

Xamarin Forms

Desenvolvimento de aplicações móveis multiplataforma



Casa do
Código

EVERTON COIMBRA DE ARAÚJO

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Adriano Almeida

Vivian Matsui

Revisão

Bianca Hubert

Vivian Matsui

[2017]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

ISBN

Impresso e PDF: 978-85-5519-240-1

EPUB: 978-85-5519-241-8

MOBI: 978-85-5519-242-5

Você pode discutir sobre este livro no Fórum da Casa do Código: <http://forum.casadocodigo.com.br/>.

Caso você deseje submeter alguma errata ou sugestão, acesse <http://erratas.casadocodigo.com.br>.

PREFÁCIO

O desenvolvimento de aplicativos para dispositivos móveis se popularizou, apresentando um crescimento extraordinário nos últimos anos. Além disso, é uma área da informática que tem despertado o interesse de desenvolvedores e de "aventureiros" no assunto.

Existem sistemas operacionais para dispositivos móveis que se destacam no mercado, como o Android, mantido pelo Google e pela Open Handset Alliance, o iOS da Apple, e o Windows Phone da Microsoft. Cada uma destas plataformas possui características particulares para o desenvolvimento nativo de seus aplicativos, que exige de nós, desenvolvedores, o conhecimento da linguagem de programação utilizada, de suas bibliotecas e dos recursos disponibilizados pela plataforma.

Enfim, para construir um aplicativo para cada uma das plataformas citadas, é necessário se especializar em todas. Não seria mais fácil desenvolver um aplicativo que funcionasse em todas elas, sem precisar reescrevê-lo com todo um ferramental tecnológico diferente?

É aí que entra o conceito de desenvolvimento de aplicativos híbridos. Um aplicativo híbrido tem características das funcionalidades nativas e da web que, dependendo da tecnologia que é utilizada, pode possuir códigos de ambos para sua criação.

Entretanto, o aplicativo híbrido necessita de um framework intermediário que atue entre o aplicativo nativo e o dispositivo hospedeiro. Podemos até aplicar o slogan da Sun Microsystems, “escreva uma vez, execute em qualquer lugar”, ao se referir à linguagem Java, que neste contexto se aplicaria no desenvolvimento

de um aplicativo em uma única linguagem e execução em diferentes sistemas operacionais.

O Xamarin, objeto de estudo deste livro, tem uma concepção de desenvolvimento um pouco diferente. Surge como uma inovação para o ramo do desenvolvimento móvel, tornando mais fácil e produtiva a entrega de aplicativos móveis para diferentes plataformas. Em conjunto com o Visual Studio, o Xamarin nos permite criar aplicativos móveis usando a linguagem C# e a plataforma .NET, a fim de ter a experiência de desenvolvimento híbrido para os principais dispositivos com iOS, Android e Windows Phone.

O diferencial do Xamarin é que o resultado final gerado a partir do código do desenvolvedor não é um aplicativo híbrido, mas um ou mais aplicativos nativos gerados a partir da mesma estrutura de código.

Este livro nos permite explorar as potencialidades do Xamarin e suas tecnologias afins. No decorrer dos capítulos, são abordados os principais conceitos para a construção de um aplicativo para dispositivos móveis, disponíveis para as plataformas mais usadas no mercado.

A abordagem prática do assunto é demonstrada de forma objetiva pelo autor, em uma linguagem clara e adequada para iniciantes no assunto, sendo que você poderá fazer a aplicação imediata de seus exemplos. Desafie-se para o aprendizado de abordagens e tecnologias inovadoras que estão em alta no mercado. Será uma experiência fantástica para a sua promoção intelectual.

O autor do livro tem grande experiência, de longo tempo, relacionado ao desenvolvimento de softwares. É professor, pesquisador, palestrante e, com destaque, um entusiasta no ramo de desenvolvimento.

Possui conhecimento em várias áreas relacionadas às tecnologias de desenvolvimento, como Orientação a Objetos, linguagem de programação Java e seus padrões JavaSE/JavaEE, padrões de projeto, programação para dispositivos móveis e, o principal foco de estudo dos recentes anos, linguagem de programação C# e as tecnologias baseadas na plataforma .NET. Assim, temos a segurança de que o conteúdo aqui tratado foi escrito por um profissional dedicado e com propriedade de conhecimento no assunto.

Faça bom proveito do livro. Uma boa leitura a todos!

Ricardo Sobjak

Universidade Tecnológica Federal do Paraná — Campus Medianeira

SOBRE O LIVRO

Este livro traz, na prática, o desenvolvimento de aplicações *cross-platform* com o Xamarin e Xamarin Forms, frameworks para desenvolvimento de aplicativos para dispositivos móveis. O desenvolvimento de um aplicativo para ser publicado em dispositivos com plataformas diferentes (iOS, Android e Windows, de forma nativa, é uma tarefa muito tranquila com o Xamarin. É possível criar uma aplicação, utilizando a linguagem C#, e ela ser publicada para as três plataformas.

O livro é desenvolvido em dez capítulos, sendo o primeiro apenas teórico, mas não menos importante, pois trago nele contextualizações sobre dispositivos móveis e as ferramentas usadas no livro. Já no segundo capítulo, apresento o processo de instalação e teste da plataforma e dos IDEs.

No terceiro capítulo, como "aperitivo", uma página onde um conjunto de dados estáticos são exibidos, e um controle chamado `ListView` é implementado. Depois, no capítulo quatro, evoluímos na maneira como o `ListView` pode ser utilizado. Este componente é, sem dúvida, o mais usado em aplicações móveis e nós vamos conhecer seus recursos e possibilidade praticamente em todos os capítulos do livro — a exceção será só nos capítulos introdutórios e no de gráficos. Em relação à inserção de dados, começaremos com uma página simples de interação com o usuário.

O capítulo cinco é intenso e muito bom. Iniciaremos as atividades com uso de componentes de terceiros, o que traz sempre grande contribuição e produtividade para nossas aplicações. Poderemos, com o auxílio destes componentes, acessar a câmera e álbum de fotos dos dispositivos. Penso que você gostará deste primeiro contato com a execução da aplicação em seus aparelhos e

não apenas em emuladores.

No sexto capítulo, o que penso ser um dos mais importantes, nós trabalharemos com acesso a bases de dados, por meio do SQLite. Implementaremos a persistência em tabelas isoladas e com relacionamentos; tudo isso pensando em objetos e fazendo uso do LINQ na recuperação dos dados.

Em relação ao `ListView`, evoluiremos muito com agrupamentos e com os `Action Context`, que são as opções de ação para cada item dele. Neste sexto capítulo, ainda veremos a criação de componentes customizados, que podem ser reutilizados e criaremos uma página de pesquisa para servir de opção quando um *combobox* ou *dropdownlist* se fizer necessário.

O sincronismo, que é um ponto muito relevante em aplicações móveis, é comentado e introduzido no capítulo sete. Implementaremos uma aplicação servidora, fazendo uso de RESTful com o ASP.NET WEB API; publicaremos esta aplicação no Windows Azure; e consumiremos os serviços na aplicação do dispositivo móvel, onde os dados serão sincronizados entre dispositivo e nuvem.

Mapas, que são também muito importantes em um dispositivo móvel, terão sua introdução no oitavo capítulo. Nele faremos uso de MVVM como arquitetura para atualização e ligação da camada de visão com a de negócio.

Muita emoção ficou reservada para os dois capítulos finais. No capítulo nove, teremos o núcleo da aplicação implementado, o pedido de um cliente. Trabalharemos uma página (tela/janela com relacionamento mestre-detalhe, enviaremos SMS e acessaremos os mapas de cada plataforma para traçar uma rota de entrega para cada pedido.

Em relação à geolocalização, trabalharemos com a posição de um dispositivo em um mapa, o que possibilitará a você criar um serviço de rastreamento ou acompanhamento da entrega de um pedido, por parte do cliente. O último capítulo, finalizando o livro, trará um componente (de um conjunto de componentes comercial, que possibilita a geração de gráficos.

Certamente, este livro pode ser usado como ferramenta em disciplinas que trabalham o desenvolvimento de dispositivos móveis, quer seja por acadêmicos ou professores. Isso porque ele é o resultado da experiência que tenho em ministrar aulas dessa disciplina, então trago para cá anseios e dúvidas dos alunos que estudam comigo.

É importante que o leitor tenha conhecimento de Orientação a Objetos e da linguagem C#, mas não é um fator impeditivo. O repositório com todos os códigos-fonte usados no livro pode ser encontrado em: <https://github.com/evertonfoz/xamarin-casa-do-codigo>.

Agradecimentos

Quero deixar registrados meus sinceros agradecimentos a toda a equipe da Casa do Código, em especial a Vivian Matsui, que foi meu contato direto durante todo o processo de escrita deste livro, sempre colaborando.

Não poderia ter testado as aplicações em dispositivos físicos se não fossem os alunos Carlos Antônio Bertoncelli Júnior, com seu Windows Phone, e o Tiago Angeli Cavaliéri, com seu Samsung S6.

Agradeço também ao meu amigo e colega de trabalho, Ricardo Sobjak, pelo constante apoio em diversas atividades na universidade e por ter aceito o meu convite para escrever o prefácio deste livro.

Aos meus colegas de trabalho do Departamento de Computação e da DIRGRAD, meu muito obrigado a todos.

A Editora Casa do Código agradece ao Carlos Panato por colaborar com a revisão técnica.

Sobre o autor

Everton Coimbra de Araújo atua na área de treinamento e desenvolvimento. É tecnólogo em processamento de dados pelo Centro de Ensino Superior de Foz do Iguaçu, possui mestrado em Ciência da Computação pela UFSC e doutorado pela UNIOESTE em Engenharia Agrícola.

É professor da Universidade Tecnológica Federal do Paraná (UTFPR, campus Medianeira, onde leciona disciplinas no Curso de Ciência da Computação e em especializações. Já ministrou aulas de Algoritmos, Técnicas de Programação, Estrutura de Dados, Linguagens de Programação, Orientação a Objetos, Análise de Sistemas, UML, Java para Web, Java EE, Banco de Dados e .NET.

Possui experiência na área de Ciência da Computação, com ênfase em Análise e Desenvolvimento de Sistemas, atuando principalmente nos seguintes temas: Desenvolvimento Web com Java e .NET e Persistência de Objetos.

O autor é palestrante em seminários de informática voltados para o meio acadêmico e empresarial.



Sumário

1 Dispositivos móveis, desenvolvimento cross-platform e o Xamarin	1
1.1 Os dispositivos móveis na atualidade	2
1.2 O desenvolvimento móvel cross-platform	5
1.3 O Xamarin	7
1.4 Conclusão	8
2 Xamarin — Instalação e testes	10
2.1 Instalação em um MacBook Pro e teste de uma execução básica	10
2.2 Instalação das ferramentas em um PC com Windows 10 e teste de uma execução básica	28
2.3 Conclusão	53
3 O início da aplicação	54
3.1 Criação da aplicação no Xamarin Studio	54
3.2 Criação da página do menu de opções	64
3.3 Conclusão	74
4 Implementação de um formulário com XAML	76
4.1 Criação da aplicação no Visual Studio	77
4.2 A listagem dos entregadores	84
4.3 A inserção de novos entregadores	88

4.4 Universal Windows Platform	95
4.5 Conclusão	97
5 Acesso à câmera e à galeria de fotos	98
5.1 Publicação da aplicação para um dispositivo iOS	98
5.2 Publicação da aplicação para um dispositivo Android	105
5.3 Publicação da aplicação para um dispositivo Windows Phone	
5.4 Inserção de imagens às listagens	106 ¹⁰⁵
5.5 Interação com a câmera e o álbum	112
5.6 Alteração de dados existentes na coleção	124
5.7 Alterando o ícone, o nome da aplicação e a cor da página de abertura	133
5.8 Conclusão	137
6 O uso de banco de dados com o SQLite	138
6.1 Instalação do SQLite na aplicação	139
6.2 Adaptação da classe de modelo para a persistência	140
6.3 Implementação da persistência para a classe TipoItemCardapio	141
6.4 Adaptação da interface com o usuário e seus comportamentos	
6.5 Recuperação de imagens da base de dados e exibindo-as no ListView	146 151
6.6 Associações/relacionamentos com o SQLite	153
6.7 A página de listagem para a classe associada	156
6.8 Controles personalizados	160
6.9 Inserção com um controle customizado	163
6.10 Uma página de pesquisa	164
6.11 Finalizando a inserção do item de cardápio	170
6.12 Exibição dos itens (associação) no ListView	174
6.13 A alteração de um item de cardápio já persistido.	179
6.14 Manipulação da base de dados do SQLite	184

6.15 Conclusão	185
7 Sincronismo com serviços REST Web API	187
7.1 A aplicação que será a servidora na web e seu modelo de negócios	188
7.2 O acesso a dados para a aplicação servidora	193
7.3 Os serviços web RESTful	196
7.4 Aplicação, o banco de dados e Windows Azure	199
7.5 Preparação da aplicação mobile para consumir os serviços RESTful	207
7.6 A interface com o usuário para a configuração do dispositivo	
7.7 A inserção de garçons na aplicação	216 ²¹¹
7.8 A sincronização do dispositivo com a aplicação servidora	222
7.9 Conclusão	228
8 Aplicação do MVVM e o uso de mapas	229
8.1 A classe de negócio e o DAL	230
8.2 O MVVM — Model-View-View Model	231
8.3 Listagem e inserção de clientes	237
8.4 Localização do cliente em um mapa	244
8.5 Alteração e remoção de um cliente já inserido	257
8.6 Conclusão	258
9 Pedido de venda, rotas em mapas e SMS	260
9.1 Classes de modelo para registro de pedidos	260
9.2 DAL para pedidos	264
9.3 A listagem dos pedidos	266
9.4 Inserção de novos pedidos	274
9.5 Transição de fases do pedido, com envio de SMS	286
9.6 Verificação da rota para a entrega do pedido	288
9.7 Registro da posição do entregador	291

9.8 Conclusão	295
10 Gráficos	296
10.1 Instalação do Syncfusion	296
10.2 Inserção das referências para o uso de gráficos do Syncfusion	
10.3 Um gráfico de barras	303 ³⁰²
10.4 Conclusão	307
11 Os estudos não param por aqui	308

CAPÍTULO 1

DISPOSITIVOS MÓVEIS, DESENVOLVIMENTO CROSS-PLATFORM E O XAMARIN

Olá! Seja bem-vindo ao primeiro capítulo deste livro. Ele será curto e conterá apenas teoria, mas nos demais compensarei com a prática. Nele buscarei trazer conteúdo sobre o panorama dos dispositivos móveis nos dias de hoje, tipos de dispositivos e suas características e opções para o desenvolvimento de aplicativos.

Caso tenha interesse, seguem os links para meus trabalhos anteriores:

- <http://www.casadocodigo.com.br/products/livro-c-sharp>
- <https://www.casadocodigo.com.br/products/livro-aspnet-mvc5>
- <http://www.visualbooks.com.br/shop/MostraAutor.asp?proc=191>

As ferramentas que serão usadas neste livro serão apresentadas conforme forem necessárias. É importante que você tenha

conhecimento de Orientação a Objetos e da linguagem C# para um perfeito acompanhamento do desenvolvimento proposto para este livro.

A aplicação proposta para ser desenvolvida neste livro, a partir do terceiro capítulo, refere-se ao atendimento oferecido por uma pizzaria, com serviços de entrega a domicílio. Mas após a implementação trabalhada no livro você terá subsídios para uma implementação com atendimento local, em mesas.

Dentre as funcionalidades previstas estão: cadastros (mesas, garçons, entregadores e cardápio), processos (registro de pedido para entrega), acompanhamentos (pedidos e entregas) e gestão (gráfico de vendas). Nestas funcionalidades, serão trabalhados tipos de telas para a aplicação, controles de entrada de dados, listagem de dados, templates para aparência da aplicação, gráficos, acesso a banco de dados, consumo de serviços web, uso de GPS e câmera. Pelas características apontadas anteriormente, é possível notar que construiremos um "iFood".

1.1 OS DISPOSITIVOS MÓVEIS NA ATUALIDADE

Até anos atrás, era comum conhecer pessoas que adquiriam computadores apenas para navegar na internet, ler e-mails e acessar algumas aplicações para leitura de livros, artigos ou documentos diversos. No lado corporativo empresarial, notebooks eram fornecidos aos colaboradores para que desempenhassem algumas atividades, como registro de uma venda para um cliente, recebimento de uma conta, anotação de um pedido e agendamento de compromissos, dentre diversas outras atividades.

Com o surgimento dos dispositivos móveis, a venda de computadores pessoais tem sofrido constantes quedas. Isso

começou lá atrás, de maneira modesta e quase despercebida, com handhelds, palmtops e PDAs (*Personal Digital Assistants*), mas que começou a ganhar destaque com a chegada do iPod e depois dos iPhones.

Lembro bem de quando Steve Jobs fez o lançamento do primeiro iPad e todos olharam para "aquilo" de maneira incrédula. Mal sabiam a revolução comportamental que aquele dispositivo traria. A massificação veio por meio do Google, com o desenvolvimento do sistema operacional Android para dispositivos mais **ace\$\$ívei\$**, em relação aos produtos da Apple.

E os notebooks? Estão cada vez mais finos, mais leves e mais parecidos com tablets. E para aqueles que "só" queriam ler seus livros em um meio que não fosse no papel, surgiram os e-readers. Porém, estes livros também podem ser lidos em smartphones e tablets.

Embora um dispositivo móvel possa permitir acesso para necessidades pessoais e corporativas, é importante ressaltar que a maneira como este mercado se desenvolveu foi bem distinta. Tem-se o lado recreativo, em que é possível ter em seu dispositivo diversos jogos; o cultural, que permite o acesso a filmes e livros; a organização pessoal, com diversos recursos para gestão financeira e de compromissos, acesso a bancos, compras; e a parte empresarial, com aplicativos corporativos, de gestão, operacional das empresas e vendas.

Hoje, os smartphones são mais utilizados como ferramentas computacionais do que como aparelho telefônico e despertador. :-)

Toda essa popularização e mudança comportamental resultaram no que é visto atualmente como um fenômeno BYOD (*Bring Your Own Device* — Traga seu Próprio Dispositivo). Ou seja, o colaborador está levando para seu ambiente de trabalho um

dispositivo potente (o seu) e que pode, em alguns casos, ser substituído por um computador, quer seja desktop ou notebook.

Com isso e o acesso à rede corporativa liberado, todos podem ter no bolso as aplicações de seu local de trabalho. Isso pode gerar mais produtividade também (ou não :-)). Porém, no que diz respeito à segurança, traz novas ameaças e vulnerabilidades, mas isso é assunto para a equipe de infraestrutura.

A massificação da internet (que ainda está longe de ser atingida) foi um fator importante para que os dispositivos móveis fossem difundidos da maneira como se encontram. E ela ainda aumentará com o surgimento de relógios e óculos inteligentes, maiores televisores, geladeiras, micro-ondas e outros que nem sabemos que surgirão.

Pense em empregos nos quais se exija atividade externa, quer seja em campo ou urbana, como a função de um engenheiro agrônomo ou um vendedor externo. Esses profissionais precisavam se dirigir à empresa, buscar blocos ou impressos específicos para coleta de dados, retornar aos seus escritórios e registrarem os dados coletados. Com um dispositivo móvel, ele não precisa nem ir ao seu escritório, pois seu equipamento já permite o registro de sua coleta ou venda no momento em que ela ocorre.

E se o acesso à internet não for possível no momento da atividade, é possível uma sincronização com os servidores da empresa tão logo uma conexão seja possível. Já viram na televisão, no jornalismo, que os repórteres se comunicam com suas centrais pelo Skype? Com transmissão ao vivo para os telespectadores? Isso é mobilidade. :-)

As empresas buscam também tirar proveito dessa realidade, quer seja dando maior liberdade para seus colaboradores, ou economizando em despesas como energia, água, condomínio ou aluguel, dentre outras.

Com o advento da "internet para o usuário", surgiu o tema relacionado à Experiência do Usuário, pois, ao se desenvolver aplicações para o ambiente web, não era desejado que nesta plataforma o usuário tivesse a mesma tela de aplicação que tinha em um ambiente desktop. Com o surgimento cada vez maior de aplicativos para dispositivos móveis, este tema reforçadamente é debatido. O usuário quer fazer uso dos recursos e características do dispositivo que usa e não ter nele a aplicação com a mesma aparência que ela tem, agora, na web.

Neste livro, trabalharemos aplicações que possam ser executadas nos sistemas operacionais iOS (iPhones e iPads), Android e dispositivos com a plataforma Windows. Nestes dois últimos, existem diversos dispositivos, nas mais diversas especificações de tamanho de tela, recursos e processamento.

1.2 O DESENVOLVIMENTO MÓVEL CROSS-PLATFORM

Não é porque os dispositivos móveis estão em uma grande ascendência que os computadores pessoais se tornaram descartáveis. Existem ainda diversas atividades e aplicativos que precisam ser realizados e utilizados em um computador que possua um teclado (que não seja na tela), uma tela grande (às vezes mais do que uma) e um desempenho ainda não alcançado pelos dispositivos móveis atuais.

Com isso posto, é importante estar ciente de que as aplicações comerciais trazidas para o ambiente móvel precisam, nas interações com o usuário, ter ou solicitar dados e informações que possam caber "confortavelmente" na tela que será usada. Nada de encher de informações que são desnecessárias para o processo atual.

O desenvolvimento para dispositivos móveis, até a presente

data, vinha sendo focado nas plataformas iOS e Android — estas sendo as mais utilizadas. Entretanto, uma terceira vem surgindo e, como traz o peso do nome, é preciso também se preocupar com ela. Estou falando do Windows.

Cada uma dessas plataformas possui suas próprias características e recursos. Com isso, você pensa em escolher uma plataforma para que seu aplicativo funcione e limitar o uso a apenas esta?

Se for uma empresa que especifique a plataforma e o aplicativo deva funcionar apenas entre os colaboradores dela, isso é uma opção sim. Mas e se sua aplicação precisar ser utilizada em plataformas diferentes? Aí esta opção já não poderia ser escolhida.

O que vem ocorrendo no desenvolvimento para dispositivos móveis é que, ao desenvolver uma aplicação para um dispositivo e uma plataforma, esta aplicação possa funcionar em qualquer dispositivo, de qualquer plataforma.

Por exemplo, deseja-se que uma aplicação desenvolvida para o iPhone 6 possa funcionar da mesma maneira em um smartphone com Android, ou ainda no tablet da Microsoft, o Surface. O que fazer para resolver este problema? Montar várias equipes de desenvolvimento, cada uma usando um ambiente de desenvolvimento diferente e focada em uma plataforma? Desenvolver diversas versões do mesmo aplicativo? Isso não é produtivo, e nem barato, concorda?

O objetivo é partir para uma ferramenta que possibilite o desenvolvimento *cross-platform*. Existem várias. Algumas geram o aplicativo para ser executado em um ambiente específico, como se fosse uma máquina virtual (conhecido como ambiente híbrido). Este tipo de aplicativo corre o risco de ter a "mesma cara" em dispositivos diferentes, não trazendo benefícios para a experiência

do usuário.

Outras ferramentas, como é o caso do Xamarin, geram aplicativos nativos para as plataformas escolhidas, permitindo fazer uso das características e recursos oferecidos por estas plataformas e seus dispositivos. No caso do Xamarin, podemos desenvolver uma aplicação que funcione nas três plataformas que adotamos, fazendo uso da mesma ferramenta (Xamarin Studio ou Visual Studio) e usando a mesma linguagem e todos os recursos oferecidos por ela, o C#.

Existem ainda aplicações desenvolvidas para a web e que são utilizadas exclusivamente para dispositivos móveis. A melhor opção é o desenvolvimento nativo, e o Xamarin permite isso.

Um ponto importante é verificar na App Store (iOS), no Google Play e na Windows Store os padrões e as regras que devem ser respeitados para a publicação oficial de sua aplicação. Por exemplo, os ícones das aplicações são diferentes em cada plataforma, assim como a barra de navegação e o padrão de cores. São alguns dos pontos que você deve verificar e garantir que sua aplicação os respeite para que a publicação seja aceita.

1.3 O XAMARIN

O Xamarin, até pouco tempo atrás, era uma plataforma proprietária, com custo para o desenvolvedor. Recentemente, ele foi adquirido pela Microsoft, que eliminou este custo, deixando-o gratuito, tanto para o Xamarin Studio como para o Visual Studio. Além disso, seguindo a nova filosofia da Microsoft, ela liberou o código-fonte do Xamarin. Ou seja, ele é free e open source.

Como uma plataforma para desenvolvimento de aplicativos cross-platform, o Xamarin tem como foco dispositivos móveis com

o iOS, Android, Windows Phone (a versão 8.1). Além destas três plataformas, é possível a criação de aplicativos Universal Windows Platform, ou seja, aplicativos para qualquer plataforma Windows.

Segundo a documentação do Xamarin, é possível a reutilização de 75% a 100% de código. Isso quer dizer que são poucas as situações em que será necessário escrever código específico para uma das plataformas citadas.

Aplicativos desenvolvidos por meio do Xamarin e do C# têm acesso nativo e total às plataformas que os executarão, podendo extrair ao máximo seus recursos. Como o Xamarin faz uso do C# e da plataforma .NET, é possível também que você faça uso de grande parte da API disponibilizada para aplicativos Windows (falo aqui do .NET), em suas aplicações mobile.

Ao utilizar o Xamarin como plataforma de desenvolvimento, é possível o compartilhamento de praticamente toda a lógica de negócio entre as plataformas alvo. Ou seja, você escreve o código uma única vez e o invoca em cada plataforma de execução.

Já o Xamarin Forms, que é uma plataforma direcionada para o desenvolvimento da camada de apresentação, permite o compartilhamento da interface com o usuário. Ou seja, você pode desenhar sua tela uma única vez e ela será renderizada, de maneira nativa, em cada plataforma móvel, usando seus controles nativos. Tudo isso, codificando em C#. As interfaces com o usuário podem ser codificadas fazendo uso de C# ou do XAML específico do Xamarin, que segue a mesma filosofia do XAML do WPF e Silverlight.

1.4 CONCLUSÃO

Bem, conforme prometido no início deste capítulo, ele foi breve.

Eu poderia trazer muito mais teoria aqui, mas penso que você quer começar logo com a prática. Entretanto, precisava lhe apresentar o texto aqui trabalhado.

É importante ler um pouco sobre dispositivos móveis, cross-platform, o Xamarin e o Xamarin Forms. É claro que o que eu trouxe para você foi apenas uma introdução, mas acredite, com ela você já está pronto para começar o desenvolvimento móvel e tem motivos para escolher o Xamarin. Isso porque você quer e precisa desenvolver para, no mínimo, três plataformas e posso assumir que não quer, ou não pode, ter uma equipe para cada plataforma.

Você precisa de produtividade e quer que sua aplicação seja desenvolvida em uma única plataforma, mas que seja executada nas três mais usadas atualmente. Então, a solução que proponho é esta: Xamarin, Xamarin Forms e C#.

No próximo capítulo, instalaremos a ferramenta em um MacBook Pro e em um PC Windows, e testaremos seu funcionamento. Até lá.

CAPÍTULO 2

XAMARIN — INSTALAÇÃO E TESTES

Quando escrevo ou leio algum livro, não gosto muito quando janelas relacionadas ao processo de instalação são representadas como figuras para o leitor. Entretanto, o maior público para os livros na atualidade são os que os leem em sua forma digital, ou eletrônica, como um e-book.

Analisando este perfil, sou forçado a pensar que a leitura pode ocorrer em qualquer lugar, no ônibus, na mesa após uma refeição, em um banco de praça ou em infinitos outros lugares. Com isso, optei por apresentar algumas figuras. Elas poderão ajudar na abstração caso a leitura esteja sendo realizada longe de seu equipamento de desenvolvimento.

Desta maneira, peço antecipadamente a compreensão pelo grande número de figuras neste capítulo, mas julguei-as necessárias para o processo, e sua leitura é fácil e rápida. Vamos lá então.

2.1 INSTALAÇÃO EM UM MACBOOK PRO E TESTE DE UMA EXECUÇÃO BÁSICA

Nossa primeira instalação será em um MacBook Pro. A configuração dele é de 4GB de RAM e 128HD. Minha máquina possui o OS X El Captain (Versão 10.11.5). Antes de iniciar a instalação, é preciso que você instale o XCode em sua máquina.

Dirija-se à App Store para isso.

Instalação do Xamarin Platform

Como primeiro passo, precisamos obter o pacote do Xamarin para começarmos. Para isso, acesse <http://xamarin.com>. No menu que aparece no topo da página, escolha a opção **Products**, e então a **Xamarin Platform**, tal qual mostra a figura a seguir.

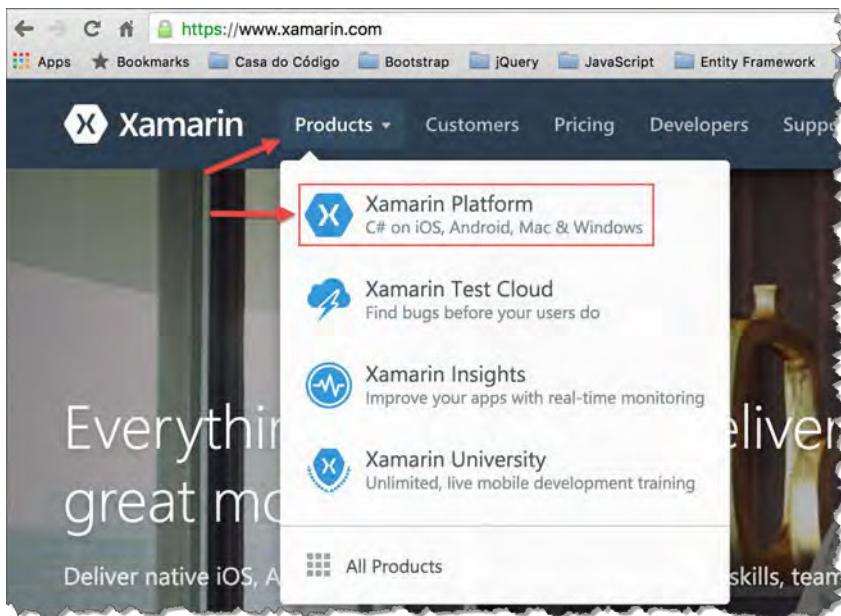


Figura 2.1: Página principal do site do Xamarin

Ao clicar no link anteriormente apresentado, você será redirecionado para a página de download do Xamarin. Nela, um link específico para download é apresentado. Veja no destaque da figura a seguir.

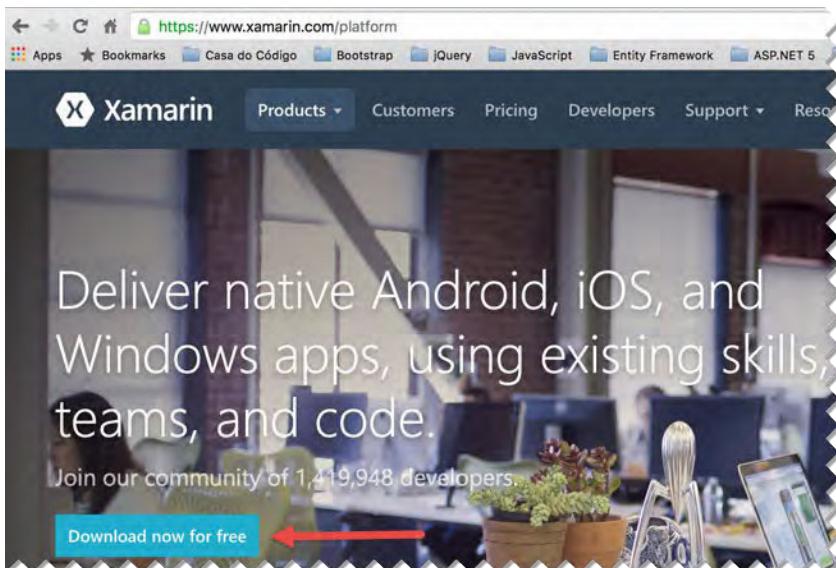


Figura 2.2: Página para acessar o download do Xamarin

Clicando no botão destacado na imagem anterior, um formulário com solicitação de poucas informações é exibido. Você precisa preenchê-lo para que o download possa ocorrer, como mostra a figura a seguir.

Tell us a bit about yourself.

Everton Coimbra de Araújo

evertongoimbra@gmail.com

UTFPR

I already have Visual Studio installed

No

My company has more than 5 developers

No

I agree to the [Terms & Conditions](#)

[Download Xamarin Studio for OS X](#)

or [Visual Studio Community with Xamarin for Windows](#)

The screenshot shows a web-based download form for Xamarin. At the top, it says "Tell us a bit about yourself." Below are four input fields: name ("Everton Coimbra de Araújo"), email ("evertongoimbra@gmail.com"), affiliation ("UTFPR"), and company developer count ("I already have Visual Studio installed" dropdown set to "No"). Another developer count dropdown is shown below. A checkbox for accepting terms and conditions is present. At the bottom, there are two prominent blue buttons: "Download Xamarin Studio for OS X" and "or Visual Studio Community with Xamarin for Windows".

Figura 2.3: Formulário para realizar o download do Xamarin

Ao confirmar, o download será iniciado e, ao ser concluído, execute o arquivo recebido. Bem, a primeira tela que será exibida refere-se à verificação de segurança na execução do arquivo baixado diretamente pela web e não pela App Store. Veja a figura a seguir. Confirme a execução abrindo o arquivo.



Figura 2.4: Alerta de segurança

Após confirmar a compreensão do alerta de segurança, você terá uma janela de execução tal qual a apresentada na figura a seguir. Dê um duplo clique na imagem da aplicação para que o processo de instalação seja iniciado. Será então apresentada uma janela com a licença de uso, confirme ciência, aceite e prossiga com a instalação.

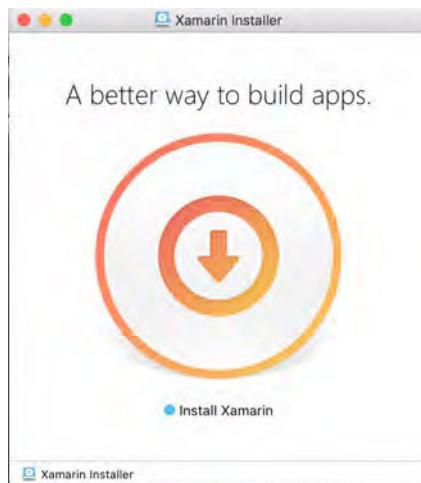


Figura 2.5: Formulário para realizar o download do Xamarin

É importante saber que a conexão com a internet deverá ser mantida por todo o processo, pois os pré-requisitos para a instalação terão seu download realizado neste momento. Outro ponto a se saber é que, durante a instalação, pode ser pedida a sua

senha de administrador da máquina onde se está executando a instalação.

A figura a seguir apresenta os componentes que são possíveis de instalar. Com exceção de `Xamarin.Mac`, todos deverão estar marcados. Porém, optei por deixar todos marcados em minha instalação.

`Xamarin.Android` e `Xamarin.iOS` são os componentes necessários para o desenvolvimento para os dispositivos móveis com Android e iOS, respectivamente. O `Xamarin.Mac` é para desenvolvimento de aplicativos Mac, e o `Intel HAXM` é utilizado para os emuladores Android.

Clique em `Continue` para começar a instalação. Durante o processo, será solicitado que você aceite a licença de uso para o SDK Android.

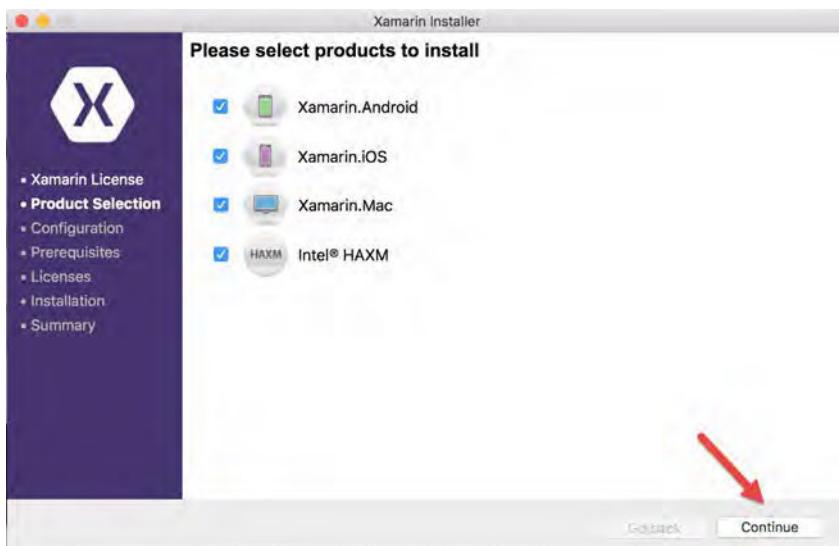


Figura 2.6: Apresentação de componentes possíveis de se instalar

Antes de ser iniciada a instalação, é exibida uma janela

apresentando os pré-requisitos que serão instalados para que os componentes anteriormente selecionados possam ser instalados. Veja a figura a seguir.

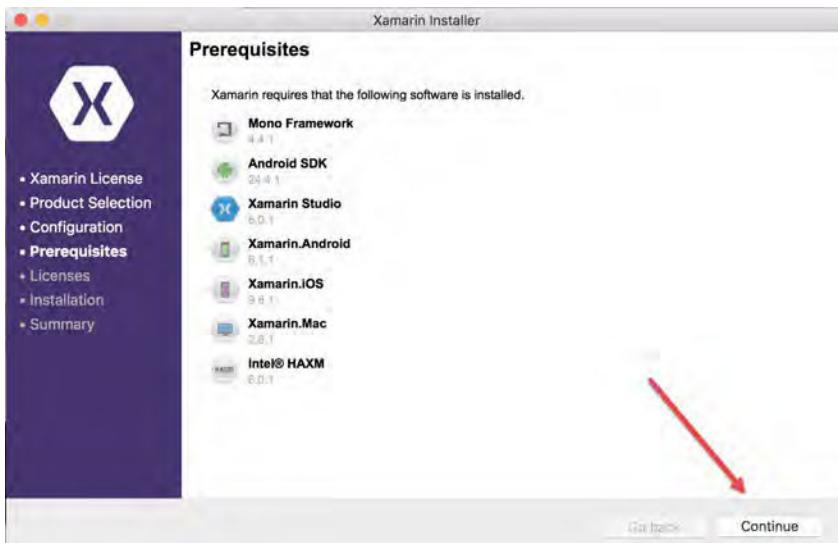


Figura 2.7: Apresentação dos pré-requisitos que serão instalados

O Mono Framework é a implementação do .NET para o Mac, necessário para a execução do Xamarin Studio e produção de suas aplicações para iOS e Android. Usando um Mac, não é possível gerar aplicações para a plataforma Windows, seja Windows Phone ou UWP. Para esta plataforma, apenas estando no Windows e utilizando o Visual Studio.

Confirme o início da instalação. O Android SDK é o componente responsável pela possibilidade de desenvolvimento de aplicações Android. O Xamarin Studio é o ambiente de desenvolvimento que utilizaremos no Mac, o IDE. Os demais componentes são as APIs Xamarin para a plataforma iOS e Android.

Se por algum momento a conexão com a internet sofrer queda,

pode ser que você receba uma mensagem de erro como a apresentada na figura a seguir. Recomendo que tente novamente a instalação em vez de seguir as orientações passo a passo sugeridas pela mensagem de erro.

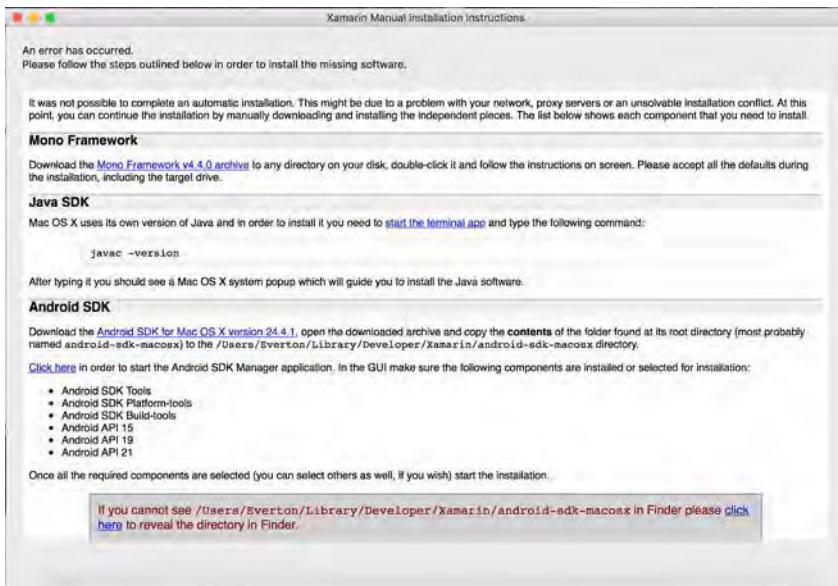


Figura 2.8: Erro relacionado à conexão com a internet durante a instalação

Se tudo estiver ocorrendo bem durante sua instalação, a janela apresentada na figura a seguir ficará sendo exibida até o término da instalação ocorra. Ao final, você receberá uma janela de sucesso da instalação e o navegador padrão será aberto na página do Xamarin.



Figura 2.9: Processo correto da instalação

Teste da instalação realizada

Localize o Xamarin Studio em sua máquina, conforme figura a seguir, e o execute.



Figura 2.10: Iniciando o Xamarin Studio

Com o Xamarin Studio aberto, é preciso verificar se existe alguma atualização estável para ele. Para isso, clique no menu

Xamarin Studio Community e, em seguida, na opção Check for Updates , conforme figura. Após a instalação das atualizações, se houver, feche a janela aberta.



Figura 2.11: Aplicando atualizações

Vamos agora criar uma solução multiplataforma (cross-platform), fazendo uso do Xamarin Studio. Esta solução não executará nada de mais, apenas uma mensagem de boas-vindas que é criada automaticamente pelo template escolhido. No próximo capítulo, começaremos a trabalhar na aplicação proposta no capítulo 1. Selecione o menu Arquivo e, nas opções que se exibem, escolha Solução , conforme figura a seguir.

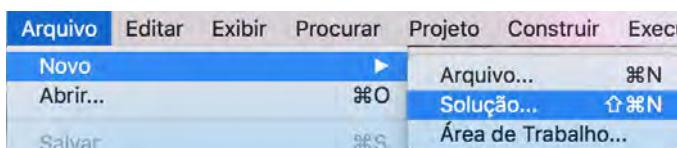


Figura 2.12: Criando uma solução

Na janela que se exibe, na categoria Multiplatform , escolha App e, nos templates exibidos à direita, escolha Forms App e

clique no botão **Next** , como é apresentado:

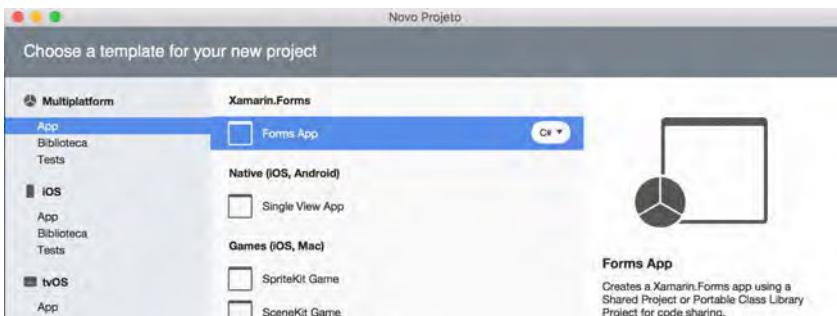


Figura 2.13: Selecionando o tipo de projeto a ser criado

Na janela seguinte, algumas informações são solicitadas, como nome da aplicação, identificador da organização, plataformas alvo, tipo de compartilhamento do código e se serão utilizados arquivos XAML para implementar a interface com o usuário. A princípio, siga os exemplos da figura a seguir e pressione o botão **Next** .

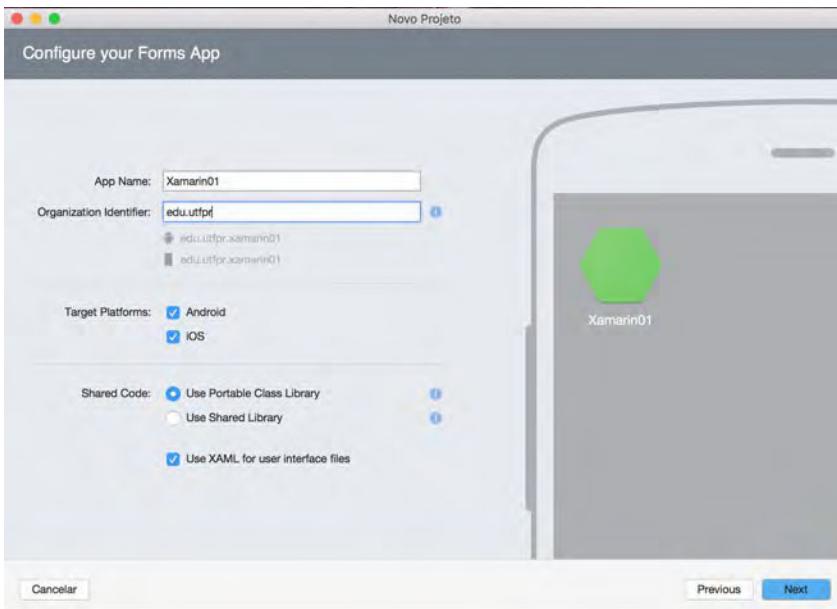


Figura 2.14: Configurando parâmetros para a criação do projeto

Quando criamos uma aplicação comum ou um projeto Library, o assembly que será o resultado do build do projeto é restrito a trabalhar na plataforma específica para que foi implementado. Com essa metodologia, o assembly gerado para o Windows Phone não pode ser reutilizado para uma aplicação `Xamarin.iOS` ou `Xamarin.Android`.

