanismos de segurança dos serviços disponibilizados pela UFSC.

Os serviços disponibilizados pela UFSC implementam uma camada de autenticação utilizando o próprio protocolo HTTP, este método é chamado de *autenticação básica*. Ele utiliza o cabeçalho do protocolo HTTP para fazer o envio de usuário e senha para o servidor. Estas informações trafegam pela rede de maneira não criptografada, codificação em base64, e o usuário e senha são separados por dois pontos: Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Devido ao fato das informações trafegarem de maneira codificada, e não criptografada, qualquer pessoa que tome posse dessa requisição HTTP, pode facilmente fazer a sua decodificação e obter os dados de acesso do usuário, por isso é importante que o mecanismo de autenticação básica seja utilizando sempre com o apoio do protocolo HTTPS, pois neste caso ele será o responsável por criptografar todos os dados enviados, inclusive o cabeçalho do protocolo.

Este tipo de autenticação não é baseado em sessão e devido ao fato de que as informações são enviadas através do cabeçalho do protocolo, é necessário que em toda e qualquer e requisição elas estejam presentes, para que assim possam garantir a autenticidade da requisição. Para este método de autenticação, é interessante que o servidor tenha algum mecanismo eficiente para verificação de autenticidade, caso o contrário pode acabar ocasionando em processamento desnecessário, como ter que realizar um processo de login para cada requisição.

O usuário e senha utilizados para consumir os Web services da UFSC são usuários específicos, criados exclusivamente para a integração de aplicações através da web. Portanto, por mais que seja necessária uma autenticação de aluno para a execução da aplicação “Minha UFSC”, a comunicação e obtenção de dados com a UFSC será feita com outro usuário. Esses usuários são criados e administrados pela equipe de tecnologia da UFSC, SETIC (Superintendência de Governança Eletrônica e Tecnologia da Informação e Comunicação).

Este usuário possibilita o consumo dos Web services da UFSC, como verificar pendências de empréstimo de livros com a B.U., obtenção do cardápio do R.U., acesso aos avisos da UFSC, e até mesmo o histórico escolar ou o índice de aproveitamento acumulado de qualquer aluno de graduação da Universidade. Diante disso, é de extrema importância que os dados de acesso sejam armazenados de

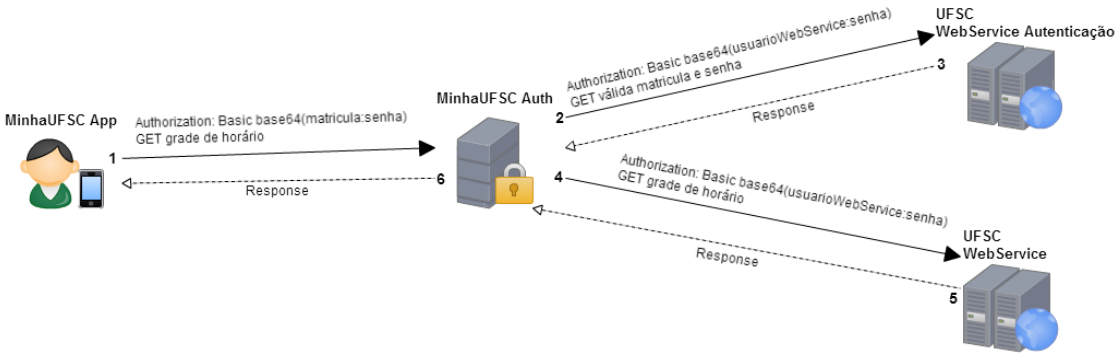
forma segura o suficiente para garantir que só as pessoas certas devam ter acesso, e também que isso não interfira na execução eficiente da aplicação.

Para garantir a segurança deste usuário de comunicação, o primeiro passo é garantir que as suas informações de acesso não estejam armazenadas no código fonte da aplicação, assim não se corre o risco de alguém mal intencionado utilizar engenharia reversa para extraí-las.

A única maneira de continuar se comunicando com os serviços da UFSC, sem que essas informações críticas estejam armazenadas no código fonte e ainda conseguir validar a matrícula e senha do usuário final da aplicação, seria através de um middleware de autenticação.