Quando se cria um projeto Portable Class Library , entretanto, é escolhida uma combinação de plataforma que se deseja que o código seja executado. As compatibilidades da escolha feita quando um projeto PCL é criado são traduzidas em um identificador de perfil, um `Profile` , que descreve quais plataformas a biblioteca suportará.

A figura a seguir apresenta um diagrama que traz a arquitetura de uma aplicação multiplataforma usando um PCL para compartilhar o código. É possível verificar ainda o uso de Injeção de Dependência para passar os recursos dependentes da plataforma.

A Wikipédia define bem a Injeção de Dependência: "*Injeção de dependência (Dependency Injection, em inglês) é um padrão de desenvolvimento de programas de computadores utilizado quando é necessário manter baixo o nível de acoplamento entre diferentes módulos de um sistema. Nesta solução, as dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infraestrutura de software (container) que é responsável por "injetar" em cada componente suas dependências declaradas. A Injeção de dependência se relaciona com o padrão Inversão de controle, mas não pode ser considerada um sinônimo deste*".

No caso do Xamarin Forms, as implementações dependentes da plataforma são passadas para o projeto PCL que é inserido em cada projeto de plataforma específica como referência ao assembly.

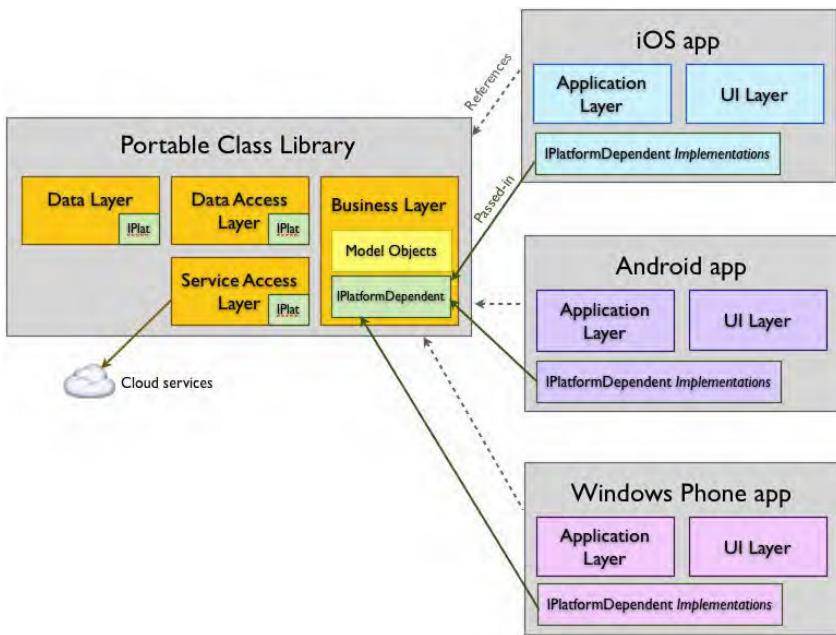


Figura 2.15: Arquitetura de um projeto PCL — https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/

A última página de configuração para a criação do projeto traz definições para o nome do projeto e solução, endereço físico onde será criado e configurações sobre Controle de Versão e Testes Automatizados. Para este momento, mantenha sua criação semelhante a apresentada na figura a seguir. Pressione o botão **Criar** para que o projeto seja criado.

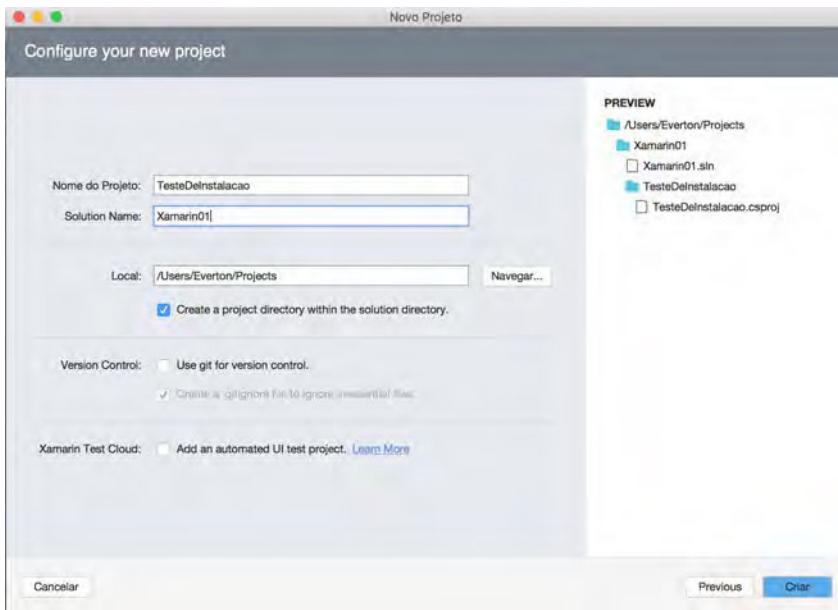


Figura 2.16: Configurando parâmetros finais para a criação do projeto

Após a criação do projeto, a janela Solução deverá estar semelhante à apresentada na figura a seguir. Note que foram criados três projetos: o PCL, um para o iOS e um para o Android. Tanto o projeto iOS como o Android fazem referência ao PCL.

É possível verificar que existem atualizações para os pacotes utilizados pelos projetos. Em minha máquina, fiz a atualização para o iOS e tudo ocorreu em perfeita ordem. Entretanto, quanto realizei a atualização para a plataforma Android, o projeto não funcionou mais.

O problema ficou na atualização do pacote `Xamarin.Forms`. Suas dependências foram atualizadas e não foram reconhecidas para ele. Tive de eliminar o projeto e criar de novo. Se este problema acontecer com você, já sabe o motivo e como solucionar. Quem sabe, no momento em que você estiver lendo o livro, esta situação já tenha sido resolvida pela equipe do Xamarin. Outra maneira de

corrigir o erro seria desinstalar a versão atualizada e baixar a anterior, via NuGet.

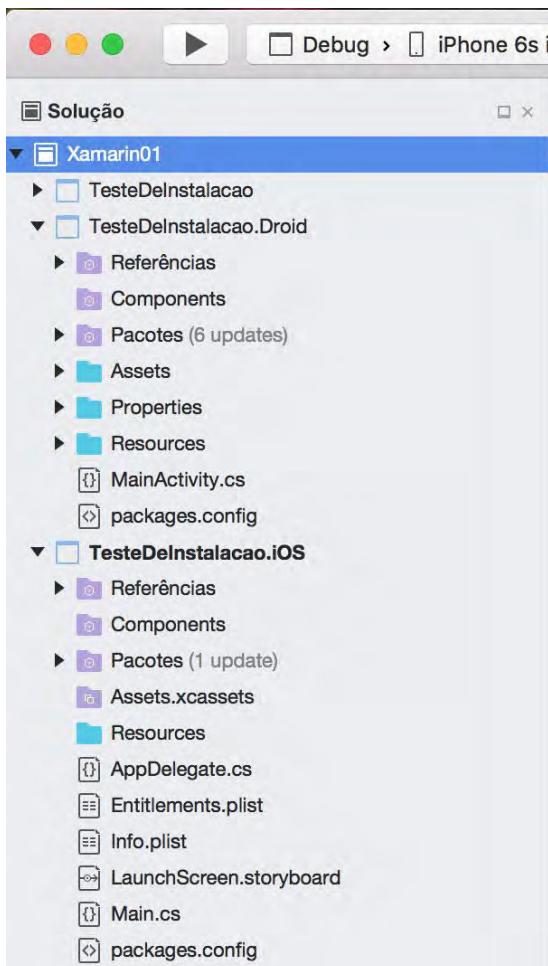


Figura 2.17: Janela da solução da aplicação

Na figura anterior, verifique que o projeto para o iOS está com seu nome em negrito. Isso quer dizer que ele é o projeto padrão para execução. Logo abaixo do menu do Xamarin Studio, é possível notar um botão de execução e, ao seu lado, está definido que o profile será o de depuração (Debug). Ao lado dele, está o nome do emulador

que será executado para deploy (distribuição) de sua aplicação. A figura a seguir apresenta em detalhe essa situação.



Figura 2.18: Escolhendo o emulador para a execução da aplicação

Na seleção do emulador, representada na figura anterior, eu escolhi o `iPhone 6 iOS 9.3`. Clique no botão de execução para que seu projeto seja construído. Se a construção for bem-sucedida, o emulador será carregado (isso pode demorar um pouco).

Após isso, a instalação de sua aplicação será realizada. Ela aparecerá para sua verificação e deve estar semelhante ao apresentado na figura a seguir. Quando terminei de escrever este livro, já estava disponível o emulador para o iPhone SE com o iOS 10, mas as telas já estavam capturadas. :-)



Figura 2.19: Aplicação funcionando em um emulador iOS

Agora precisamos testar a aplicação em uma emulador Android. Para isso, precisamos definir o projeto Android como o Projeto Inicial . Para isso, clique com o botão direito do mouse sobre o nome do projeto e clique na opção Definir como Projeto Inicial , tal qual é apresentado na figura:

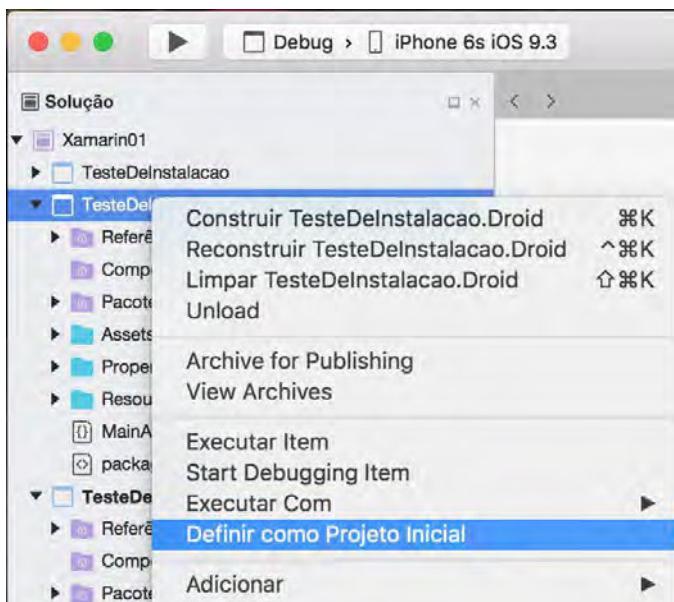


Figura 2.20: Definindo um projeto como o inicial

Verifique, na seleção de emuladores, que agora são apresentados alguns específicos para o Android. Para o teste, selecionei o `Android_Accelerated_x86`. Faça sua seleção e execute a aplicação. Lembre-se de que a inicialização do emulador pode demorar um pouco. Veja o resultado na figura a seguir.

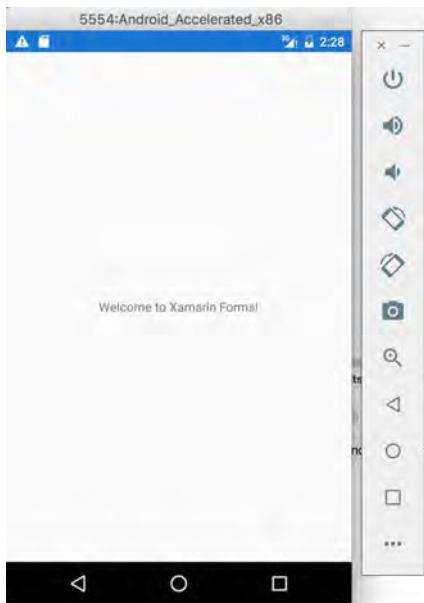


Figura 2.21: Aplicação funcionando em um emulador Android

2.2 INSTALAÇÃO DAS FERRAMENTAS EM UM PC COM WINDOWS 10 E TESTE DE UMA EXECUÇÃO BÁSICA

Semelhante ao que fizemos para a plataforma Mac OS, teremos como primeiro passo para a plataforma Windows a instalação da ferramenta de desenvolvimento da Microsoft, que é o Visual Studio 2015 Community Edition que, a exemplo do Xamarin Studio, é gratuita.

É importante ressaltar alguns problemas que obtive durante o processo de instalação e testes, para que você não os tenha. Farei isso conforme for necessário.

Instalação do Visual Studio 2015 Community Edition

Antes de começarmos, é preciso que você saiba que é necessário,

como mínimo, o Windows Professional — a edição Home não permite execução de emuladores Windows Phone. Estou usando o Windows 10 64 Profissional Edition. Nele, precisamos habilitar o Hyper-V.

Para isso, procure no Windows o Ativar ou Desativar Recursos do Windows e ative esta opção, tal qual é exibido na figura a seguir, sempre que precisar executar o emulador Windows Phone. Não ative este recurso agora.



Figura 2.22: Ativando o Hyper-V

Antes de instalar o Visual Studio, precisamos instalar a última versão do Java Development Kit, o JDK. E você pode fazer isso obtendo-o no endereço <http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html>.

Eu tinha em minha máquina a versão 1.7 e precisei baixar a 1.8, por erros apontados pelo Visual Studio na execução de projetos

Android. Desta maneira, faça isso em sua máquina.

Com o JDK devidamente instalado, para obter o Visual Studio, acesse <https://www.visualstudio.com/pt-br/downloads/download-visual-studio-vs.aspx>. Opte por baixar a versão Community Edition, como mostra a figura a seguir.

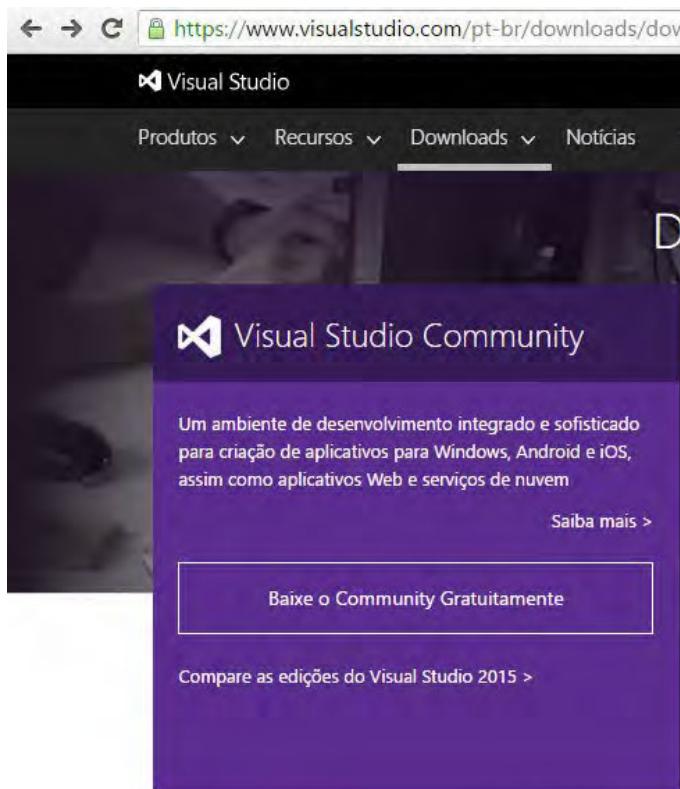


Figura 2.23: Página principal do site do Visual Studio

Após o download do instalador ser concluído, execute-o. Pode acontecer de ele solicitar que você reinicie sua máquina antes de dar prosseguimento. Na primeira janela do instalador, selecione a instalação customizada, indicada na figura a seguir. Feito isso, clique no botão **Next**.

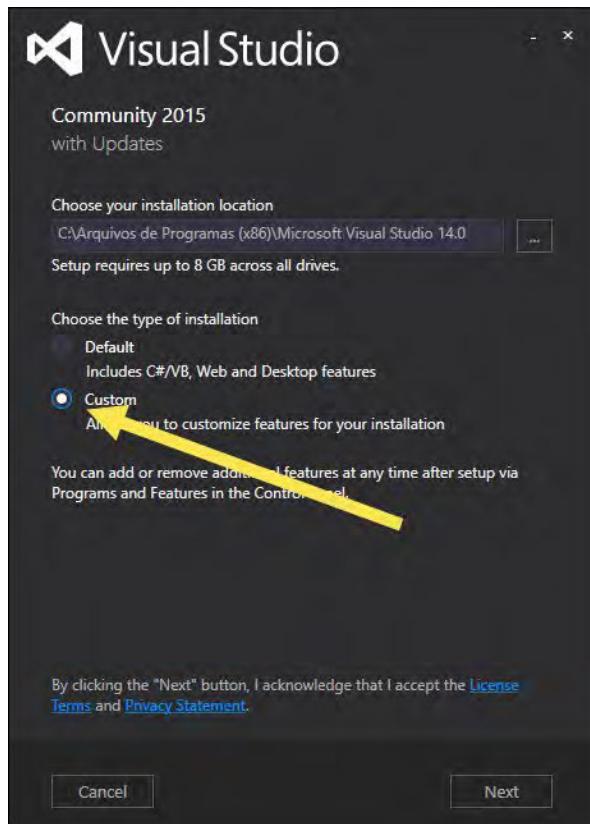


Figura 2.24: Janela inicial de instalação do Visual Studio

Na janela que é exibida na sequência da instalação, é preciso selecionar alguns componentes que não são instalados por padrão. Verifique a figura a seguir e configure sua instalação de acordo a ela. Ao final, pressione o botão **Next**.

Uma janela com as licenças de uso será exibida, pressione o botão **Install** para que a instalação inicie. É preciso uma boa paciência agora, pois é um processo que pode ser demorado. Em minha máquina, que é um i7, com 8GB de RAM, levou 4 horas. Minha conexão com a internet não era das melhores, e isso pode ser um diferencial durante sua instalação.

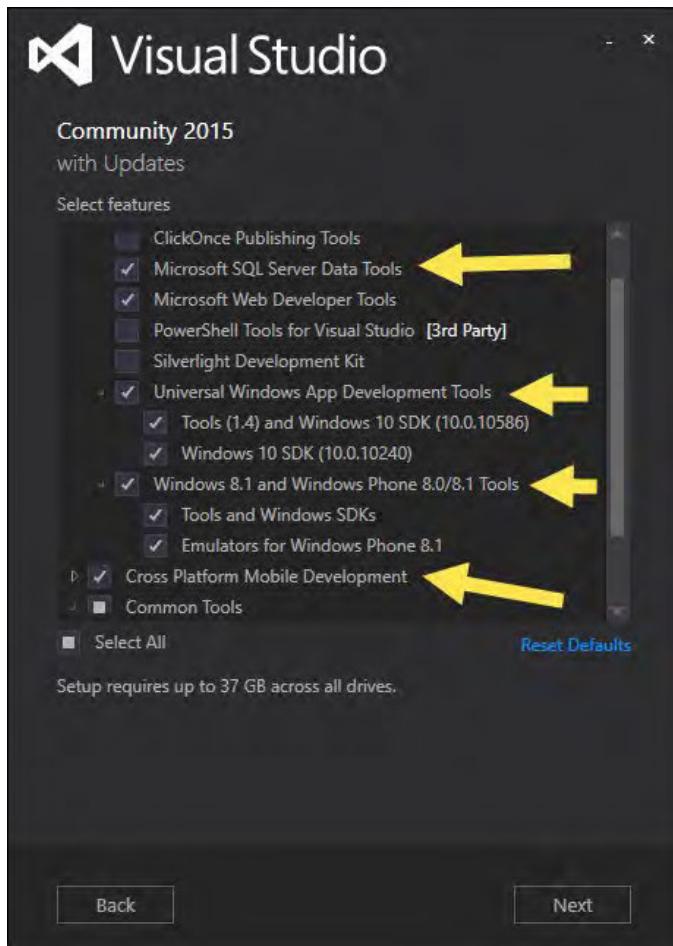


Figura 2.25: Configurando os componentes para a instalação do Visual Studio

Ao terminar minha instalação, foi apresentado um conjunto de erros em relação ao Android, como pode ser visto na figura a seguir. A primeira tentativa é executar novamente o setup e tentar realizar o download do que foi apresentado como erro. Se para você não houve problema algum, isso é ótimo, pule esta etapa.

Estes erros que ocorreram estão relacionados ao download e instalação dos componentes do Android. Preciso ressaltar que eles podem não ocorrer com você, mas comigo ocorreram duas vezes,

por isso julguei importante informar aqui.

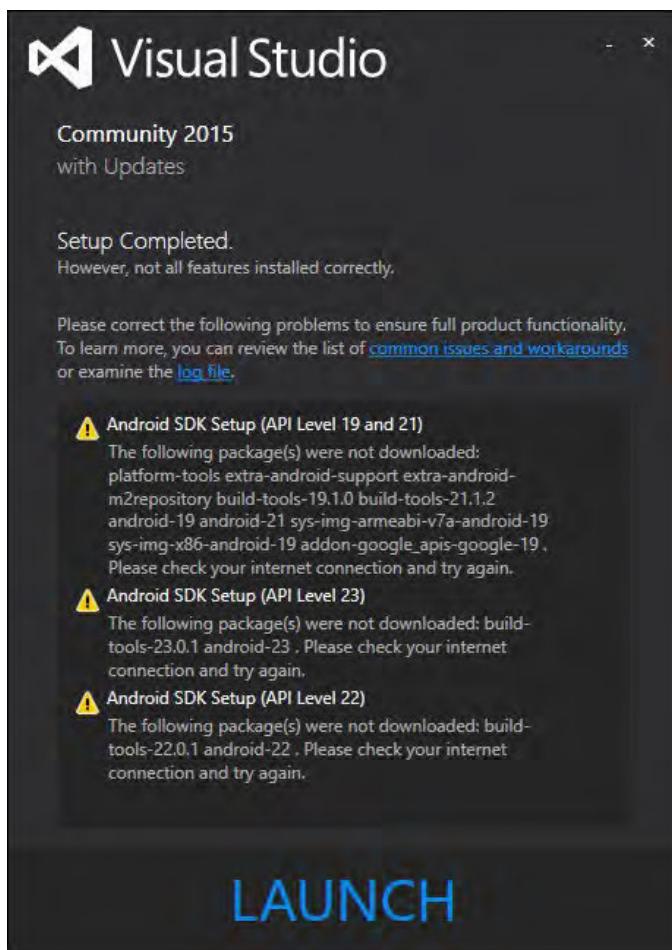


Figura 2.26: Erros durante a instalação do Visual Studio

Caso os erros comentados anteriormente tenham ocorrido com você, acesse novamente o setup e clique no botão **Modify**, pois modificaremos a instalação já realizada. Na janela seguinte, selecione os componentes que não foram instalados. Clique no botão **Next** e, na janela de licenças, clique no botão **Install**. O processo agora deverá levar menos tempo e logo teremos o Visual Studio instalado.

Caso o mesmo erro, ou um erro diferente ocorra, precisamos partir para outra estratégia. Instalar estas APIs diretamente pelo Android SDK , que já deve estar instalado em sua máquina. Em minha máquina, ele está em C:\Program Files (x86)\Android\android-sdk .

Execute o arquivo SDK Manager e terá a tela semelhante a apresentada na figura a seguir. Note na parte de baixo da janela que é iniciado um processo de atualização. Veja que isso deverá ser feito apenas se você não obteve sucesso na primeira tentativa de instalação.

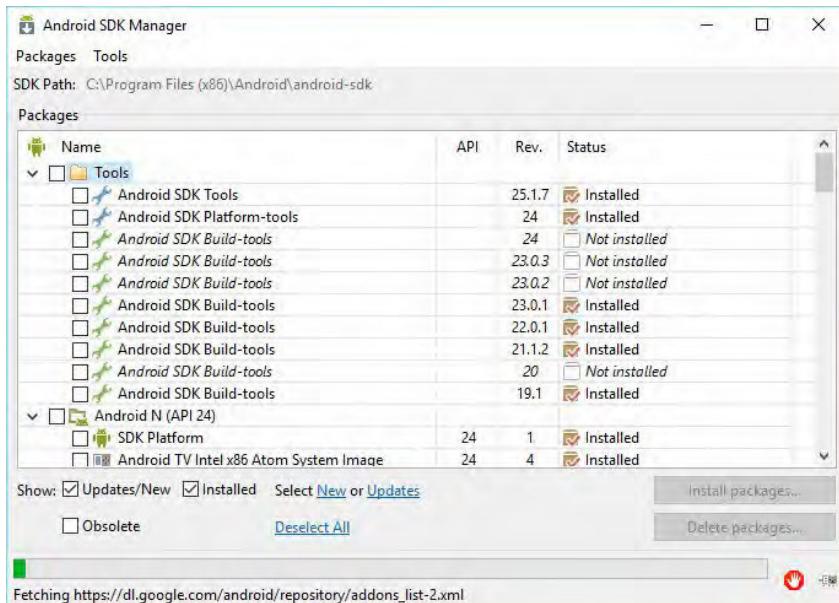


Figura 2.27: Android SDK Manager para instalação de SDKs

Com o processo de atualização concluído, marque as APIs que não foram instaladas pelo Visual Studio. Marque apenas a SDK Platform. Clique no botão Install e, em seguida, confirme o aceite às licenças de uso. Não feche ainda o Android SDK Manager.

Busque pelo Visual Studio em seu equipamento e o execute. A primeira execução pode demorar um pouco. Seja paciente. :-)

Com o Visual Studio aberto, é preciso agora realizar a atualização do Xamarin para o Visual Studio. Para isso, acesse o menu **Tools** -> **Options** e, no lado esquerdo, ao final das opções, clique e expanda as opções de **Xamarin**. Clique em **Other**. Veja a figura a seguir.

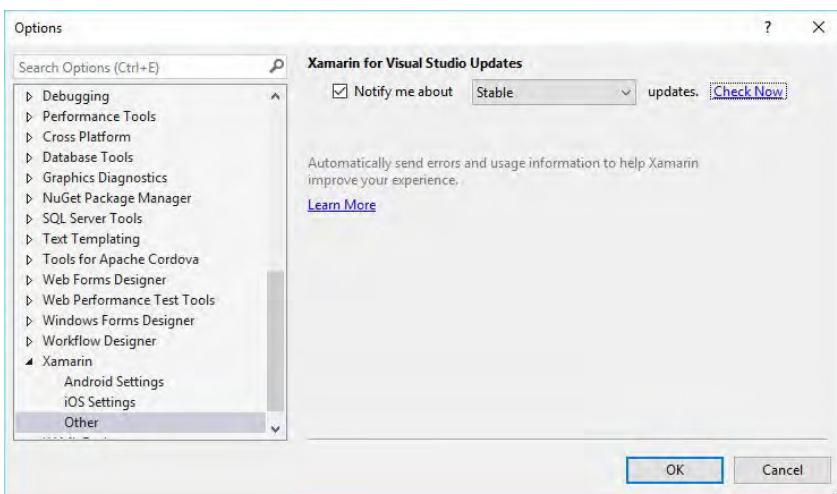


Figura 2.28: Verificando atualizações para o Xamarin

Com a janela da figura anterior aberta, clique em **Check now** e, caso existam atualizações, realize o download delas. Com o download concluído, surgirá um botão para iniciar a instalação, clique nele. Assim que ela começar, feche o Visual Studio e siga as orientações para a atualização.

Agora, vamos verificar se o Visual Studio tem configurado o JDK que foi feito o download e instalado anteriormente. Vá novamente em **Tools** -> **Options** e, na categoria **Xamarin**, em **Android Settings**, confirme o **Java Development Location**. Caso não seja a mesma, altere e reinicie o Visual Studio.

Antes de criarmos o projeto de teste, precisamos instalar o Intel x86 Emulator Accelerator (HAXM installer), e isso pode ser feito pelo Android SDK Manager. Vá até o final da janela de pacotes e procure por ele, tal qual pode ser visto na figura a seguir. Esta ferramenta possibilita a execução, de maneira mais rápida, dos emuladores Android.

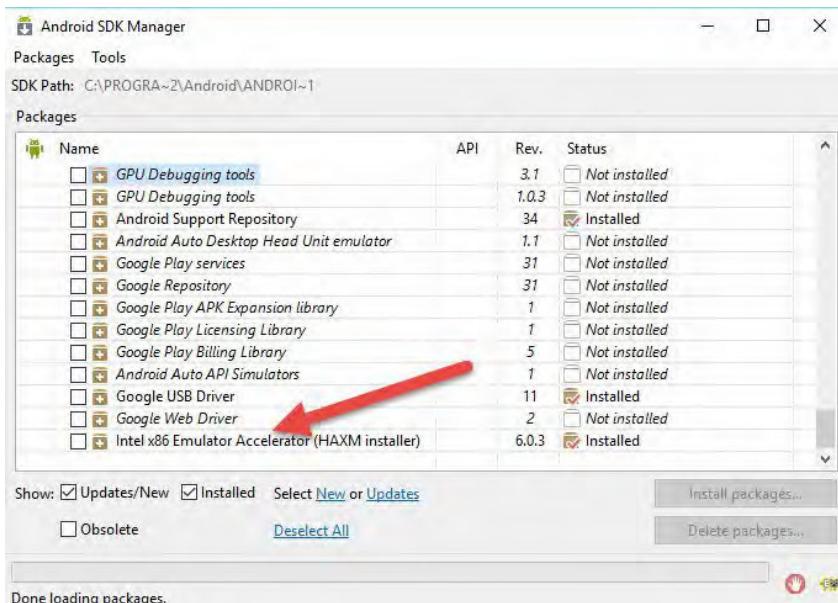


Figura 2.29: Baixando o instalador do HAXM

Marque este pacote e clique no botão para instalação. Ocorre que a instalação não é executada, o pacote é apenas baixado. É preciso localizar este pacote em seu Windows Explorer e executá-lo. Em minha máquina, ele estava em `C:\Program Files (x86)\Android\android-sdk\extras\intel\Hardware_Accelerated_Execution_Manager`.

Ao acessar esta pasta, execute o arquivo `intelhaxm-android` e siga as orientações para concluir a instalação. Após a instalação, pode fechar o Android SDK Manager.

Se você tem o Hyper-V já habilitado em sua máquina, não conseguirá instalar este pacote. Lembra de que anteriormente mostrei como habilitar, mas pedi que não o fizesse? Bem, se você o habilitou ou já o tem habilitado, precisa desabilitá-lo e depois disso instalar o HAXM. É um pouco demorado e chato este processo de habilitar ou desabilitar o Hyper-V. Tenha paciência.

Criação de um projeto cross-platform no Visual Studio

É importante saber que, da mesma maneira que não é possível criar uma aplicação Windows Phone em um Mac, também não é possível em uma plataforma Windows criar uma aplicação iOS. Na realidade, é possível criar, mas realizar o build e executar esta aplicação é que não é possível.

Entretanto, por meio de acesso remoto a uma máquina Mac, este trabalho se torna viável. Vamos começar esta sessão configurando isso. Em seu Mac, acesse as Preferências do sistema e, dentro delas, Compartilhamento , como pode ser visto na figura a seguir.

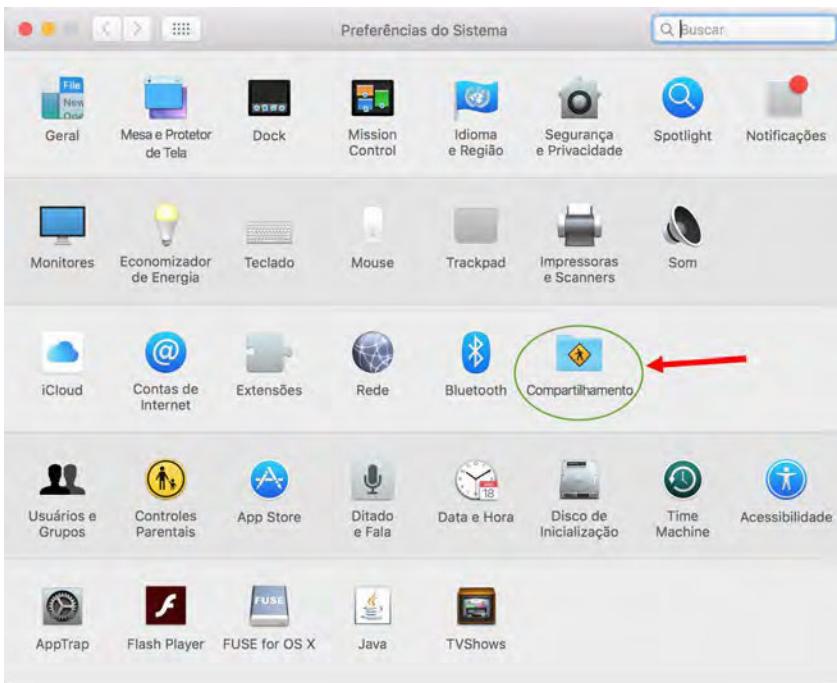


Figura 2.30: Acessando regras para compartilhamento em um Mac

Na janela que se abrir, ative o acesso remoto, tal qual é exibido na figura a seguir. Mantenha também a configuração para os usuários. Confirme essas alterações e vamos para a criação da aplicação.

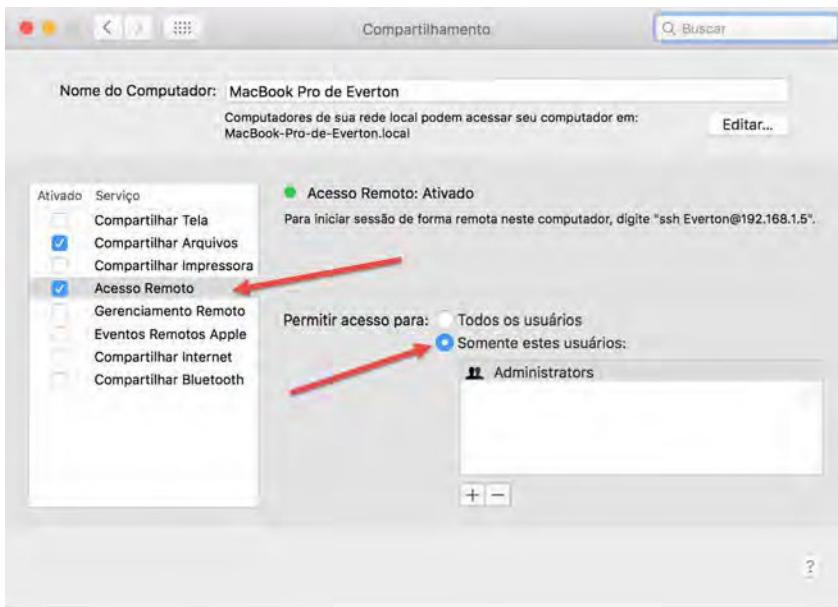


Figura 2.31: Configurando o acesso remoto

Abra novamente o Visual Studio, pois já podemos criar nossa aplicação Mobile Cross-platform de teste, tal qual fizemos na plataforma Mac OS. Para isso, clique no menu `File -> New -> Project`, conforme a figura a seguir.

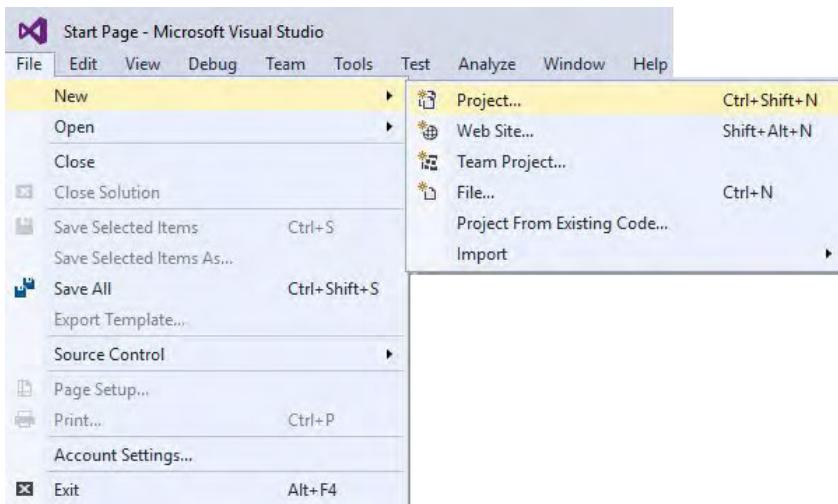


Figura 2.32: Criando um novo projeto pelo Visual Studio

Na janela que se abre, escolha a linguagem Visual C#, a categoria Windows e, dentro dela, a Cross-Platform. Nos templates que são exibidos ao lado direito, selecione o Blank Xaml App (Xamarin.Forms Portable) , tal qual mostra a figura a seguir. Nomeie-o *TesteDeInstalacao* e atribua *Xamarin02* para o nome da Solução . Clique no botão Ok, e aguarde.

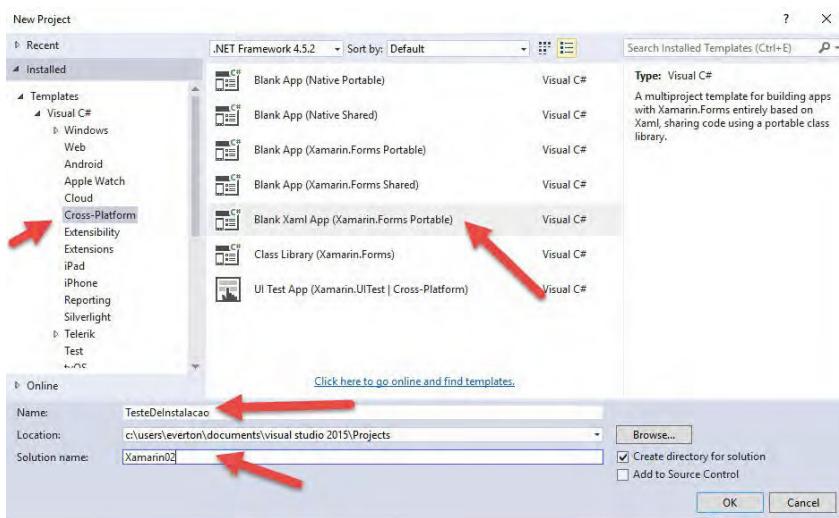


Figura 2.33: Configurando a criação do novo projeto

Durante o processo de criação do projeto, algumas janelas podem aparecer. Uma delas é a definição de plataforma para o desenvolvimento de aplicações Universal Windows Project , ou UWP , como mostra a figura a seguir. Eu mantive a sugestão padrão, apresentada na figura.

Em outra máquina de testes, a configuração padrão deu erro na criação do projeto, pois trazia como Target Version uma versão recente da plataforma. Tive de mudar para uma versão anterior e funcionou.

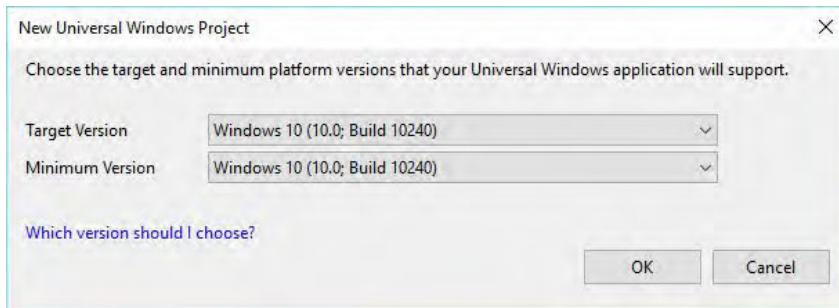


Figura 2.34: Configurando as plataformas para aplicações UWP

Na sequência, como será criado um projeto para a plataforma iOS, são apresentadas informações como configurar a conexão do Visual Studio com seu Mac, mas siga as que escrevi anteriormente. Clique no botão **Ok**, como destacado na figura a seguir.

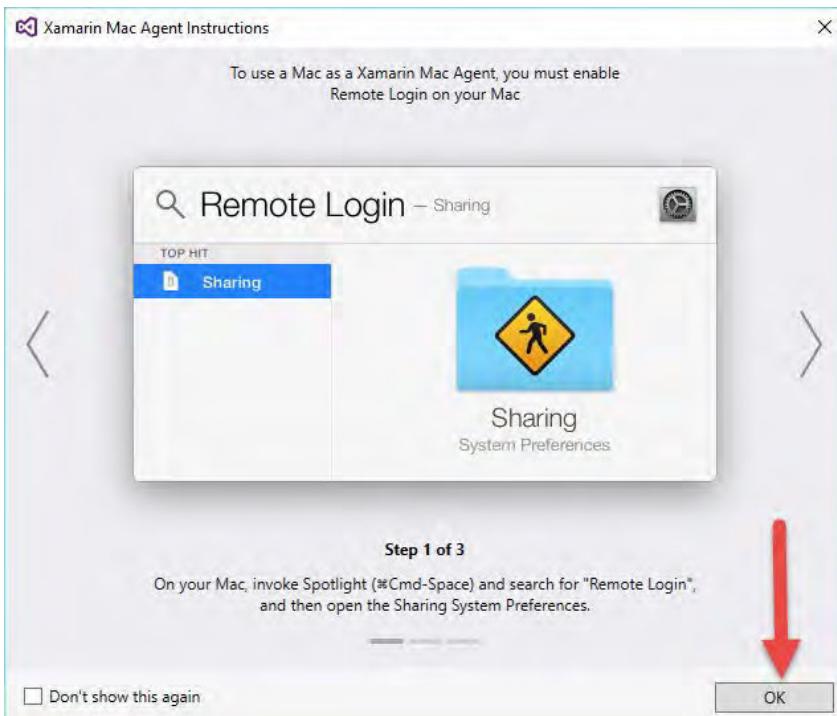


Figura 2.35: Orientações para conectar o Visual Studio com um Mac na rede

Muito bem. Se seu Mac estiver ligado e na rede, sua janela deverá estar semelhante a apresentada pela figura da sequência, a qual exibe o Mac já identificado.

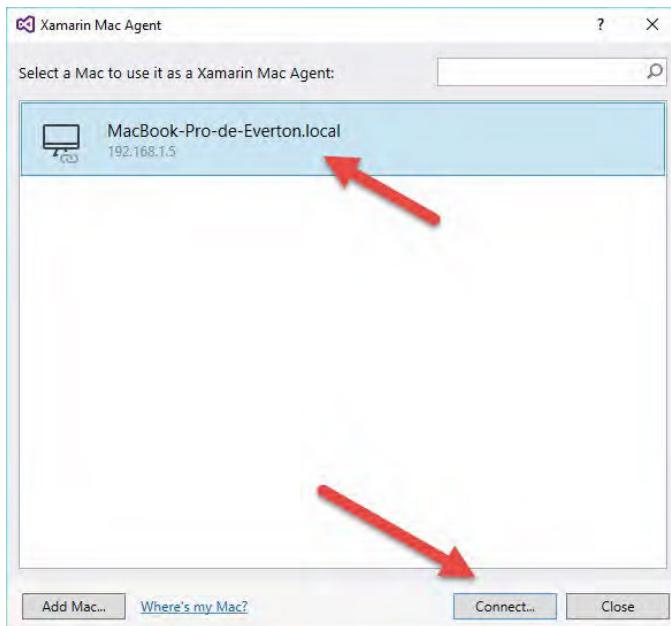


Figura 2.36: Configurando as plataformas para aplicações UWP

Clique no nome de seu Mac e, em seguida, no botão `Connect`. Informe seu usuário e senha, e os confirme. Com isso, a criação do projeto estará concluída e você deverá possuir uma `Solution Explorer` com todos os projetos criados, tal qual apresenta a figura a seguir.

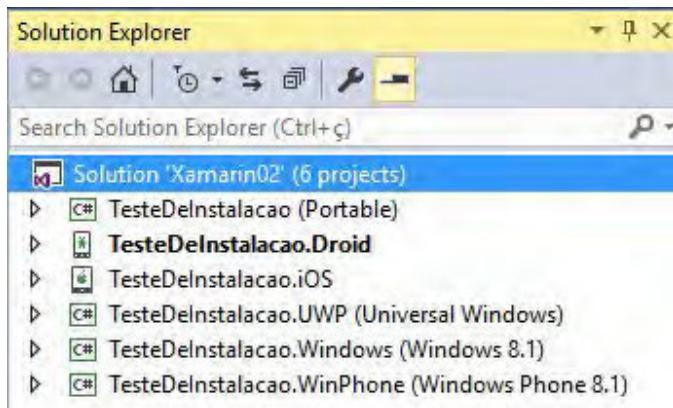


Figura 2.37: Solution Explorer com todos os projetos criados

Teste dos projetos criados

Como pode ser verificado em destaque na figura anterior, o projeto padrão para a aplicação, tal qual foi no Xamarin Studio, é o para Android. Dessa maneira, vamos executá-lo.

Veja, na figura a seguir, a barra de tarefas já com o emulador para o Android habilitado. Ao lado do nome do emulador, existe uma série de botões, cujas funcionalidades serão explanadas conforme eles forem sendo usados.



Figura 2.38: Barra de tarefas com destaque para emuladores

Infelizmente, em meu caso, não foi possível fazer o emulador criado como padrão ser executado e aceitar o deploy da aplicação. Precisei criar um novo emulador. Para isso, clique, da esquerda para a direita, no segundo botão da figura anterior, o Open Android Emulator Manager (AVD). Quando ele abrir, deve estar semelhante ao apresentado na figura a seguir. A exceção é que deve ter apenas um emulador criado.

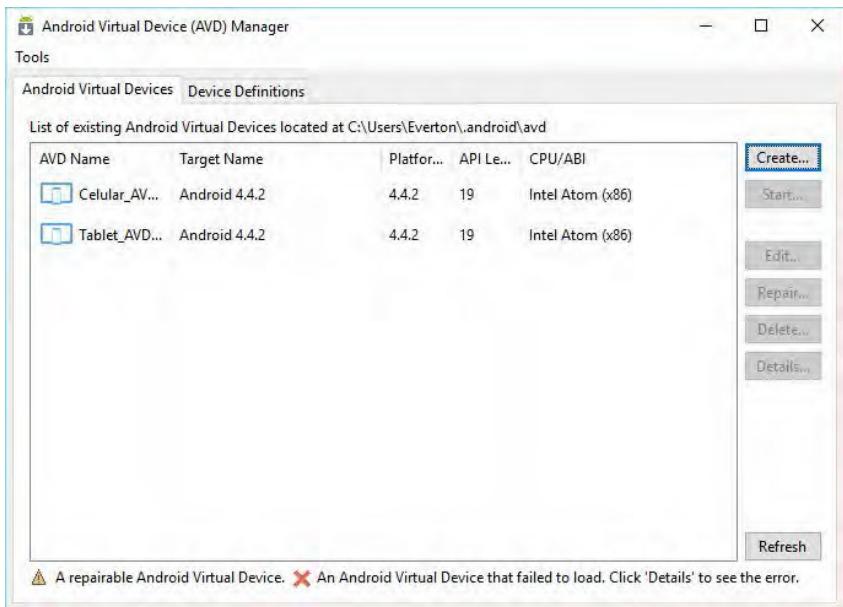


Figura 2.39: Android Virtual Device (AVD) Manager

Na janela da figura anterior, clique na guia **Device Definitions** e localize a configuração que seja semelhante a apresentada na figura a seguir. Ao localizar, clique nela e, depois, no botão **Create AVD**.

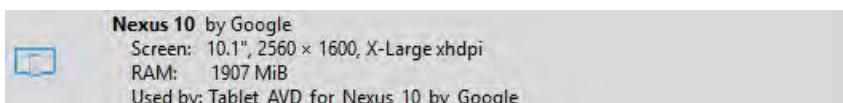


Figura 2.40: Selecionando o dispositivo que terá o emulador criado

Adapte os campos destacados na figura a seguir e clique no botão **OK** para que o emulador seja criado. Veja que na CPU está configurado o **Intel Atom (x86)**. Esta configuração permite o uso do HAXM (que instalamos anteriormente) como acelerador. A outra opção disponível é a **ARM (armeabi-v7a)**, que é muito, mas muito lenta mesmo.

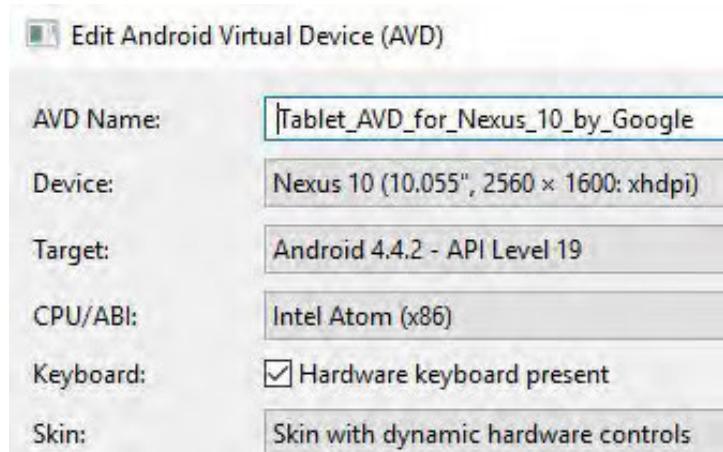


Figura 2.41: Criando o emulador para Tablet

Vamos agora criar um emulador para celular. O processo é semelhante: identifique o dispositivo com as características apontadas na figura a seguir e altere os campos, semelhante ao que foi feito anteriormente, mudando apenas o nome para celular.

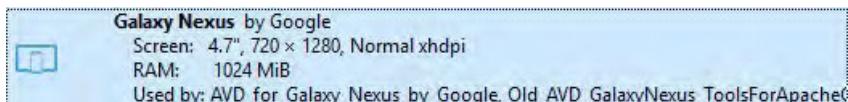


Figura 2.42: Selecionando o dispositivo que terá o emulador criado

Bem, vamos agora voltar ao Visual Studio e executar a aplicação. Se ao executar a aplicação (o que pode demorar, pois o Emulador precisa ser carregado), for informado que houve erro de deploy, não se assuste, aconteceu comigo também.

O primeiro erro que apareceu foi o `The name 'InitializeComponent' does not exist in the current context`. Agora, a solução para ele, acredite, foi fazer alguma alteração nos arquivos XAML e gravá-los novamente. Infelizmente, este erro ocorre com certa frequência durante o desenvolvimento. Veja na figura a seguir o emulador do celular em execução com sua

aplicação.



Figura 2.43: Emulador de celular em execução

Como você pode ter verificado, o processo de distribuição e execução é lento, devido à carga do emulador. Desta maneira, minha recomendação é que você deixe-o ativo entre as execuções e testes de sua aplicação. Com isso, já temos nossa aplicação testada para o Android.

Agora, na sequência, vamos testar a aplicação para Windows

Phone 8.1. Antes de tudo, é preciso ativar o Hyper-V, faça isso. Tenha paciência, pois você precisará reiniciar sua máquina.

Com o Hyper-V ativado, configure o projeto WinPhone para ser o de inicialização, clicando com o botão direito sobre ele e escolhendo Set as Startup Project. Ao fazer isso, você poderá notar na barra de tarefas que o emulador para a execução da aplicação já mudou. Clique nele para executar. Quando tudo estiver OK, sua janela deverá estar semelhante a apresentada pela figura a seguir.



Figura 2.44: Emulador para aplicação Windows Phone

Antes de o emulador ser inicializado, pode ser que apareça uma janela de direitos, pedindo para adicionar o Hyper-V ao grupo de

Administradores. Confirme clicando no botão `Retry`. Pode ser que o emulador demore muito para estar disponível. Pode ser inclusive que alguns erros apareçam no Visual Studio, mas tudo isso será devido à lentidão da inicialização do emulador.

Se o erro aparecer, espere o emulador estar todo inicializado (aparecerá a janela básica de aplicativos dele), e então tente novamente executar a aplicação. Da mesma maneira que foi orientado para o emulador do Android, fica aqui o registro de manter o emulador ativo durante as execuções, para evitar este tempo todo que ele leva para ser inicializado.

SOBRE OS EMULADORES WINDOWS PHONE E ANDROID

Aqui cabe uma observação sobre os emuladores para Windows Phone e Android. Se você mantiver o Hyper-V instalado, não será possível executar aplicações no emulador Android, com o Intel Atom (x86). Pois é, isso é chato. Não é possível ter o Hyper-V e o HAXM habilitado ao mesmo tempo em sua máquina.

Você pode optar pela outra opção disponível como CPU/ABI para o emulador Android, mas saiba que ela é lenta. Este problema é discutido muito bem em:
<http://docplayer.com.br/1949580-Hyper-v-e-intel-haxm-ativando-a-virtualizacao.html>. É importante que você leia.

No momento da escrita deste livro, a plataforma atual da Microsoft para dispositivos móveis é a 10, que faz uso da Universal Platform Windows, e que a usaremos quando formos testar os aplicativos em dispositivos físicos.

Em meu Visual Studio, quando selecionei o projeto UWP como de startup, não aparecia emuladores. Precisei fazer os downloads deles, acessando o site <https://developer.microsoft.com/en-us/windows/downloads/sdk-archive>. Eu fiz o download dos emuladores e do SDK. Isso foi um processo demorado também.

Após a inicialização, precisamos executar a aplicação neste emulador, que é um pouco diferente do que vimos até agora. É preciso, antes de executar, realizar o Deploy . Selecione um emulador e então, clicando com o botão direito do mouse sobre o nome do projeto UWP, clique em Deploy .

Depois do deploy ter sido concluído, execute a aplicação. A figura a seguir apresenta o emulador executando a aplicação em minha máquina. Registro aqui que, ao realizar o deploy pela primeira vez, o Visual Studio não obteve sucesso, exibindo uma mensagem de erro relacionada ao uso de GPU com RemoteFlex . Precisei acessar o Gerenciador do Hyper-V e, em Configurações do Hyper-V..., na categoria GPUs Físicas , desabilitei o item Usar esta GPU com RemoteFX .



Figura 2.45: Emulador para aplicação UWP

Bem, nos resta agora testar a aplicação iOS. Defina este projeto como de startup. Pode ser que neste momento seja solicitado que localize novamente a máquina Mac. Siga os mesmos passos que foram dados anteriormente.

Para executar a aplicação, na barra de tarefas, em `Device`, escolha um dos que são disponibilizados e execute a aplicação. Eu escolhi `iPhone 6 Plus` e, no meu Mac, apareceu a janela apresentada na figura a seguir. É importante que, a cada troca de projeto padrão, seja realizado um `Clean` e novo `Build` na aplicação. Isso evita alguns erros de execução.



Figura 2.46: Emulador para aplicação iOS

Emuladores Android para o Visual Studio e o Xamarin Android Player

A Microsoft oferece emuladores para Android, que podem ser executados (lentamente) com o Hyper-V instalado. Se sua instalação do Visual Studio ocorreu bem, eles estão disponíveis no menu Tools do Visual Studio. Caso eles não tenham sido instalados, você pode obtê-los no link <https://visualstudiogallery.msdn.microsoft.com/74e85e34-46ea-4188-9fef-55efb815534f>.

Já o Xamarin Android Player, faz uso do Virtual Box e pode ser obtido em <https://developer.xamarin.com/releases/android/android-player/>. Não farei uso destas duas opções neste livro, fica a seu critério realizar a instalação e uso.

Se fizer uso dos Emuladores para o Visual Studio, recomendo a leitura de https://developer.xamarin.com/guides/android/deployment,_testing

[,_and_metrics/debug-on-emulator/visual-studio-android-emulator/](#).

2.3 CONCLUSÃO

Ufa. Chegamos ao final deste capítulo. Sei que ele foi mais um tutorial de como instalar as ferramentas e executar a aplicação de testes. De fora, parece algo simples, mas alguns problemas podem ocorrer e isso pode decepcionar você. Por isso optei por ter este capítulo no livro.

Não entrei em detalhes quanto às estruturas dos projetos. Farei isso com calma, nos próximos capítulos. Já no próximo, criaremos a primeira etapa de nosso projeto, aquele que foi detalhado no capítulo 1. Relaxe um pouco agora, tome um ar e vamos para o terceiro capítulo.

CAPÍTULO 3

O INÍCIO DA APLICAÇÃO

Como já temos o Xamarin instalado e testado, tanto em uma plataforma Mac como em uma Windows, resta-nos agora começarmos nossa aplicação. A primeira etapa, escolhida por mim, será a criação de um menu de opções para o usuário e a listagem de garçons registrados no sistema.

Optei por não fazer neste momento nenhum diagrama de classes ou qualquer artefato relacionado a uma análise de requisitos. Vamos puramente implementar a aplicação, seguindo um fluxo de necessidades e recursos que selecionei.

Nossa aplicação possuirá diversos módulos. O primeiro em que trabalharemos será o responsável por inserir os dados que serão consumidos pelos demais.

Poderíamos pensar em uma aplicação Desktop (ou uma Web) para esta necessidade, mas quero fazer o máximo que for possível usando dispositivos móveis. Desta maneira, prepare-se para sua primeira aplicação desenvolvida para dispositivos móveis.

A aplicação deste capítulo será desenvolvida no Xamarin Studio, em um Mac, inicialmente testável em um dispositivo Apple e em um Android. É importante que você saiba que teremos também um pouco de teoria e conceitos durante o desenvolvimento.

3.1 CRIAÇÃO DA APLICAÇÃO NO XAMARIN

STUDIO

Com o Xamarin Studio aberto, clique no menu Arquivo -> Novo -> Solução . Na janela que se abre, nas categorias do lado esquerdo, escolha Multiplatform -> App e, ao lado direito, Forms App . Confirme a linguagem C# , e depois clique no botão Next no lado inferior direito da janela.

Na janela que se abre, chamada de Configure your Forms App , nomeie a aplicação como CasaDoCodigoFoods . Coloque um domínio como identificador e mantenha as plataformas de destino iOS e Android selecionadas. Quanto ao código compartilhado, selecione o PCL (use Portable Class Library) e desmarque o Use XAML for user interface files . Feito tudo isso, clique no botão Next .

Na janela seguinte, chamada Configure your new project , em Nome do Projeto , informe Modulo1 e clique no botão Criar . Bem, optei por não trazer nenhuma figura para esta atividade, pois todas as figuras que poderiam representar neste parágrafo já foram inseridas e discutidas no capítulo anterior. O que mudei, para esta aplicação, foi não optar por páginas XAML para a interface.

Vamos começar codificando nossa interface com usuário fazendo uso de C#, depois voltamos para o XAML. Vamos agora entender um pouco sobre as estruturas dos projetos criados.

Estrutura básica de um projeto iOS

Na figura a seguir, veja a estrutura que foi criada para o projeto iOS pelo Xamarin Studio . Após a figura, apresento uma descrição de cada item criado e, ao final, uma figura que representa o ciclo de execução da aplicação.

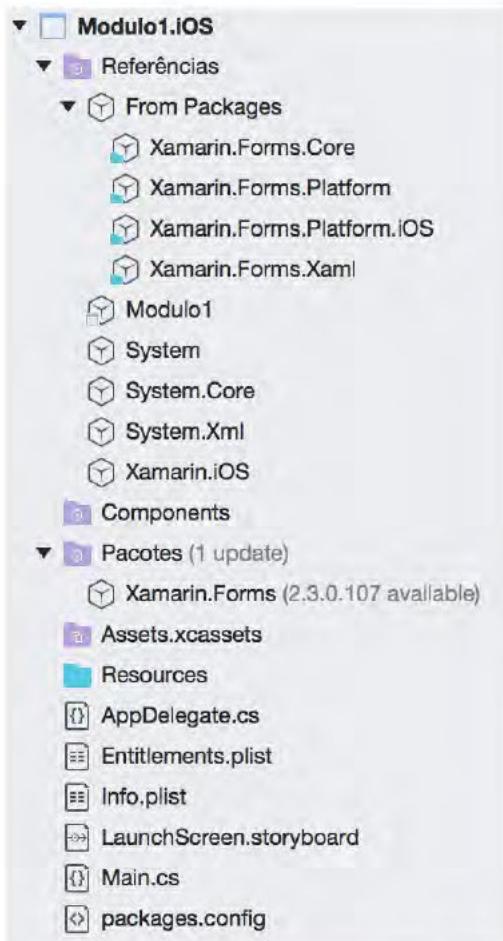


Figura 3.1: Estrutura criada para o projeto iOS

- **References** — Esta pasta contém os assemblies requeridos para construir e executar a aplicação. Veja na figura anterior que, além dos assemblies referentes ao Xamarin, existem os específicos do .NET e Modulo1, que é nosso projeto PCL.
- **Components** — Esta pasta mantém recursos prontos e disponíveis na **Xamarin Components Store**, um local público para código Xamarin. É similar ao **NuGet**.

`Gallery`, para quem é familiar ao Visual Studio. Se você ainda não viu este termo, o NuGet é um repositório de componentes. Você pode dar uma visualizada em <https://www.nuget.org/>. O Visual Studio oferece mecanismos para busca, instalação e atualização de componentes.

- **Pacotes** — Referem-se aos assemblies disponíveis e baixados para sua aplicação e específicos para a plataforma, que podem ser obtidos pela NuGet Gallery.
- **Resources** — Aqui são armazenados ícones, imagens de abertura e outros tipos de mídia.
- **Main.cs** — Este arquivo contém o ponto principal de entrada da aplicação, é aqui que tudo começa. Para iniciar, é passado o `AppDelegate` no nome da aplicação principal, como pode ser verificado na figura a seguir. Veja, no destaque, a instanciação da classe `App`, que está no projeto PCL, que a princípio é o projeto que possuirá nossa aplicação.

```
1 using UIKit;
2
3 namespace Module1.iOS
4 {
5     public class Application
6     {
7         // This is the main entry point of the application.
8         static void Main(string[] args)
9         {
10             // If you want to use a different Application Delegate class from "AppDelegate"
11             // you can specify it here.
12             UIApplication.Main(args, null, "AppDelegate");
13         }
14     }
15 }
```

Figura 3.2: Código da classe Application do arquivo Main.cs

- **AppDelegate.cs** — Este arquivo contém a classe principal da aplicação e é responsável por criar a janela, construir a interface com o usuário e escutar os eventos vindos do sistema operacional. Veja o código deste arquivo na figura a seguir.

```

1 using Foundation;
2 using UIKit;
3
4 namespace Modulo1.iOS
5 {
6     [Register("AppDelegate")]
7     public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
8     {
9         public override bool FinishedLaunching(UIApplication app, NSDictionary options)
10        {
11            global::Xamarin.Forms.Forms.Init();
12
13            LoadApplication(new App());
14
15            return base.FinishedLaunching(app, options);
16        }
17    }
18 }

```

Figura 3.3: Código da classe AppDelegate do arquivo AppDelegate.cs

- **LaunchScreen.storyboard** — O storyboard contém o desenho da interface com o usuário da aplicação. Arquivos storyboard são abertos em um editor gráfico chamado iOS Designer, que faz parte do Xamarin Forms. Se você quiser fazer uma aplicação iOS pura, é neste tipo de arquivo que você insere e configura os controles visuais. Não faz parte do escopo deste livro abordar este tema.
- **Info.plist** — É onde configuramos as propriedades para a aplicação, como seu nome, ícones, imagens de inicialização, dentre outros.
- **Entitlements.plist** — Este arquivo permite a especificação de recursos (também chamados de Tecnologias App Store), como o iCloud, por exemplo.

Estes três últimos arquivos serão descritos de melhor maneira quando forem requeridos. A figura a seguir apresenta o ciclo de execução de uma aplicação iOS.

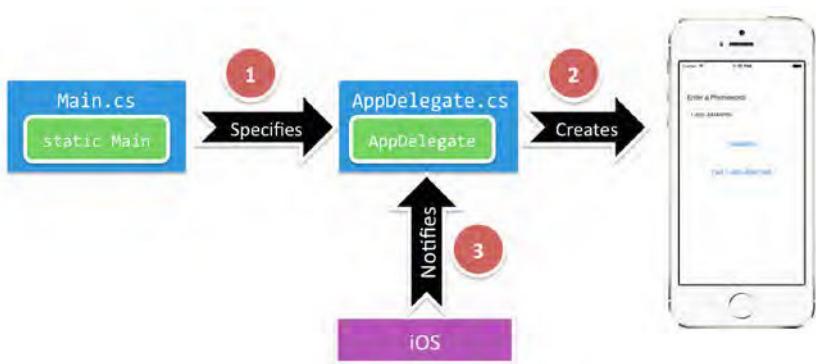


Figura 3.4: Ciclo de execução de uma aplicação iOS
[\(https://developer.xamarin.com/guides/ios/getting_started/hello,_iOS/hello,_iOS_deepdive/\)](https://developer.xamarin.com/guides/ios/getting_started/hello,_iOS/hello,_iOS_deepdive/)

Estrutura básica de um projeto Android

A figura a seguir apresenta a estrutura criada para o projeto **Droid**, pelo **Xamarin Studio**.



Figura 3.5: Estrutura criada para o projeto Android

- **References, Components, Pacotes e Resources**
— Semelhantes às pastas de mesmo nome no projeto iOS .
- **Assets** — Contém os arquivos que a aplicação precisa para executar, incluindo fontes, arquivos de dados locais e arquivos de textos. Maiores explicações sobre Assets serão dadas quando eles forem usados.
- **Properties** — Contém o arquivo `AndroidManifest.xml` que descreve todos os requisitos para a aplicação `Xamarin.Android` ,

incluindo nome, versão e permissões. Esta pasta também hospeda o assembly .NET AssemblyInfo.cs , que contém metadados cujo preenchimento é recomendado.

Na aplicação Android, a pasta Resources possui algumas subpastas, com suas descrições expostas a seguir:

- `drawable` , `drawable-hdpi` , `drawable-xhdpi` e `drawable-xxhdpi` — Estes diretórios armazenam recursos como imagens e bitmaps. No template criado, ele traz o arquivo ícone da aplicação, chamado `Icon.png` . Cada pasta representa uma resolução da imagem. Uma discussão sobre estas pastas e as resoluções das imagens que elas hospedam pode ser visualizada em <http://stackoverflow.com/questions/14381965/image-size-drawable-hdpi-ldpi-mdpi-xhdpi>.
- `layout` — Esta pasta contém arquivos de desenho do Android que definem a interface de usuário para cada janela ou Activity. Se você for implementar uma aplicação puramente Android, são nestes arquivos que desenharia a interface com o usuário.
- `values` — Esta pasta hospeda arquivos que armazenam valores simples, como strings, inteiros e cores.
- `Resource.designer.cs` — Também conhecido como a classe `Resource` . Este arquivo é um `partial class` que armazena os IDs únicos atribuídos para cada recurso. Ele é automaticamente criado pelas ferramentas Xamarin.Android e é regerado sempre que necessário. Não mexa neste arquivo, pois qualquer mudança que fizer será sobrescrita.

O arquivo `MainActivity.cs`, na raiz do projeto, implementa a classe `MainActivity`, que pode ser verificada na figura a seguir. Observe os atributos decoradores para a classe, e veja no destaque a invocação da classe `App`, que está no projeto PCL.

```
1 using Android.App;
2 using Android.Content.PM;
3 using Android.OS;
4
5 namespace Modulo1.Droid
6 {
7     [Activity(Label = "Modulo1.Droid", Icon = "@drawable/icon", Theme = "@style/MyTheme", MainLauncher = true,
8             ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
9     public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
10    {
11        protected override void OnCreate(Bundle bundle)
12        {
13            TabLayoutResource = Resource.Layout.Tabbar;
14            ToolbarResource = Resource.Layout.Toolbar;
15
16            base.OnCreate(bundle);
17
18            global::Xamarin.Forms.Forms.Init(this, bundle);
19
20            LoadApplication(new App()); ←
21        }
22    }
23 }
```

Figura 3.6: Classe `MainActivity`

O projeto PCL

De maneira inicial, posso dizer que é no projeto PCL que sua aplicação será implementada. Podemos nele definir a interface com o usuário, controladores para esta interface, o modelo de negócio e a camada de persistência.

Você pode ver, na figura a seguir, que o conteúdo do arquivo `Modulo1.cs` é a implementação da classe `App`, instanciada tanto pelo projeto iOS como pelo projeto Droid. E, ainda no código da figura a seguir, está a implementação do construtor da classe `App`, que implementa a interface que o usuário verá. Logo mudaremos isso.

```

1 using Xamarin.Forms;
2
3 namespace Modulo1
4 {
5     public class App : Application
6     {
7         public App()
8         {
9             // The root page of your application
10            var content = new ContentPage
11            {
12                Title = "Modulo1",
13                Content = new StackLayout
14                {
15                    VerticalOptions = LayoutOptions.Center,
16                    Children = {
17                        new Label {
18                            HorizontalTextAlignment = TextAlignment.Center,
19                            Text = "Welcome to Xamarin Forms!"
20                        }
21                    }
22                };
23            };
24        }
25        MainPage = new NavigationPage(content);
26    }
}

```

Figura 3.7: Classe App implementada no arquivo Modulo1 do projeto PCL

Anatomia de uma aplicação Xamarin Forms

Precisamos ter uma noção sobre a anatomia de uma aplicação Xamarin Forms, antes de implementarmos nossa primeira interface com o usuário. Desta maneira, é importante saber que todos os objetos que aparecem na tela de um dispositivo móvel são chamados de elementos visuais, e são divididos em três categorias: `page` , `layout` e `view` .

Estes elementos não são abstratos. A API do Xamarin Forms define classes chamadas `VisualElement` , `Page` , `Layout` e `View` . Estas classes e suas descendentes formam a espinha dorsal da interface com o usuário do Xamarin Forms. Um objeto `VisualElement` é qualquer coisa que ocupa espaço na tela.

Uma aplicação Xamarin Forms consiste, normalmente, de uma ou mais páginas. Uma página geralmente ocupa toda uma janela. Um tipo de página muito utilizada é a página de conteúdo, que é representada pela classe `ContentPage` .

Os componentes visuais são organizados, em cada página, em uma hierarquia pai-filho. O filho de uma página de conteúdo é, geralmente, um layout, que organiza os componentes visuais na tela. Alguns layouts possuem um único filho, mas muitos layouts possuem múltiplos filhos, que são organizados dentro dele.

Estes filhos podem ser outros layouts ou views. O termo `view` para o Xamarin Forms refere-se a tipos de objetos de apresentação ou interação. São normalmente chamados de controles ou widgets em outros ambientes de programação.

3.2 CRIAÇÃO DA PÁGINA DO MENU DE OPÇÕES

Quando vamos criar uma aplicação, quer seja ela grande ou não, precisamos pensar em camadas. E quando pensamos em camadas, podemos pensar em alguns princípios, que nos levam a separar nossa aplicação em projetos.

Entretanto, neste momento do livro, adotarei a estratégia de separação por pastas, o que implicará em namespaces diferentes. Mais à frente, separemos isso em alguns projetos. Uma breve discussão sobre estas estratégias pode ser vista em: <http://stackoverflow.com/questions/36577930/xamarin-architecture-projects-or-namespaces>.

SEPARAÇÃO EM CAMADAS

Quando desenvolvemos uma aplicação, independente de seu tamanho e complexidade, é interessante a separarmos por responsabilidades. Estas responsabilidades podem ser vistas como camadas e estas camadas, quando trazidas da abstração para o físico, se convertem em pastas ou projetos.

Quando optamos por separar as camadas em projetos, possibilitamos a coesão e acoplamento físico. Por exemplo, poderíamos ter um projeto para a camada de modelo, e este projeto ser referenciado em um projeto desktop, web ou mobile.

Outra estratégia, e que vou utilizar, é ter estas camadas em pastas. É claro que a camada de modelo que estou implementando nesta estratégia ficará atrelada a este projeto, mas para o que proponho no momento, isso não será problema.

Bem, vamos lá. No nome do projeto (Modulo1), clique com o botão direito do mouse e escolha Adicionar -> Nova Pasta , nomeie-a Pages . Para o Xamarin Forms, os formulários são vistos como páginas.

Dentro desta pasta, clicando com o botão direito novamente, clique agora em Adicionar -> Novo Arquivo e, na janela que se exibe, do lado esquerdo, escolha a categoria Forms e, ao lado direito, Forms ContentPage . Nomeie este arquivo de MenuPage e clique no botão Novo , tal qual mostra a figura a seguir.

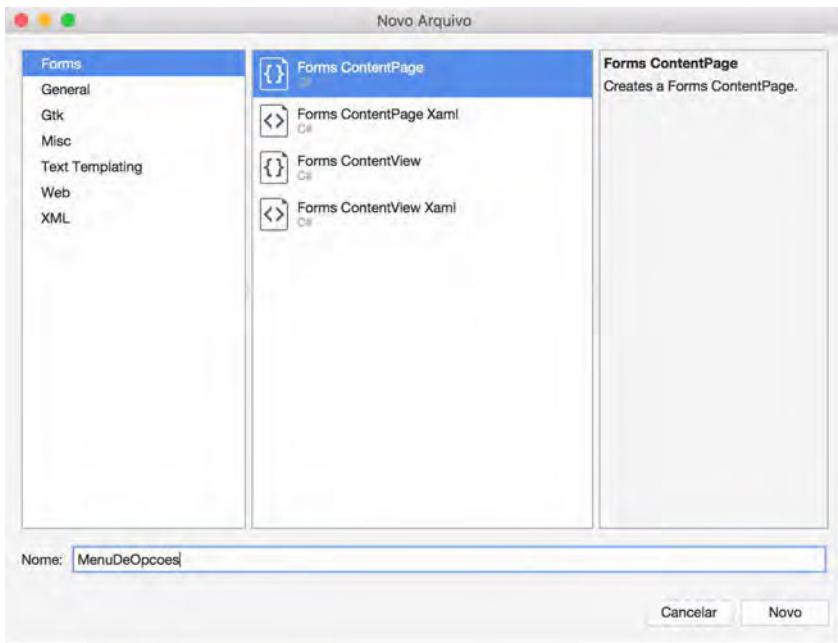


Figura 3.8: Janela de criação de uma nova página

Após a criação do novo arquivo, deixe sua implementação tal qual mostra o código adiante. Note que a classe estende `ContentPage` e, no construtor, configuramos o título da página e seu conteúdo por meio da propriedade `Content`. Uma `ContentPage` é uma página que exibe apenas um controle visual (`view`).

A princípio, o conteúdo desta página refere-se a um `StackLayout`, que a exemplo da propriedade `Content` pode ser visto como um container. Entretanto, o `StackLayout`, além de abrigar controles, é responsável pela forma como estes controles serão dispostos dentro dele; neste caso, serão empilhados, um após o outro.

Mais sobre tipos de páginas e tipos de layout podem ser consultados respectivamente em:

- <https://developer.xamarin.com/guides/xamarin-forms/controls/pages/>
- <https://developer.xamarin.com/guides/xamarin-forms/user-interface/layouts/>

Estas páginas trazem a documentação oficial do Xamarin Forms no que se refere aos tipos de pages e layouts.

```
namespace Modulo1.Pages
{
    public class MenuPage : ContentPage
    {
        public MenuPage()
        {
            Title = "Menu de opções";
            Content = new StackLayout
            {
                VerticalOptions = LayoutOptions.Center,
                Children = {
                    new Button() {
                        Text = "Garçons",
                        Image = "icone_garcons.png",
                        Command = new Command(() => Navigation.PushAsync(new GarconsPage())))
                }
            };
        }
    }
}
```

Continuando sobre o código anterior, verifique que, ao instanciar `StackLayout`, configuramos a disposição dos controles adicionados a ele, por meio da propriedade `VerticalOptions`. Esta recebe `LayoutOptions.Center`, que faz com que os controles

adicionados a ele sejam centralizados verticalmente.

A estratégia que adotei para inserir controles no `StackLayout` foi a de inserção deles já no construtor, atribuindo-os à propriedade `Children`. Preferi esta adoção pelo fato de, ao instanciar o objeto, já terei nele tudo que será preciso. Você poderia instanciar apenas o objeto, sem as propriedades iniciais, e atribuir valores a elas após a instanciação.

O objeto adicionado refere-se a um botão (`Button`), que recebe um título, uma imagem e o que deve ser executado quando o botão for pressionado. Você pode estar acostumado a usar eventos para isso em vez de uma propriedade. O uso de eventos também é comum, mas quis apenas introduzir o `Command`, que pode ser mais bem compreendido com uma leitura em <https://blog.xamarin.com/simplifying-events-with-commanding/>.

O uso de Commands está mais relacionado ao desenvolvimento com Data Bindings e com o uso do padrão Model-View-View-Model, que veremos mais para a frente.

É possível ver, ainda no código anterior, que a página `GarconsPage`, que será exibida quando o usuário clicar no botão, está sendo enviada como argumento para o método `PushAsync()`, de `Navigation`. Embora a página `GarconsPage` seja uma `TabbedPage`, como você pode ver no código seguinte, ela terá o comportamento de uma `NavigationPage`, que exibe no início da página o nome da página que a instanciou.

Uma `TabbedPage` consiste de uma lista de guias/tabs e uma área onde o conteúdo de cada área pode ser exibido. Se você quiser maiores detalhes sobre este tipo de página, pode visitar <https://developer.xamarin.com/guides/xamarin-forms/user-interface/navigation/tabbed-page/>. Em relação à imagem `icone-garcons.png`, que é um recurso, falaremos sobre ela após o código

a seguir.

```
using System;  
  
using Xamarin.Forms;  
  
namespace Modulo1.Pages.Garcons  
{  
    public class GarconsPage : TabbedPage  
    {  
        public GarconsPage()  
        {  
            Children.Add(new GarconsListPage()  
            {  
                Title = "Listagem",  
                Icon = "icone_list.png"  
            });  
            Children.Add(new GarconsNewPage()  
            {  
                Title = "Inserir Novo",  
                Icon = "icone_new.png"  
            });  
        }  
    }  
}
```

No código anterior, no construtor da classe, é possível verificar a adição de dois objetos ao conteúdo da página, por meio do método `Add()`, da propriedade `Children`. Cada objeto refere-se a uma página, que você precisa criar e implementar.

Estas páginas receberão, por meio da instanciação, um título (`Title`), exibido em cada `Tab`, e uma imagem, que aparece também na `Tab` para o iOS. O código da classe `GarconsListPage` pode ser verificado na listagem a seguir.

Verifique que, no construtor, a propriedade `Content` recebe o retorno do método `GetGarcons()`, criado e implementado na própria classe. Este método retorna um `ListView`, que veremos com maiores detalhes em usos futuros. Para este momento, saiba que ele gerará uma listagem com os valores atribuídos à propriedade `ItemsSource`.

```

using System;
using System.Collections;
using System.Collections.ObjectModel;
using Xamarin.Forms;

namespace Modulo1
{
    public class GarconsListPage : ContentPage
    {
        public GarconsListPage()
        {
            Content = GetGarcons();
        }

        private ListView GetGarcons()
        {
            var garcons = new ListView();
            garcons.ItemsSource = new string[] {
                "Brauzio", "Asdrugio", "Entencius", "Gesfredio", "Cartucious",
                "Gesfrundio", "Adoliterio", "Kentencio", "Castrogildo", "Gesifrelio"
            };
            return garcons;
        }
    }
}

```

Inserção de imagens no projeto

Sempre que preciso usar alguma imagem, procuro alguma que esteja disponível no link <http://findicons.com/>. Caso você queira usar uma imagem sua, fique à vontade em usá-la. Para isso, grave a imagem, com o nome utilizado no código anterior, no projeto iOS em Resources ; e no Droid, dentro de drawable , que está em Resources .

A figura que baixei tem resolução 32x32. Após isso, precisamos incluir esta imagem no projeto, pois a copiamos externamente para a pasta dele, ela não faz parte do projeto no ambiente Xamarin Studio. Para isso, no projeto iOS, clique com o botão direito do mouse no nome da pasta Resources e, depois, em Adicionar ->

Add files from Folder . . .

Na janela que se abre, localize a pasta `Resources`, e em seguida clique em `Open`. Marque o arquivo e clique no botão `OK`, tal qual mostra a figura a seguir.

Maiores informações sobre como trabalhar com imagens no Xamarin Forms podem ser obtidas em <https://developer.xamarin.com/guides/xamarin-forms/working-with/images/>. Use apenas letras, números e o sublinhado como nome dos arquivos de imagem.

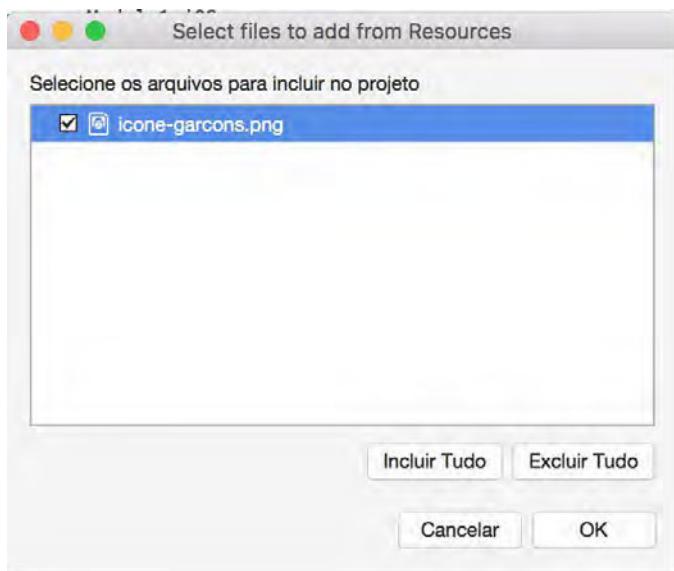


Figura 3.9: Adicionando um arquivo à pasta resources do projeto iOS

Para finalizar a implementação e executar o que fizemos até agora, vamos verificar como fica a classe `App`, que está no arquivo `Modulo1.cs`. O código dela é apresentado na sequência. Verifique

que substituí todo o conteúdo apenas pela atribuição do objeto `MenuPage` à propriedade `MainPage`.

```
using Xamarin.Forms;

namespace Modulo1
{
    public class App : Application
    {
        public App()
        {
            MainPage = new MenuPage();
        }
    // O código restante não foi alterado
```

Agora, vamos ver em execução o que implementamos até aqui. Selecionei o emulador iPhone 4s iOS 9.3 e depois o Android_Accelerated_x86, e obtive a figura seguinte. Tomarei como padrão mostrar a interface iOS do lado esquerdo e a Android do lado direito.

Lembre-se de que, em caso de erro na execução da aplicação Android, pode ser lentidão de inicialização do emulador. Se ocorrer, tente parar e executar novamente o projeto.

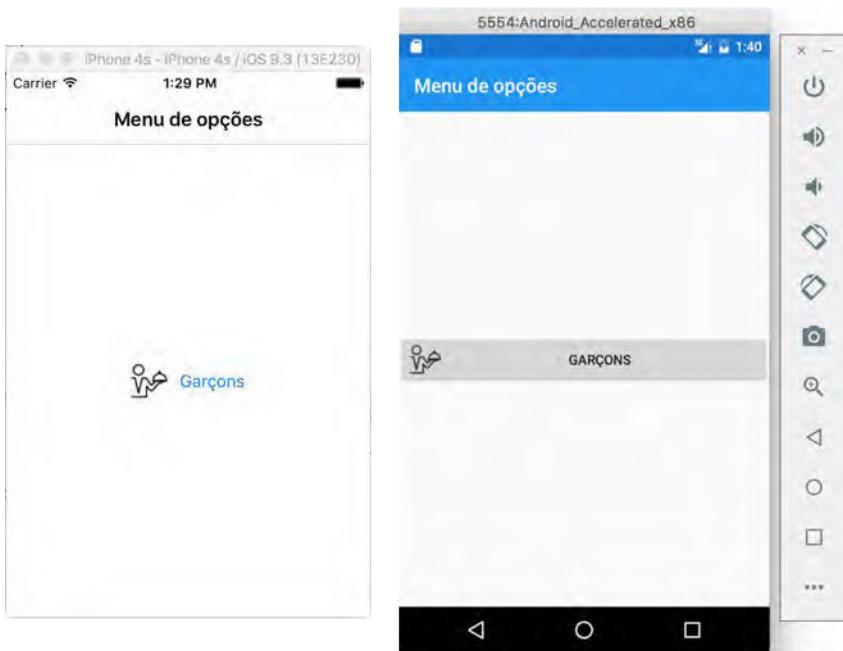


Figura 3.10: Emulador iOS e Android com a aplicação inicializada

Na tela representada pela figura anterior, para acessarmos a área restrita aos dados de Garçons, clique no botão que a representa. Você visualizará a janela apresentada na figura a seguir.

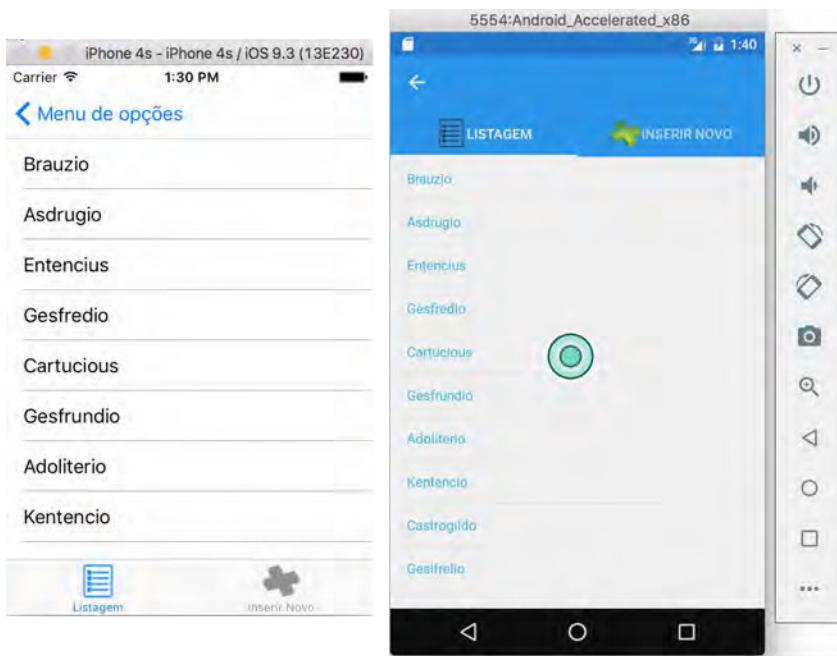


Figura 3.11: Emulador iOS e Android com a janela restrita a operações com dados de garçons

Agora, a tela de seu emulador deve estar exibindo a lista de nomes dos garçons, tal como a figura anterior. Você pode clicar na Tab Inserir Novo e ver o texto padrão definido quando criou o arquivo GarconsNewPage.cs .

3.3 CONCLUSÃO

Chegamos ao final do capítulo, no qual foi possível realmente criar nossa primeira aplicação mobile com o Xamarin Forms. Criamos uma página onde as opções de trabalho serão exibidas. Para a opção implementada neste capítulo, criamos outra página, com uma listagem de dados estáticos, armazenados em um array de Strings.

Para esta aplicação, usamos três tipos de páginas:

`ContentPage` , `NavigationPage` e `TabbedPage` . Toda a implementação foi feita diretamente por meio de código C#, e utilizamos o Xamarin Studio em um Mac.

No próximo capítulo, criaremos esta funcionalidade para outro modelo de negócio, desta vez em uma plataforma Windows, por meio do Visual Studio. Faremos uso de XAML para a criação da interface com o usuário, teremos agora uma aplicação Windows Phone e UWP, e armazenaremos dados em uma coleção. Será um capítulo muito interessante. Até lá.

CAPÍTULO 4

IMPLEMENTAÇÃO DE UM FORMULÁRIO COM XAML

No capítulo anterior, implementamos nossa primeira aplicação, já visando ao nosso sistema. A princípio, ela apenas se referia a uma listagem de garçons, fazendo uso de uma array de strings.

Neste capítulo, trabalharemos com entregadores, desenvolveremos uma nova página para listagem e outra para cadastro. Com o que apresentarei para entregadores, você será capaz de fazer a mesma implementação para garçons, melhorando o código do projeto do capítulo anterior.

O registro de dados se dará em uma coleção, que só terá os dados inseridos enquanto a aplicação estiver ativa. O correto, e veremos mais para a frente, é fazer uso de uma base de dados para armazenamento destes dados.

Como desenvolvemos a aplicação do capítulo anterior no Mac/Xamarin Studio, começaremos agora a trabalhar com o Visual Studio, na plataforma Windows. A interface com o usuário, no capítulo anterior, foi toda codificada em C#. Neste capítulo, faremos uso da XAML, uma linguagem de apresentação do Xamarin Forms, semelhante ao XAML do WPF (para quem conhece).

Antes de começarmos a prática, julguei importante trazer aqui uma informação quanto ao teste da aplicação no emulador para Windows Phone e Windows 10 Mobile. Sempre que eu tentava

executar a aplicação, o emulador travava e ela não podia ser testada. Lembrei de que havia por várias vezes ativado e desativado o Hyper-V, então fui verificar os dispositivos de rede, pois na ativação ele cria um específico. Havia muitos.

Eu desativei o Hyper-V, removi todos adaptadores de rede no Gerenciamento de Dispositivos do Windows, reiniciei a máquina, ativei o Hyper-V, reiniciei a máquina mais uma vez, e tudo deu certo. É trabalhoso, mas para testarmos nas três plataformas, precisamos deste *workaround*. Bem, vamos ao trabalho.

4.1 CRIAÇÃO DA APLICAÇÃO NO VISUAL STUDIO

Com o Visual Studio aberto, clique no menu File -> New -> Project . Na janela que se abre, nas categorias do lado esquerdo, escolha Templates -> Visual C# -> Cross-Platform e, ao lado direito, Blank XAML App (Xamarin.Forms.Portable) .

Nomeie sua solução (eu dei o nome de *CasaDoCodigoFoods*) e seu projeto (atribuí *Modulo1*). Tal qual no capítulo anterior, optei por não trazer nenhuma figura para esta atividade, pois todas que poderiam representar este parágrafo já foram inseridas e discutidas no capítulo 2.