Este middleware teria a responsabilidade de, receber toda e qualquer requisição oriunda da aplicação “Minha UFSC”, fazer a validação de matrícula e senha do usuário e fazer o redirecionamento da requisição para o serviço solicitado da USFC. Com isso a aplicação não se comunicaria direto com os serviços da UFSC, e sim através deste módulo de segurança. A figura a seguir detalha quais seriam as etapas realizadas em uma requisição para obtenção da grade de horário. Nela podemos perceber que a autenticação entre a aplicação e o modulo de segurança também utiliza o método de autenticação básica. Após isso é necessário fazer a validação de matricula e senha, portanto o “MinhaUFSC Auth” se comunica com o serviço de autenticação da UFSC, enviando as informações a serem validadas. Caso elas estejam corretas, a identificação do serviço destino é feita através de parâmetros enviados dentro da requisição original, e o redirecionamento é feito para o serviço solicitado.

Figura 73 - Ciclo da chamada de uma das funcionalidades, desde a origem (Aplicativo) ao Web service da UFSC.



Fonte: elaborado pelos autores (2013).

A construção do método comunicação com o módulo de segurança é através de serviços web, baseados em *REST* (Representational State Transfe), segundo Stefan Tilkov, editor líder da comunidade SOA da InfoQ, traduzido por Gomes:

> REST é um conjunto de princípios que definem como Web Standards como HTTP e URIs devem ser usados (o que frequentemente difere um pouco do que muitas pessoas atualmente fazem). A promessa é que se você aderir a princípios REST enquanto estiver desenhando sua aplicação, você terá um sistema que explora a arquitetura da Web em seu benefício. (2008)

A construção de uma interface *Web service* utilizando *REST* é bastante simples, e tem a vantagem de não precisar de uma interface de descrição de funcionalidade, como os arquivos *WSDL* (Web Services Description Language) utilizados no protocolo *SOAP* (Simple Object Access Protocol).

Para facilitar ainda mais a implementação deste módulo, optamos por armazenar toda a regra de negócio na aplicação “Minha UFSC”, assim ele seria responsável apenas por fazer redirecionamentos para os serviços correspondentes, além de realizar a validação de matrícula e senha.

A comunicação com a UFSC é feita através do protocolo SOAP, e como toda a regra de negócio esta armazenada na aplicação, é importante que o módulo de segurança atue apenas como um redirecionador, sem realizar qualquer alteração na requisição. Esta abordagem tem a vantagem de funcionar como uma camada de abstração, responsável apenas pela segurança. Para a aplicação “Minha UFSC”, ela estará se comunicando direto com os serviços da UFSC através do protocolo SOAP, portanto, deve-se fazer todas as implementações necessárias para a sua utilização, como a utilização do arquivo *WSDL*.

O módulo de segurança consegue manter esse comportamento devido a maneira com que ele foi implementando, toda a comunicação se da através de uma Uniform Resource Identifier (URI) de entrada, como por exemplo: *http://localhost/minhaufsc-auth/rest.* Esta *URI* recebe como parâmetro uma *Uniform Resource Locator (URL)* o endereço destino do serviço. Desta maneira garante que toda a lógica e definição de serviço sejam de responsabilidade da aplicação “Minha UFSC”.

A *URI* é acessível apenas através de autenticação básica do protocolo HTTP, portanto antes de iniciar a aplicação do smartphone deverá ser inserido um número

de matricula e senha, para que possam ser utilizados durante essa autenticação. A aplicação faz o POST/GET do envelope *SOAP* direto para o “MinhaUFSC-Auth”, exatamente como seria feito se fosse direto para os serviços da UFSC. O módulo fará a validação de matrícula e senha, e caso esteja válido fará o redirecionamento do envelope *SOAP* para a *URL* enviada como parâmetro. Este redirecionamento também irá utilizar o método de autenticação básica, mas não utilizando matrícula e senha, e sim credenciais de outro usuário específico, um usuário de comunicação, criado apenas para esta finalidade.

A construção deste serviço foi feita através da linguagem Java, utilizando o framework Spring/Spring-Security/Spring-MVC para criação do serviço *REST* e criação da autenticação básica, além disso, foi utilizada a geração automática de códigos do Eclipse para arquivos *WSDL* para realizar a validação de matrícula e senha do usuário.

A figura abaixo demonstra em linhas de códigos como é feito o redirecionamento da requisição. Uma nova requisição é criada utilizando o mesmo método, seja GET ou POST, após isso é feito uma cópia da requisição original e adicionado o cabeçalho de autenticação básica utilizando o usuário de comunicação. O resultado dessa requisição é copiado direto para o fluxo de dados (Stream) da requisição original, concluindo assim os passos de redirecionamento.