O modelo

Nossa primeira implementação na solução é a criação de uma pasta chamada *Modelo* , e nela criaremos a classe *Entregador* . Para isso, clique com o botão direito do mouse sobre o nome do projeto PCL, e então em Add -> New Folder . Depois, clicando com o botão direito sobre a pasta *Modelo* , selecione Add -> Class... . Sua classe deverá ter o código semelhante ao

apresentado na sequência.

```
namespace Modulo1.Modelo
{
    public class Entregador
    {
        public long Id { get; set; }
        public string Nome { get; set; }
        public string Telefone { get; set; }
    }
}
```

A camada de acesso aos dados

Agora, como defini que trabalharemos com uma coleção de dados, precisamos pensar em como implementar a inserção e recuperação dos dados desta coleção. Como dito no capítulo anterior, trabalharemos em camadas, separando-as em pastas em vez de projetos, nesse momento. Sendo assim, o acesso a dados será realizado por uma nova camada (já temos a de modelos), a de acesso aos dados.

O .NET especifica esta camada como DAL (*Data Access Layer*). Para criarmos nossa camada, crie no projeto PCL uma pasta chamada `Dal`, e nela crie a classe `EntregadorDAL`, com o código apresentado a seguir. Note, neste código, que existem dois campos: `Entregadores` e `EntregadorInstance`.

O primeiro, é a coleção, do tipo `ObservableCollection`, que permite que os controles que fazem uso da coleção sejam atualizados quando a coleção for atualizada. Em nosso caso, será o `ListView`, que vimos no capítulo anterior. Já o segundo campo refere-se a um objeto da própria classe, que será usado para termos uma instância única dele, em toda a aplicação. Para isso, faremos uso do padrão de projeto `Singleton`.

O método construtor da classe instancia os objetos que serão inseridos na coleção. É claro que, quando estivermos fazendo uso de

base de dados, isso será retirado. A classe termina com três métodos: um que retorna a instância do DAL, um para retornar todos os elementos da coleção e outro para inserir um novo elemento na coleção.

```
using Modulo1.Modelo;
using System.Collections.ObjectModel;

namespace Modulo1.Dal
{
    public class EntregadorDAL
    {
        private ObservableCollection<Entregador> Entregadores =
            new ObservableCollection<Entregador>();
        private static EntregadorDAL EntregadorInstance = new EntregadorDAL();

        private EntregadorDAL()
        {
            Entregadores.Add(new Entregador() {
                Id = 1, Nome = "Brauzio", Telefone = "Asdrugio"
            );
            Entregadores.Add(new Entregador() {
                Id = 2, Nome = "Entencius", Telefone = "Gesfredio"
            });
            Entregadores.Add(new Entregador() {
                Id = 3, Nome = "Cartucious", Telefone = "Gesfrundio"
            });
            Entregadores.Add(new Entregador() {
                Id = 4, Nome = "Adoliterio", Telefone = "Kentencio"
            });
            Entregadores.Add(new Entregador() {
                Id = 5, Nome = "Castrogildo", Telefone = "Gesifrelio"
            });
            Entregadores.Add(new Entregador() {
                Id = 6, Nome = "Asdrugio", Telefone = "Brauzio"
            );
            Entregadores.Add(new Entregador() {
                Id = 7, Nome = "Gesfredio", Telefone = "Entencius"
            });
            Entregadores.Add(new Entregador() {
                Id = 8, Nome = "Gesfrundio", Telefone = "Cartuciou
s"
            });
            Entregadores.Add(new Entregador() {
                Id = 9, Nome = "Kentencio", Telefone = "Adoliterio"
            });
            Entregadores.Add(new Entregador() {
```

```
        Id = 10, Nome = "Gesifrelio", Telefone = "Castrogi  
ldo" }));  
    }  
  
    public static EntregadorDAL GetInstance()  
    {  
        return EntregadorInstance;  
    }  
  
    public ObservableCollection<Entregador> GetAll()  
    {  
        return Entregadores;  
    }  
  
    public void Add(Entregador entregador)  
    {  
        this.Entregadores.Add(entregador);  
    }  
}  
}
```

A camada de apresentação

Precisamos agora nos preocupar com a camada de visão. Repetirei aqui o código da página de opções, com a nova opção agora. Crie no projeto uma pasta chamada `Pages` e, dentro dela, a página `MenuPage`.

Para criar a página, clique com o botão direito do mouse sobre a pasta `Pages`, e então em `Add -> New Item`. Na categoria `Cross-Platform`, escolha `Forms Xaml Page`. Atribua `MenuPage` para o nome do arquivo. Implemente-o para que fique como mostra o código a seguir.

Uma curiosidade: ao fazer isso em minha máquina, por algum bug do Visual Studio, a categoria `Cross-Platform` não aparecia. Tive de fechar o Visual Studio e reiniciá-lo. Fica a dica para você quando (e se) isso acontecer.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.MenuPage"
        Title="Menu de Opções">
    <StackLayout VerticalOptions="Center">
        <Button Text="Garçons"
            Image="icone_garcons.png"
            Clicked="GarconsOnClicked"/>
        <Button Text="Entregadores"
            Image="icone_entregadores.png"
            Clicked="EntregadoresOnClicked"/>
    </StackLayout>
</ContentPage>
```

A camada controladora

Bem, o código anterior é nossa primeira implementação XAML, então, é preciso entender as declarações dela. Em primeiro lugar, é preciso que você tenha em mente que o XAML se refere a camada de visão e que, ao criar a página, um arquivo C# também foi criado e vinculado a ele.

Note, na penúltima linha da tag `<ContentPage>`, a declaração `x:Class="Modulo1.Pages.MenuPage"`. Ela associa o XAML à classe implementada no arquivo C#. O título da página é definido na última declaração da mesma tag. As demais cláusulas, `<StackLayout>` e `<Button>`, têm o mesmo comportamento do que foi implementado no capítulo anterior, via código C#.

Veja a definição da propriedade `Clicked`, que se refere ao método que deveremos implementar na classe. Note também que a propriedade `Image` faz referência às imagens que você deverá inserir nos projetos, da mesma maneira que foi feito no capítulo anterior. Para inserir imagens no projeto WinPhone e UWP, elas devem estar na raiz do projeto.

Para acessar a classe criada para o arquivo XAML, expanda o

nome do arquivo na Solution Explorer e dê duplo clique no nome que aparece. Esta classe pode ser vista como controlador para a visão, e seu código pode ser verificado na sequência.

```
using System;
using Xamarin.Forms;

namespace Modulo1.Pages
{
    public partial class MenuPage : ContentPage
    {
        public MenuPage()
        {
            InitializeComponent();
        }

        private async void GarconsOnClicked(object sender, EventArgs args)
        {
            await Navigation.PushAsync(new GarconsPage());
        }

        private async void EntregadoresOnClicked(object sender, EventArgs args)
        {
            await Navigation.PushAsync(new EntregadoresPage());
        }
    }
}
```

Os dois métodos implementados na classe apresentada anteriormente são os responsáveis por capturar o evento de clique dos botões que serão disponibilizados na página de opções. Nós ainda não temos as páginas `GarconsPage` e `EntregadoresPage`, vamos criá-las.

Crie, dentro da pasta `Pages`, duas pastas: uma chamada `Garcons`, e outra `Entregadores`. E em cada uma destas pastas, crie as referidas páginas. Após isso, importe os namespaces na classe apresentada anteriormente. Para mim, eles ficaram como o código a seguir.

```
using Modulo1.Pages.Entregadores;
```

```
using Modulo1.Pages.Garcons;
using System;
using Xamarin.Forms;
```

Podemos agora testar nossa aplicação e ver como fica a página de menu de opções. Para isso, precisamos mudar a classe `App` para que, ao iniciar a aplicação, invoque a página que acabamos de implementar. Expanda o arquivo `App.xaml` na Solution Explorer, e dê duplo clique no arquivo C#. O construtor da classe deverá estar semelhante ao código seguinte.

```
public App()
{
    InitializeComponent();
    MainPage = new NavigationPage(new MenuPage());
}
```

Executando a aplicação

Ok, agora que já temos tudo certo para a nossa página de opções, vamos executá-la. Veja na figura a seguir a aplicação emulada no iOS, Android e Windows Phone. Note a diferença da interface visual, características dos controles e configurações padrões para cada plataforma. As orientações para iniciar os emuladores foram todas vistas no capítulo 2.



Figura 4.1: Página do Menu de Opções

4.2 A LISTAGEM DOS ENTREGADORES

Muito bem, já criamos a página que conterá as opções disponíveis ao usuário e já a testamos também. Agora, vamos implementar a página que conterá as opções de trabalho com os Entregadores.

Para isso, na pasta `Entregadores`, que está dentro de `Pages`, clique com o botão direito do mouse e adicione uma nova página. Dê a ela o nome de `EntregadoresPage`.

Observe, no código a seguir da página, a definição do elemento

```
xmlns:ep="clr-  
namespace:Modulo1.Pages.Representantes;assembly=Modulo1"  
. Tenha em mente que ele funcionará na página como o using  
funciona no C#, ou seja, teremos acesso a todas as classes do  
namespace referenciado no elemento.
```

Verifique também que a página está definida como `TabPage` e não `ContentPage`. Com isso, você precisa mudar a

extensão da classe no código C# dela para TabbedPage também.

Dentro da tag <TabbedPage.Children> , temos a definição das tabs que serão exibidas ao usuário na página, já informando quais páginas serão renderizadas. Sabendo disso, precisamos criar estas duas páginas na pasta Entregadores . Lembra como fazer? Botão direito do mouse no nome da pasta, Add -> New Item -> Cross-Platform -> Forms Xaml Page .

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:ep="clr-namespace:Modulo1.Pages.Entregadores;assembly=Modulo1"
             x:Class="Modulo1.Pages.Entregadores.EntregadoresPage">

    <TabbedPage.Children>
        <ep:EntregadoresListPage Title="Listagem" Icon="icone_list.png"
        x:Name="listagem"/>
        <ep:EntregadoresNewPage Title="Inserir Novo" Icon="icone_new.png"
        x:Name="inserir"/>
    </TabbedPage.Children>
</TabbedPage>
```

Com a página em XAML implementada, precisamos criar as páginas referenciadas nela, que são: EntregadoresListPage e EntregadoresNewPage . Crie-as na pasta Entregadores . Após ter criado as duas páginas, vamos trabalhar nos códigos.

A página de listagem trará um ListView , com a relação dos entregadores registrados, tal qual foi feito para garçons. Já a página de novo item terá um formulário com os controles de entradas de dados sobre entregadores. A listagem a seguir traz o XAML da página EntregadoresListPage .

Observe na listagem que, dentro da tag <ContentPage.Content> , foi definido um StackLayout , que será o container responsável pelos controles visuais a serem exibidos ao usuário. Neste container, temos dois controles: um Label , que

será o título da página, e um `ListView`, que exibirá os entregadores registrados.

Verifique que eles possuem algumas propriedades que são configuradas. No `ListView`, atribuímos um valor a propriedade `Name` do objeto `x`, pois desta maneira, poderemos nos referenciar a este controle no código C# da página.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.Entregadores.EntregadoresListP
age">
    <ContentPage.Content>
        <StackLayout>
            <Label Text="Entregadores" VerticalOptions="Center" Font="25"
HorizontalOptions="Center"/>
            <ListView x:Name="lvEntregadores" RowHeight="70">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="5, 0, 5, 0" Orientation="Verti
cal">
                                <Label Text="{Binding Nome}" TextColor="Blue" Fo
ntSize="Large"/>
                                <Label Text="{Binding Telefone}" TextColor="Gre
n" FontSize="Small"/>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Nosso `ListView` agora exibirá os dados de uma maneira personalizada. Teremos duas linhas para cada entregador, com um label em cada linha e configurações diferenciadas neles. A formatação dos dados pode ser realizada pelas tags `ItemTemplate` e `DataTemplate`, que definem uma `ViewCell`, responsável pela geração da listagem exibida no `ListView`.

O `DataTemplate` permite que você configure os dados que serão exibidos da maneira que você quiser. Veja que os `Labels`, além de uma formatação visual, liga (`Binding`) a propriedade `Text` do controle às propriedades `Nome` e `Telefone`. Mas de onde vêm os dados? Vamos ver o código C# da classe `EntregadoresListPage` na listagem a seguir.

Veja no construtor a chamada a propriedade `ItemsSource`, que recebe o resultado do método `GetAll()` do DAL, definido antes do construtor. O `StackLayout` dos labels tem a propriedade `Padding` definida em `5, 0, 5, 0`. Definindo um espaço de 5 pixels para os lados esquerdo e direito. Esta foi a medida que eu achei adequada, mas teste diferentes valores e verifique o resultado em seu emulador.

```
using Modulo1.Dal;
using Xamarin.Forms;

namespace Modulo1.Pages.Entregadores
{
    public partial class EntregadoresListPage : ContentPage
    {
        private EntregadorDAL dalEntregador = EntregadorDAL.GetInstance();

        public EntregadoresListPage()
        {
            InitializeComponent();
            lvEntregadores.ItemsSource = dalEntregador.GetAll();
        }
    }
}
```

Alguns links interessantes sobre DataTemplate e ViewCell que merecem uma leitura são:

- <https://developer.xamarin.com/guides/xamarin-forms/templates/data-templates/creating/>
- <https://developer.xamarin.com/guides/xamarin-forms/custom-renderer/viewcell/>

Execute sua aplicação, clique no botão de Entregadores e você verá a página responsável por gerar a listagem dos entregadores registrados. Isso pode ser verificado na figura a seguir.

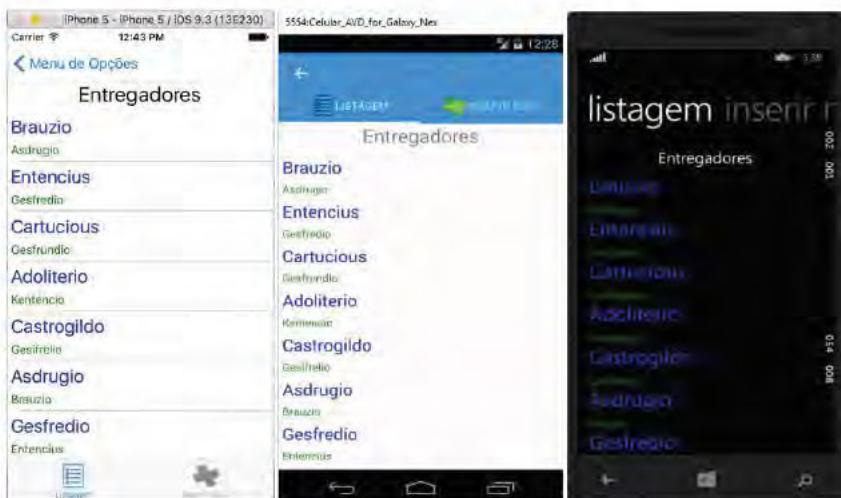


Figura 4.2: Listagem de Entregadores

4.3 A INSERÇÃO DE NOVOS ENTREGADORES

No início deste capítulo, definimos o modelo para nosso problema, a classe `Entregador`. E por último, criamos uma visão que obtém, por uma chamada a um método DAL, todos os entregadores registrados.

Precisamos agora prover ao usuário o mecanismo para inserir um novo Entregador e fazer com que ele já apareça na listagem. Já temos a página `EntregadoresNewPage`, mas vamos deixá-la semelhante ao código a seguir. Ele é grande, mas coloquei alguns comentários para auxiliar.

Após o código, tratarei alguns pontos mais relevantes. Fique atento, pois faço uso de uma nova imagem, que você precisará inserir em seus projetos.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.Entregadores.EntregadoresNewPa
ge">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout VerticalOptions="Center">

<!-- 1. Definição do GRID em que os controles serão inseridos na p
ágina
    São configuradas as quantidades de linhas e colunas. As li
nhas
    estão definidas com altura automática, a única coluna ocup
ará
    100% do tamanho da página --&gt;
            &lt;Grid Padding="5,10,5,10"&gt;
                &lt;Grid.RowDefinitions&gt;
                    &lt;RowDefinition Height="Auto"/&gt;
                    &lt;RowDefinition Height="Auto"/&gt;
                    &lt;RowDefinition Height="Auto"/&gt;
                    &lt;RowDefinition Height="Auto"/&gt;
                &lt;/Grid.RowDefinitions&gt;
                &lt;Grid.ColumnDefinitions&gt;
                    &lt;ColumnDefinition Width="Auto"/&gt;
                &lt;/Grid.ColumnDefinitions&gt;

<!-- 2. Definição do Frame que exibirá o título da página, com uma
    imagem e o valor do ID que será atribuído ao novo entregad
or.
    Este Frame vai na primeira linha e primeira coluna do GRID
--&gt;
                &lt;Frame Grid.Row="0" Grid.Column="0" OutlineColor="Black"
                    BackgroundColor="Yellow" HasShadow="True"</pre>
```

```

        Padding="5,5,5,5">
    <StackLayout>
<!-- 3. Definição do GRID para o Frame -->
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>

<!-- 4. Inserção de uma imagem e dois labels no GRID do FRAME -->

        <Image Source="icone_entregador.png" Grid.Row="0"
Grid.Column="0"/>
        <Label Grid.Row="0" Grid.Column="1" Text="Novo Ent
regador"
            Font="24" TextColor="Blue" HorizontalOptions=
"Start"
            VerticalOptions="Center"/>
        <Label Grid.Row="0" Grid.Column="2" Text="Id" Hori
zontalOptions="End"
            Font="Bold, 24" TextColor="Blue" x:Name="id
entregador"
            VerticalOptions="Center"/>
    </Grid>
    </StackLayout>
</Frame>

<!-- 5. Definição do FRAME e GRID que receberá os dados informados
pelo usuário -->
<Frame Grid.Row="1" Grid.Column="0" OutlineColor="Black"
HasShadow="True">
        Padding="5,5,5,5">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Entry Placeholder="Nome do Entregador" Placeholde
rColor="Gray"
                Grid.Row="0" Grid.Column="0"

```

```

        x:Name="nome"/>
    <Entry Placeholder="Telefone do Entregador" PlaceholderColor="Gray"
           Grid.Row="1" Grid.Column="0"
           x:Name="telefone" Keyboard="Telephone"/>
    </Grid>
</StackLayout>
</Frame>

<!-- 6. Frame e Grid que hospedarão os botões de ação para o formulário
      de entrada de dados para inserção do novo Entregador -->

<Frame Grid.Row="2" Grid.Column="0" OutlineColor="Black"
       HasShadow="True"
       Padding="5,5,5,5">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Button Grid.Row="0" Grid.Column="0" Text="Gravar"
                   Clicked="BtnGravarClick"/>
        </Grid>
    </StackLayout>
</Frame>
</Grid>
</StackLayout>
</ScrollView>
</ContentPage.Content>
</ContentPage>
```

No passo 1, foi criado um `Grid`, com 4 linhas e 1 coluna. As linhas e colunas terão altura automática, relativa ao conteúdo delas. No passo 2, é definido um `Frame`, que será inserido na linha 0 e coluna 0, relativas a primeira linha e coluna do grid criado no passo 1. O `Frame` tem uma configuração visual estabelecida em suas propriedades, cor de fundo amarela, contorno preto e com sombra.

O passo 3 define um `Grid` interno ao `Frame` anteriormente criado. Ele possui 1 linha, com altura automática, e 3 colunas, sendo as duas primeiras de comprimento automático e a terceira

ocupando o espaço que sobrar.

O passo 4 insere uma imagem e 2 labels no `Grid` interno ao `Frame`, sendo um controle em cada célula. Veja nos labels as configurações `HorizontalOptions` e `VerticalOptions`. Procure tentar outros valores; basta pressionar `CTRL+Espaço` entre as aspas do valor para verificar as opções.

O passo 5 define um novo `Frame`, novamente com um `Grid` interno. As configurações de altura e comprimento já foram explicadas. Perceba que os controles `Entry` são os responsáveis pela entrada de dados. Veja que é habilitado um teclado (`keyboard`) para os controles. A ideia do teclado é limitar os caracteres disponíveis para os campos. Pressione `CTRL+Espaço` nas aspas e veja as opções.

O `PlaceHolder` apresenta um texto no controle de entrada, até que o usuário comece a digitar. É uma dica do que deve ser preenchido. O passo 6 traz também configurações comuns, com exceção do controle `Button`, que tem configurado qual o método implementará o comportamento para o evento `Clicked`.

No código anterior, a tag `<ScrollView>` possibilita que a página possa ser arrastada para cima e para baixo, caso ela ocupe um espaço maior que a dimensão da tela do dispositivo. O primeiro `<Grid>` define valores para Padding dos quatro lados do controle. Tente variar estes valores e verifique em seu emulador os resultados.

Para utilizar um `<Grid>`, precisamos configurar suas linhas e colunas. A quantidade de linhas e colunas de um `Grid` é relativa a quantidade de configurações destes itens que fazemos. Um `<Frame>` é um controle que permite hospedar um único objeto, e tem um efeito visual bonito nas aplicações. Você pode querer testar sua aplicação sem usá-lo.

A tag `<Entry>` renderizará um controle que se assemelha a uma caixa de texto, para entrada de dados por parte do usuário. Veja que, no controle para informação de telefone, a propriedade `Keyboard` recebe `Telephone`, o que faz com que o teclado que será disponibilizado ao usuário seja apenas para informar número de telefone. Para mais, acesse o link <https://developer.xamarin.com/recipes/cross-platform/xamarin-forms/controls/choose-keyboard-for-entry/> para saber mais sobre os tipos de teclados disponíveis.

Antes de testarmos nossa aplicação, precisamos implementar o método `BtnGravarClick`, referenciado no `<Button>` de Gravar, e o comportamento necessário para que o Entregador seja gravado. Precisamos validar os dados informados (se estão corretos), chamar o método `Add()` do `DAL`, e depois liberar os controles para uma nova entrada de dados.

Veja o código na sequência. Note a declaração logo no início do DAL. No construtor, há a chamada a um método implementado na própria classe, chamado `PreparaParaNovoEntregador()`. Este método obtém o valor do próximo ID a ser registrado, por meio de LINQ, e limpa os controles para uma nova entrada.

O método que captura o clique do usuário no botão Gravar verifica se os dados estão preenchidos. Caso não estejam, uma janela de alerta é exibida ao usuário; caso contrário, um novo objeto, com os dados informados, é inserido na coleção, por meio da chamada `Add`.

Veja que é feita referência ao nome do controle dado no arquivo XAML. Como nossa coleção faz uso do tipo de dado `ObservableCollection`, a listagem já exibirá o novo dado inserido.

```
using Modulo1.Dal;  
using Modulo1.Modelo;
```

```

using System;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Pages.Entregadores
{
    public partial class EntregadoresNewPage : ContentPage
    {
        private EntregadorDAL dalEntregadores = EntregadorDAL.GetInstance();

        public EntregadoresNewPage()
        {
            InitializeComponent();
            PreparaParaNovoEntregador();
        }

        public void BtnGravarClick(object sender, EventArgs e)
        {
            if (nome.Text.Trim() == string.Empty || telefone.Text
== string.Empty)
            {
                this.DisplayAlert("Erro",
                    "Você precisa informar o nome e telefone para
o novo entregador.",
                    "Ok");
            }
            else
            {
                dalEntregadores.Add(new Entregador()
                {
                    Id = Convert.ToInt32(identregador.Text),
                    Nome = nome.Text,
                    Telefone = telefone.Text
                });
                PreparaParaNovoEntregador();
            }
        }

        private void PreparaParaNovoEntregador()
        {
            var novoId = dalEntregadores.GetAll().Max(x => x.Id) +
1;
            identregador.Text = novoId.ToString().Trim();
            nome.Text = string.Empty;
            telefone.Text = string.Empty;
        }
    }
}

```

}

Agora vamos testar. Execute sua aplicação. Quando aparecer a listagem dos entregadores, clique na Tab `Inserir Novo`. Sua página deve estar semelhante à apresentada na figura a seguir.

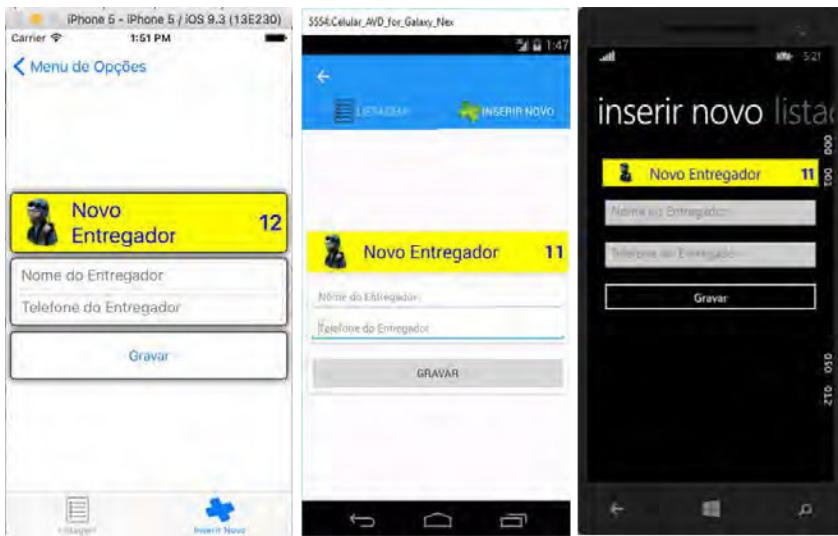


Figura 4.3: Inserindo um novo entregador

4.4 UNIVERSAL WINDOWS PLATFORM

Se você tem um dispositivo com o Windows 10 ou está usando o Windows 10, é possível testar o projeto `UWP`. Veja na figura a seguir as imagens da execução nesta plataforma.

Antes de executar o projeto, como `Local Machine`, é preciso realizar a instalação do projeto. Faça isso clicando com o botão direito do mouse sobre o nome do projeto `UWP` e selecione `Deploy`.



Figura 4.4: Janelas da simulação do projeto UWP — Local Machine

Agora, da mesma maneira que foi apresentada no capítulo 2 a execução do projeto UWP em um emulador, a figura a seguir representa as páginas implementadas para entregadores em um emulador para o Windows 10 Mobile que será nossa plataforma alvo para aplicações Windows a partir do próximo capítulo (onde trabalharemos com dispositivos físicos).

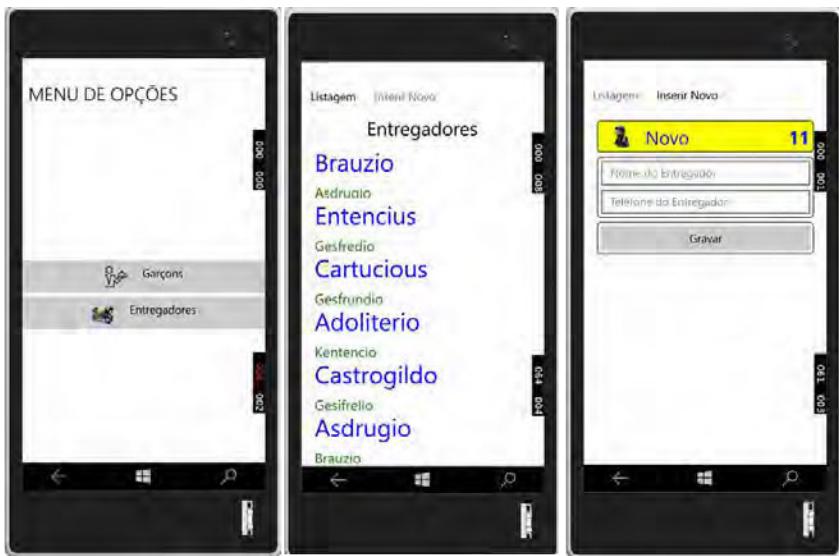


Figura 4.5: Janelas da simulação do projeto UWP em um emulador

4.5 CONCLUSÃO

Concluímos mais um capítulo. Vimos novos recursos nele, e configuramos a maneira como os itens de uma `ListView` podem ser exibidos. Criamos uma página de inserção de dados e vimos bastante código XAML.

O próximo capítulo será muito interessante, pois faremos uso da câmera fotográfica do celular e também veremos como acessar a galeria de fotos. Para que possamos fazer estes testes, faremos o deploy da aplicação para um dispositivo real, e não um emulador. Será um capítulo cheio de novidades. Respire um pouco e pegue fôlego para ele.

CAPÍTULO 5

ACESSO À CÂMERA E À GALERIA DE FOTOS

No capítulo anterior, implementamos a funcionalidade de registro de um novo dado a uma coleção existente. Com esta implementação, já temos os modelos de Garçons e Entregadores implementados, faltando apenas as operações de alteração e exclusão, que vamos implementar neste capítulo.

Podemos também supor que, tanto para Garçom como para Entregador, é interessante termos uma foto de cada um e que ela possa ser exibida não apenas na página de cadastro, mas também na listagem dos dados já registrados. Faremos isso aqui.

Para que uma foto seja selecionada da galeria de fotos ou uma nova seja tirada, precisamos acessar o dispositivo em si, pois os emuladores não trazem este recurso. Desta maneira, além das funcionalidades anteriores, instalaremos (distribuiremos) o aplicativo para um aparelho celular e testaremos a aplicação nele. De quebra, mudaremos o ícone e nome da aplicação quando distribuída. Este capítulo terá muita novidade, então mãos à obra.

5.1 PUBLICAÇÃO DA APLICAÇÃO PARA UM DISPOSITIVO IOS

Para que você se torne um desenvolvedor certificado pela Apple e possa fazer parte do `Apple Developer Program`, existe uma

série de tramitações que devem ser seguidas e que estão muito bem documentadas em https://developer.xamarin.com/guides/ios/getting_started/installation/device_provisioning/. Entretanto, para fazer um deploy como um desenvolvedor "simples" e sem custo, existe um processo mais simples de ser realizado, e é ele que apresentarei aqui.

O primeiro requisito é ter um id Apple. Se não tem, obtenha um em <https://appleid.apple.com/account#!&page=create>. Em seu Mac, acesse o XCode. A versão mínima exigida é a 7.

Com o XCode ativo, acesse o XCode Menu -> Preferences . Na categoria Accounts , clique na base esquerda, no botão + para adicionar seu Id Apple. Sua senha será solicitada neste momento e a janela a seguir é exibida.

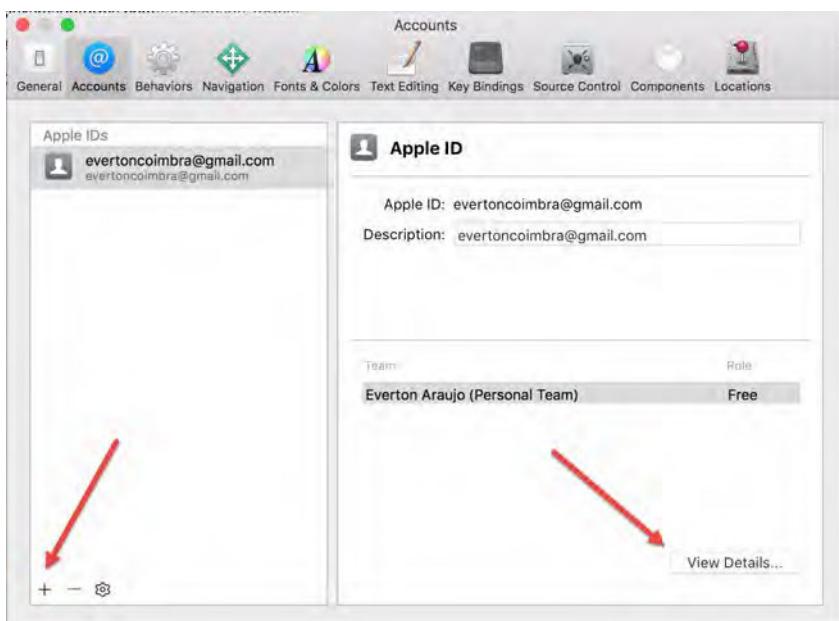


Figura 5.1: Criando uma conta no XCode

Com a conta criada, precisamos agora criar uma Signing

Identity. Para isso, clique no botão `View Details...` e a janela apresentada na figura a seguir é exibida. Clique no botão `Create` que aparece ao lado de `iOS Development`.

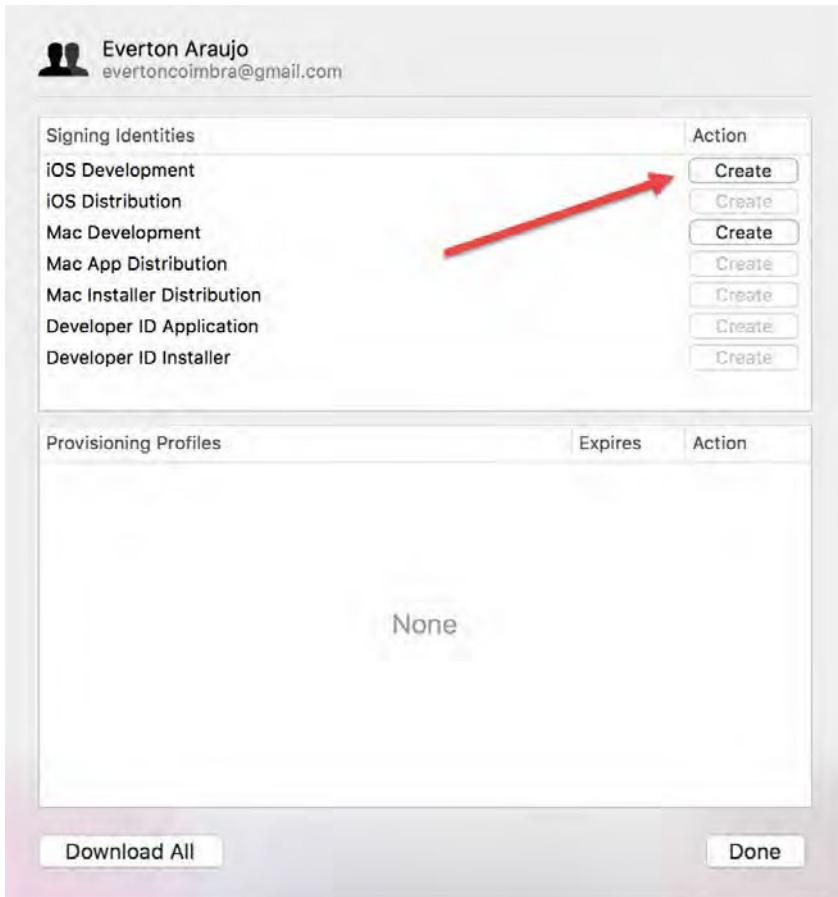


Figura 5.2: Criando uma Signing Identity

Precisamos criar um projeto no XCode e instalá-lo no dispositivo que usaremos para teste. Sendo assim, conecte seu dispositivo iOS em seu Mac. Em meu caso, estou usando um iPhone SE com o iOS 9.3.4. No XCode, selecione o menu `File -> New -> Project` e, na categoria `iOS -> Application`, escolha `Single View Application` e clique no botão `Next`.

Na janela que se abre, precisamos fornecer algumas informações. A mais importante é a `Organization Identifier`, que deverá ser a mesma na aplicação Xamarin. Muita atenção neste item. Veja a figura a seguir. Após registrar as informações, clique em `Next`. Selecione a página para salvar e clique no botão `Create` na nova janela que se abre.

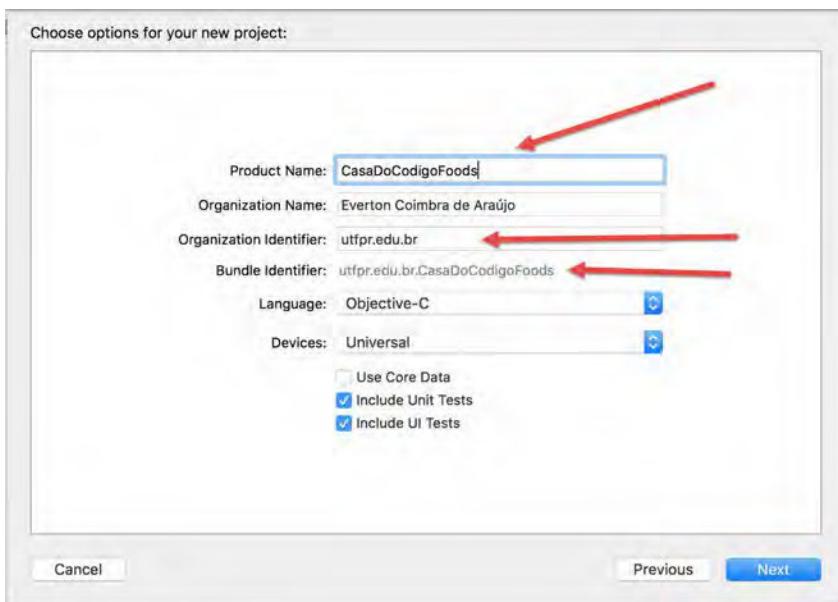


Figura 5.3: Criando uma aplicação no XCode

Vamos agora fazer a configuração do `Signing Identity` e a versão mínima para uso de sua aplicação. Quando você clicou no botão `Create`, uma janela com diversas guias apareceu. Em `Team`, selecione sua conta criada anteriormente, e informe a versão 8.4 como mínima, que é a padrão oferecida e atende às versões 5 e superior do iPhone.

É preciso também configurar a execução para seu dispositivo, e depois clicar no botão `Fix Issue` para criar o perfil de configuração. Veja a figura a seguir. Execute a aplicação. Se neste

momento uma janela de erro aparecer em seu dispositivo, informando-o para verificar configurações de confiança, não se preocupe, isso ocorre na primeira execução de uma aplicação via o mecanismo que estamos utilizando, o de depuração. Realize a configuração indicada pelo dispositivo, que é bem simples, e execute novamente sua aplicação.

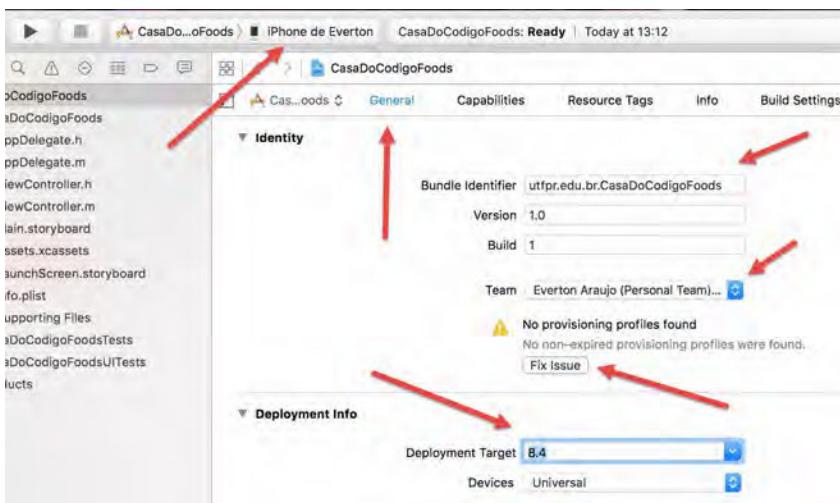


Figura 5.4: Configurando o Signing Identity no projeto XCode

Se tudo deu certo, uma aplicação, sem exibir nada, será executada em seu dispositivo. Na figura a seguir, você pode ver a aplicação instalada em meu iPhone. Lembre de encerrar a execução da aplicação, no XCode, ou no próprio iPhone.

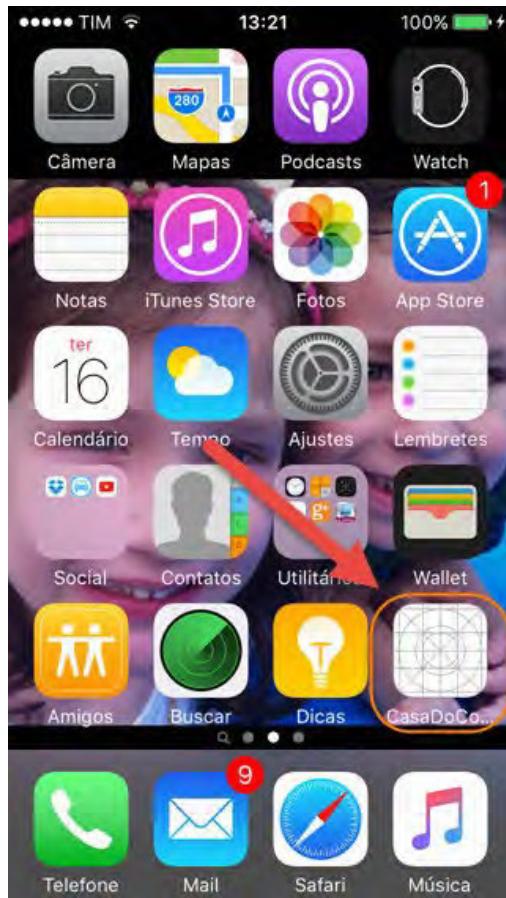


Figura 5.5: Aplicação XCode instalada no dispositivo

Precisamos agora configurar nossa aplicação (no Visual Studio) para o processo de deploy no dispositivo. O seu dispositivo deve ficar conectado no Mac. Na `Solution Explorer`, clique com o botão direito do mouse sobre o nome do projeto iOS, e em seguida em `Properties`.

Na janela que se abre, clique no lado esquerdo, na categoria `iOS Application`. Do lado direito, configure o `Identifier` para o mesmo valor que utilizou no XCode, assim como a versão mínima para deploy. Tenha certeza de que, no topo direito, esteja

selecionada a plataforma iPhone . Veja a figura a seguir.

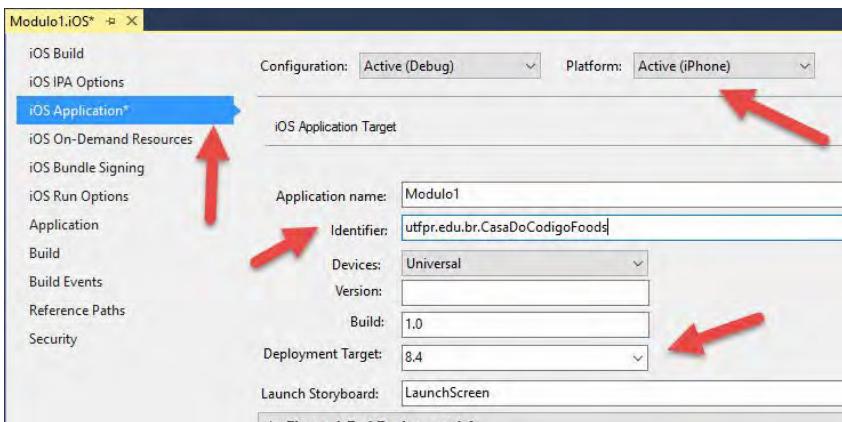


Figura 5.6: Configurando a aplicação iOS no Visual Studio

Na sequência, clique também do lado esquerdo, na categoria **iOS Bundle Signing** , e selecione os arquivos que foram criados em seu Mac, conforme a seguir.



Figura 5.7: Configurando o Signing Identity no Visual Studio

Vamos testar. Na barra de tarefas no Visual Studio, verifique se os valores estão corretos para a CPU e para a execução em seu dispositivo. Deve estar semelhante ao apresentado na figura adiante. Execute sua aplicação.

Quando fui executar, a aplicação não era inicializada no iPhone. Eu encerrei todas as aplicações nele e tentei novamente. Deu certo. Maiores detalhes podem ser vistos em https://developer.xamarin.com/guides/ios/getting_started/installation/

[on/device_provisioning/free-provisioning/](#), para auxiliar na compreensão deste processo.



Figura 5.8: Verificando atalhos e configurações para execução da aplicação no dispositivo

5.2 PUBLICAÇÃO DA APLICAÇÃO PARA UM DISPOSITIVO ANDROID

A distribuição e execução de uma aplicação em um dispositivo Android são mais simples do que o que vimos para um Apple. Estou realizando os testes em um Samsung S6 com Android 6.

O primeiro passo é habilitar as opções para o desenvolvedor (*Developer Options*). Para isso, vá em Configurações -> Sobre o dispositivo -> Informações de Software e clique 7 vezes sobre o Número de compilação (*build number*). Após isso, retorne para Configurações e clique em Opções do Desenvolvedor (*Developer Options*) e habilite a opção USB Debugging .

Pronto. Com isso, o dispositivo aparecerá na listagem de dispositivos para execução da aplicação se seu projeto de inicialização for o Android. Para o Android inferior à versão 4.2, veja o link https://developer.xamarin.com/guides/android/getting_started/installation/set_up_device_for_development/. Execute sua aplicação e veja-a executando em seu dispositivo.

5.3 PUBLICAÇÃO DA APLICAÇÃO PARA UM DISPOSITIVO WINDOWS PHONE

O desenvolvimento de uma aplicação para funcionar em um

dispositivo com o Windows Phone 8 tem uma complexidade: você precisa ter uma conta de desenvolvedor na Microsoft, o que implica em **inve\$timento**. Isso não é mais necessário para o desenvolvimento UWP, bastando apenas configurar o aparelho para opções de desenvolvedor.

Estou utilizando um Nokia Lumia 925 com o Windows 10 Mobile. Vá em Configurações -> Atualização e Segurança -> Para desenvolvedores . Marque a opção Modo de desenvolvedor e a Torne seu dispositivo visível para conexões USB e sua rede local .

Pronto, com isso seu dispositivo já aparecerá no Visual Studio. Esta será a plataforma alvo para as aplicações Windows.

5.4 INSERÇÃO DE IMAGENS ÀS LISTAGENS

Para a inserção de imagens que possam ser exibidas no ListView , poderíamos adaptar as páginas de Entregadores e Garçons . Porém, vamos criar um novo modelo e, com base nas mudanças, você atualiza as páginas já implementadas.

O novo modelo será relacionado à categorização dos pratos e bebidas oferecidos pelo estabelecimento. Pense em Pizzas, Filés, Frangos, Sanduíches, Bebidas e assim por diante. Vamos nomear esta classe de TipoItemCardapio . Sendo assim, em sua pasta de modelos, crie esta classe conforme a listagem a seguir.

```
namespace Modulo1.Modelo
{
    public class TipoItemCardapio
    {
        public long Id { get; set; }
        public string Nome { get; set; }
        public string CaminhoArquivoFoto { get; set; }
    }
}
```

Precisamos adicionar na página de opções a nova opção que implementaremos. Busque uma imagem e insira na página MenuPage essa nova opção. O código para ela está na sequência.

```
<Button Text="Tipos de Itens"  
       Image="icone_tipositenscardapio.png"  
       Clicked="TiposItensCardapioOnClicked"/>
```

Como pode ser verificado na listagem anterior e seguindo o que fizemos nos capítulos anteriores, precisamos implementar o método que captura o evento `Click` do botão. Lembre-se de que ele está na classe da própria página. Veja o código na sequência.

```
private async void TiposItensCardapioOnClicked(object sender, Event  
tArgs args)  
{  
    await Navigation.PushAsync(new TiposItensCardapioPage());  
}
```

Para que o código anterior possa ser compilado corretamente, precisamos criar a página anteriormente implementada. Crie, então, uma pasta chamada `TiposItensCardapio` e, dentro dela, crie a página conforme o código a seguir.

A implementação é semelhante à que fizemos para Entregadores, mudando apenas os nomes das páginas, namespace e classe. Lembre-se de mudar a extensão da classe para `TabbedPage` também, pois, como pode ser visto na tag a seguir, a página é um `TabbedPage`. E quando criamos uma página, o padrão é ter a classe estendida de `ContentPage`.

```
<?xml version="1.0" encoding="utf-8" ?>  
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"  
           xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
  
           xmlns:ep="clr-namespace:Modulo1.Pages.TiposItensCarda  
pio;assembly=Modulo1"  
           x:Class="Modulo1.Pages.TiposItensCardapio.TiposItensC  
ardapioPage">  
    <TabbedPage.Children>  
        <ep:TiposItensCardapioListPage Title="Listagem" Icon="icone_li
```

```

st.png" x:Name="listagem"/>
    <ep:TiposItensCardapioNewPage Title="Inserir Novo" Icon="icone
_new.png" x:Name="inserir"/>
</TabbedPage.Children>
</TabbedPage>

```

Como a página anterior referencia duas novas páginas, para que o código possa ser compilado, precisamos criá-las. Desta maneira, crie-as com os nomes referenciados no código. A listagem a seguir traz a implementação para a página `TiposItensCardapioListPage`. Em relação à página semelhante para Entregadores, muda apenas a retirada de um `Label` e a inserção da instrução `<Image Source="{Binding CaminhoArquivoFoto}" />`. Para que uma imagem seja exibida no `ListView`, adicionei o `VerticalOptions="Center"` no `Label` e coloquei o título da página dentro de um `StackLayout`.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.TiposItensCardapio.TiposItensCardapioPage">
    <ContentPage.Content>
        <StackLayout>
            <StackLayout Padding="5, 20, 5, 20">
                <Label Text="Tipos de Itens Cardápio" VerticalOptions="Ce
nter" Font="25"
                    HorizontalOptions="Center"/>
            </StackLayout>
            <ListView x:Name="lvTiposItensCardapio" RowHeight="60">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="5, 0, 5, 0" Orientation="Horiz
ontal">
                                <Image Source="{Binding CaminhoArquivoFoto}" />
                                <Label Text="{Binding Nome}" TextColor="Blue" Font
Size="Large"
                                    VerticalOptions="Center"/>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>

```

```

        </ListView>
    </StackLayout>
</ContentPage.Content>
</ContentPage>

```

Para que possamos ligar o `ListView` a uma fonte de dados, precisamos antes da camada de persistência. Sendo assim, implemente na pasta `Dal` a classe `TipoItemCardapioDAL`, conforme código a seguir. Perceba que, por enquanto, ainda faço uso de uma coleção e insiro nela alguns itens iniciais. Veja a referência às figuras, você precisará importá-las para os projetos.

```

using Modulo1.Modelo;
using System.Collections.ObjectModel;

namespace Modulo1.Dal
{
    public class TipoItemCardapioDAL
    {
        private ObservableCollection<TipoItemCardapio> TiposItensCardapio =
            new ObservableCollection<TipoItemCardapio>();
        private static TipoItemCardapioDAL TipoItemCardapioInstanc
e = new TipoItemCardapioDAL();

        private TipoItemCardapioDAL()
        {
            TiposItensCardapio.Add(new TipoItemCardapio()
            {
                Id = 1, Nome = "Pizza", CaminhoArquivoFoto = "pizz
as.png"
            });
            TiposItensCardapio.Add(new TipoItemCardapio()
            {
                Id = 2, Nome = "Bebidas", CaminhoArquivoFoto = "be
bidas.png"
            });
            TiposItensCardapio.Add(new TipoItemCardapio()
            {
                Id = 3, Nome = "Saladas", CaminhoArquivoFoto = "sa
ladas.png"
            });
            TiposItensCardapio.Add(new TipoItemCardapio()
            {
                Id = 4, Nome = "Sanduíches", CaminhoArquivoFoto =
"sanduiches.png"
            });
        }
    }
}

```

```

        });
        TiposItensCardapio.Add(new TipoItemCardapio()
        {
            Id = 5, Nome = "Sobremesas", CaminhoArquivoFoto =
"sobremesas.png"
        });
        TiposItensCardapio.Add(new TipoItemCardapio()
        {
            Id = 6, Nome = "Carnes", CaminhoArquivoFoto = "car
nes.png"
        });
    }

    public static TipoItemCardapioDAL GetInstance()
    {
        return TipoItemCardapioInstance;
    }

    public ObservableCollection<TipoItemCardapio> GetAll()
    {
        return TiposItensCardapio;
    }

    public void Add(TipoItemCardapio tipoItemCardapio)
    {
        this.TiposItensCardapio.Add(tipoItemCardapio);
    }
}
}

```

Agora, para que possamos enfim testar nossa aplicação, que até aqui tem a mudança no menu de opções e a nova listagem, precisamos ligar o `ListView` com uma fonte de dados. Semelhante ao realizado no capítulo anterior, vamos implementar isso na classe da página de listagem. Veja o código da classe na sequência.

```

using Modulo1.Dal;
using Xamarin.Forms;

namespace Modulo1.Pages.TiposItensCardapio
{
    public partial class TiposItensCardapioListPage : ContentPage
    {
        private TipoItemCardapioDAL dalTipoItemCardapio = TipoItem
CardapioDAL.GetInstance();
    }
}

```

```

public TiposItensCardapioListPage()
{
    InitializeComponent();
    lvTiposItensCardapio.ItemsSource = dalTipoItemCardapio
    .GetAll();
}
}
}

```

Agora é possível executarmos a aplicação e vermos as mudanças. Veja na sequência as figuras da página de listagem. Preferi não exibir a página de opções, por ter pouca mudança.

A partir deste momento, começamos a ver alguns problemas com as aplicações direcionadas para o Windows Phone e UWP. No emulador do Windows Phone, as figuras não aparecem na listagem e, no UWP, os itens e imagens só apareceram no emulador após eu rotacioná-lo.

A compatibilidade com a plataforma Windows não é das melhores. Quem sabe, com a Microsoft assumindo a plataforma, estes problemas possam ser corrigidos.

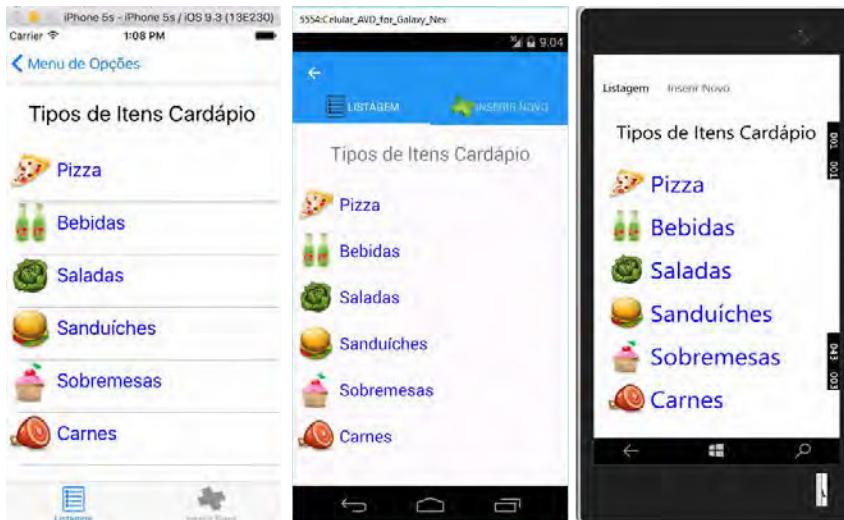


Figura 5.9: Página com a listagem dos itens registrados

5.5 INTERAÇÃO COM A CÂMERA E O ÁLBUM

Vamos agora ver uma implementação bem interessante, vamos fazer com que nossa aplicação interaja com a câmera, permitindo que uma foto possa ser tirada, armazenada no dispositivo e também na galeria de fotos. Também veremos como selecionar e usar uma imagem armazenada nessa galeria.

Implementar esta funcionalidade apenas com o Xamarin Forms não é uma tarefa trivial, pois cada plataforma tem sua API de comunicação com a câmera e álbum. Precisaríamos implementar esta funcionalidade em cada projeto e não no PCL.

Eu fiz isso como um teste, gerou muito código e a complexidade foi grande. Este tipo de implementação é conhecida como *DependencyService*, e você pode ver mais detalhes sobre isso em <https://developer.xamarin.com/guides/xamarin-forms/dependency-service/>.

Uma das maravilhas do Xamarin Forms é a componentização. Existem vários componentes que podem nos auxiliar, e muito. Os homologados pela equipe do Xamarin podem ser vistos em <https://components.xamarin.com/>.

Usaremos dois destes recursos em nossa aplicação agora, que são:

- Media Plugin —
<https://components.xamarin.com/view/MediaPlugin>
- File System Plugin (ou PCL Storage) —
<https://components.xamarin.com/view/pclstorage>

Vamos instalá-los em **todos** os projetos, via Nuget. Para isso, clique com o botão direito no nome de cada projeto e escolha Manage NuGet Packages... . Busque por estes componentes e

instale-os. Veja os destaques na figura a seguir.

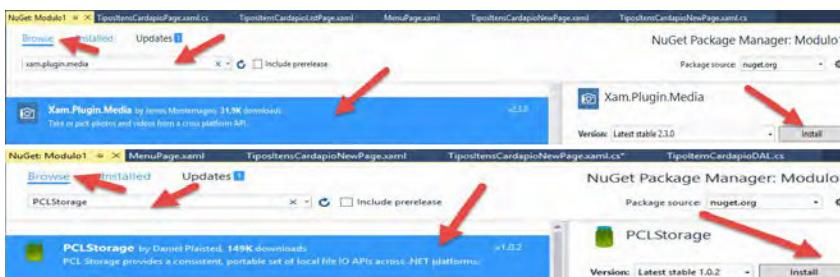


Figura 5.10: Instalando os componentes

Precisamos agora desenhar a página que permitirá o acesso à câmera e ao álbum, e exibirá a foto tirada ou selecionada. Para facilitar, vou mostrar esta figura antes do código e falarei sobre a funcionalidade dela. Veja a figura a seguir.

Os testes para as três plataformas foram também realizados em dispositivos físicos. O deploy para o dispositivo Windows teve de ser usando a CPU ARM.

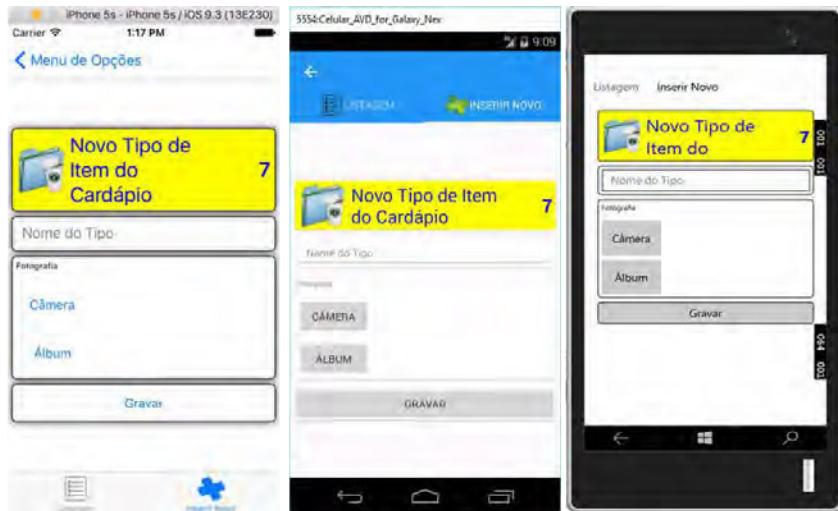


Figura 5.11: Página de inserção de um novo tipo de item do cardápio

A informação do nome e o botão de gravar são semelhantes ao que implementamos para Entregadores, sendo a parte da fotografia o diferencial nesta página. Nesta parte, temos dois botões, Câmera e Álbum , que acessam os recursos de seu nome.

Na primeira vez que o usuário tentar acessar um destes recursos, o dispositivo solicitará permissão para este acesso. Ao tirar uma foto e confirmar o uso dela, ou selecionar uma foto do álbum, ela aparecerá na página. Após gravar, você poderá visualizar a listagem de itens, que já mostra o item novo registrado e sua foto. A figura seguinte mostra a captura de tela dos dispositivos.

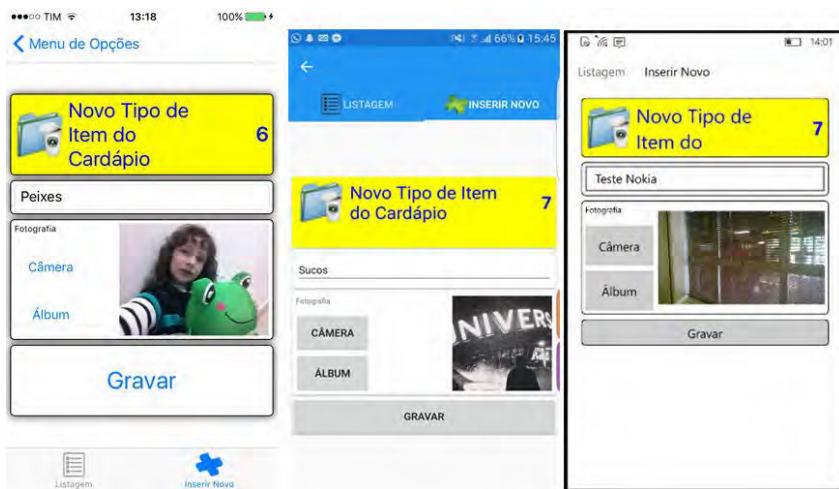


Figura 5.12: Página no dispositivo após a seleção de uma foto

Ok. Agora que você já viu as imagens da aplicação em execução, vamos para o código. O primeiro é o XAML da página `TiposItensCardapioNewPage` . Veja o código a seguir. Praticamente é a mesma coisa que fizemos para Entregadores. Talvez você queira dar uma olhada especial no `Frame` da linha (Row) 2 do Grid, que é o específico para os dados da fotografia.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

x:Class="Modulo1.Pages.TiposItensCardapio.TiposItensCardapioNewPage"
<ContentPage.Content>
    <ScrollView>
        <StackLayout VerticalOptions="Center">
            <Grid Padding="5,10,5,10">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="100*"/>
                </Grid.ColumnDefinitions>
                <Frame Grid.Row="0" Grid.Column="0" OutlineColor="Black"
                    BackgroundColor="Yellow" HasShadow="True"
                    Padding="5,5,5,5">
                    <StackLayout>
                        <Grid>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="Auto"/>
                            </Grid.RowDefinitions>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="20*"/>
                                <ColumnDefinition Width="60*"/>
                                <ColumnDefinition Width="20*"/>
                            </Grid.ColumnDefinitions>
                            <Image Source="icone_tipotemcardapio.png" Grid.Row="0" Grid.Column="0"/>
                            <Label Grid.Row="0" Grid.Column="1" Text="Novo Típo de Item do Cardápio"
                                Font="24" TextColor="Blue" HorizontalOptions="Start"
                                VerticalOptions="Center"/>
                            <Label Grid.Row="0" Grid.Column="2" Text="Id" HorizontalOptions="End"
                                Font="Bold, 24" TextColor="Blue" x:Name="idtipotemcardapio"
                                VerticalOptions="Center"/>
                        </Grid>
                    </StackLayout>
                </Frame>
                <Frame Grid.Row="1" Grid.Column="0" OutlineColor="Black"
                    HasShadow="True"
                    Padding="5,5,5,5">

```

```

<StackLayout>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100*"/>
        </Grid.ColumnDefinitions>
        <Entry Placeholder="Nome do Tipo" PlaceholderColor="Gray"
               Grid.Row="0" Grid.Column="0" Text="{Binding Nome}"
               x:Name="nome"/>
    </Grid>
</StackLayout>
</Frame>

<!-- Frame específico para a parte de fotografia -->
<Frame Grid.Row="2" Grid.Column="0" OutlineColor="Black"
       HasShadow="True"
       Padding="5,5,5,5" x:Name="framefoto">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="20"/>
                <RowDefinition Height="50"/>
                <RowDefinition Height="50"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="30*"/>
                <ColumnDefinition Width="70*"/>
            </Grid.ColumnDefinitions>
            <Label Grid.Row="0" Grid.Column="0" Text="Fotografia" FontSize="10"
                  HorizontalOptions="Start"/>
            <Button Grid.Row="1" Grid.Column="0" Text="Câmera"
                   x:Name="BtnCamera"/>
            <Button Grid.Row="2" Grid.Column="0" Text="Álbum"
                   x:Name="BtnAlbum"/>
            <Image Grid.Row="0" Grid.Column="1" Grid.RowSpan="3"
                   Grid.ColumnSpan="2" x:Name="fototipoitemcardapio" HorizontalOptions="End"/>
        </Grid>
    </StackLayout>
</Frame>

<Frame Grid.Row="3" Grid.Column="0" OutlineColor="Black"

```

```

HasShadow="True"
    Padding="0">
<StackLayout>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100*"/>
        </Grid.ColumnDefinitions>
        <Button Grid.Row="0" Grid.Column="0" Text="Gravar"
            Clicked="BtnGravarClick"/>
    </Grid>
</StackLayout>
</Frame>
</Grid>
</StackLayout>
</ScrollView>
</ContentPage.Content>
</ContentPage>

```

Vamos ver agora como ficou a implementação da classe da página anterior. Vou colocar o código em partes, para que eu possa comentá-lo de maneira que facilite sua compreensão. Começo com o início da classe, os usings, campos definidos e construtor. Veja a seguir.

Temos dois campos criados na classe: o DAL, que já utilizamos na página de Entregadores, e agora um campo que é auxiliar. Ele conterá o nome do arquivo que representa a foto tirada ou selecionada no álbum. Quando tirarmos ou selecionarmos uma foto, esta variável terá a atribuição e, quando formos gravar o item, usaremos este valor.

O construtor possui três chamadas a métodos (que logo apresento) além do `InitializeComponent()`. O primeiro prepara o formulário para uma nova entrada de dados. Já o segundo e terceiro registram um método para o evento click dos botões relacionados à câmera e ao álbum, sendo que, para a câmera, envio o Id do novo item a ser inserido, pois ele comporá o nome do

arquivo.

Para o álbum, não utilizei esta estratégia, mantendo o nome original da foto no álbum. Quando usarmos banco de dados, tudo isso muda.

```
using Modulo1.Dal;
using Modulo1.Modelo;
using Plugin.Media;
using System;
using System.Linq;
using Xamarin.Forms;
using PCLStorage;

namespace Modulo1.Pages.TiposItensCardapio
{
    public partial class TiposItensCardapioNewPage : ContentPage
    {
        private TipoItemCardapioDAL dalTiposItensCardapio = TipoItemCardapioDAL.GetInstance();
        private string caminhoArquivo;

        public TiposItensCardapioNewPage()
        {
            InitializeComponent();
            PreparaParaNovoTipoItemCardapio();
            RegistraClickBotaoCamera(idtipoitemcardapio.Text.Trim());
            RegistraClickBotaoAlbum();
        }
    }
}
```

Começo agora a apresentar os métodos que foram implementados para atender aos eventos do formulário de inserção de item. Veja na sequência o método

`PreparaParaNovoTipoItemCardapio()`. Note que a implementação do método faz uso do LINQ para obter o maior valor para a propriedade `Id` nos elementos da coleção, por meio a chamada ao método `Max()`, enviando como argumento uma expressão lambda, um recurso do C#. As instruções restantes do método apenas limpam os controles de interação com o usuário, para que novos dados possam ser inseridos.

Você pode obter uma introdução sobre LINQ e Expressões Lambdas em:

- <https://msdn.microsoft.com/pt-br/library/bb397906.aspx>
- <https://msdn.microsoft.com/pt-br/library/bb397687.aspx>.

```
private void PreparaParaNovoTipoItemCardapio()
{
    var novoId = dalTiposItensCardapio.GetAll().Max(x => x.Id) + 1;
    idtipoitemcardapio.Text = novoId.ToString().Trim();
    nome.Text = string.Empty;
    fototipoitemcardapio.Source = null;
}
```

Na sequência, veja o método que registra o evento clique para o botão que habilitará a câmera e permitirá que seja tirada uma foto. Note que o método é declarado como assíncrono, por ter invocações a métodos que podem demorar para responder. Desta maneira, a aplicação não fica travada.

Inseri alguns comentários no código para facilitar. Perceba que mantive a opção de salvar a foto também no álbum apenas para que você pudesse ver esta possibilidade.

```
private void RegistraClickBotaoCamera(string idparafoto)
{
    // Criação do método anônimo para captura do evento click do botão
    BtnCamera.Clicked += async (sender, args) =>
    {
        // Inicialização do plugin de interação com a câmera e álbum
        await CrossMedia.Current.Initialize();

        // Verificação de disponibilização da câmera e se é possível tirar foto
    };
}
```

```

        if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)
        {
            DisplayAlert("Não existe câmera", "A câmera não está disponível.", "OK");
            return;
        }

        /* Método que habilita a câmera, informando a pasta onde a foto deverá ser armazenada, o nome a ser dado ao arquivo e se é ou não para armazenar a foto também no álbum */
        var file = await CrossMedia.Current.TakePhotoAsync(
            new Plugin.Media.Abstractions.StoreCameraMediaOptions
        {
            Directory = FileSystem.Current.LocalStorage.Name,
            Name = "tipotipoitem_" + idparafoto + ".jpg",
            SaveToAlbum = true
        });

        // Caso o usuário não tenha tirado a foto, clicando no botão cancelar
        if (file == null)
            return;

        // Armazena o caminho da foto para ser utilizado na gravação do item
        caminhoArquivo = file.Path;

        // Recupera a foto e a atribui para o controle visual
        fototipoitemcardapio.Source = ImageSource.FromStream(() =>
    {
        var stream = file.GetStream();
        file.Dispose();
        return stream;
    });
}
}

```

Precisamos ver agora o método que possibilita o acesso ao álbum. Veja-o na sequência. Procure observar no código que o arquivo é recuperado e gravado em outro local, para uso pela aplicação. Isso garante que a foto esteja disponível para a aplicação mesmo que o usuário elimine-a do álbum.

Neste código, fizemos uso dos três componentes anteriormente instalados. Procure depurar esta aplicação para acompanhar os valores. Sim, você pode depurar normalmente a aplicação executando no dispositivo.

```
private void RegistraClickBotaoAlbum()
{
    // Criação do método anônimo para captura do evento click do botão
    BtnAlbum.Clicked += async (sender, args) =>
    {
        // Inicialização do plugin de interação com a câmera e álbum
        await CrossMedia.Current.Initialize();

        // Verificação de disponibilização do álbum
        if (!CrossMedia.Current.IsPickPhotoSupported)
        {
            DisplayAlert("Álbum não suportado", "Não existe permissão para acessar o álbum de fotos", "OK");
            return;
        }

        // Método que habilita o álbum e permite a seleção de uma foto
        var file = await CrossMedia.Current.PickPhotoAsync();

        // Caso o usuário não tenha selecionado uma foto, clicando no botão cancelar
        if (file == null)
            return;

        /* Nas instruções abaixo é feito uso dos components PCLStorage
        e PCLSpecialFolder */

        // Recupera o arquivo com base no caminho da foto selecionada
        var getArquivoPCL = FileSystem.Current.GetFileFromPathAsync(file.Path);

        // Recupera o caminho raiz da aplicação no dispositivo
        var rootFolder = FileSystem.Current.LocalStorage;

        /* Caso a pasta Fotos não exista, ela é criada para armazenamento da foto selecionada */
        var folderFoto = await rootFolder.CreateFolderAsync("Fotos")
```

```

, CreationCollisionOption.OpenIfExists);

    // Cria o arquivo referente à foto selecionada
    var setArquivoPCL = folderFoto.CreateFileAsync(getArquivoPCL.Result.Name, CreationCollisionOption.ReplaceExisting);

        // Guarda o caminho do arquivo para ser utilizado na gravação do item criado
        caminhoArquivo = setArquivoPCL.Result.Path;

        // Recupera o arquivo selecionado e o atribui ao controle no formulário
        fototipoitemcardapio.Source = ImageSource.FromStream(() =>
    {
        var stream = file.GetStream();
        file.Dispose();
        return stream;
    });
}
}

```

Nesse código, a instrução `var getArquivoPCL = FileSystem.Current.GetFileFromPathAsync(file.Path);` não funciona no Windows 10 Mobile. O arquivo não é recuperado. A instrução refere-se ao componente PCL Storage e, infelizmente, deve ter algum conflito com a plataforma, embora a documentação não traga nada a respeito disso.

A chamada ao método retorna uma `UnauthorizedException`, mas todos os recursos estão habilitados na configuração do arquivo. Talvez a saída, se for preciso deste recurso, seja buscar por outro componente que pudesse resolver este problema, ou desenvolver uma implementação específica para a plataforma, por meio do `Dependency Service`.

Não farei esta implementação aqui. Fica apenas o registro deste problema. Mas fique tranquilo, no código de alteração do tipo (classe `TipoItensCardapioEditPage`), trago outra solução para esse problema.

Para finalizar esta implementação, nos resta o método que

implementa a gravação do item. Veja-o na sequência. Não inseri comentários neste código, pois ele é simples e fácil de entender.

Há a validação de informação do nome para o tipo, e a foto não é obrigatória. Caso os dados estejam informados, o objeto é instanciado e inserido na coleção por meio de uma chamada ao método `Add()` do DAL. Após este procedimento, o formulário é preparado para uma nova inserção.

```
public void BtnGravarClick(object sender, EventArgs e)
{
    if (nome.Text.Trim() == string.Empty)
    {
        this.DisplayAlert("Erro",
            "Você precisa informar o nome para o novo tipo de item
do cardápio.",
            "Ok");
    }
    else
    {
        dalTiposItensCardapio.Add(new TipoItemCardapio()
        {
            Id = Convert.ToInt32(idtipoitemcardapio.Text),
            Nome = nome.Text,
            CaminhoArquivoFoto = caminhoArquivo
        });
        PreparaParaNovoTipoItemCardapio();
    }
}
```

Com esta implementação, finalizo esta seção. Teste sua aplicação agora. Faça os testes nos emuladores e depois nos dispositivos.

Vamos agora partir para a alteração dos dados. Se seu dispositivo iOS for iOS 10 ou superior, é preciso inserir a definição `NSCameraUsageDescription` em seu arquivo `info.plist` para ser permitido o acesso à câmera, e `NSPhotoLibraryUsageDescription` para ter o acesso à galeria de fotos liberado.

É interessante você acessar e ler o link

https://developer.xamarin.com/guides/ios/platform_features/introduction-to-ios10/. Ele traz uma introdução aos recursos e configurações para o desenvolvimento para a plataforma iOS 10.

5.6 ALTERAÇÃO DE DADOS EXISTENTES NA COLEÇÃO

O ListView oferece um recurso chamado Context Action , que se resume à oferta de possíveis interações com cada item renderizado. Este efeito pode ser verificado quando, no iOS, você pressiona um item e o arrasta para a esquerda. No Android e UWP, eles aparecem quando você mantém pressionada uma linha por algum tempo. Veja isso na figura a seguir.

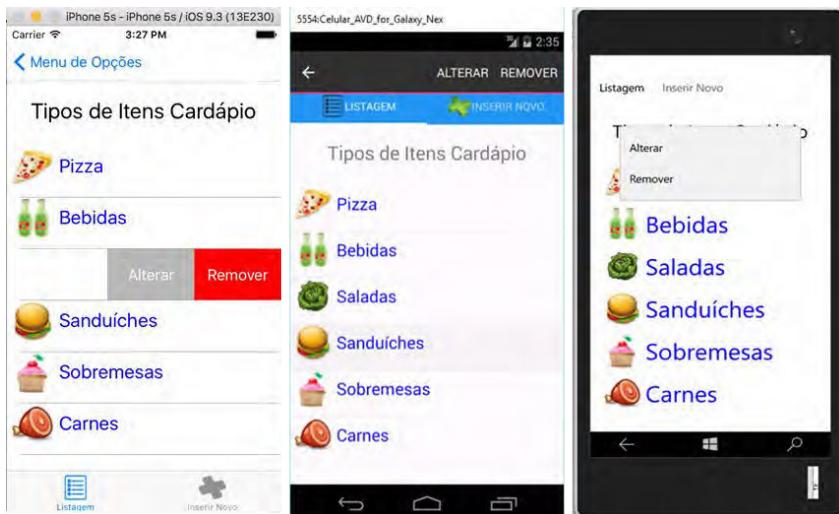


Figura 5.13: Visualizando Command Actions no ListView

A execução da aplicação quando o usuário interage é simples. Quando ele clicar na Action Remover, uma mensagem de diálogo aparecerá, perguntando se confirma a remoção. Se confirmar, o item é removido da coleção e, consequentemente, da ListView . Ao confirmar a alteração, uma nova página é renderizada, com os

dados já aparecendo, possibilitando ao usuário alterar o dado que quiser e depois gravá-lo.

Ao confirmar os novos dados, eles já aparecerão na `ListView` atualizados. O primeiro código que apresentarei é o do que foi alterado no `ListView`. Veja-o na sequência. Note que ele está logo abaixo da abertura da `<ViewCell>`.

```
<!-- código omitido -->
<ViewCell>
    <ViewCell.ContextActions>
        <MenuItem Clicked="OnAlterarClick" CommandParameter="{Binding .}"
        Text="Alterar" />
        <MenuItem Clicked="OnRemoverClick" CommandParameter="{Binding .}"
        Text="Remover" IsDestructive="True" />
    </ViewCell.ContextActions>
<!-- código omitido -->
```

Ao implementar o código anterior, precisamos implementar os métodos `OnAlterarClick()` e `OnRemoverClick()` na classe da página. Veja que a propriedade `Text` contém o texto a ser exibido para cada Context Action. Já a `CommandParameter` enviará para o método o objeto ligado à linha onde os Context Actions foram exibidos.

Veja a implementação do método de remoção na sequência. Note que é obtido o objeto recebido como argumento, convertido para uma referência de `MenuItem` e, consequentemente, para `TipoItemCardapio`. Tudo isso poderia estar em uma única linha.

A variável `opcao` recebe um boolean após a interação do usuário com a janela gerada pelo `DisplayAlert()`, sendo `true` caso o botão `Sim` seja o pressionado, e `false` caso seja o `Não`. Se a confirmação para remoção ocorra, o objeto é removido da coleção.

```
public async void OnRemoverClick(object sender, EventArgs e)
{
    var mi = ((MenuItem)sender);
```

```

var item = mi.CommandParameter as TipoItemCardapio;
var opcao = await DisplayAlert("Confirmação de exclusão",
    "Confirma excluir o item " + item.Nome.ToUpper() + "?", "Sim",
    "Não");
if (opcao)
{
    dalTipoItemCardapio.Remove(item);
}
}

```

Para que a remoção anterior possa ocorrer, precisamos implementar os métodos `Equals()` e `GetHashCode()` na classe de modelo `TipoItemCardapio`. Veja esta implementação na listagem seguinte.

```

public override bool Equals(object obj)
{
    TipoItemCardapio tipoItemCardapio = obj as TipoItemCardapio;
    if (tipoItemCardapio == null)
    {
        return false;
    }
    return (Id.Equals(tipoItemCardapio.Id));
}

public override int GetHashCode()
{
    return Id.GetHashCode();
}

```

Com a remoção implementada, vamos para a alteração. Veja seu código na sequência. Observe que existe o mesmo trabalho para obtenção do item em referência na `ListView`. O comportamento esperado é que muda, pois precisamos de uma página para que os dados sejam alterados.

Neste instante, teremos muita redundância de código, pois a interface com o usuário para a alteração é semelhante à da inserção. O mesmo ocorrerá para a implementação do comportamento da interface na classe da página, entretanto, neste momento deixaremos assim. Logo veremos como minimizar este problema.

```
public async void OnAlterarClick(object sender, EventArgs e)
```

```

{
    var mi = ((MenuItem)sender);
    var item = mi.CommandParameter as TipoItemCardapio;
    await Navigation.PushModalAsync(new TiposItensCardapioEditPage
(item));
}

```

Verifique no código anterior que a página `TiposItensCardapioPage` será renderizada como `Modal`, ou seja, uma camada acima da página que a invoca. Desta maneira, quando terminarmos o processo de alteração, apenas fechamos esta página, trazendo o controle da aplicação para a página do `ListView`. Na sequência, você pode visualizar o código XAML da página referenciada no código anterior, a `TiposItensCardapioEditPage`.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    x:Class="Modulo1.Pages.TiposItensCardapio.TiposItensCardapioEditPage">
    <ContentPage.Content>
        <ScrollView>
            <StackLayout VerticalOptions="Center">
                <Grid Padding="5,10,5,10">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="100*"/>
                    </Grid.ColumnDefinitions>
                    <Frame Grid.Row="0" Grid.Column="0" OutlineColor="Black"
                        BackgroundColor="Yellow" HasShadow="True"
                        Padding="5,5,5,5">
                        <StackLayout>
                            <Grid>
                                <Grid.RowDefinitions>
                                    <RowDefinition Height="Auto"/>
                                </Grid.RowDefinitions>
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="20*"/>

```

```

        <ColumnDefinition Width="60*" />
        <ColumnDefinition Width="20*" />
    </Grid.ColumnDefinitions>
    <Image Source="icone_tipotemcardapio.png" Grid.Row="0" Grid.Column="0"/>
    <Label Grid.Row="0" Grid.Column="1" Text="Alterando o Tipo de Item do Cardápio"
        Font="24" TextColor="Blue" HorizontalOptions="Start"
        VerticalOptions="Center" />
    <Label Text="{Binding Id}" Grid.Row="0" Grid.Column="2" HorizontalOptions="End"
        Font="Bold, 24" TextColor="Blue" x:Name="id_tipotemcardapio"
        VerticalOptions="Center" />
</Grid>
</StackLayout>
</Frame>

<Frame Grid.Row="1" Grid.Column="0" OutlineColor="Black" HasShadow="True">
    Padding="5, 5, 5, 5">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="100*" />
            </Grid.ColumnDefinitions>
            <Entry Placeholder="Nome do Tipo" PlaceholderColor="Gray"
                Grid.Row="0" Grid.Column="0" Text="{Binding Nome}"
                x:Name="nome" />
        </Grid>
    </StackLayout>
</Frame>

<Frame Grid.Row="2" Grid.Column="0" OutlineColor="Black" HasShadow="True">
    Padding="5, 5, 5, 5" x:Name="framefoto">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="20" />
                <RowDefinition Height="50" />
                <RowDefinition Height="50" />

```

```

        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="30*" />
            <ColumnDefinition Width="70*" />
        </Grid.ColumnDefinitions>
        <Label Grid.Row="0" Grid.Column="0" Text="Fotografia" FontSize="10"
               HorizontalOptions="Start"/>
        <Button Grid.Row="1" Grid.Column="0" Text="Câmera" x:Name="BtnCamera"/>
        <Button Grid.Row="2" Grid.Column="0" Text="Álbum" x:Name="BtnAlbum"/>
        <Image Source="{Binding CaminhoArquivoFoto}" Grid.Row="0" Grid.Column="1" Grid.RowSpan="3" Grid.ColumnSpan="2" x:Name="fototipoitemcardapio" HorizontalOptions="End"/>
    </Grid>
</StackLayout>
</Frame>

<Frame Grid.Row="3" Grid.Column="0" OutlineColor="Black" HasShadow="True">
    <Padding>"0"</Padding>
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="100*" />
            </Grid.ColumnDefinitions>
            <Button Grid.Row="0" Grid.Column="0" Text="Gravar" Clicked="BtnGravarClick"/>
        </Grid>
    </StackLayout>
</Frame>
</Grid>
</StackLayout>
</ScrollView>
</ContentPage.Content>
</ContentPage>

```

Agora, na sequência, apresento todo o código da classe da página anterior. Veja na chamada que o construtor receberá um argumento, que é o objeto a ser alterado. Algumas mudanças foram

feitas no comportamento dos métodos, mas todas fáceis de compreender, por isso não inseri comentários neste código.

Veja o comportamento dos métodos que registram os eventos cliques para os botões de acesso à câmera e ao álbum. Eles estão mais enxutos. O clique do botão Gravar invoca o método `Update`, que ainda não implementamos no DAL.

```
using Modulo1.Dal;
using Modulo1.Modelo;
using PCLStorage;
using Plugin.Media;
using System;

using Xamarin.Forms;

namespace Modulo1.Pages.TiposItensCardapio
{
    public partial class TiposItensCardapioEditPage : ContentPage
    {
        private TipoItemCardapio tipoItemCardapio;
        private string caminhoArquivo;
        private TipoItemCardapioDAL dalTiposItensCardapio = TipoItemCardapioDAL.GetInstance();

        public TiposItensCardapioEditPage(TipoItemCardapio tipoItemCardapio)
        {
            InitializeComponent();
            PopularFormulario(tipoItemCardapio);
            RegistraClickBotaoCamera(idtipoitemcardapio.Text.Trim());
            RegistraClickBotaoAlbum();
        }

        private void PopularFormulario(TipoItemCardapio tipoItemCardapio)
        {
            this.tipoItemCardapio = tipoItemCardapio;
            idtipoitemcardapio.Text = tipoItemCardapio.Id.ToString();
            nome.Text = tipoItemCardapio.Nome;
            caminhoArquivo = setArquivoPCL.Result.Path;
            fototipoitemcardapio.Source = ImageSource.FromFile(tipoItemCardapio.CaminhoArquivoFoto);
        }
    }
}
```

```

private void RegistraClickBotaoAlbum()
{
    BtnAlbum.Clicked += async (sender, args) =>
    {
        await CrossMedia.Current.Initialize();

        if (!CrossMedia.Current.IsPickPhotoSupported)
        {
            DisplayAlert("Álbum não suportado", "Não existe permissão para acessar o álbum de fotos", "OK");
            return;
        }

        var file = await CrossMedia.Current.PickPhotoAsync();
        var getArquivoPCL = FileSystem.Current.GetFileFromPathAsync(file.Path);
        var rootFolder = FileSystem.Current.LocalStorage;
        var folderFoto = await rootFolder.CreateFolderAsync("Fotos", CreationCollisionOption.OpenIfExists);
        var setArquivoPCL = folderFoto.CreateFileAsync(getArquivoPCL.Result.Name, CreationCollisionOption.ReplaceExisting);
        var arquivoLido = getArquivoPCL.Result.ReadAllBytesAsync();
        var arquivoEscrito = setArquivoPCL.Result.WriteAllBytesAsync(arquivoLido.Result);
        caminhoArquivo = setArquivoPCL.Result.Path;

        if (file == null)
            return;

        fototipoitemcardapio.Source = ImageSource.FromStream(() =>
        {
            var stream = file.GetStream();
            file.Dispose();
            return stream;
        });
    };
}

private void RegistraClickBotaoCamera(string idparafoto)
{
    BtnCamera.Clicked += async (sender, args) =>
    {
        await CrossMedia.Current.Initialize();

```

```

        if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)
        {
            await DisplayAlert("Não existe câmera", "A câmera não está disponível.", "OK");
            return;
        }

        var file = await CrossMedia.Current.TakePhotoAsync(
(
        new Plugin.Media.Abstractions.StoreCameraMediaOptions
        {
            Directory = FileSystem.Current.LocalStorage.Name,
            Name = "tipoitem_" + idparafoto.Trim() + ".jpg"
        });
    }

    if (file == null)
        return;

    fototipoitemcardapio.Source = ImageSource.FromFile(file.Path);
    caminhoArquivo = file.Path;
}
}

public async void BtnGravarClick(object sender, EventArgs e)
{
    if (nome.Text.Trim() == string.Empty)
    {
        this.DisplayAlert("Erro",
            "Você precisa informar o nome para o novo tipo de item do cardápio.",
            "Ok");
    }
    else
    {
        this.tipoItemCardapio.Nome = nome.Text;
        this.tipoItemCardapio.CaminhoArquivoFoto = caminhoArquivo;

        dalTiposItensCardapio.Update(this.tipoItemCardapio);
        await Navigation.PopModalAsync();
    }
}

```

```
        }  
    }  
}
```

No método `BtnGravarClick()`, implementado na listagem anterior, após a gravação das alterações realizadas pelo método `Update()` (na classe `TipoItemCardapioDAL`), apresentado na sequência, é possível verificar que a página de alteração é disposta pela chamada ao método `PopModalAsync()`, e o controle da aplicação retorna para a janela de listagem.

```
public void Update(TipoItemCardapio tipoItemCardapio)  
{  
    this.TiposItensCardapio[this.TiposItensCardapio.IndexOf(tipoIt  
emCardapio)] = tipoItemCardapio;  
}
```

5.7 ALTERANDO O ÍCONE, O NOME DA APLICAÇÃO E A COR DA PÁGINA DE ABERTURA

Apresentarei agora como trocar o ícone da aplicação e a cor da página de abertura. Como nos exemplos anteriores, começarei com o iOS.

Alteração da aplicação iOS

Vamos lá. Clique com o botão direito do mouse sobre o nome do projeto iOS e clique na guia `iOS Application`. Na caixa `Application Name`, digite o nome que deseja para a aplicação. Eu coloquei `CC Foods`.

Agora, para mudar o ícone, desça a janela de configurações até a seção `App. Spotlight and Setting icons`. Verifique que existem várias possibilidades para a configuração de ícones. Para saber a resolução dos ícones para cada configuração, passe o mouse sobre a área dos ícones e espere até o `hint` aparecer com as

informações.

Para as configurações que não existem ícones pré-definidos, a resolução aparece como imagem. Busque pelo ícone que deseja utilizar e clique sobre a configuração desejada. Veja a imagem da minha configuração na figura a seguir.

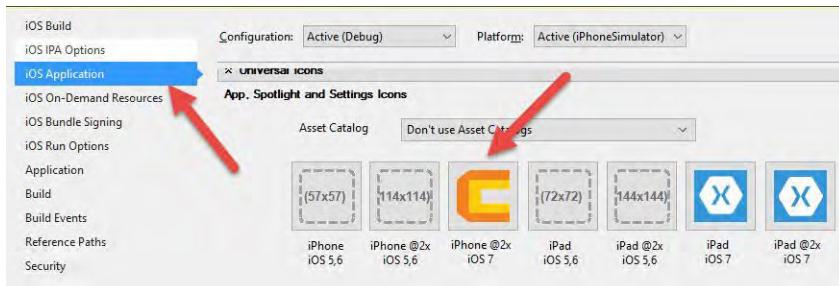


Figura 5.14: Configurando o ícone da aplicação

Precisamos agora mudar a configuração da cor da página de abertura. Para isso, dê duplo clique no arquivo `LaunchScreen.storyboard`, que está na pasta `Resources` do projeto `iOS`. Você deverá visualizar a área de design para projetos `iOS`, mas como estamos usando o `Xamarin Forms`, não mexeremos nisso para os controles. Entretanto, para a configuração do `Background` da página de inicialização, faremos isso.

Abra sua janela de propriedades (`F4` ou `Menu View`). Altere a cor de fundo, de acordo com o seu gosto. Veja a minha configuração na figura a seguir. Para alterar o ícone da janela de apresentação, clique sobre ele e selecione o novo na janela de propriedades do ícone. Realize sua alteração e execute sua aplicação para ver as mudanças.

Legal, não é? Leia maiores detalhes em: https://developer.xamarin.com/guides/ios/application_fundamentals/working_with_images/app-icons/.