Figura 74 - Classe principal do Web service em Rest.

```
@Controller
@RequestMapping("/rest")
public class RestController {

    @RequestMapping(method = { RequestMethod.GET, RequestMethod.POST })
    public final void proxyAjaxCall(@RequestParam(required = true, value = "url") String url,
            HttpServletRequest request, HttpServletResponse response) throws Exception {
        url = URLDecoder.decode(url, "utf-8");
        HttpClient client = new DefaultHttpClient();
        HttpRequestBase method = null;

        if (request.getMethod().equals("GET"))
            method = new HttpGet(url);

        else if (request.getMethod().equals("POST")) {
            method = new HttpPost(url);
            Enumeration<String> paramNames = request.getParameterNames();
            List<NameValuePair> urlParameters = new ArrayList<NameValuePair>();
            while (paramNames.hasMoreElements()) {
                String paramName = paramNames.nextElement();
                urlParameters.add(new BasicNameValuePair(paramName, request.getParameter(paramName)));
            }
            ((HttpPost) method).setEntity(new UrlEncodedFormEntity(urlParameters));
        }
        String auth = Base64.encodeBase64String([redacted].getBytes());
        method.setHeader("Authorization", "Basic " + auth);

        HttpResponse httpResponse = client.execute(method);
        IOUtils.copy(httpResponse.getEntity().getContent(), response.getOutputStream());
    }
```

Fonte: elaborado pelos autores (2013).

A figura a seguir demonstra em linhas de códigos a maneira com que a matrícula e senha do usuário são validadas. Tudo se baseia na utilização do serviço criado automaticamente através da geração de código pelo Eclipse *(AutenticacaoServiceService).*

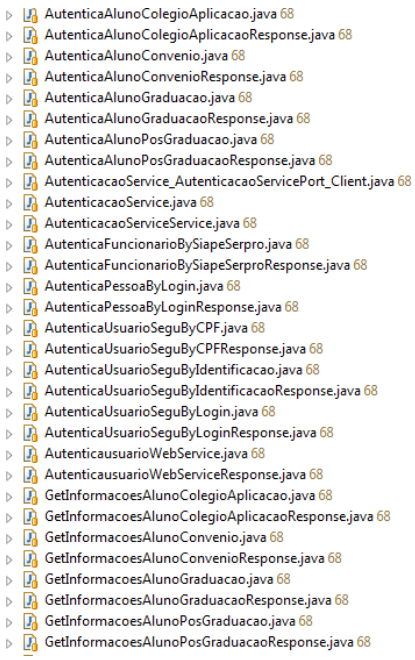
Figura 75 - Método de validação das credenciais do aluno

```
@Override
public Boolean isUsuarioValido(Long matricula, String senha) {

    URL wsdlURL = AutenticacaoServiceService.WSDL_LOCATION;

    AutenticacaoServiceService ss = new AutenticacaoServiceService(wsdlURL, SERVICE_NAME);
    AutenticacaoService port = ss.getAutenticacaoServicePort();

    try {

        AutenticacaoInfo autenticacao = port.autenticaAlunoGraduacao(matricula, senha);
        return autenticacao.getMatricula().getValue().equals(matricula);
    } catch (WebServiceException e) {
        System.out.println("Expected exception: WebServiceException has occurred.");
        System.out.println(e.toString());
        return false;
    }

}
```

Fonte: elaborado pelos autores (2013).

O Eclipse gerou automaticamente mais de 60 classes para o consumo completo de todos os métodos do serviço relacionado à autenticação, algumas dessas classes podem ser vista na figura a seguir.

Figura 76 - Classes geradas para serm utilizadas nos Web services



Fonte: elaborado pelos autores (2013).

8. CONSIDERAÇÕES FINAIS

Este trabalho resultou em uma aplicação para smartphone Android e em um módulo de autenticação.

Durante o desenvolvimento houve alguns desvios em relação ao planejamento, principalmente em relação ao módulo de autenticação, pois o mesmo não estava previsto, junto com ele veio à necessidade de um servidor web específico, o que estreitou ainda mais o cronograma.