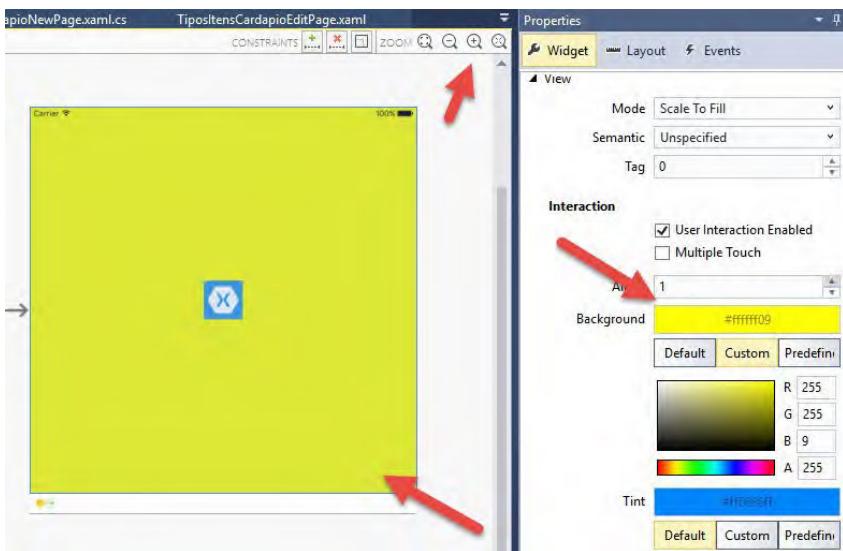


Figura 5.15: Configurando a cor de fundo para a página de início da aplicação iOS

Alteração dos ícones para Android

A documentação do Xamarin aponta para que as mudanças sejam feitas em sua janela de propriedades (botão direito do mouse sobre o nome do projeto e `Properties`), entretanto, para a mudança do nome da aplicação, foi preciso mexer no código. Veja na figura a seguir a configuração realizada em dita janela. Atente que o ícone (figura chamada `icon.png`) precisa estar em todas as pastas `drawable` .

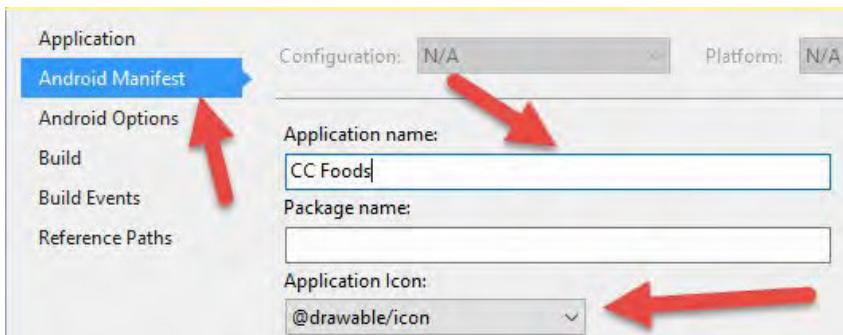


Figura 5.16: Configurando ícone e nome da aplicação Android

Se você realizou as configurações anteriores e executou sua aplicação, notou que o ícone mudou, mas o nome da aplicação não. Para isso, abra o arquivo `MainActivity.cs` na raiz do projeto, e mude o valor dado para o elemento `Label`. Veja o meu código na sequência.

```
[Activity(Label = "CC Foods", Icon = "@drawable/icon", Theme = "@style/MainTheme", MainLauncher = true, ConfigurationChanges = ConfigurationChanges.ScreenSize | ConfigurationChanges.Orientation)]
```

Você pode ver a documentação do Xamarin para deploy em Android no link: https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_application/part_1_-_preparing_an_application_for_release/#Specify_the_Application.Icon.

Alteração dos ícones para UWP

A alteração dos ícones da aplicação para esta plataforma é extremamente simples. Verifique as imagens, semanticamente nomeadas na pasta `Assets`, e crie suas figuras substituindo-as.

O nome da aplicação pode ser alterado nas propriedades da aplicação. Para isso, clique com o botão direito do mouse sobre o

nome do projeto e altere o `Assembly Name`.

5.8 CONCLUSÃO

Ufa, chegamos ao final deste capítulo. Como prometido, trouxe muita coisa nova e interessante. Foi intenso.

Você pôde trabalhar com componentes de terceiros, em que foi possível trabalhar com a câmera, álbum e sistemas de arquivos dos dispositivos. Com isso, você pôde testar sua aplicação em um dispositivo físico, o que torna mais legal ainda o processo de desenvolvimento. Infelizmente, foi possível verificar que a aplicação gerada para a UWP não funciona como nas outras duas plataformas.

Vimos como empilhar uma página em outra, quando trabalhamos com a alteração de dados. Esta alteração foi possível também ao utilizarmos o novo recurso de arrastar uma linha do `ListView` e ter opções de interação, pseudobotões.

Finalizamos o capítulo com a troca de ícone e nome da aplicação. Muita coisa para colocar em prática no que construímos nos capítulos anteriores. Agora, você pode dar uma relaxada, pegar um fôlego e voltar para o próximo capítulo, que abordará o uso de persistência física dos dados. Faremos isso com o SQLite. Prepare-se.

CAPÍTULO 6

O USO DE BANCO DE DADOS COM O SQLITE

Já vimos vários recursos até o capítulo anterior, recursos estes relacionados a controles para interface com o usuário e para o uso de recursos do dispositivo, como a câmera e o álbum. Temos dados persistidos e recuperados de uma coleção, mas nada permanente.

Agora, chegou a hora de termos os dados persistidos em uma base de dados, para que possam ser recuperados e utilizados sempre que a aplicação os solicitar. Para realizarmos esta atividade, faremos uso do SQLite (<https://sqlite.org/>), um pequeno e poderoso sistema para persistência de dados.

Como primeira atividade, trabalharemos na implementação que fizemos anteriormente, com tipos de itens para o cardápio. Faremos com que os dados sejam persistidos e obtidos em uma base de dados do SQLite. Para isso, faremos uso de novos componentes para o Xamarin.

Após conseguirmos implementar um CRUD em uma tabela simples (mapeada de uma classe), faremos a persistência de uma classe que possui associações (relacionamento). Para isso, vamos utilizar um segundo componente para o Xamarin.

Também faz parte deste capítulo a criação de um controle reutilizável em XAML e a implementação de novos recursos para o `ListView`, como o agrupamento de dados. Veremos ainda como

criar um conversor de dados, um recurso muito interessante oferecido pelo Xamarin. Tome um fôlego e vamos ao trabalho.

6.1 INSTALAÇÃO DO SQLITE NA APLICAÇÃO

Quando trabalhamos com aplicações complexas, robustas, seja para o ambiente web ou desktop, temos sempre um servidor de banco de dados funcionando na persistência dos dados do sistema. Pode ser o Oracle, o SQL Server, Postgres ou MySQL. São produtos mundialmente conhecidos.

Quando o desenvolvimento de uma aplicação tem como plataforma alvo um dispositivo móvel, fica difícil imaginar uma solução dessa. Quem sabe no futuro isso seja possível, mas, atualmente, o que se utiliza na persistência de dados em dispositivos móveis tem sido, em sua maioria, o SQLite.

O SQLite possui uma licença simples de uso e recomendo que você a leia no site do produto, anteriormente informado. Ele não requer nenhuma configuração no dispositivo em que será usado. O acesso aos dados pode ser por instruções SQL diretas ou por meio do LINQ.

Existem algumas versões do SQLite disponíveis para serem instaladas por meio do NuGet, sendo a `SQLite-net PCL` a mais utilizada e inclusive referenciada na documentação do Xamarin Forms. Mas nós faremos uso de um projeto que é uma evolução deste componente, o `SQLite.Net PCL`.

Vamos instalar este componente em nossa solução, o que fará com que ele seja atualizado em todos os projetos. Clique com o botão direito do mouse no nome da solução e escolha `Manage NuGet Packages for Solution...`. Pesquise por `SQLite.Net` e, ao encontrá-lo, instale-o. Veja a figura a seguir.

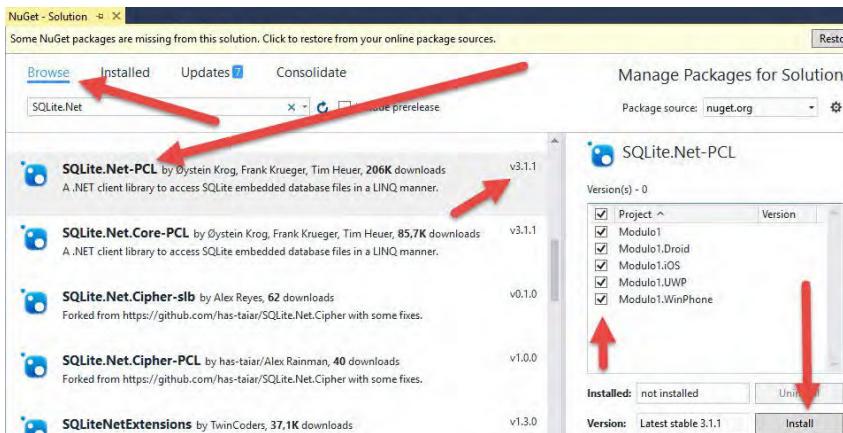


Figura 6.1: Instalação do SQLite.NET-PCL na solução

6.2 ADAPTAÇÃO DA CLASSE DE MODELO PARA A PERSISTÊNCIA

A nossa classe de modelo `TipoItemCardapio` agora será persistida em uma base de dados por meio do SQLite. Desta maneira, precisamos fazer algumas mudanças no código. Veja-o na listagem a seguir.

Note logo no início o uso de um namespace do SQLite. Na propriedade `TipoItemCardapioId`, há agora os atributos `PrimaryKey` e `AutoIncrement`, tornando esta propriedade o campo que será chave primária na tabela, como autoincremento, tirando de nossa responsabilidade a atribuição para seu valor. Veja que já declarei a propriedade da foto como um array de bytes, o que será mapeado para um campo BLOB na tabela.

```
using SQLite.Net.Attributes;

namespace Modulo1.Modelo
{
    public class TipoItemCardapio
    {
        [PrimaryKey, AutoIncrement]
        public long? TipoItemCardapioId { get; set; }
```

```

        public string Nome { get; set; }
        public byte[] Foto { get; set; }

        public override bool Equals(object obj)
        {
            TipoItemCardapio tipoItemCardapio = obj as TipoItemCardapio;
            if (tipoItemCardapio == null)
            {
                return false;
            }

            return (TipoItemCardapioId.Equals(tipoItemCardapio.TipoItemCardapioId));
        }

        public override int GetHashCode()
        {
            return TipoItemCardapioId.GetHashCode();
        }
    }
}

```

6.3 IMPLEMENTAÇÃO DA PERSISTÊNCIA PARA A CLASSE TIPOITEMCARDAPIO

Com o modelo adaptado, precisamos agora configurar nossa aplicação para acessar uma base de dados do SQLite. Como cada plataforma possui sua particularidade no que diz respeito ao acesso realizado pelo SQLite ao arquivo de base de dados, precisaremos fazer uso dos recursos de `Dependency Service`, já comentado por mim quando trabalhamos com acesso à câmera, no capítulo anterior.

O primeiro passo é criarmos uma interface que deverá ser implementada em cada projeto, de acordo com as particularidades de cada plataforma. Sendo assim, no projeto PCL, crie uma nova pasta, chamada `Infraestructure`, e nela uma interface chamada `IDatabaseConnection`, com o código apresentado na listagem a seguir.

Veja que a interface define apenas um método, o que será responsável por obter a conexão com a base de dados. Veja novamente o `using` para o SQLite.

```
using SQLite.Net;

namespace Modulo1.Infraestructure
{
    public interface IDatabaseConnection
    {
        SQLiteConnection DbConnection();
    }
}
```

Configuração da dependência no projeto iOS

Agora, precisamos realizar a implementação do `DependeceService` em cada plataforma, e começaremos pela `iOS`. Na raiz deste projeto, crie uma classe chamada `DatabaseConnection_iOS`, com o código a seguir. Veja que ela implementa a interface que criamos anteriormente.

Antes disso, para o namespace, é realizado um registro de dependência para a classe que estamos implementando. É com esta instrução que o projeto PCL requisitará esta classe. No corpo do método, há instruções para nomear e gravar o arquivo da base de dados. O nome que estamos dando é `CCFoodsDb.db3`. Obtemos a pasta que desejamos que o arquivo seja criado e retornamos uma instância de `SQLiteConnection`, que permitirá a interação com a base de dados.

```
using Modulo1.Infraestructure;
using Modulo1.iOS;
using SQLite.Net;
using SQLite.Net.Platform.XamarinIOS;
using System;
using System.IO;

[assembly: Xamarin.Forms.Dependency(typeof(DatabaseConnection_iOS))]
namespace Modulo1.iOS
```

```

{
    public class DatabaseConnection_iOS : IDatabaseConnection
    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CCFoodsDb.db3";
            string personalFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            string libraryFolder = Path.Combine(personalFolder, ".",
            Library");
            var path = Path.Combine(libraryFolder, dbName);
            var platform = new SQLitePlatformIOS();
            return new SQLiteConnection(platform, path);
        }
    }
}

```

Configurando a dependência no projeto Android

Para o funcionamento de uma aplicação com o SQLite na plataforma Android, precisamos criar também a classe que atenderá pela Dependency Service . Para isso, no projeto Droid, crie a classe `DatabaseConnection_Android` com o código a seguir. Note que é muito semelhante à implementação que fizemos para iOS. Damos um nome à base de dados, obtemos o caminho onde ela será criada, e então retornamos a conexão para ela.

```

using Modulo1.Infraestructure;
using Modulo1.Droid;
using SQLite.Net.Platform.XamarinAndroid;
using System.IO;
using SQLite.Net;

[assembly: Xamarin.Forms.Dependency(typeof(DatabaseConnection_Android))]
namespace Modulo1.Droid
{
    public class DatabaseConnection_Android : IDatabaseConnection
    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CCFoodsDb.db3";
            string documentsFolder = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);
            string path = Path.Combine(documentsFolder, dbName);

```

```

        var platform = new SQLitePlatformAndroid();
        return new SQLiteConnection(platform, path);
    }
}

```

Configuração da dependência no projeto UWP

Para que você consiga êxito na execução de sua aplicação com SQLite na plataforma UWP, é preciso adicionar um pacote VSIX para o Visual Studio, disponibilizado no site do SQLite (<http://sqlite.org>). Este pacote é o `sqlite-uwp-3140100.vsix`. Baixe-o e o execute.

Lembre-se de ter fechado o Visual Studio antes. Após a instalação, clique com o botão direito do mouse em `References` do projeto UWP, e depois em `Add Reference...`. Na janela que se abre, clique em `Extensions`, na categoria `Universal Windows`. Na direita, marque as opções `SQLite for Universal Windows Platform` e `Visual C++ 2015 Runtime for Universal Windows Platform Apps`.

Agora, crie a classe `DatabaseConnection_UWP`, com o código a seguir. Note uma vez mais que a lógica é a mesma aplicada para o iOS e Android. Precisamos dar um nome ao arquivo que será a base de dados e obtermos o caminho onde ela será gravada. Tendo isso, já é possível retornar a conexão com o banco.

```

using Modulo1.Infraestructure;
using Modulo1.UWP;
using SQLite.Net;
using SQLite.Net.Platform.WinRT;
using System.IO;
using Windows.Storage;

[assembly: Xamarin.Forms.Dependency(typeof(DatabaseConnectio_UWP))]
namespace Modulo1.UWP
{
    public class DatabaseConnectio_UWP : IDatabaseConnection

```

```

    {
        public SQLiteConnection DbConnection()
        {
            var dbName = "CCFoodsDb.db3";
            string path = Path.Combine(ApplicationData.Current.LocalFolder.Path, dbName);
            var platform = new SQLitePlatformWinRT();
            return new SQLiteConnection(platform, path);
        }
    }
}

```

Implementação do DAL

Agora precisamos implementar os métodos de acesso e persistência, e vamos fazê-los na classe `TipoItemCardapioDAL`. Modificaremos a implementação que existe nela, que trabalhava com coleções, pela que se segue, que faz uso da classe anteriormente implementada.

Observe no construtor que a chamada ao método `Get()` de `DependencyService` retorna a conexão com o SQLite, pela implementação que fizemos anteriormente nos projetos específicos de cada plataforma. Após isso, criaremos a tabela pela qual esta classe DAL é responsável. Se a tabela já existir, nada é feito. Os métodos que se seguem são de fácil compreensão, pois representam a implementação do CRUD. Veja que é feito uso do LINQ.

```

using Modulo1.Infraestructure;
using Modulo1.Modelo;
using SQLite.Net;
using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Dal
{
    public class TipoItemCardapioDAL
    {
        private SQLiteConnection sqlConnection;

        public TipoItemCardapioDAL()

```

```

    {
        this.sqlConnection = DependencyService.Get<IDatabaseConnection>().DbConnection();
        this.sqlConnection.CreateTable<TipoItemCardapio>();
    }

    public IEnumerable<TipoItemCardapio> GetAll()
    {
        return (from t in sqlConnection.Table<TipoItemCardapio>()
                select t).OrderBy(i => i.Nome).ToList();
    }

    public TipoItemCardapio GetItemById(long id)
    {
        return sqlConnection.Table<TipoItemCardapio>().FirstOrDefault(t => t.TipoItemCardapioId == id);
    }

    public void DeleteById(long id)
    {
        sqlConnection.Delete<TipoItemCardapio>(id);
    }

    public void Add(TipoItemCardapio tipoItemCardapio)
    {
        sqlConnection.Insert(tipoItemCardapio);
    }

    public void Update(TipoItemCardapio tipoItemCardapio)
    {
        sqlConnection.Update(tipoItemCardapio);
    }
}

```

6.4 ADAPTAÇÃO DA INTERFACE COM O USUÁRIO E SEUS COMPORTAMENTOS

Precisamos agora realizar adaptações nas páginas de interação com o usuário para que possamos usar a base de dados. Vamos a elas?

A página de listagem de tipos de itens

A primeira adaptação que precisamos fazer na interface com o usuário é na página que apresenta a listagem de todos os itens registrados. Na primeira execução, você verá que nada aparecerá, o que é óbvio, pois a base de dados foi apenas criada. De início, não mudaremos nada no arquivo XAML.

Na classe, temos duas alterações. A primeira é que agora precisamos instanciar o DAL e não obter uma instância única dele. Isso está garantido na implementação que fizemos. Sendo assim, altere a declaração do DAL para `private TipoItemCardapioDAL dalTipoItemCardapio = new TipoItemCardapioDAL();`.

A segunda mudança que faremos é substituir a chamada ao método `Remove()` do método `OnRemoveClick()` pela `dalTipoItemCardapio.DeleteById((long) item.TipoItemCardapioId)`. Também adaptei como o `ListView` era populado. Fazíamos isso no construtor, pois tínhamos nossa coleção observável, e qualquer mudança nela automaticamente atualizaria o `ListView`. Isso precisou mudar, pois agora recuperamos os dados de uma base de dados.

Bem, para que este tipo já aparecesse ao inserir um novo tipo de item, retornando para a página de listagem, sobrescrevi o método `OnAppearing()`, que é invocado cada vez que a página se torna visível. É um método herdado da classe `Page`. Com isso, retirei a chamada ao método `GetAll()` do DAL do construtor e implementei o método a seguir.

```
protected override void OnAppearing()
{
    base.OnAppearing();
    lvTiposItensCardapio.ItemsSource = dalTipoItemCardapio.GetAll();
}
```

Aqui cabe uma observação muito importante. Em meus projetos, estava com `Xamarin Forms v2.0` e, quando executava a

aplicação na plataforma UWP ou Android, este evento não era capturado, mas funcionava no iOS. Precisei fazer o update do pacote.

Para que você faça isso, clique com o botão direito do mouse no nome de cada projeto e, no `Manage NuGet Packages...`, na guia `Updates`, atualize o `Xamarin Forms` para a versão mais recente. Eu não atualizei os demais componentes, apenas o `Xamarin Forms`.

A inserção de um novo item

Na página de inserção, também houve pouca mudança. Nada a alterar no XAML. Na busca de uma imagem no álbum, mudei o código para que ele fizesse uso de stream e não de arquivo físico, o que permitiu a seleção na plataforma UWP — que apresentei problema no final do capítulo anterior, quando não foi possível selecionar uma imagem do álbum. Veja o código na sequência.

Note que, ao final do método, atribuo o stream como uma matriz de bytes. A variável `bytesFoto`, que será usada na gravação do tipo, é declarada junto com o DAL, como `private byte[] bytesFoto;`.

```
BtnAlbum.Clicked += async (sender, args) =>
{
    await CrossMedia.Current.Initialize();
    if (!CrossMedia.Current.IsPickPhotoSupported)
    {
        DisplayAlert("Álbum não suportado", "Não existe permissão
para acessar o álbum de fotos", "OK");
        return;
    }
    var file = await CrossMedia.Current.PickPhotoAsync();

    if (file == null)
        return;

    var stream = file.GetStream();
    var memoryStream = new MemoryStream();
```

```

        stream.CopyTo(memoryStream);
        fototipoitemcardapio.Source = ImageSource.FromStream(() =>
    {
        var s = file.GetStream();
        file.Dispose();
        return s;
    });
    bytesFoto = memoryStream.ToArray();
});

```

A mesma mudança em relação à nova técnica, que mantém a foto como um array de bytes para que possa ser persistida no banco foi tomada para a captura de uma fotografia pela câmera. Veja o código a seguir.

```

private void RegistraClickBotaoCamera(string idparafoto)
{
    BtnCamera.Clicked += async (sender, args) =>
    {
        await CrossMedia.Current.Initialize();
        if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)
        {
            DisplayAlert("Não existe câmera", "A câmera não está disponível.", "OK");
            return;
        }
        var file = await CrossMedia.Current.TakePhotoAsync(
            new Plugin.Media.Abstractions.StoreCameraMediaOptions
            {
                Directory = FileSystem.Current.LocalStorage.Name,
                Name = "tipoitem_" + idparafoto + ".jpg"
            });
        if (file == null)
            return;

        var stream = file.GetStream();
        var memoryStream = new MemoryStream();
        stream.CopyTo(memoryStream);
        fototipoitemcardapio.Source = ImageSource.FromStream(() =>
    {
        var s = file.GetStream();
        file.Dispose();
        return s;
    });
    bytesFoto = memoryStream.ToArray();
}

```

```
    };
}
```

Para finalizar, precisamos alterar a invocação do método `Add()` do DAL, pois atribuímos um valor para o ID, e ele agora é autoincremento. Ou seja, o SQLite gerará seu valor no momento da inserção. Desta maneira, veja a seguir o código do método que captura o evento clique do botão.

```
public void BtnGravarClick(object sender, EventArgs e)
{
    if (nome.Text.Trim() == string.Empty)
    {
        this.DisplayAlert("Erro",
            "Você precisa informar o nome para o novo tipo de item
do cardápio.",
            "OK");
    }
    else
    {
        dalTiposItensCardapio.Add(new TipoItemCardapio()
        {
            Nome = nome.Text,
            Foto = bytesFoto
        });
        PreparaParaNovoTipoItemCardapio();
    }
}
```

A alteração de um item já cadastrado

Para o processo de alteração de um tipo já cadastrado, não foi necessário alterar nada, nem no XAML, nem na classe. Aqui uma situação que trabalharei na implementação das páginas de Itens de Cardápio. Notou que o XAML da inserção e alteração são praticamente iguais? E que os comportamentos dos controles também são bem semelhantes?

Pois é. Cadê a reutilização? Logo veremos isso. Seja paciente.

A remoção de um item registrado

Quando trabalhamos com coleções, até o capítulo anterior, elas eram de um tipo que possibilitava a atualização do `ListView` quando seus valores sofriam alterações. Quando mudamos para base de dados neste capítulo, este comportamento também mudou, levando-o à necessidade de, após a remoção, atualizarmos novamente o `ListView`.

Desta maneira, após a chamada ao método `DeleteById()`, no evento que captura o clique do botão de remover, insira a instrução

```
lvTiposItensCardapio.ItemsSource =  
dalTipoItemCardapio.GetAll(); .
```

6.5 RECUPERAÇÃO DE IMAGENS DA BASE DE DADOS E EXIBINDO-AS NO LISTVIEW

Na seção anterior, você pode ter visto que armazenar uma imagem em um campo de uma tabela na base de dados não é algo de outro mundo. Precisamos apenas ter um campo na classe que seja um array de bytes. Assim, tanto na inserção como na atualização, o SQLite trata isso de maneira transparente.

Entretanto, para que esta imagem possa ser exibida no `ListView`, não é algo trivial. Lá em nosso `ListView`, ainda temos o elemento `<Image>`, inclusive apontando para um campo que nem faz parte mais de nosso modelo. Precisamos converter o campo BLOB, recuperado para uma propriedade `byte[]`, para um `Stream`, que é um tipo de dado que o controle `<Image>` aceita.

Esta conversão que precisamos fazer é facilmente possível com o Xamarin Forms. Basta escrevermos um `Converter` específico e nos referenciarmos a ele. Ele receberá o valor com o tipo original e retornará para o controle que o utiliza o valor já com o tipo convertido. Vamos lá.

Crie uma pasta chamada `Converters` em seu projeto PCL. Nesta pasta, crie uma classe chamada `ByteToImageSourceConverter`, que estende `IValueConverter`, e a codifique conforme a listagem a seguir. Nela note que o método `Convert()` recebe quatro argumentos, sendo o primeiro que nos interessa, já que é ele que receberá o array de bytes.

Veja no corpo do método que o valor é simplesmente transformado em um `Stream` e carregado como tal, a um `ImageSource`, que, ao final, será o valor atribuído ao controle `<Image>`.

```
using System;
using System.Globalization;
using System.IO;
using Xamarin.Forms;

namespace Modulo1.Converters
{
    public class ByteToImageSourceConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            ImageSource objImageSource;
            if (value != null)
            {
                byte[] byteImageData = (byte[])value;
                objImageSource = ImageSource.FromStream(() => new MemoryStream(byteImageData));
            }
            else
            {
                objImageSource = null;
            }
            return objImageSource;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

}

Agora, com o conversor criado, precisamos consumi-lo, e isso começa com a declaração dele no início do XAML. Insira
 xmlns:conv="clr-
namespace:Modulo1.Converters;assembly=Modulo1" antes da declaração da classe. Agora, antes do elemento
<ContentPage.Content>, insira a implementação a seguir. Veja que <conv> é o nome que demos para o objeto na declaração anterior.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <conv:ByteToImageSourceConverter x:Key="convImage"/>
    </ResourceDictionary>
</ContentPage.Resources>
```

Para finalizar, adapte o Binding do elemento <Image> para {Binding Foto, Converter={StaticResource convImage}} . Vamos testar? As imagens inseridas precisam aparecer agora na listagem.

Bem, com isso, terminamos a adaptação do trabalho com tipos de itens de cardápio para que pudesse ter seus dados persistidos e recuperados de uma base de dados. Mas como fazer para trabalhar com associações? Este é o assunto da próxima seção.

6.6 ASSOCIAÇÕES/RELACIONAMENTOS COM O SQLITE

Como você pôde ver na seção anterior, o trabalho com o SQLite é bem tranquilo, sem nada de especial ou dificultoso para configurar ou realizar as operações básicas. Mas e as associações?

Como referenciamos em nossos projetos, o SQLite por si só não habilita este trabalho. Precisamos de um componente conhecido como `SQLite.Net Extensions`, que tem sua página no Nuget em

<https://www.nuget.org/packages/SQLiteNetExtensions/> e uma ótima referência em <https://bitbucket.org/twincoders/sqlite-net-extensions>.

Vamos instalá-lo em nossos projetos? Siga o mesmo passo apresentado para a instalação do SQLite, clicando com o botão direito do mouse sobre o nome da solução e acessando Manage NuGet Packages for Solution.... Verifique a figura a seguir.

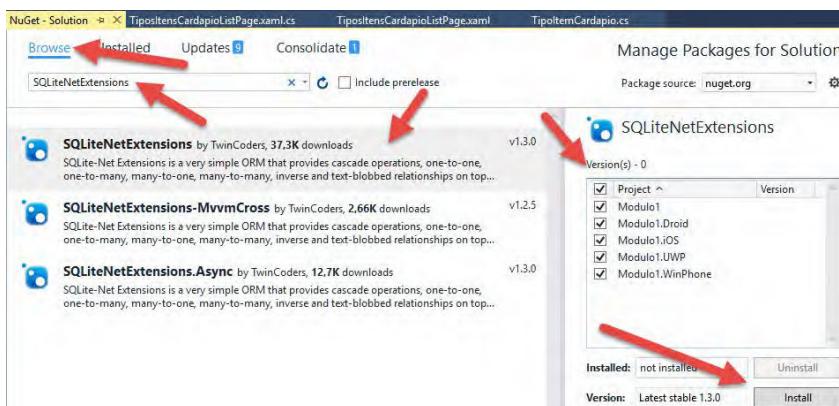


Figura 6.2: Instalação do SQLiteNetExtensions na solução

Criação de uma nova classe e mapeamento das associações

Nesta nova seção, criaremos uma nova classe, a `ItemCardapio`. Cada objeto dela pertencerá a um único objeto de `TipoItemCardapio`, e cada objeto de `TipoItemCardapio` poderá possuir muitos objetos de `ItemCardapio`. Ou seja, é uma associação bidirecional, um-para-muitos e muitos-para-um (*one-to-many* e *many-to-one*).

Veja na sequência o código da nova classe. Observe os atributos `ForeignKey` e `ManyToOne`. Eles comunicam ao SQLite que os dados associados devem ser recuperados. Observe o valor para o

parâmetro `CascadeOperations`. Veja que já implementei também o método `Equals()` e `GetHashCode()`.

```
using SQLite.Net.Attributes;
using SQLiteNetExtensions.Attributes;

namespace Modulo1.Modelo
{
    public class ItemCardapio
    {
        [PrimaryKey, AutoIncrement]
        public long? ItemCardapioId { get; set; }
        public string Nome { get; set; }
        public string Descricao { get; set; }
        public double Preco { get; set; }
        public byte[] Foto { get; set; }

        [ForeignKey(typeof(TipoItemCardapio))]
        public long? TipoItemCardapioId { get; set; }

        [ManyToOne(CascadeOperations = CascadeOperation.CascadeRea
d)]
        public TipoItemCardapio TipoItemCardapio { get; set; }

        public override bool Equals(object obj)
        {
            ItemCardapio itemCardapio = obj as ItemCardapio;
            if (itemCardapio == null)
            {
                return false;
            }

            return (ItemCardapioId.Equals(itemCardapio.ItemCardapi
oId));
        }

        public override int GetHashCode()
        {
            return ItemCardapioId.GetHashCode();
        }
    }
}
```

Para configurar a associação no lado da classe `TipoItemCardapio`, inseri as seguintes instruções ao final da relação das propriedades.

```
[OneToMany]  
public List<ItemCardapio> Itens { get; set; }
```

6.7 A PÁGINA DE LISTAGEM PARA A CLASSE ASSOCIADA

Nesta seção, começaremos a criar nosso `ListView` para os itens do cardápio. Trarei para esta implementação o agrupamento, mas, para que isso fique visível, precisamos de dados. Desta maneira, criaremos da maneira que já sabemos e depois aprimoraremos.

A listagem a seguir traz a implementação da página, em XAML, da listagem de itens de cardápio. Nomeie o arquivo como `ItensCardapioPage` em uma nova pasta, chamada `ItensCardapio`.

Note que não estou fazendo uso agora de uma `TabbedPage`. Faz parte dos novos recursos que veremos. Veja, no início do código, na tag `<ContentPage>` a definição do título da página e a cor de fundo para ela, agora usando hexadecimal.

Para brincar com as cores, dê uma olhada no link:
http://www.w3schools.com/colors/colors_picker.asp.

Note também na definição do `<Grid>` que temos agora uma personalização do `Padding`, de acordo com a plataforma em que a aplicação for executada. Legal isso, não é?

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
             xmlns:conv="clr-namespace:Modulo1.Converters;assembly
```

```

=Modulo1"
        x:Class="Modulo1.Pages.ItensCardapio.ItensCardapioPage"
    <ContentPage.Resources>
        <ResourceDictionary>
            <conv:ByteToImageSourceConverter x:Key="convImage"/>
        </ResourceDictionary>
    </ContentPage.Resources>
<ContentPage.Content>
    <Grid>
        <Grid.Padding>
            <OnPlatform x:TypeArguments="Thickness">
                <OnPlatform.iOS>
                    5, 10, 5, 10
                </OnPlatform.iOS>
                <OnPlatform.WinPhone>
                    5, 0, 5, 35
                </OnPlatform.WinPhone>
                <OnPlatform.Android>
                    5, 10, 5, 10
                </OnPlatform.Android>
            </OnPlatform>
        </Grid.Padding>
        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*"/>
        </Grid.ColumnDefinitions>

        <ListView Grid.Row="0" Grid.Column="0" x:Name="lvItensCardapio" BackgroundColor="#ffffe6">
            <ListView.ItemTemplate BackgroundColor="#ffffe6">
                <DataTemplate>
                    <ViewCell>
                        <Grid Padding="5, 5, 20, 5">
                            <Grid.RowDefinitions>
                                <RowDefinition Height="Auto"/>
                                <RowDefinition Height="*"/>
                            </Grid.RowDefinitions>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="50*"/>
                                <ColumnDefinition Width="*"/>
                                <ColumnDefinition Width="100"/>
                            </Grid.ColumnDefinitions>
                        </Grid>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </Grid>
</ContentPage.Content>

```

```

        <Image Source="{Binding Foto, Converter={StaticRes
ource convImage}}"
            Grid.Row="0" Grid.Column="0" HorizontalOpti
ons="FillAndExpand"
            VerticalOptions="FillAndExpand" Grid.RowSpar
="2" />
        <Label Text="{Binding Nome}" TextColor="Blue" Font
Size="Medium"
            Grid.Row="0" Grid.Column="1" />
        <Label Text="{Binding Preco, StringFormat='{0:C}'}"
            TextColor="Red" HorizontalOptions="End"
            Grid.Row="0" Grid.Column="2"/>
        <Label Text="{Binding Descricao}" TextColor="Green"
FontSize="Small"
            Grid.Row="1" Grid.Column="1" Grid.ColumnSp
an="2"
            HorizontalOptions="FillAndExpand"/>
    </Grid>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
<ListView.GroupHeaderTemplate>
<DataTemplate>
<ViewCell>
    <StackLayout VerticalOptions="FillAndExpand" Horizo
ntalOptions="FillAndExpand"
        BackgroundColor="#ffd11a">
        <Grid VerticalOptions="FillAndExpand" HorizontalOp
tions="FillAndExpand">
            <Label Text="{Binding Key.Nome}" FontSize="Large"
TextColor="#003300"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand"/>
        </Grid>
    </StackLayout>
</ViewCell>
</DataTemplate>
</ListView.GroupHeaderTemplate>
</ListView>
<StackLayout Grid.Row="1" Grid.Column="0" Padding="0" Backgr
oundColor="#ff8c1a">
    <Button Text="Inserir novo item" x:Name="BtnNovoItem" Image
="icone_new.png"/>
</StackLayout>
</Grid>
</ContentPage.Content>

```

</ContentPage>

Para que você possa executar e testar sua aplicação, é preciso criar a opção de acesso a ela na página `MenuPage`. Faça isso e depois execute sua aplicação. A figura a seguir apresenta a nova página.

Destaque para o botão na base da página, seu título e as cores usadas, saindo da cor padrão dos temas dos dispositivos. Observe que, no dispositivo Android, o botão não herda a cor de fundo do `StackLayout`.

Talvez aqui você possa pensar em colocar a cor de fundo no `button`, ou trabalhar com diferenciamento de plataforma, como feito anteriormente. Lembre-se de que, na classe, a extensão precisa ser para `Grid` e não `ContentPage`, que é o padrão. É possível que sua página não esteja formatada semelhante à apresentada na figura a seguir, pois você pode escolher um emulador diferente, mas o importante é que você já sabe como configurar isso. Agora, você já tentou rotacionar seu emulador para a posição de paisagem?

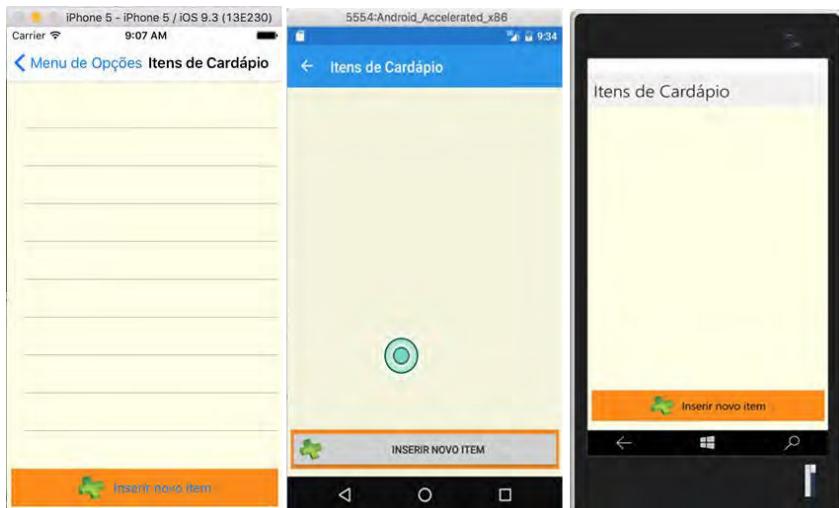


Figura 6.3: Página de listagem de itens de cardápio

6.8 CONTROLES PERSONALIZADOS

Nosso seguinte passo na aplicação é criar a entrada de dados, ou seja, o registro de novos itens de cardápio. Já pensando em uma página de alteração, identificamos que a interface com o usuário, entre a inserção e alteração, é a mesma. Por que repetir então? A ideia é reutilização.

Quando falamos de Orientação a Objetos, poderíamos pensar rapidamente em uma classe que abstraia a interface com o usuário e reutilizá-la. Em XAML, é a mesma coisa. Crie uma pasta chamada `Controls` dentro da pasta `ItensCardapio`, e nela implemente uma nova página, chamada `GridCustomControl` com o seguinte código.

Dê uma lida com calma no código e tente abstrair o seu layout. Tirando o fato de que o primeiro elemento é um `<Grid>` e não um uma página (isso mesmo, estamos especializando um Grid), não há nada de novo. Observe que faço uso novamente de especificidades para a plataforma Windows, agora para a altura de algumas células dos Grids.

```
<?xml version="1.0" encoding="utf-8" ?>
<Grid xmlns="http://xamarin.com/schemas/2014/forms"
      xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

      x:Class="Modulo1.Pages.ItensCardapio.Controls.GridCustomControl">
    <StackLayout VerticalOptions="FillAndExpand" Padding="0">
      <Grid Padding="5,10,5,10">
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto"/>
          <RowDefinition Height="Auto"/>
          <RowDefinition Height="Auto"/>
          <RowDefinition Height="90"/>
          <RowDefinition>
            <RowDefinition.Height>
              <OnPlatform x:TypeArguments="GridLength" WinPhone="140"
                        Android="170" iOS="190"/>
            </RowDefinition.Height>
```

```

        </RowDefinition>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100*"/>
    </Grid.ColumnDefinitions>

    <StackLayout Padding="0" Grid.Row="0" Grid.Column="0">
        <StackLayout HeightRequest="1" BackgroundColor="Black"
                    Padding="0"/>
        <Grid Padding="5,10,5,10">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>
            <Label Text="" x:Name="idTipo" Grid.Row="0" Grid.Column=
"0"
                  WidthRequest="0"/>
            <Label Text="Selecione o Tipo do Item" x:Name="nomeTipo"
                  Grid.Row="0" Grid.Column="1">
                <Label Text=">" HorizontalOptions="End" Grid.Row="0"
                      Grid.Column="2" BackgroundColor="Gray">
                </Label>
            </Grid>
            <StackLayout HeightRequest="1" BackgroundColor="Black"
                        Padding="0"/>
        </StackLayout>

        <StackLayout Padding="0" Grid.Row="1" Grid.Column="0">
            <Entry Placeholder="Nome do Item" PlaceholderColor="Gray"
                  x:Name="nome"/>
            <StackLayout HeightRequest="1" BackgroundColor="Black"
                        Padding="0"/>
        </StackLayout>

        <StackLayout Padding="0" Grid.Row="2" Grid.Column="0">
            <Entry Placeholder="Preço" PlaceholderColor="Gray"
                  x:Name="preco" Keyboard="Numeric"/>
            <StackLayout HeightRequest="1" BackgroundColor="Black"
                        Padding="0"/>
        </StackLayout>
    
```

```

<StackLayout Padding="0" Grid.Row="3" Grid.Column="0"
             VerticalOptions="FillAndExpand">
    <Editor x:Name="descricao" VerticalOptions="FillAndExpand">
        Keyboard="Chat" />
    <StackLayout HeightRequest="1" BackgroundColor="Black"
                 Padding="0"/>
</StackLayout>

<StackLayout Grid.Row="4" Grid.Column="0" Padding="0">
    <Grid Padding="0">
        <Grid.RowDefinitions>
            <RowDefinition>
                <RowDefinition.Height>
                    <OnPlatform x:TypeArguments="GridLength" WinPhone="110" Android="130" iOS="170"/>
                </RowDefinition.Height>
            </RowDefinition>
            <RowDefinition>
                <RowDefinition.Height>
                    <OnPlatform x:TypeArguments="GridLength" WinPhone="30" Android="40" iOS="20"/>
                </RowDefinition.Height>
            </RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100*"/>
        </Grid.ColumnDefinitions>
        <Image x:Name="fotoAlbum" Aspect="AspectFill"
               HorizontalOptions="FillAndExpand"
               VerticalOptions="FillAndExpand" Grid.Row="0"
               Grid.Column="0"/>
        <Button Text="Recuperar foto do álbum"
               Grid.Row="1" Grid.Column="0"/>
    </Grid>
</StackLayout>

<StackLayout Grid.Row="5" Grid.Column="0" Padding="20, 0, 20
, 0"
             BackgroundColor="#009933"
             Orientation="Horizontal" HorizontalOptions="Fill
AndExpand">
    <StackLayout HorizontalOptions="CenterAndExpand"
                 Orientation="Horizontal">
        <Image Source="icone_save.png" HorizontalOptions="Center">
        </Image>
        <Label Text="Gravar Item" VerticalOptions="Center">
    </StackLayout>
</StackLayout>

```

```

        FontAttributes="Bold">
    </Label>
</StackLayout>
</StackLayout>
</Grid>
</StackLayout>
</Grid>

```

Veja, no final desse código, que fazemos uso de uma imagem como ícone para o botão de gravar o item.

6.9 INSERÇÃO COM UM CONTROLE CUSTOMIZADO

Precisamos agora criar uma página na pasta `ItensCardapio`, chamada `ItensCardapioNewPage`. Nela utilizaremos o controle criado anteriormente. Veja o código desta página na sequência.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:controls="clr-namespace:Modulo1.Pages.ItensCardapio.Controls;assembly=Modulo1"
    x:Class="Modulo1.Pages.ItensCardapio.ItensCardapioNewPage"
    Title="Novo Item de Cardápio"
    BackgroundColor="#c2f0c2">
<ContentPage.Content>
    <ScrollView>
        <controls:GridCustomControl x:Name="gridControl"/>
    </ScrollView>
</ContentPage.Content>
</ContentPage>

```

Para testarmos a inserção, precisamos implementar o comportamento para o evento `Click` do botão `BtnNovoItem` da página `ItensCardapioPage`. Para isso, no construtor da classe da página, após a chamada ao método `InitializeComponent()`, insira a instrução `BtnNovoItem.Clicked += BtnNovoItemClick;`. Ela define o método que será executado

quando o botão for clicado e, na sequência, a implementação deste método.

```
private async void BtnNovoItemClick(object sender, EventArgs e)
{
    await Navigation.PushAsync(new ItensCardapioNewPage());
}
```

Agora você pode testar sua aplicação. Clique no botão de inserir novo item. Sua página deve estar semelhante à apresentada na figura a seguir.

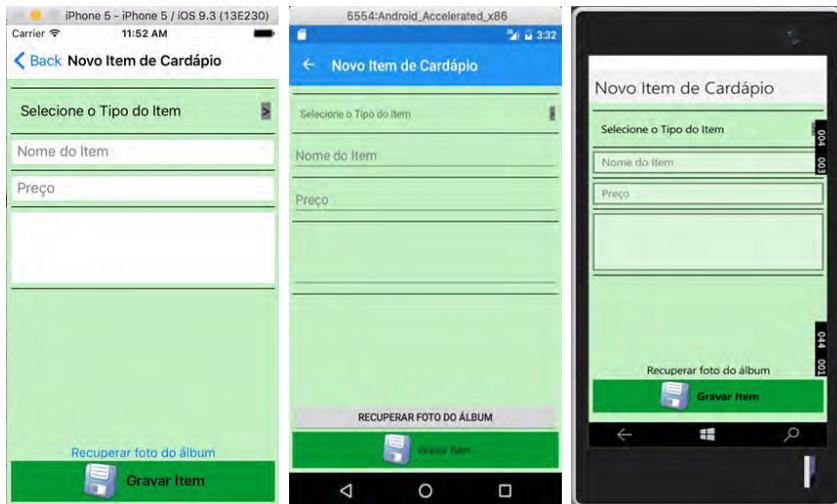


Figura 6.4: Página de inserção de um item de cardápio

6.10 UMA PÁGINA DE PESQUISA

Os dispositivos móveis não trazem nativamente um controle que simule um *combobox* ou *dropdownlist*, controles conhecidos nas plataformas desktop e web. Como fazer para simular isso?

Existe um controle chamado `Picker` que pode ser usado para estas situações. Porém, eu não vou utilizá-lo, pois não é indicado para nosso problema. Quero trazer aqui um novo conceito, que é comum em algumas aplicações mobile.