Outra dificuldade encontrada foi na criação da interface, por se tratar de um conceito relativamente novo comparado com aplicações web, a dificuldade em planejar uma interface para esse tipo de plataforma pela falta de experiência e infinidades de recursos disponíveis, oferecendo infinitas possibilidades, acabou gerando um pouco de dificuldade.

A disponibilização do orientador e da equipe do SETIC foi de fundamental importância para a realização deste trabalho, sem eles isso não seria possível.

O objetivo principal com esta aplicação era proporcionar para universidade e para os alunos um novo canal de acesso à informação. Para atingir esse objetivo era necessário construir uma aplicação de qualidade, confiável, estável e segura, de fácil utilização e com uma boa experiência de usuário. Mas acima de tudo ela deve agregar valor, ou seja, resolver problemas de fato. Ninguém irá utilizar a aplicação somente pelo fato de ela ter sido bem desenvolvida ou por possuir uma aparência agradável, mas sim por apresentar soluções, e é por isso que os autores classificam como objetivo atingido na elaboração deste trabalho.

9. TRABALHOS FUTUROS

Para trabalhos futuros, algumas melhorias no gerenciamento das informações consumidas do *Web service*, desenvolvido nesse projeto, traria ao aplicativo maior agilidade na obtenção das informações. A ideia seria que esse *Web service* armazenasse informações consumidas dos *Web services* da Universidade em banco ou memória e quando o aplicativo solicitar informações, a respeito de alguma funcionalidade, esse *Web service* só faria uma nova consulta em algum *Web service* da Universidade quando as informações armazenadas nele não forem as mais atuais.

Além disso, seria interessante a adição de novas funcionalidades, como por exemplo, a possibilidade de realizar a rematrícula diretamente pelo aplicativo a cada semestre, assim como também trazer algumas funções do *Moodle*.

A funcionalidade de fazer a rematrícula pelo aplicativo traria a ele uma maior independência para com o sistema da Universidade na qual realiza essa tarefa hoje, como também daria ao aluno mais um meio para á realização dessa função, seja ele dentro do ônibus na volta pra casa ou no intervalo de uma aula, e isso com apenas poucos toques no seu dispositivo móvel.

Já a funcionalidade do *Moodle,* seria muito importante fazer o upload de arquivos para a entrega de tarefas e trabalhos diretamente pelo aplicativo, fazendo com o que o aplicativo seja utilizado dia a dia pelos alunos, além de poder enviar e receber mensagens para o professor de cada disciplina.

Para que essas funcionalidades trabalhem de acordo com muitos aplicativos disponibilizados para dispositivos móveis nos dias atuais, deverá ser disponibilizado que o aplicativo envie notificações para o aluno avisando que novas informações estão disponíveis, como o início e o fim do período de rematrícula, que o professor disponibilizou a nota do semestre e etc.

Realizar feedback com os usuários é de extrema importância para garantir a constante evolução do sistema. Uma maneira de atingir isso é possibilitar o usuário a fornecer sugestões diretas da aplicação, com isso geraria uma sensação de maior proximidade do usuário junto com os criadores da aplicação. Outra maneira de realizar esse feedback é através do mapeamento do comportamento do usuário, existem bibliotecas que fornecem isso de maneira simples e transparente.

Algumas melhorias poderiam ser aplicadas no módulo de autenticação, a fim de garantir a sua disponibilidade e de aumentar a velocidade. Hoje, este módulo é o ponto central de comunicação, todas as informações trafegam por ele, se em algum pico de utilização ele ficar sobrecarregado ele pode afetar toda a cadeia de usuários, e tornar a aplicação inacessível.

Por fim para que esse aplicativo seja utilizado por todos os alunos da Universidade seria desenvolvido esse aplicativo para IOS e para Windows Phone.

REFERÊNCIAS

ACHARTENGINE. Disponível em: < http://www.achartengine.org/> Acesso em: 25 ago. 2013.

ANDROIDE. **DEVELOPER**. <http://developer.android.com/tools/sdk/eclipse-adt.html> Acesso em: 23 abr. 2013.

ASSEMBLA. Disponível em: < https://www.assembla.com/code/minha-ufsc/subversion/nodes /> Acesso em: 31 out. 2013

CAGR. Disponível em: <http://cagr.sistemas.ufsc.br/> Acesso em: 15 mar 2013.

CARVALHO, Suelen Goularte. **Entenda quando usar tag fragment ou tag de layout**. 2013. Disponível em: <http://suelengc.com.br/blog/?cat=18>. Acesso em 25 aug 2013.

CHICOUT, Fabio. **Single-sign-on**. 2011. <http://fabiocesar.wordpress.com/2011/07/24/single-sign-on/> Acesso em: 20 mar. 2013.

CIDRAL, Beline. **Afinal, o que é Android?** 2012. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2011/01/afinal-o-que-e-android.html>. Acesso em: 16 mar. 2013.

DIMARZIO, JEROME F.; **Android**: A programmer's Guide,New York. McGrawHill, 2008.

ECLIPSE (Org.). **About the Eclipse Foundation**. Disponível em: <http://www.eclipse.org/org/>. Acesso em: 15 mar. 2013.

GOMES, André Faria. **Uma rápida Introdução ao REST**. 2008. Disponível em: <http://www.infoq.com/br/articles/rest-introduction/>. Acesso em: 31 out. 2013.

HIGA, Paulo. **Nokia 808 PureView é o último smartphone com Symbian**. 2013. <http://tecnoblog.net/122712/morre-menino-symbian/> Acesso em: 20 mar. 2013.

LECHETA, Ricardo R. **Google Android**: Aprenda a criar aplicações para dispositivos móveis com o Android SDK Sáo Paulo, Novatec. 2010, 608 p, 2ed.

MCQUEEN, ROD; BALSILLIE, JIM; LAZARIDIS, MIKE.BLACKBERRY. **Inside Story of Research in Motion**. New York: H.B. Fenn & Company, 2010.

MINHA UFSC. Disponível em <http://minhaufsc.homologacao.ufsc.br> Acesso em 15 mar. 2013;

OPEN HANDSET ALLIANCE (Org.). 2011. <http://www.openhandsetalliance.com/> Acesso em: 16 mar. 2013.

PACHECO JÚNIOR, Marco Antônio; CASTRO, Reinaldo de Oliveira. **Um estudo de caso da plataforma Android com Interfaces Adaptativas**. São Carlos: UFSCAR. 2011 p. 4

PASSOS, Thiago de Souza. **Android, Arquitetura e Desenvolvimento**. 2009, 75 p. Monografia. Pontifícia Universidade Católica de Minas Gerais.

PORTAL BU. Disponível em: < http://portalbu.ufsc.br /> Acesso em: 31 out. 2013
APACHE MAVEN PROJECT (Org.). **What is Maven?** 2013. Disponível em: <http://maven.apache.org/what-is-maven.html>. Acesso em: 20 mar. 2013.

PRAE. Disponível em: < http://prae.ufsc.br /> Acesso em: 31 out. 2013

RESTAURANTE UNIVERSITÁRIO. Disponível em <http://ru.ufsc.br/ru/>. Acesso em 20 mar. 2013.

SILVA, Ricardo P. E. **Como modelar com UML 2**. Florianópolis, SC: Visual Books, 2009. 320p.

TAURION, Cesar**. Eclipse e Open Innovation Network**. 2007. Disponível em: <https://www.ibm.com/developerworks/community/blogs/ctaurion/entry/eclipse_e_open_innovation_network?lang=en>. Acesso em: 20 mar. 2013.

TOZETTO, Claudia. **Smartphones superam celulares básicos em vendas pela primeira vez, diz estudo**. Disponível em: <http://tecnologia.ig.com.br/2013-04-26/smartphones-superam-celulares-basicos-em-vendas-pela-primeira-vez-diz-estudo.html>. Acesso em: 15 mar. 2013.

XAVIER, Léo. **O internet móvel e o internauta mobile**: Ou sobre o ano do mobile. Disponível em: <http://iabbrasil.net/portal/o-internet-movel-e-o-internauta-mobile-ou-sobre-o-ano-do-mobile/>. Acesso em: 15 mar. 2013.

WIKIPEDIA. **Ficheiro**: Android-System-Architecture <http://pt.wikipedia.org/wiki/Ficheiro:Android-System-Architecture.svg> Acesso em: 20 mar. 2013.

APÊNDICE(S) (A, B, C...)

Junte cópia do instrumento de coleta de dados e outros documentos de sua autoria usados na pesquisa.

ANEXO(S) (A, B, C...)

Anexe cópia de documentos comprobatórios ou ilustrativos que não sejam de sua autoria, se for o caso.