Vamos criar uma página de busca, com um controle `SearchBar` ligado com um `ListView`. E quando o usuário selecionar um item dos itens exibidos no `ListView`, de acordo com o texto digitado no `SearchBar`, este item será retornado para a página de inserção. Um exemplo de uso do `Picker` pode ser visto em

<https://developer.xamarin.com/api/type/Xamarin.Forms.Picker/>.

Como chamar esta página de pesquisa? Se notar na figura anterior, no topo dela, temos o texto `Selecione o Tipo do Item`, seguido de um `>`. Os dois são `Labels` e não `Buttons`. Como fazer para que, quando o usuário pressione um dos labels, nossa página possa ser exibida?

Os dispositivos móveis trazem um conceito de reconhecedores de gestos (*Gesture Recognizers*), e é com isso que trabalharemos, mais precisamente com o `Tap Gesture`, que é o pressionar com o dedo a tela do dispositivo (no emulador é o clicar do mouse).

Se você tiver interesse em se aprofundar nisso, pode encontrar mais detalhes em
<https://developer.xamarin.com/guides/xamarin-forms/user-interface/gestures/>.

Adapte no XAML da página anterior (que é o controle do Grid) estes labels para ficarem igual ao que apresento na sequência. Observe o número de vezes que os controles precisam ser pressionados (`NumberOfTapsRequired`) para que o evento seja disparado (`Tapped`).

```
<Label Text="Selecione o Tipo do Item" x:Name="nomeTipo"
       Grid.Row="0" Grid.Column="1">
    <Label.GestureRecognizers>
```

```

<TapGestureRecognizer
    Tapped="OnTapLookForTipos"
    NumberOfTapsRequired="1" />
</Label.GestureRecognizers>
</Label>
<Label Text=">" HorizontalOptions="End" Grid.Row="0"
    Grid.Column="2" BackgroundColor="Gray">
<Label.GestureRecognizers>
<TapGestureRecognizer
    Tapped="OnTapLookForTipos"
    NumberOfTapsRequired="1" />
</Label.GestureRecognizers>
</Label>

```

Agora, para que a página de pesquisa possa ser acessada, vamos implementar na classe de nosso Grid customizado o método `OnTapLookForTipos`, que está apontado no código XAML anterior. Veja na sequência este método.

Nele note que a chamada ao construtor da página é composta de dois argumentos, dois labels, responsáveis por armazenar o `id` e `valor` do tipo de item de cardápio selecionado. Estes objetos serão recebidos pelo construtor e atualizados de acordo com a seleção do usuário.

Em relação ao `id` do tipo, se você retornar à listagem da página `GridCustomControl`, verá que ele existe lá, com comprimento zero: `WidthRequest="0"`. Ou seja, será como um controle oculto que será usado quando o item for inserido ou atualizado.

```

private async void OnTapLookForTipos(object sender, EventArgs args)
{
    await Navigation.PushAsync(new TiposDeItensParaItensPage(idTipo,
        nomeTipo));
}

```

Vamos à criação da página. Veja na sequência o seu código. Eu a nomeei de `TiposDeItensCardapioSearchPage`, e a criei na pasta `TiposItensCardapio`.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.TiposItensCardapio.TiposItensCardapioSearchPage"
        Title="Seleção de tipo de item">
    <StackLayout Orientation="Vertical" HorizontalOptions="FillAndExpand" Padding="5,20,5,0">
        <SearchBar Placeholder="Digite o nome do tipo ..." TextColor="Black"/>
        <ListView x:Name="lvTipos" HasUnevenRows="True">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout Orientation="Vertical">
                            <Label Text="{Binding TipoItemCardapioId}" TextColor="Blue" FontSize="0"/>
                            <Label Text="{Binding Nome}" TextColor="Blue" FontSize="Large"/>
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>

```

Resta agora implementar a classe da página do XAML, que está na sequência. Veja que já está previsto o DAL para obtenção dos dados e uma coleção para os itens que serão recuperados. O construtor invoca o método `GetAll()` do DAL e atribui o resultado à coleção `itens`, que será onde aplicaremos a pesquisa.

Poderíamos pensar em fazer a busca direto na base de dados, criando um método DAL para isso, mas precisamos pensar no processamento e também na situação. Haverá alteração constante nos dados para que se justifique uma pesquisa constante na tabela? Acho que não.

Os valores recebidos pelo construtor são armazenados nas variáveis da classe. Você poderia optar, no construtor, em atribuir ao `ListView` todos os itens. Não vejo problema nisso. Apenas

optei, em meu caso, em não mostrar nada de início.

```
using Modulo1.Dal;
using Modulo1.Modelo;
using System.Collections.Generic;
using Xamarin.Forms;

namespace Modulo1.Pages.TiposItensCardapio
{
    public partial class TiposItensCardapioSearchPage : ContentPage
    {
        private TipoItemCardapioDAL dalTipoItemCardapio = new Tipo
ItemCardapioDAL();
        private Label displayValue;
        private Label keyValue;
        private IEnumerable<TipoItemCardapio> itens;

        public TiposItensCardapioSearchPage(Label keyValue, Label
displayValue)
        {
            InitializeComponent();
            itens = dalTipoItemCardapio.GetAll();
            this.keyValue = keyValue;
            this.displayValue = displayValue;
        }
    }
}
```

Com as implementações anteriores realizadas, já podemos testar nossa aplicação e verificar a página de pesquisa em execução. Veja a página na figura a seguir.

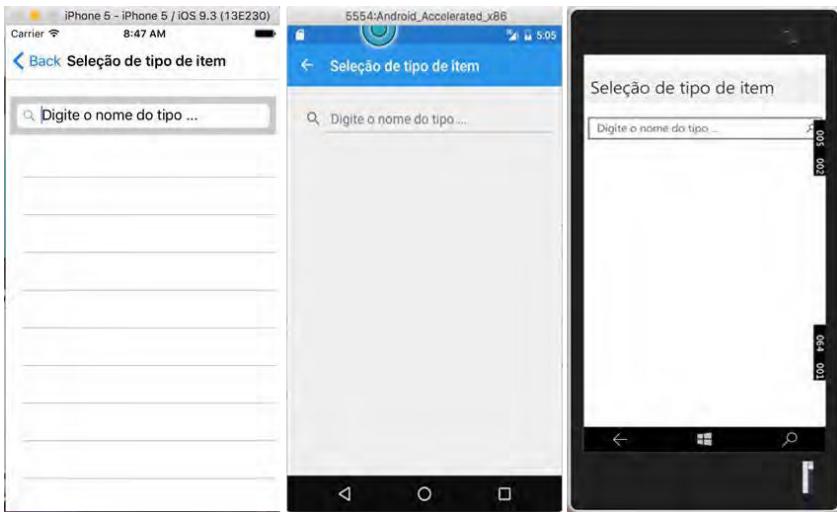


Figura 6.5: Página de pesquisa de tipos de itens de cardápio

Precisamos agora implementar o comportamento para a página. O primeiro é declarar no controle `SearchBar` qual é o método que será disparado quando for digitado algo em sua caixa de texto. Podemos fazer isso adicionando o código `TextChanged="OnTextChanged"` na declaração do controle. A implementação para este método está na sequência.

Observe que a lógica está entre a chamada aos métodos `BeginRefresh()` e `EndRefresh()`, para que a aplicação saiba que o `ListView` está em estado de atualização. No caso deste método, optei em exibir todos os tipos caso o usuário apague o que escreveu no `SearchBar`. O que acha de testar sua aplicação e ver se ela está realizando a pesquisa?

```
private void OnTextChanged(object sender, TextChangedEventArgs e)
{
    lvTipos.BeginRefresh();
    if (string.IsNullOrEmpty(e.NewTextValue))
        lvTipos.ItemsSource = itens;
    else
        lvTipos.ItemsSource = itens.Where(i => i.Nome.Contains(e.NewTextValue));
```

```
    lvTipos.EndRefresh();
}
```

Para concluir o processo de pesquisa, e para que o tipo encontrado e selecionado retorne para a página de inserção, precisamos alterar a declaração do `ListView`, para que tenha `ItemTapped="OnItemTapped"`. Agora, novamente na classe do Grid customizado, precisamos implementar este método. Ele será disparado quando o usuário pressionar a linha com o tipo desejado. Veja-o na sequência.

Observe que capturamos o item que está ligado na linha selecionada e obtemos valores dele para atribuir àqueles parâmetros que foram recebidos no construtor da página. Também retornamos para a página que foi a chamadora da página de pesquisa. Vamos implementar este método e testar?

```
public async void OnItemTapped(object o, ItemTappedEventArgs e)
{
    var item = (o as ListView).SelectedItem as TipoItemCardapio;
    this.keyValue.Text = item.TipoItemCardapioId.ToString();
    this.displayValue.Text = item.Nome;
    await Navigation.PopAsync();
}
```

6.11 FINALIZANDO A INSERÇÃO DO ITEM DE CARDÁPIO

Agora que terminamos o processo de pesquisa, precisamos trabalhar nas demais interações possíveis da página de inserção (que é nosso custom Grid). O passo seguinte é implementar a captura da imagem do álbum, que já fizemos em outra ocasião, mas não com a foto vinda da base de dados como um array de bytes.

Precisamos mudar a nossa tag do botão que abrirá o álbum de fotografias. Para isso, adicione `Clicked="OnAlbumClicked"` ao elemento. Em seguida, adicione à classe, antes do construtor, a

instrução `private byte[] bytesFoto;`, para que tenhamos a imagem selecionada no momento de gravar.

Agora, para finalizar, implemente o método responsável pela abertura do álbum e recuperação da foto. Ele está na sequência.

A primeira mudança está na chamada ao `DisplayAlert()`, pois estamos em uma classe `Grid` e este método pertence à `Page`, então recuperamos a página principal da aplicação. Depois, copiamos o `Stream` recuperado para um `MemoryStream`, para que possamos invocar, ao final, o método `ToArray()` e atribuir seu resultado para a variável `bytesFoto`.

Vamos verificar se está funcionando? Teste sua aplicação e clique no botão de abertura do álbum.

```
public async void OnAlbumClicked(object sender, EventArgs e)
{
    await CrossMedia.Current.Initialize();

    if (!CrossMedia.Current.IsPickPhotoSupported)
    {
        await App.Current.MainPage.DisplayAlert("Álbum não suporta
do", "Não existe permissão para acessar o álbum de fotos", "OK");
        return;
    }

    var file = await CrossMedia.Current.PickPhotoAsync();
    if (file == null)
        return;
    var stream = file.GetStream();
    var memoryStream = new MemoryStream();
    stream.CopyTo(memoryStream);
    fotoAlbum.Source = ImageSource.FromStream(() =>
    {
        var s = file.GetStream();
        file.Dispose();
        return s;
    });
    bytesFoto = memoryStream.ToArray();
}
```

A última interação disponibilizada para o usuário nesta página é

ainda não implementada é a de gravação do item de cardápio. Se você verificar no código do Custom Grid, verá que a imagem que aparece como botão é, na realidade, uma composição entre um `<Image>` e um `<Label>`.

Como fazer para que seja uma área de interação? Usando `Gesture` novamente. Desta maneira, procure aplicar isso em sua página e compare as tags com o que apresento na sequência. Fez igual? Parabéns!

```
<Image Source="icone_save.png" HorizontalOptions="Center">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
            Tapped="OnTappedSaveItem"
            NumberOfTapsRequired="1" />
    </Image.GestureRecognizers>
</Image>
<Label Text="Gravar Item" VerticalOptions="Center"
    FontAttributes="Bold">
    <Label.GestureRecognizers>
        <TapGestureRecognizer
            Tapped="OnTappedSaveItem"
            NumberOfTapsRequired="1" />
    </Label.GestureRecognizers>
</Label>
```

Antes de completarmos a inserção de um item, o que faremos quando for clicado em um dos controles? Salvaremos na base de dados, certo? Na tabela de Itens de Cardápio. Mas como faremos isso? Ainda não temos o DAL para este modelo. Então, vamos implementá-lo.

Crie-o na pasta `Dal` com o nome `ItemCardapioDAL`. Veja o código na sequência. Eu já aproveitei e, além do método `Add()`, adicionei outros que usaremos no futuro e que são semanticamente compreensíveis.

```
using Modulo1.Infraestructure;
using Modulo1.Modelo;
using SQLite.Net;
using SQLiteNetExtensions.Extensions;
```

```

using System.Collections.Generic;
using Xamarin.Forms;
using System.Linq;

namespace Modulo1.Dal
{
    public class ItemCardapioDAL
    {
        private SQLiteConnection sqlConnection;

        public ItemCardapioDAL()
        {
            this.sqlConnection = DependencyService.Get<IDatabaseConnection>().DbConnection();
            this.sqlConnection.CreateTable<ItemCardapio>();
        }

        public void Add(ItemCardapio itemCardapio)
        {
            sqlConnection.Insert(itemCardapio);
        }

        public void DeleteById(long id)
        {
            sqlConnection.Delete<ItemCardapio>(id);
        }

        public IEnumerable<ItemCardapio> GetAllWithChildren()
        {
            return sqlConnection.GetAllWithChildren<ItemCardapio>()
                .OrderBy(i => i.Nome).ToList();
        }

        public ItemCardapio GetItemById(long id)
        {
            return sqlConnection.GetAllWithChildren<ItemCardapio>()
                .FirstOrDefault(i => i.ItemCardapioId == id);
        }
    }
}

```

Para completar o processo de inserção, precisamos implementar o método que será invocado quando ocorrer o pressionamento em um dos controles. Este código está na sequência.

Observe que instanciamos o DAL, e invocamos o método

`Add()` com um objeto que é criado na chamada ao método. Consta do código a invocação e implementação de um método específico para inicialização dos controles, para que um novo item possa ser inserido na sequência.

Não foi feita nenhuma validação nos dados. Isso fica por sua conta. Já vimos como fazer em implementações anteriores.

```
private async void OnTappedSaveItem(object sender, EventArgs args)
{
    var dal = new ItemCardapioDAL();
    dal.Add(new ItemCardapio() {
        Nome = nome.Text,
        Descricao = descricao.Text,
        Preco = Convert.ToDouble(preco.Text),
        TipoItemCardapioId = Convert.ToInt32(idTipo.Text),
        Foto = bytesFoto
    });
    await App.Current.MainPage.DisplayAlert("Inserção de item", "Item inserido com sucesso", "Ok");

    ClearControls();
}

private void ClearControls()
{
    nome.Text = string.Empty;
    descricao.Text = string.Empty;
    preco.Text = string.Empty;
    bytesFoto = null;
    idTipo.Text = string.Empty;
    nomeTipo.Text = "Selecione o Tipo do Item";
    fotoAlbum.Source = null;
}
```

6.12 EXIBIÇÃO DOS ITENS (ASSOCIAÇÃO) NO LISTVIEW

Realizamos um grande processo de implementação para podermos inserir um item de cardápio, mas valeu a pena. Vimos muita coisa interessante. Precisamos agora exibir os itens de cardápio que são inseridos no `ListView`, pois, até o momento,

nada é exibido nele.

Faremos algo diferente e veremos um novo recurso do `ListView`. Trabalharemos com agrupamento. Isso quer dizer que os itens de cardápio serão separados por seus tipos. Isso dá um efeito visual interessante, e ainda é possível gerar uma listagem de atalho para os grupos.

Para fazer isso, é preciso saber e entender que a fonte de dados precisa ser diferente. Ela precisa aplicar o conceito de associação, no qual teremos uma coleção de tipos e, dentro de cada tipo, uma coleção de itens. Para isso, implementaremos uma classe que abstrai esta associação. Ela está no código a seguir.

Criei uma pasta chamada `HelperControls` e nomeei a classe como `ListViewGrouping`. Veja, no código, que cada objeto possuirá uma chave (`Key`) e, como a classe herda de `Collection`, ela possui `Items`, que receberá os itens recebidos pelo construtor.

```
using System.Collections.Generic;
using System.Collections.ObjectModel;

namespace Modulo1.HelpersControls
{
    public class ListViewGrouping<K, T> : Collection<T>
    {
        public K Key { get; private set; }

        public ListViewGrouping(K key, IEnumerable<T> items)
        {
            Key = key;
            foreach (var item in items)
                this.Items.Add(item);
        }
    }
}
```

O segundo passo é configurar o `ListView` para trabalhar com agrupamentos. Desta maneira, adapte a declaração dele na página `ItensCardapioPage` para o código a seguir. Note a propriedade

`IsGroupingEnabled` habilitando o agrupamento, a `HasUnevenRows` informando que nem todas as linhas do `ListView` terão alturas iguais, e as de ligação (*Binding*), para o nome do grupo e do nome curto, para a área de atalho.

```
<ListView Grid.Row="0" Grid.Column="0" x:Name="lvItensCardapio" Ba  
ckgroundColor="#fffffe6"  
    HasUnevenRows="True" GroupDisplayBinding="{Binding Key.Nom  
e}"  
    GroupShortNameBinding="{Binding Key.TipoItemCardapioId}"  
    IsGroupingEnabled="true">
```

Precisamos agora recuperar os dados para serem exibidos no `ListView`. Nossa DAL possui, por enquanto, apenas o método `Add()`. Precisamos criar um novo método que nos retorne os itens. Entretanto, se você verificar atentamente o que falei sobre agrupamento anteriormente, precisamos recuperar os tipos de itens, e não os itens, pois queremos os itens agrupados por tipo.

Lembra-se de quando criamos a associação no início do capítulo? A classe `TipoItemCardapio` tem uma coleção de `ItemCardapio`. Veja na sequência a implementação do novo método na classe `TipoItemCardapioDAL()`. Veja o nome do método e qual o método que é invocado em nossa coleção. Isso faz parte do `SQLiteNetExtensions`.

Este método, como o próprio nome diz, recupera os objetos da classe referenciada e todos os objetos de todas as associações existentes. Fica mais prático usarmos este do que o `GetAll()` que já temos.

Se utilizássemos o `GetAll()`, precisaríamos criar um método em `ItemCardapioDAL` para retornar os itens de um determinado tipo. Teríamos diversas chamadas ao SQLite, e com essa nova implementação, temos apenas uma. É claro que ela faz um `join`, mas é tudo em uma única chamada.

```
public IEnumerable<TipoItemCardapio> GetAllWithChildren()
```

```
{  
    return sqlConnection.GetAllWithChildren<TipoItemCardapio>().Or  
derBy(i => i.Nome).ToList();  
}
```

Com o método que recupera os itens que precisamos (ainda não agrupados), precisamos agora obter estes dados na página, agrupá-los e atribuí-los ao `ListView`. Primeiro, insira, antes do construtor, a instrução `private TipoItemCardapioDAL dalTipoItemCardapio = new TipoItemCardapioDAL();`, para que possamos invocar o método criado anteriormente.

Veja na sequência a implementação do método que invoca o DAL e prepara os dados da maneira necessária para o agrupamento. Observe no código a criação da coleção, obtenção dos dados e uma varredura neles, para que sejam adicionados na coleção de acordo com a necessidade requerida pelo agrupamento do `ListView`.

```
private Collection<ListViewGrouping<TipoItemCardapio, ItemCardapio  
{  
    var dadosAgrupados = new Collection<ListViewGrouping<TipoItemC  
ardapio, ItemCardapio>>();  
    var tipos = dalTipoItemCardapio.GetAllWithChildren();  
    foreach (var tipo in tipos)  
    {  
        dadosAgrupados.Add(new ListViewGrouping<TipoItemCardapio,  
ItemCardapio>(tipo, tipo.Itens));  
    }  
    return dadosAgrupados;  
}
```

Para fecharmos, precisamos atribuir ao `ListView` a coleção retornada pelo método anterior. Seguiremos um exemplo anteriormente adotado, que é fazer isso no evento `OnAppearing`.

Note que, para este `ListView`, optei em ter os dados sempre obtidos da base de dados, pois o usuário poderá inserir um novo item e retornar para o `ListView`, querendo ver este item lá. Veja o código a seguir e o implemente. Vamos testar sua aplicação?

```
protected override void OnAppearing()
{
    base.OnAppearing();
    lvItensCardapio.ItemsSource = GetDataByGroup();
}
```

Na figura adiante, você tem a representação da página de listagem dos itens de cardápio, agrupados por tipos. Insira novos itens, em diversas categorias, e veja o resultado. Alguns detalhes são interessantes e os comento.

Quando fui testar a aplicação nos emuladores para Android e Windows, foi me retornado um erro de inexistência da tabela `ItemCardapio`. Isso é algo meio que óbvio, pois, ao chamar os itens para popular o `ListView`, ele trará os itens de cardápio juntos. E como eu havia testado a inserção de itens apenas no iOS, esta tabela não foi criada.

Para resolver o problema, adicionei a instrução `private ItemCardapioDAL dalItemCardapio = new ItemCardapioDAL();` antes do construtor. O menu de atalho, que vemos no iOS, para ver no Windows é preciso clicar no título de um grupo. E, infelizmente, no Android, este recurso não está disponível.

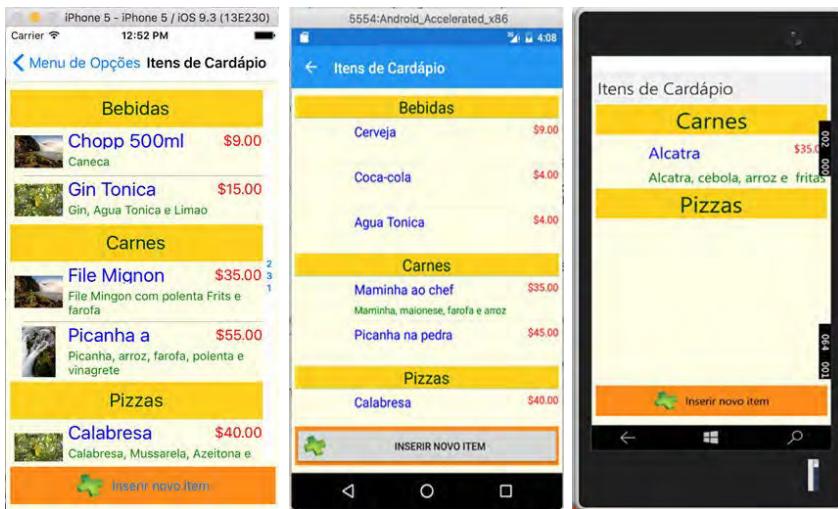


Figura 6.6: Página de pesquisa de tipos de itens de cardápio

6.13 A ALTERAÇÃO DE UM ITEM DE CARDÁPIO JÁ PERSISTIDO.

A interface que será utilizada para a alteração de um item já cadastrado é a mesma que usamos para a inserção, por isso criamos um controle customizado. Entretanto, o comportamento para este controle precisará de alguma adaptação, pois na inserção ele não exibia nenhum dado nos controles, mas na alteração, ele deverá exibir os dados do item desejado.

Vamos começar criando a página de edição. Faça isso na pasta `ItensCardapio` e dê o nome de `ItensCardapioEditPage`. Implemente nela o código a seguir. Veja como ele é semelhante ao que implementamos na página de inserção.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:controls="clr-namespace:Modulo1.Pages.ItensCardapio.Controls;assembly=Modulo1"
    x:Class="Modulo1.Pages.ItensCardapio.ItensCardapioEdi
```

```

tPage"
    Title="Alterando Item de Cardápio"
    BackgroundColor="#c2f0c2">
<ContentPage.Content>
    <ScrollView>
        <controls:GridCustomControl x:Name="gridControl"/>
    </ScrollView>
</ContentPage.Content>
</ContentPage>

```

Precisamos agora invocar esta página. Para fazer isso, faremos uso dos menus de Action Contexts, que vimos no capítulo anterior. Para que isso possa ocorrer, adicione o código seguinte logo abaixo da tag `<ViewCell>` do `ListView` da página `ItensCardapioPage`. Veja que já implementamos também a Action responsável pela exclusão de um item de cardápio.

```

<ViewCell.ContextActions>
    <MenuItem Clicked="OnAlterarClick" CommandParameter="{Binding .}"
    Text="Alterar" />
    <MenuItem Clicked="OnRemoverClick" CommandParameter="{Binding .}"
    Text="Remover" IsDestructive="True" />
</ViewCell.ContextActions>

```

Para que possamos testar nossa aplicação e ver se os Context Actions aparecem, precisamos implementar os dois métodos nomeados e indicados no código anterior. Faça isso, codificando-os de acordo com o apresentado na sequência.

Se você for compará-los com os métodos que implementamos na listagem de tipos de itens de cardápio, verá que são semelhantes, pois a lógica é a mesma. Na chamada ao construtor da classe `ItensCardapioEditPage`, o item que foi selecionado pelo usuário é enviado, pois precisaremos dele para popular os controles visuais. Adapte seu construtor para receber um `ItemCardapio`.

Veja o comentário na chamada ao método `DeleteById()`. Mantenha-o assim, pois logo o implementaremos. A princípio, teste

sua aplicação e veja se ela funciona até aqui. Depois concluirmos esta implementação.

```
public async void OnAlterarClick(object sender, EventArgs e)
{
    var mi = ((MenuItem)sender);
    var item = mi.CommandParameter as ItemCardapio;
    await Navigation.PushAsync(new ItensCardapioEditPage(item));
}

public async void OnRemoverClick(object sender, EventArgs e)
{
    var mi = ((MenuItem)sender);
    var item = mi.CommandParameter as ItemCardapio;
    var opcao = await DisplayAlert("Confirmação de exclusão",
        "Confirma excluir o item " + item.Nome.ToUpper() + "?", "Sim",
        "Não");
    if (opcao)
    {
        //dalItemCardapio.DeleteById((long)item.ItemCardapioId);
        this.lvItensCardapio.ItemsSource = GetDataByGroup();
    }
}
```

Espero que você tenha conseguido executar sua aplicação e chegar até a página de alteração. Se não conseguiu, reveja seu código.

Agora, precisamos popular os controles com os dados recebidos pelo construtor. Mas, se você lembrar, nossos controles não estão na página de alteração, pois eles fazem parte do Custom Grid que fizemos. Precisamos resolver esta situação na classe do Grid, a `GridCustomGrid`.

O primeiro passo é criarmos uma variável na classe que receba o item que será alterado. Isso pode ser feito com a instrução `private ItemCardapio itemCardapio = null;` antes do construtor. Em seguida, precisamos criar o método que será responsável por popular os controles visuais, e ele está na sequência.

Após implementarmos este método no construtor da classe de

alteração, é preciso invocá-lo. Insira no construtor, como última instrução, o código

gridControl.PopularControles(itemCardapio); . . Teste novamente sua aplicação e verifique se os controles agora estão populados.

```
public void PopularControles(ItemCardapio itemCardapio)
{
    this.itemCardapio = itemCardapio;
    nome.Text = this.itemCardapio.Nome;
    descricao.Text = this.itemCardapio.Descricao;
    preco.Text = this.itemCardapio.Preco.ToString("N");
    if (this.itemCardapio.Foto != null)
    {
        fotoAlbum.Source = ImageSource.FromStream(() => new MemoryStream(this.itemCardapio.Foto));
        bytesFoto = this.itemCardapio.Foto;
    }
    nomeTipo.Text = this.itemCardapio.TipoItemCardapio.Nome;
    idTipo.Text = this.itemCardapio.TipoItemCardapioId.ToString();
}
```

Precisamos agora adaptar nosso processo de gravação do item, pois ele está adicionando o item à base de dados. Com esta implementação, nosso Grid customizado deverá, além de inserir, atualizar um item, quando estiver na página de alteração. Para isso, precisamos, antes de tudo, implementar o método `Update()` na classe `ItemCardapioDAL`, como se segue.

```
public void Update(ItemCardapio itemCardapio)
{
    sqlConnection.Update(itemCardapio);
}
```

Agora, para finalizar este processo, adaptamos o método que captura o gesto do Label e imagem de gravar para ser semelhante ao que apresento na sequência. Veja que já são previstas as operações de inserir e atualizar.

Observe que, seguindo a Orientação a Objetos, deveríamos refatorar este método para outro método, com responsabilidade de

preparar o objeto para ser utilizado. Teste agora alterar um item de cadastro e veja se funciona.

```
private async void OnTappedSaveItem(object sender, EventArgs args)
{
    var dal = new ItemCardapioDAL();
    if (this.itemCardapio == null)
    {
        this.itemCardapio = new ItemCardapio();
    }
    this.itemCardapio.Nome = nome.Text;
    this.itemCardapio.Descricao = descricao.Text;
    this.itemCardapio.Preco = Convert.ToDouble(preco.Text);
    this.itemCardapio.TipoItemCardapioId = Convert.ToInt32(idTipo.
Text);
    this.itemCardapio.Foto = bytesFoto;

    if (this.itemCardapio.ItemCardapioId == null)
    {
        dal.Add(this.itemCardapio);
        await App.Current.MainPage.DisplayAlert("Inserção de item"
, "Item inserido com sucesso", "Ok");
    }
    else
    {
        dal.Update(this.itemCardapio);
        await App.Current.MainPage.DisplayAlert("Atualização de it
em", "Item atualizado com sucesso", "Ok");
    }

    ClearControls();
}
```

Precisamos agora finalizar nosso CRUD, e faremos isso implementando o método `DeleteById()` na classe `ItemCardapioDAL`. Ele é simples; veja-o na sequência.

Antes de testar sua aplicação, remova o comentário que temos na chamada a este método na classe `ItensCardapioPage`, no método que captura o gesto de pressionar o Label e imagem.

```
public void DeleteById(long id)
{
    sqlConnection.Delete<ItemCardapio>(id);
}
```

6.14 MANIPULAÇÃO DA BASE DE DADOS DO SQLITE

Muitas vezes se faz necessário acessar a base de dados que nosso dispositivo manipula, neste caso, nossos emuladores. Existem diversas ferramentas para isso, mas eu utilizo o Db Browser for SQLite . Você pode obtê-lo em: <http://sqlitebrowser.org>. A instalação é bem simples, então não vou comentá-la.

Para acessar a base de dados criada para os emuladores, recomendo que você obtenha o caminho por meio de depuração, inserindo um break-point na classe que criamos como Dependency Service em cada projeto e verificando o valor do caminho. Veja isso na figura a seguir.

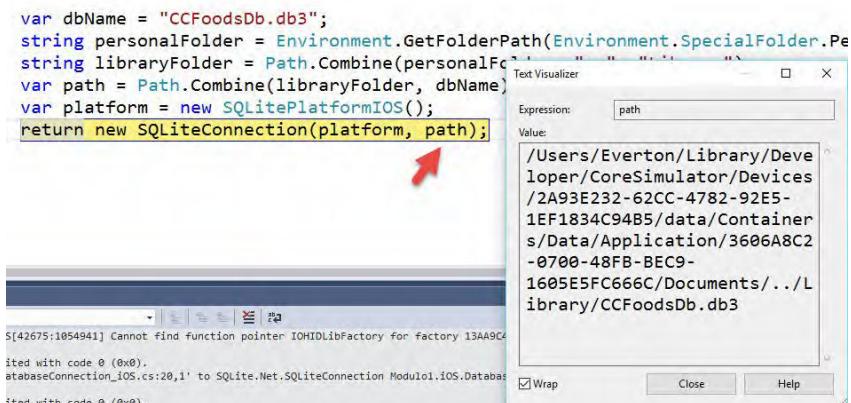


Figura 6.7: Break-point para obter o valor da variável com o caminho da base de dados

Com este caminho em mãos, precisamos agora abrir o arquivo no DB Browser for SQLite . Clique em Abrir banco de dados no topo da janela, ou clique no menu Arquivo e selecione esta opção.

A figura a seguir apresenta o aplicativo com a base de dados aberta. Procure navegar pelas janelas e investigar seu uso. É uma

ferramenta simples, útil e de fácil uso.

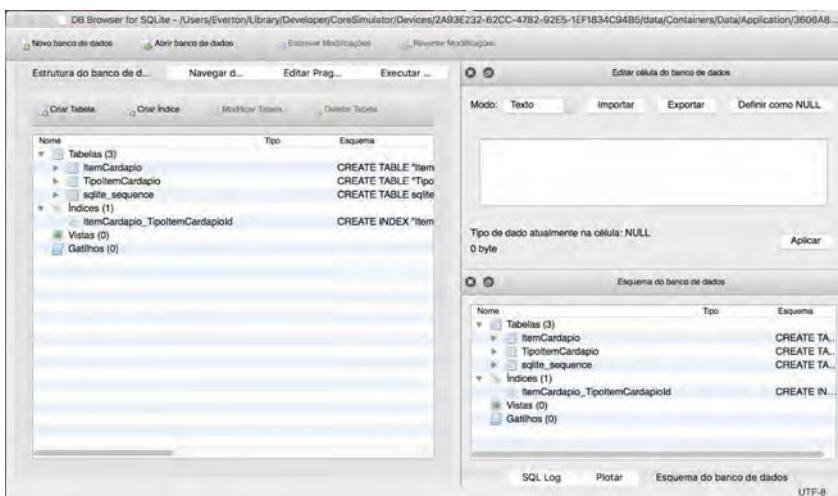


Figura 6.8: DB Browser for SQLite com a base de dados aberta

6.15 CONCLUSÃO

Chegamos ao fim deste importante capítulo. Nele você pôde fazer uso de uma base de dados para persistir e recuperar seus objetos. Agora não precisamos mais trabalhar com coleções. Você pôde também, além de implementar um CRUD em uma simples tabela, trabalhar com tabelas relacionadas, ou seja, implementamos uma associação entre classes.

Para estas implementações, fizemos uso do SQLite e suas extensões em todos os projetos. Vimos como implementar uma Dependency Service nos projetos específicos das plataformas, e consumi-las no projeto PCL para o uso do SQLite e seus plugins para Xamarin Forms.

Apresentei também a criação de um controle customizado, que permitiu a reutilização em nossa aplicação. Implementamos uma página de pesquisa, fazendo uso de um novo controle, o

`SearchBar`. Vimos ainda um recurso importante do Xamarin Forms, que são os conversores de tipos de dados. E finalizamos com a utilização de um novo recurso para o `ListView`, o agrupamento de dados.

Foi um extenso capítulo, com conceitos novos, novas ferramentas e novas técnicas. Acho que agora você merece uma trégua. Um momento de relaxamento. No próximo capítulo, trabalharemos com REST para obtenção e atualização de dados com um servidor de aplicação na web. Até lá.

CAPÍTULO 7

SÍNCRONISMO COM SERVIÇOS REST WEB API

Com o trabalho realizado no capítulo anterior, nossa aplicação já persiste dados em uma base de dados, tornando-os permanente. Como trabalhamos com dispositivos móveis e nossa aplicação certamente terá mais do que um dispositivo a utilizando, precisamos pensar na possibilidade real de que os dados persistidos devam ser compartilhados entre dispositivos.

O sincronismo de dados não é uma tarefa trivial, é muito complexo. Imagine a situação de alteração de um dado de um registro em um dispositivo, alteração de outro dado do mesmo registro em outro dispositivo, a inserção de um registro em dispositivos diferentes, exclusão deles. Enfim, é algo complexo.

O objetivo deste capítulo é apresentar como acessar um servidor web, que disponha da base de dados da aplicação (com dados de todos os dispositivos) e como atualizar, nesta base de dados, os registros persistidos localmente, nos dispositivos. Para realizarmos esta atividade, faremos uso do Azure da Microsoft, para hospedar nossa aplicação central, base de dados e serviços web.

Uma solução gratuita pode ser o App Harbor (<https://appharbor.com/>) ou o Amazon Web Services (<https://aws.amazon.com/pt/>). Mas, por opção pessoal, adotei o Azure.

Já vi situações em que a aplicação oferece um cliente web para o usuário, além do cliente para os dispositivos. A ideia é os dados estarem na nuvem. Vale a pena pensar nisso.

Apesar de usarmos novas tecnologias neste capítulo, o foco não é detalhar como cada uma delas funciona. Caso você tenha o interesse em se aprofundar em Azure, SQL Server (que será a base de dados no servidor), REST e Web API, existem diversos livros sobre estes temas e recomendo uma pesquisa sobre isso. No momento de uso destas tecnologias, apontarei alguns sites e livros que possam auxiliá-lo neste estudo.

Em relação a um sincronismo, existem ferramentas próprias para isso e que certamente você acabará adotando uma delas, caso seu projeto assim necessite. Alguns exemplos são: Zumero (<http://zumero.com/>) e Azure Mobile Services, que possui um pacote Nuget a ser instalado em sua aplicação.

Para uma breve descrição e introdução sobre o assunto, acesse:

- <http://blog.alectucker.com/post/2014/08/01/azure-mobile-services-with-xamarin-forms.aspx>
- <https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-xamarin-forms-get-started-offline-data/>
- <https://developer.xamarin.com/guides/xamarin-forms/web-services/sync/azure-mobile-apps/>

Vale a pena uma leitura, talvez até antes de continuar neste capítulo.

7.1 A APLICAÇÃO QUE SERÁ A SERVIDORA NA WEB E SEU MODELO DE NEGÓCIO

Para criarmos nossa aplicação servidora, faremos uso dos

templates do Visual Studio. Normalmente, gosto de criar a aplicação do zero e adicionar nela os recursos necessários. Entretanto, trabalhar com uma aplicação Web e Web API não é o foco deste livro.

Se você estiver interessado em começar a desenvolver aplicações ASP.NET MVC, recomendo meu outro livro, *ASP.NET MVC5: Crie aplicações web na plataforma Microsoft®* (<https://www.casadocodigo.com.br/products/livro-aspnet-mvc5>), e para Web API, recomendo o *Web Services REST com ASP .NET Web API e Windows Azure*, do Paulo Siécola (<https://www.casadocodigo.com.br/products/livro-web-services-rest>).

Com o Visual Studio ativo, crie um novo projeto. Na primeira janela que se abre, escolha a categoria `Web` , informe o caminho onde gravará a solução e o nome para ela. Confirme as opções e, na janela que se abre, faça as configurações tal qual apresento na figura a seguir. Clique no botão OK e aguarde a criação de sua aplicação.

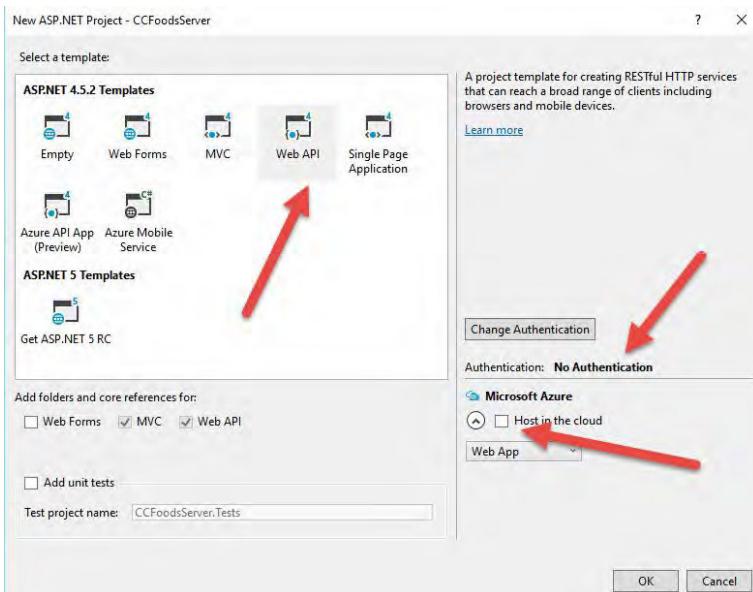


Figura 7.1: Criando a aplicação Web API

Com a aplicação criada, vamos para o modelo de negócio de nossa aplicação. Não trarei um novo modelo para este capítulo, retrabalharemos a classe `Garcom`, que foi feita logo no capítulo 2 e sem persistência de dados.

A nossa base de dados nos dispositivos móveis está fazendo uso do SQLite para persistência dos dados. Criaremos uma aplicação servidora, que fiz uso do SQL Server Express 2014 em minha máquina, para atender às solicitações de sincronismo realizadas pela aplicação mobile.

Essa aplicação servidora deverá estar disponibilizada na nuvem no Windows Azure, e nele teremos a persistência dos dados no SQL Server oferecido por ele. Ufa, três bancos, mas isso será tranquilo.

Enfim, resumindo, nossa aplicação terá os dados sincronizados entre os dispositivos e a nuvem. Sendo assim, precisamos pensar na sincronização. Criarei neste projeto nosso modelo, o que redundará

também na aplicação mobile, no nosso projeto PCL.

A aplicação criada já traz uma pasta chamada `Models`, que é o padrão do ASP.NET MVC. Crie nesta pasta a classe `Garcom`, tal qual o código adiante.

Note na classe os atributos que marcam a propriedade `GarcomId`. O valor desta propriedade será composto por dois identificadores, o `DispositivoID` e o `EntityId`. Logo veremos mais sobre isso. Veja também a implementação sobreescrita para os métodos `Equals()` e `GetHashCode()`, precisaremos disso no sincronismo.

```
using CCFoodServer.Models.Enums.Modulo1.Modelo.Enums;
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Modulo1.Modelo
{
    public class Garcom
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public string GarcomId { get; set; }
        public string Nome { get; set; }
        public string Sobrenome { get; set; }
        public byte[] Foto { get; set; }

        public long? DispositivoID { get; set; }
        public long? EntityId { get; set; }
        public OperacaoSincronismo OperacaoSincronismo { get; set; }
    }

    public override bool Equals(object obj)
    {
        var garcom = obj as Garcom;
        return this.GarcomId.Equals(garcom.GarcomId);
    }

    public override int GetHashCode()
    {
        return Convert.ToInt32(DispositivoID + EntityId);
    }
}
```

```
    }  
}
```

Note, na listagem anterior, que temos uma propriedade `OperacaoSincronismo`. Este tipo de dado é um `Enum`. Vamos criá-lo. Crie na pasta `Models` uma nova pasta chamada `Enums` e, dentro dela, crie a classe com o código a seguir. É por meio deste tipo de dado que controlaremos o estado dos objetos da classe no dispositivo.

Nosso sincronismo será simples. Apenas publicaremos no servidor os registros inseridos localmente nos dispositivos, e traremos do servidor para os dispositivos todos os dados, atualizando no dispositivo os ainda não existentes nele.

```
namespace CCFoodsServer.Models.Enums  
{  
    namespace Modulo1.Modelo.Enums  
    {  
        public enum OperacaoSincronismo  
        {  
            InseridoDispositivo,  
            Sincronizado  
        }  
    }  
}
```

Faz parte também do modelo que trabalharemos uma classe que será responsável por manter a configuração do dispositivo junto ao servidor da aplicação. O código dela está na sequência e ela precisa ser criada na pasta `Models`, junto com `Garcom`.

Cada dispositivo terá um código único na aplicação e, este código, como dito anteriormente, comporá o identificador de cada objeto, a chave primária da tabela. Não será permitido ao usuário acessar páginas relacionadas a garçons (em nosso exemplo), sem ter este identificador de dispositivo.

```
namespace CCFoodsServer.Models  
{
```

```

public class ConfiguracaoDispositivo
{
    public long? ConfiguracaoDispositivoId { get; set; }
    public string EMail { get; set; }
}

```

7.2 O ACESSO A DADOS PARA A APLICAÇÃO SERVIDORA

Para o acesso a dados em nossa aplicação, faremos uso do Entity Framework, e precisamos instalá-lo em nosso projeto. O EF é um framework de mapeamento de objetos em um modelo relacional. Não há como entrar em detalhes sobre ele, pois ele, por si só, é conteúdo para um livro.

Clique com o botão direito do mouse sobre o nome do projeto e selecione `Manage NuGet Packages`. Se ele não aparecer para você, clique na categoria `Browse` e digite seu nome. Após encontrá-lo, selecione-o e clique em `Install`, tal qual a figura a seguir.

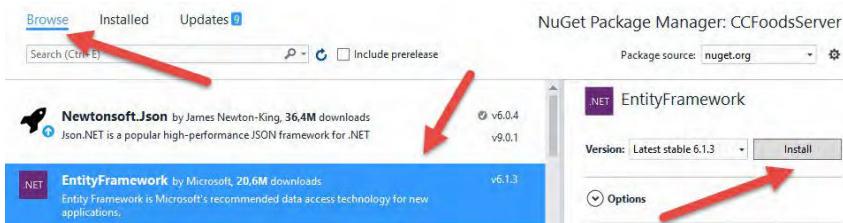


Figura 7.2: Instalando o Entity Framework

No projeto, crie uma pasta chamada `Persitência` e, dentro dela, uma classe chamada `CCFoodsContext`, como o código a seguir. Nessa classe, que estende `DbContext`, invocamos o construtor da superclasse enviando o nome da string de conexão com o banco, a `ConnectionString`. Implementamos as propriedades do contexto, que representam o mapeamento com as tabelas na base de dados.

```

using CCFoodsServer.Models;
using Modulo1.Modelo;
using System.Data.Entity;

namespace CCFoodsServer.Persistence
{
    public class CCFoodsContexts : DbContext
    {
        public CCFoodsContexts() : base("CCFoods_CS")
        {

        }

        public DbSet<Garcom> Garcons { get; set; }
        public DbSet<ConfiguracaoDispositivo> ConfiguracoesDispositivos { get; set; }
    }
}

```

Precisamos agora inserir a string de conexão `CCFoods_CS` no arquivo `Web.config`, para que o Entity Framework possa recuperá-la e realizar corretamente o acesso aos dados. Ela está na sequência. Insira-a após o fechamento da tag `<configSections>`.

Lembre-se de que você precisa ter instalado o SQL Server Express em sua máquina. Não estou usando aqui o SQL Server LocalDb, ok? Você pode realizar o download pelo link <https://www.microsoft.com/pt-br/download/details.aspx?id=42299>.

Não detalharei aqui o processo de instalação desta ferramenta, por ser relativamente trivial. O SQL Server LocalDb é uma instância reduzida, mas poderosa, do SQL Server.

```

<connectionStrings>
    <add name="CCFoods_CS" connectionString="Data Source=EVERTON-DELL\SQLEXPRESS;Initial Catalog=ccfoods_db;Integrated Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>

```

Para finalizar o acesso a dados, criaremos as classes com estas funcionalidades agora. Na pasta `Persistencia`, crie a classe `ConfiguracaoDispositivoDAL`, assim como o código seguinte.

Observe que, como cada dispositivo só poderá ser inserido uma vez na aplicação, o método responsável pela inserção tem de garantir isso, procurando por um registro do e-mail recebido como parâmetro. Caso o e-mail enviado ainda não exista na base de dados, ele será inserido e então o contexto será salvo.

```
using CCFoodssServer.Models;
using System.Linq;

namespace CCFoodssServer.Persistence
{
    public class ConfiguracaoDispositivoDAL
    {
        private CCFoodssContexts contexto = new CCFoodssContexts();

        public ConfiguracaoDispositivo Insert(string eMail)
        {
            ConfiguracaoDispositivo cd = GetConfiguracaoDispositivo(eMail);
            if (cd == null)
            {
                cd = contexto.ConfiguracoesDispositivos.Add(
                    new ConfiguracaoDispositivo() { EMail = eMail });
            }
            contexto.SaveChanges();
        }
        return cd;
    }

    private ConfiguracaoDispositivo GetConfiguracaoDispositivo
    (string email)
    {
        return contexto.ConfiguracoesDispositivos.Where(e => e
.EMail == email).FirstOrDefault();
    }
}
```

Agora, a classe `GarcomDAL`, que tem seu código apresentado na sequência. Apenas dois métodos estão sendo implementados para o problema proposto para este capítulo.

```
using Modulo1.Modelo;
using System.Collections.Generic;
```

```

namespace CCFoodsServer.Persistencia
{
    public class GarcomDAL
    {
        private CCFoodsContexts contexto = new CCFoodsContexts();

        public IEnumerable<Garcom> GetAll()
        {
            return contexto.Garcons;
        }

        public void Insert(Garcom garcom)
        {
            contexto.Garcons.Add(garcom);
            contexto.SaveChanges();
        }
    }
}

```

7.3 OS SERVIÇOS WEB RESTFUL

Com o modelo de dados e o acesso a base de dados implementados, precisamos implementar o serviços que nossa aplicação atenderá. Começaremos com o serviço que registrará o dispositivo no servidor e retornará o identificador dele para a aplicação.

Na pasta `Controllers`, clique com o botão direito sobre ela, e selecione `Add -> Controller`. Nomeie-o `ConfiguracaoDispositivoController`. Na janela que se exibe, escolha `Web API 2 Controller` e clique no botão `Add`. Implemente o seguinte código nesta nova classe.

Observe a definição de um DAL, da rota para o serviço e da chamada ao método `Insert()`. Este serviço será atendido pelo método `HTTP GET`.

```

using CCFoodsServer.Persistencia;
using System.Web.Http;

```

```

namespace CCFoodsServer.Controllers
{
    public class ConfiguracaoDispositivoController : ApiController
    {
        private ConfiguracaoDispositivoDAL configuracaoDispositivo
        DAL = new ConfiguracaoDispositivoDAL();

        [Route("dispositivos/configuracao/")]
        public long Get(string eMail)
        {
            return (long) configuracaoDispositivoDAL.Insert(eMail)
            .ConfiguracaoDispositivoId;
        }
    }
}

```

Na sequência, precisamos implementar os serviços que disponibilizaremos para o trabalho com garçons. Veja em seguida a classe `GarcomController`, que você deve adicionar ao projeto, tal qual foi feito para o `ConfiguracaoDispositivoController`.

Observe que agora possuímos dois serviços: um `HTTP GET`, que retorna todos os garçons registrados no sistema (não apenas o do dispositivo), pois este é um dado comum para todos os clientes em nosso sistema; e um `HTTP POST`, que receberá um objeto `Garcom`, que terá seu estado de sincronismo alterado e então será inserido na base de dados central, a base da nuvem.

```

using CCFoodsServer.Persistencia;
using Modulo1.Modelo;
using System.Collections.Generic;
using System.Web.Http;

namespace CCFoodsServer.Controllers
{
    public class GarcomController : ApiController
    {
        private GarcomDAL garcomDAL = new GarcomDAL();

        // GET: api/Garcom
        [Route("garcons/todos")]
        public IEnumerable<Garcom> Get()
        {
            return garcomDAL.GetAll();
        }
    }
}

```

```
    }

    [Route("garcom/insert")]
    public void PostInsertGarcom(Garcom garcom)
    {
        garcom.OperacaoSincronismo = Models.Enums.Modulo1.Modo
1o.Enums.OperacaoSincronismo.Sincronizado;
        garcomDAL.Insert(garcom);
    }
}

}
```

Vamos testar nossa aplicação com a chamada ao serviço [http://localhost:63179/dispositivos/configuracao?
email=everttoncoimbra@gmail.com](http://localhost:63179/dispositivos/configuracao?email=everttoncoimbra@gmail.com). Veja que na URL está o servidor e porta de minha máquina, logo, é preciso ver como estará na sua. Note também que envio uma *Query string* com o nome do parâmetro que o serviço recebe e o valor atribuído a ele.

A primeira execução, que deve ser feita com o pressionamento da tecla F5 (ou pelo menu *Debug*), pode demorar um pouco, pois será criada a base de dados e as tabelas mapeadas na classe de contexto. Você pode depois acessar o *SQL Server Management Studio*, que é instalado junto com o *SQL Express 2014*, para verificar a base de dados criada e suas tabelas.

O resultado da requisição anterior é <long
*xmlns="http://schemas.microsoft.com/2003/10/Serialization/">1</long> . Ou seja, no meu caso, o Id para o dispositivo com e-mail *everttoncoimbra@gmail.com* será 1 .*

Atente que não estou preocupado neste momento com segurança no acesso a uma aplicação web. Quis apenas demonstrar a requisição a um serviço web que não está com a rota mapeada. O recomendado aqui seria o uso de um serviço HTTP POST, além de autenticação e autorização.

Se você quiser, você pode inserir alguns registros na tabela de

garçons e testar requisitar o serviço `garcons/todos`. Esta inserção pode ser por meio de código, `Server Explorer` no Visual Studio ou no SQL Server Management Studio. Fica a seu critério. Nossa teste aqui se dará depois, por meio da aplicação, que se repetirá para a chamada ao serviço HTTP POST, que inserirá os garçons inseridos via dispositivo.

Precisamos agora publicar nossa aplicação na nuvem, no Windows Azure. Mas para isso, é necessário antes criar nossa conta e recursos nele. Vamos lá.

7.4 APLICAÇÃO, O BANCO DE DADOS E WINDOWS AZURE

Terminada a implementação de nossa aplicação que estará na nuvem, atendendo a todos os dispositivos, precisamos agora criá-la em um servidor, para que os dispositivos possam acessá-la. Para isso, faremos uso do Windows Azure (<https://azure.microsoft.com>).

Para acessá-lo, é preciso que você tenha uma conta nele. Ele não é gratuito, mas oferece um crédito em reais para uso com os recursos. Ao acessar o site, logo de início existe um link chamado **Conta Gratuita**, conforme destaca a figura a seguir.

Por se tratar de um portal dinâmico, essa página pode sofrer alterações, o que vale, da mesma maneira, para as políticas de acesso. Você precisará de um cartão de crédito internacional para informar e será cobrado US\$ 1,00 nele. Não detalharei aqui o processo de criação desta conta, é algo trivial.



Figura 7.3: Acessando o Microsoft Azure

Depois da conta criada e em futuros acessos, você deverá acessar o link **Minha Conta** e em seguida o **Portal Azure**, como também destaca a figura a seguir.

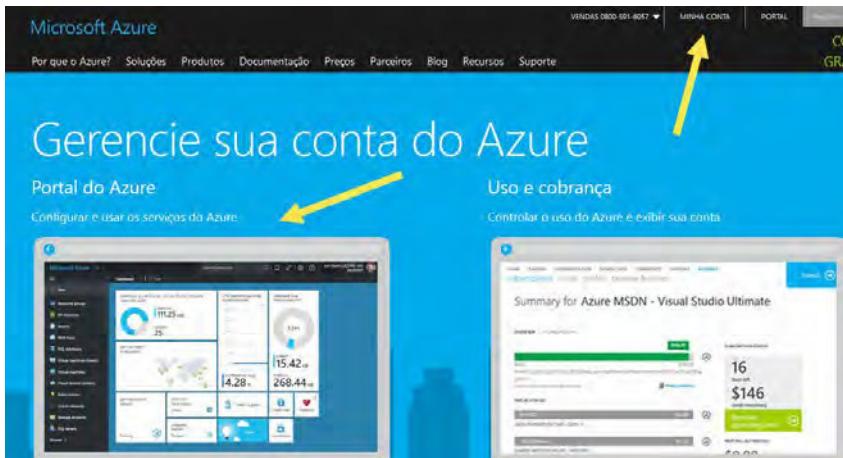


Figura 7.4: Acessando a conta do Windows Azure

Ao acessar o portal, que é conhecido também como **Console do Windows Azure**, um menu lateral é apresentado com opções de acesso. Primeiramente criaremos nossa aplicação. Sendo assim, no menu, clique em **Novo**, depois em **Web + Celular** e em **Aplicativo Web**.

Este processo pode demorar um pouco, pois depende de sua conexão com a internet. A minha aplicação segue os parâmetros da figura a seguir. Em localização, escolhi **South Central US**.

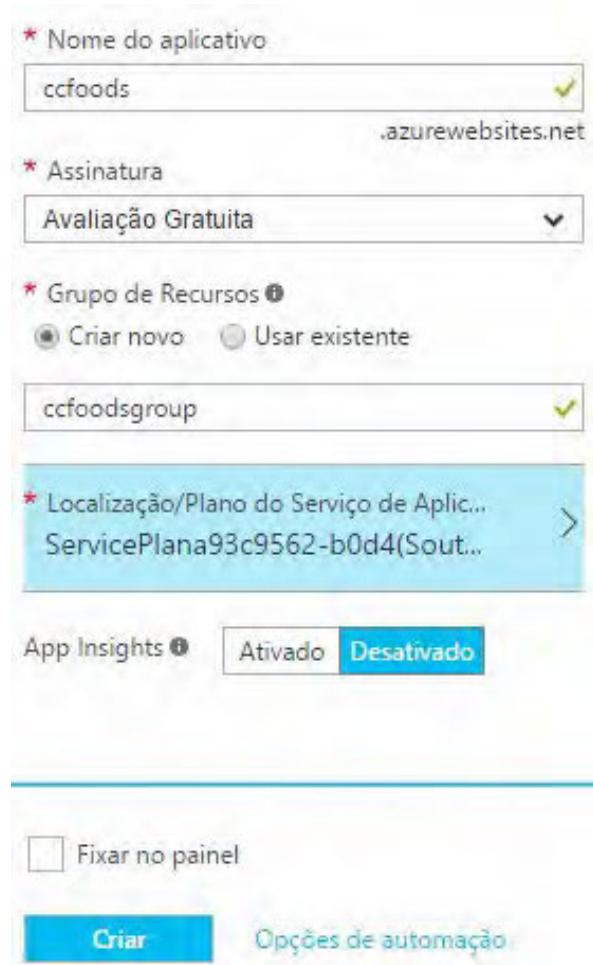


Figura 7.5: Criando a aplicação web

O passo seguinte é criar o servidor de banco de dados e a base de dados que nossa aplicação utilizará. Para isso, uma vez mais no menu **Novo**, selecione agora **Dados + Armazenamento**, e em seguida **SQL Database**. Na janela que se abre, configurei meus parâmetros tal qual consta na figura a seguir, do lado esquerdo.

Nesta figura, quando for criar o servidor de banco de dados, também serão solicitados valores para os parâmetros, os quais estão

ao lado direito dela. Não esqueça de seu usuário e senha.

The screenshot shows the Azure portal's 'Create a new web app' wizard. The first step, 'Select a database', is completed. The second step, 'Configure your web app', is currently active. The configuration fields include:

- Nome do banco de dados:** ccfoods
- Assinatura:** Avaliação Gratuita
- Grupo de recursos:** Criar novo (ccfoods)
- Selecionar fonte:** Banco de dados em branco
- Servidor:** ccfoods (Centro-Sul dos EUA) - This field is highlighted with a blue border.
- Tipo de preço:** S2 Standard
- Fixar no painel:** (unchecked)

The third step, 'Configure your server', is visible below:

- Nome do servidor:** ccfoods.database.windows.net
- Logon de administrador do servidor:** everton
- Senha:** (redacted)
- Confirmar senha:** (redacted)
- Localização:** Centro-Sul dos EUA
- Criar servidor V12 (atualização mais recente):** Sim (selected)
- Permitir que os serviços do Azure acessem o servidor:** (checked)

At the bottom are two buttons: 'Criar' (Create) and 'Opções de automação' (Automation options).

Figura 7.6: Criando a aplicação web

Para que possamos publicar nossa aplicação, precisamos realizar uma alteração na maneira que o Entity Framework acessa nossa base de dados. Se você lembrar, não fizemos nada de especial nas classes e o banco foi criado pela string de conexão implementada no `Web.config`. Para a aplicação distribuída no Azure, precisamos adaptar nosso contexto para que faça uso de um recurso do EF, que é o `Migrations Kit`.

Para habilitá-lo, estando com seu projeto ativo no Visual Studio, clique no menu `Tools -> NuGet Package Manager -> Packager Manager Console` e, na console que se abre, digite `Enable-Migrations`. Não entrarei em detalhes sobre esta ferramenta, pois como disse no início do capítulo, é um recurso que usarei, mas sem

espaço para detalhar. Apenas saiba que, com este recurso instalado, é possível trabalhar com upgrades e downgrades de versões da estrutura da base de dados.

Como criamos nosso projeto no Visual Studio sem a criação simultânea da aplicação e base de dados, precisamos baixar as credenciais da aplicação, que está no Windows Azure, para que possamos realizar a publicação. Na console do Windows Azure, acesse o item de menu **Todos os recursos** e, na página que se abre, clique na aplicação que criamos anteriormente.

Veja que cada recurso tem um ícone diferente na listagem. O tipo da aplicação web é **Serviço de aplicativo**. Nessa nova janela, no topo, tem uma opção chamada **...Mais**, clique nela e, no menu que se abre, selecione **Obter perfil de...**. Veja a figura a seguir.

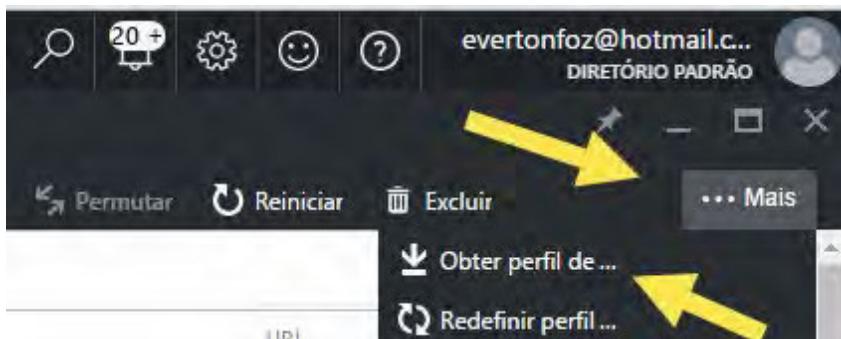


Figura 7.7: Obtendo as credenciais da aplicação para uso no Visual Studio

Agora teremos uma sequência de telas que trarei como figuras aqui. Para começarmos a publicação, clique com o botão direito no nome do projeto, na Solution Explorer, do Visual Studio e escolha **Publish...**. Na janela que se abre, clique na opção **Import** e localize o arquivo baixado anteriormente.

Após a leitura do arquivo, o Visual Studio exibirá a figura a

seguir. Não precisa alterar nenhum valor nela. Clique no botão **Validate Connection** e, após a confirmação, no botão **Next**.

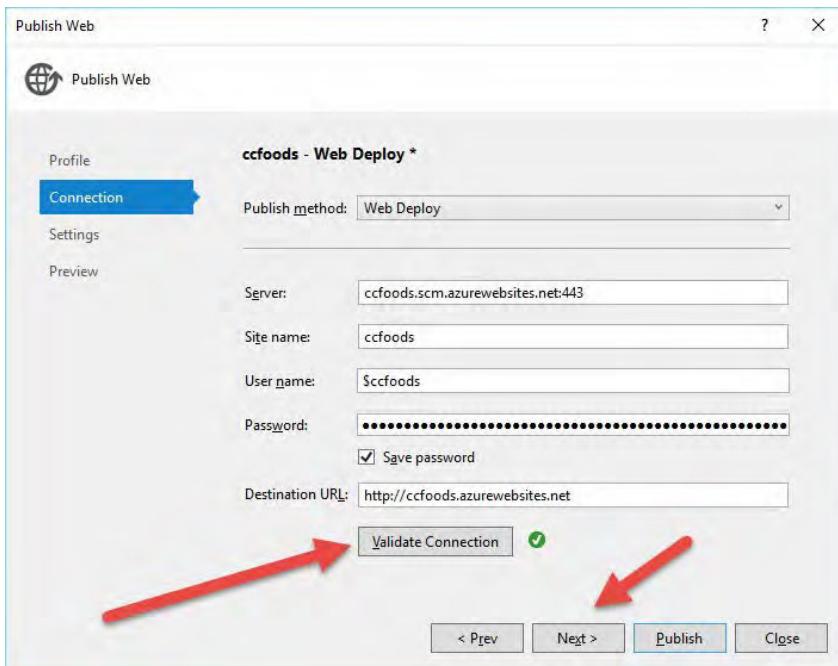


Figura 7.8: Dados da aplicação obtidos pelas credenciais importadas

A janela que se exibe agora precisará da nossa Connection String, pois a publicação identificou a existência de uma configuração no `Web.Config`. Precisamos obter este dado no Azure.

Novamente, clique no menu **Todos os recursos** e selecione o recurso de tipo **Banco de dados SQL**. Na janela que se abre, clique em **Mostrar cadeias de conexão com banco de dados**. Veja a figura a seguir.

Na janela que se abre, clique no botão ao lado direito do campo que exibe a string de conexão para `ADO.NET`. Cole esta string em um arquivo de texto, e insira seu nome de usuário e senha nos

campos que estão solicitando estes dados.

Fundamentos ^	
Grupo de recursos	Nome do servidor
ccfoods	ccfoods.database.windows.net
Status	Versão do servidor
Online	V12
Localização	Cadeias de conexão
South Central US	Mostrar cadeias de conexão do banco de d...
Nome da assinatura	Camada de preços
Avaliação Gratuita	S2 Standard (50 DTUs)
ID da Assinatura	Função de Replicação Geográfica
4a5b4909-f6c6-417e-a120-085bc75b91ad	Não configurado

Figura 7.9: Obtendo a connection string com o banco de dados do Azure

Voltemos ao Visual Studio. Na janela exibida e representada pela figura a seguir, marque as opções em destaque na figura e cole sua Connection String na caixa de texto marcada. Após isso, clique no botão **Next**.

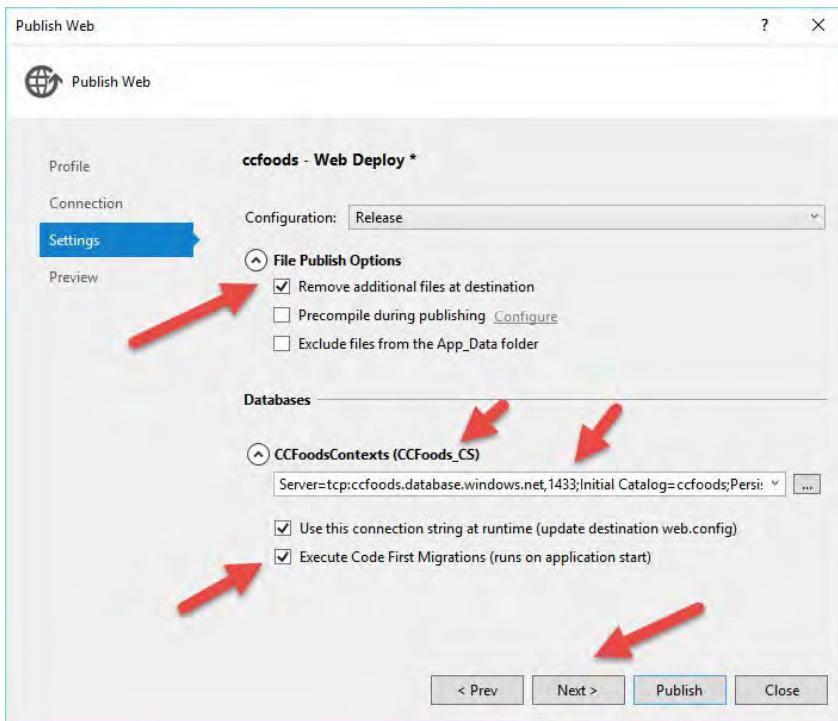


Figura 7.10: Configuração do acesso a dados pela aplicação

A última janela, que não vou apresentar a figura, traz um botão logo no centro, `Start Preview`. Clique nele e será buscado em sua aplicação local o que foi alterado em relação a última publicação no Azure e qual ação deverá ser tomada.

Como esta é nossa primeira publicação, serão exibidos todos os arquivos do projeto, com a ação `Add`. Após visualizar estas mudanças, clique no botão `Publish`. Ao concluir, o navegador será aberto, com o domínio da aplicação como página atual.

Vamos verificar se os serviços estão disponíveis? Complemente a URL para ficar semelhante a <http://ccfoods.azurewebsites.net/dispositivos/configuracao?email=evertoncoimbra@gmail.com>. O retorno é para ser o mesmo

que obtivemos localmente, `<long xmlns="http://schemas.microsoft.com/2003/10/Serialization/">1</long>`. Isso quer dizer que nossa aplicação foi bem publicada e está funcionando.

7.5 PREPARAÇÃO DA APLICAÇÃO MOBILE PARA CONSUMIR OS SERVIÇOS RESTFUL

Com a aplicação servidora pronta para o trabalho que temos de realizar com garçons, precisamos agora implementar como esta funcionalidade será consumida em nossa aplicação mobile. O primeiro passo é criarmos nossas classes de modelo de negócio, e elas redundarão o que criamos na aplicação servidora.

Sendo assim, na pasta `Modelo`, crie a pasta `Enums` e, dentro dela, o arquivo `OperacaoSincronismo`, como segue.

```
namespace Modulo1.Modelo.Enums
{
    public enum OperacaoSincronismo
    {
        InseridoDispositivo,
        Sincronizado
    }
}
```

Agora precisamos implementar as classes `ConfiguracaoDispositivo` e `Garcom`, que têm seus códigos apresentados sequencialmente, logo a seguir, ambas na pasta `Modelo`.

```
namespace Modulo1.Modelo
{
    public class ConfiguracaoDispositivo
    {
        public long? ConfiguracaoDispositivoId { get; set; }
        public string EMail { get; set; }
    }
}
```

Note na listagem a seguir a implementação, aqui também, dos métodos `Equals()` e `GetHashCode()`.

```
using Modulo1.Modelo.Enums;
using SQLite.Net.Attributes;
using System;

namespace Modulo1.Modelo
{
    public class Garcom
    {
        [PrimaryKey]
        public string GarcomId { get; set; }
        public string Nome { get; set; }
        public string Sobrenome { get; set; }
        public byte[] Foto { get; set; }

        public long? DispositivoId { get; set; }
        public long? EntityId { get; set; }
        public OperacaoSincronismo OperacaoSincronismo { get; set; }
    }

    public override bool Equals(object obj)
    {
        var garcom = obj as Garcom;
        return this.GarcomId.Equals(garcom.GarcomId);
    }

    public override int GetHashCode()
    {
        return Convert.ToInt32(DispositivoId + EntityId);
    }
}
```

Os objetos das classes anteriores serão todos persistidos no SQLite. Desta maneira, precisamos implementar as classes DAL para eles.

A primeira é a classe `ConfiguracaoDAL`, que deve ser criada na pasta `Dal` e ter o código apresentado na sequência. Lembre-se de que cada dispositivo terá sempre apenas um objeto/registro, ou seja, os seus dados de configuração.

```
using Modulo1.Infraestructure;
```

```

using Modulo1.Modelo;
using SQLite.Net;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Dal
{
    public class ConfiguracaoDAL
    {
        private SQLiteConnection sqlConnection;

        public ConfiguracaoDAL()
        {
            this.sqlConnection = DependencyService.Get<IDatabaseCo
nnexion>().DbConnection();
            this.sqlConnection.CreateTable<ConfiguracaoDispositivo
>();
        }

        public ConfiguracaoDispositivo GetConfiguracao()
        {
            return (from t in sqlConnection.Table<ConfiguracaoDisp
ositivo>() select t).FirstOrDefault();
        }

        public void Add(ConfiguracaoDispositivo configuracaoDisp
ositivo)
        {
            sqlConnection.Insert(configuracaoDispositivo);
        }
    }
}

```

O mesmo recurso deverá ser disponibilizado para os garçons. Assim, precisamos criar a classe `GarcomDAL`, também na pasta `DAL`. Veja na sequência o código para esta classe.

Ela possui mais métodos, além de declararmos como campo da classe um objeto DAL para a configuração. O construtor, o `Update()` e o `GetAll()` são triviais, dispensando explicações.

O método `GetAllInseridoDispositivo()` faz uma seleção mais criteriosa, pois ele será usado para atualizar os dados na nuvem (na aplicação servidora). Ele seleciona apenas os objetos ainda não

sincronizados e que sejam do dispositivo atual. O `Add()` obtém os valores para a composição do identificador do objeto (chave primária na tabela), define a propriedade de sincronismo, e então insere o objeto na base de dados.

```
using Modulo1.Infraestructure;
using Modulo1.Modelo;
using SQLite.Net;
using System;
using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Dal
{
    public class GarcomDAL
    {
        private SQLiteConnection sqlConnection;
        private ConfiguracaoDAL configuracaoDAL = new Configuracao
DAL();

        public GarcomDAL()
        {
            this.sqlConnection = DependencyService.Get<IDatabaseCo
nnection>().DbConnection();
            this.sqlConnection.CreateTable<Garcom>();
        }

        public void Update(Garcom garcom)
        {
            this.sqlConnection.Update(garcom);
        }

        public IEnumerable<Garcom> GetAll()
        {
            return (from t in sqlConnection.Table<Garcom>() select
t).OrderBy(i => i.Nome).ToList();
        }

        public IEnumerable<Garcom> GetAllInseridoDispositivo()
        {
            var configuracaoId = configuracaoDAL.GetConfiguracao()
.ConfiguracaoDispositivoId;
            return (from t in sqlConnection.Table<Garcom>() where
t.OperacaoSincronismo == Modelo.Enums.OperacaoSincronismo.Inserido
Dispositivo && t.DispositivoId == configuracaoId select t).OrderBy
(i => i.Nome).ToList();
        }
    }
}
```

```

    }

    public void Add(Garcom garcom)
    {
        garcom.DispositivoId = configuracaoDAL.GetConfiguracao
        ().ConfiguracaoDispositivoId;
        garcom.EntityId = GetMaxId();
        garcom.GarcomId = garcom.DispositivoId.ToString().Trim
        () + "/" + garcom.EntityId.ToString().Trim();
        garcom.OperacaoSincronismo = Modelo.Enums.OperacaoSinc
        ronismo.InseridoDispositivo;
        sqlConnection.Insert(garcom);
    }

    private long GetMaxId()
    {
        var id = sqlConnection.Table<Garcom>().Max(g => g.Enti
        tyId);
        return Convert.ToInt32(id) + 1;
    }
}
}

```

7.6 A INTERFACE COM O USUÁRIO PARA A CONFIGURAÇÃO DO DISPOSITIVO

Vamos implementar agora a interface com o usuário para a configuração do dispositivo. Novamente relembro de que não estou me preocupando aqui com o processo de autenticação e autorização, assumindo que, se o usuário está na aplicação, ele está autenticado e autorizado a todos os serviços oferecidos.

Sendo assim, na pasta `Pages`, crie uma pasta chamada `Configuracao`. Nela, crie uma página chamada `ConfiguracoesPage`, com o código a seguir.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="Modulo1.Pages.Configuracoes.ConfiguracoesPag
e"
              Title="Configurações">

```

```

<ContentPage.Content>
    <Frame>
        <StackLayout VerticalOptions="Center" Padding="10">
            <Entry Placeholder="Email" PlaceholderColor="Gray" x:Name="eMail"/>
            <Label x:Name="dispositivoId" Text="Sem Id" HorizontalOptions="Center"/>
            <Button Text="Obter Id para o dispositivo" Clicked="OnClickedObter" x:Name="BtnObterId"/>
        </StackLayout>
    </Frame>
</ContentPage.Content>
</ContentPage>

```

Não há nada de diferente no código apresentado anteriormente. Precisamos criar agora o comportamento para esta página. Veja no código a seguir esta implementação. No código ainda não implementei o método que captura o clique do botão da página para obter o `id`, portanto, não tente executar sua aplicação ainda, pois dará erro.

Veja, no construtor, que é chamado o método `VerificarIdDispositivo()`, que em caso de já existir configuração, invoca o método `AtualizarControles()`, que recebe os dados recuperados para o controle e desabilita os controles visuais para alteração. Caso não exista ainda um controle para o dispositivo, uma mensagem é dada como aviso para o usuário e os controles permitirão a interação.

```

using Modulo1.Dal;
using Modulo1.Modelo;
using System;
using Xamarin.Forms;

namespace Modulo1.Pages.Configuracoes
{
    public partial class ConfiguracoesPage : ContentPage
    {
        private ConfiguracaoDAL configuracaoDAL = new Configuracao
DAL();
        public ConfiguracoesPage()
        {

```

```

        InitializeComponent();
        VerificarIdDispositivo();
    }

    private void VerificarIdDispositivo()
    {
        var cd = configuracaoDAL.GetConfiguracao();
        if (cd != null)
        {
            AtualizarControles(cd);
        }
        else
        {
            DisplayAlert("Erro", "ID não pôde ser recuperado o
u não foi ainda criado", "OK");
        }
    }

    private void AtualizarControles(ConfiguracaoDispositivo cd
)
{
    dispositivoId.Text = "ID do dispositivo: " + cd.Config
uracaoDispositivoId.ToString();
    eMail.Text = cd.EMail;
    BtnObterId.Visible = false;
    eMail.IsEnabled = false;
}
}
}

```

Para implementarmos o comportamento para o evento que captura o clique do botão, que buscará na aplicação servidora o id atribuído ao dispositivo, precisamos lembrar de que isso será feito por meio de consumo do web service RESTful que criamos anteriormente e publicamos no Azure. Então, vamos lá.

Crie uma pasta no projeto chamada `RESTServices`. E nela uma classe chamada `ConfiguracaoDispositivoREST`, com o código seguinte. Veja a criação do cliente para a aplicação servidora no construtor, definindo um número máximo de bytes para o buffer de resposta para o conteúdo lido pelo cliente.

No método `GetDispositivoIdAsync()`, temos a definição da

URL relativa ao serviço que desejamos consumir. Caso a conexão seja bem realizada e o serviço consumido com sucesso, o retorno é lido e deserializado, sendo ele retornado ao método chamador. Não optei por trabalhar com exceções na requisição do serviço, mas você poderia considerar isso em sua aplicação.

```
using Modulo1.Modelo;
using Newtonsoft.Json;
using System;
using System.Net.Http;
using System.Threading.Tasks;

namespace Modulo1.RESTServices
{
    public class ConfiguracaoDispositivoREST
    {
        private HttpClient client;
        public ConfiguracaoDispositivo ConfiguracaoDispositivo { get; set; }

        public ConfiguracaoDispositivoREST()
        {
            client = new HttpClient();
            client.MaxResponseContentBufferSize = 256000;
        }

        public async Task<long?> GetDispositivoIdAsync(string eMail)
        {
            long? id = null;
            // Testei em uma rede esta URL e por problema com a configuração troquei o domínio para o IP e funcionou. Observe também o HTTPS
            var uri = new Uri(string.Format("https://ccfoods.azurewebsites.net/dispositivos/configuracao?email={0}", eMail));
            var response = await client.GetAsync(uri);
            if (response.IsSuccessStatusCode)
            {
                var content = await response.Content.ReadAsStringAsync();
                id = JsonConvert.DeserializeObject<long>(content);
            }
            return id;
        }
    }
}
```

Agora sim, com esta classe implementada, podemos também implementar o método que captura o evento de clique do botão da página que mantém a configuração do dispositivo. Veja o código dele na sequência.

```
public async void OnClickedObter(object sender, EventArgs args)
{
    await DisplayAlert("Aviso", "Este processo depende de sua conexão com a internet. Ele pode ser lento, ou até falhar", "Ok");
    var services = new ConfiguracaoDispositivoREST();
    IsBusy = true;
    try
    {
        var id = await services.GetDispositivoIdAsync(eMail.Text);
        var cd = new ConfiguracaoDispositivo()
        {
            EMail = eMail.Text,
            ConfiguracaoDispositivoId = id
        };
        configuracaoDAL.Add(cd);
        AtualizarControles(cd);
        await DisplayAlert("Configuração de Id", "ID para o dispositivo criado/recuperado com sucesso", "Ok");
    }
    catch (Exception ex)
    {
        await DisplayAlert("Erro", ex.Message, "OK");
    }
    IsBusy = false;
}
```

Ufa! Quanto código para podermos testar nossa aplicação. Estamos quase lá. Precisamos apenas adaptar a página `MenuPage` para que possa ter a opção de chamar a página que implementamos.

Insira a instrução `<Button Text="Configurações" Image="icone_settings.png" Clicked="ConfiguracoesOnClicked"/>` nela, observando que faço uso de uma imagem. Na classe, precisamos implementar o método `ConfiguracoesOnClicked()`, que tem seu código na sequência.

```
private async void ConfiguracoesOnClicked(object sender, EventArgs
```

```
args)
{
    await Navigation.PushAsync(new ConfiguracoesPage());
}
```

Com esta implementação realizada, vamos executar a aplicação. Acesse o menu e a opção de configurações. A tela é simples e não me preocuparei em apresentá-la aqui. Informe seu e-mail (ou qualquer valor) e clique no botão. Feche sua aplicação e acesse novamente esta página.

Você verá que o Id já aparece e não é possível alterar seu e-mail. Isso foi uma opção minha. Você poderia criar um botão de reset para este id, se for o caso para sua necessidade. Se você tiver acesso à rede e não houver bloqueio nela, é para conseguir acesso sem problemas. Se não conseguir acessar via emulador, faça o deploy para seu dispositivo e tente por ele.

7.7 A INSERÇÃO DE GARÇONS NA APLICAÇÃO

A interface para que o usuário possa inserir garçons é semelhante ao processo que fizemos no capítulo 4, para entregadores. Como pode ser verificado no código a seguir, retirei apenas o acesso à câmera, mantendo o ao álbum de fotos. Também criei na pasta Garcons , que já deve existir em sua pasta Pages e nomeei como GarconsNewPage .

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.Garcons.GarconsNewPage">
<ContentPage.Content>
    <ScrollView>
        <StackLayout VerticalOptions="Center">
            <Grid Padding="5,10,5,10">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/>
```

```

<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="1*"/>
</Grid.ColumnDefinitions>
<Frame Grid.Row="0" Grid.Column="0" OutlineColor="Black"
       BackgroundColor="Yellow" HasShadow="True"
       Padding="5,5,5,5">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="1*"/>
                <ColumnDefinition Width="2*"/>
            </Grid.ColumnDefinitions>
            <Image Source="icone_garcom.png" Grid.Row="0" Grid
                  .Column="0"/>
            <Label Grid.Row="0" Grid.Column="1" Text="Novo Gar
                  çom"
                  Font="24" TextColor="Blue" HorizontalOptions=
                  "Start"
                  VerticalOptions="Center"/>
        </Grid>
    </StackLayout>
</Frame>
<Frame Grid.Row="1" Grid.Column="0" OutlineColor="Black"
       HasShadow="True"
       Padding="5,5,5,5">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="100*"/>
            </Grid.ColumnDefinitions>
            <Entry Placeholder="Nome" PlaceholderColor="Gray"
                  Grid.Row="0" Grid.Column="0" Text="{Binding N
                  ome}"
                  x:Name="nome"/>
        </Grid>
    </StackLayout>
</Frame>

```

```

<Frame Grid.Row="2" Grid.Column="0" OutlineColor="Black"
HasShadow="True"
    Padding="5,5,5,5">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="100*"/>
            </Grid.ColumnDefinitions>
            <Entry Placeholder="Sobrenome" PlaceholderColor="G
ray"
                Grid.Row="0" Grid.Column="0" Text="{Binding S
obrenome}"
                x:Name="sobrenome"/>
        </Grid>
    </StackLayout>
</Frame>

<Frame Grid.Row="3" Grid.Column="0" OutlineColor="Black"
HasShadow="True"
    Padding="5,5,5,5" x:Name="framefoto">
    <StackLayout>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="20"/>
                <RowDefinition Height="70"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="30*"/>
                <ColumnDefinition Width="70*"/>
            </Grid.ColumnDefinitions>
            <Label Grid.Row="0" Grid.Column="0" Text="Fotograf
ia" FontSize="10"
                HorizontalOptions="Start"/>
            <Button Grid.Row="1" Grid.Column="0" Text="Álbum"
                x:Name="BtnAlbum" VerticalOptions="CenterA
ndExpand" HorizontalOptions="Start"/>
                <Image Grid.Row="0" Grid.Column="1" Grid.RowSpan="
2" Grid.ColumnSpan="2"
                    x:Name="fotogarcom" HorizontalOptions="End"
/>
        </Grid>
    </StackLayout>
</Frame>

<Frame Grid.Row="4" Grid.Column="0" OutlineColor="Black"

```

```

HasShadow="True"
    Padding="0">
<StackLayout>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="100*"/>
        </Grid.ColumnDefinitions>
        <Button Grid.Row="0" Grid.Column="0" Text="Gravar"
            Clicked="BtnGravarClicked"/>
    </Grid>
</StackLayout>
</Frame>
</Grid>
</StackLayout>
</ScrollView>
</ContentPage.Content>
</ContentPage>

```

Antes de visualizarmos a interface em execução, precisamos implementar os métodos que capturam os eventos cliques dos botões de acesso ao álbum e do botão de gravar o garçom. Na sequência, para simplificar, está a listagem da classe completa.

```

using Modulo1.Dal;
using Modulo1.Modelo;
using Plugin.Media;
using System;
using System.IO;

using Xamarin.Forms;

namespace Modulo1.Pages.Garcons
{
    public partial class GarconsNewPage : ContentPage
    {
        private byte[] bytesFoto;

        public GarconsNewPage()
        {
            InitializeComponent();
            RegistraClickBotaoAlbum();
        }

        private void RegistraClickBotaoAlbum()

```

```

{
    BtnAlbum.Clicked += async (sender, args) =>
    {
        await CrossMedia.Current.Initialize();
        if (!CrossMedia.Current.IsPickPhotoSupported)
        {
            DisplayAlert("Álbum não suportado", "Não existe permissão para acessar o álbum de fotos", "OK");
            return;
        }
        var file = await CrossMedia.Current.PickPhotoAsync();
        if (file == null)
            return;

        var stream = file.GetStream();
        var memoryStream = new MemoryStream();
        stream.CopyTo(memoryStream);
        fotogarcom.Source = ImageSource.FromStream(() =>
        {
            var s = file.GetStream();
            file.Dispose();
            return s;
        });
        bytesFoto = memoryStream.ToArray();
    };
}

private async void BtnGravarClicked(object sender, EventArgs e)
{
    var dal = new GarcomDAL();
    var garcom = new Garcom();
    garcom.Nome = nome.Text;
    garcom.Sobrenome = sobrenome.Text;
    garcom.Foto = bytesFoto;
    dal.Add(garcom);
    ClearControls();
    await App.Current.MainPage.DisplayAlert("Inserção de garçom", "Garçom inserido com sucesso", "Ok");
}

private void ClearControls()
{
    nome.Text = string.Empty;
    sobrenome.Text = string.Empty;
}

```

```
        bytesFoto = null;
        fotogarcom.Source = null;
    }
}
}
```

Agora sim, altere o método do clique do botão Garçons na MenuPage para que instancie esta classe e não mais a GarconsPage , como está lá. Também precisamos garantir que só seja possível inserir um garçom se existir a configuração para o dispositivo, pois o identificador dos objetos (chave primária) faz uso desta configuração, lembra? Veja o código da sequência para este método.

```
private async void GarconsOnClicked(object sender, EventArgs args)
{
    ConfiguracaoDAL dal = new ConfiguracaoDAL();
    if (dal.GetConfiguracao() == null)
    {
        DisplayAlert("Erro", "Dispositivo sem configuração", "Ok")
    ;
    }
    else
    {
        await Navigation.PushAsync(new GarconsNewPage());
    }
}
```

Ao executar, seus emuladores devem mostrar as seguintes figuras. Insira alguns garçons para que possamos realizar o sincronismo.

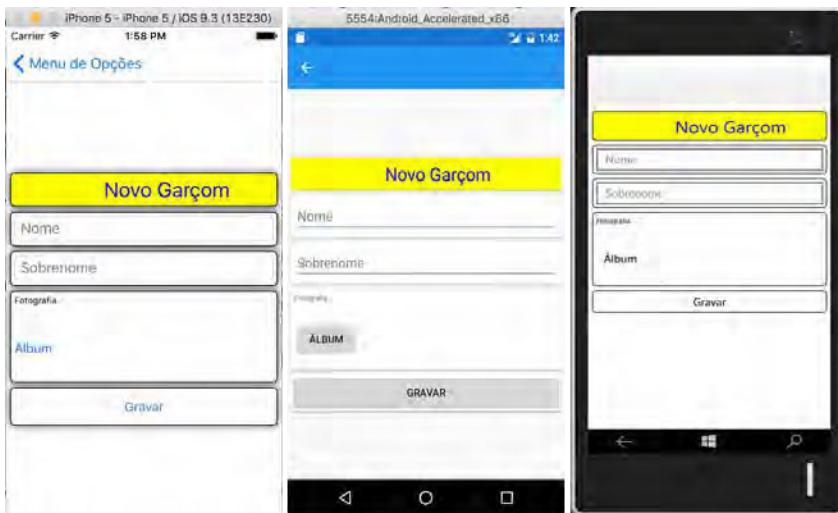


Figura 7.11: Página de inserção de garçons

7.8 A SINCRONIZAÇÃO DO DISPOSITIVO COM A APLICAÇÃO SERVIDORA

Para a implementação do sincronismo, faremos uso de outro recurso do `ListView`, que ainda não vimos, o de puxar a listagem para baixo e realizar a operação conhecida como `Pull to refresh`.

Infelizmente, implementaremos um workaround para a aplicação Windows, pois esta plataforma não implementa esta funcionalidade. Para isso, adaptaremos a página `GarconsPage` que temos. Deixe-a com o código a seguir.

Veja na listagem, na tag `<ListView>`, a configuração de dois novos atributos: `IsPullToRefreshEnabled="True"` e `Refreshing="lvGarconsRefreshing"`. O primeiro habilita o recurso e o segundo aponta para o método que atenderá ao evento. Observe que, no final da listagem, existe a definição de um botão, com configurações específicas para que ele seja exibido apenas

quando a aplicação estiver executando em uma plataforma Windows. Legal, não é?

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:conv="clr-namespace:Modulo1.Converters;assembly=Modulo1"
    x:Class="Modulo1.Pages.Garcons.GarconsPage"
    Title="Garçons">
<ContentPage.Resources>
    <ResourceDictionary>
        <conv:ByteToImageSourceConverter x:Key="convImage"/>
    </ResourceDictionary>
</ContentPage.Resources>
<ContentPage.Content>
    <StackLayout>
        <Grid Padding="5, 10, 5, 10">
            <Grid.RowDefinitions>
                <RowDefinition Height="1*"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="1*"/>
            </Grid.ColumnDefinitions>
            <ListView x:Name="lvGarcons" HasUnevenRows="True" Grid.Row="0"
                Grid.Column="0" IsPullToRefreshEnabled="True"
                Refreshing="lvGarconsRefreshing">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <StackLayout Padding="5, 5, 5, 0" Orientation="Horizontal">
                                <Grid>
                                    <Grid.RowDefinitions>
                                        <RowDefinition Height="Auto"/>
                                    </Grid.RowDefinitions>
                                    <Grid.ColumnDefinitions>
                                        <ColumnDefinition Width="50"/>
                                        <ColumnDefinition Width="1*"/>
                                    </Grid.ColumnDefinitions>
                                    <Image Source="{Binding Foto, Converter={StaticResource convImage}}"
                                        Grid.Row="0" Grid.Column="0" Horizontal
```

```

        Options="FillAndExpand"
                    VerticalOptions="FillAndExpand"/>
                <Label Text="{Binding Nome}" TextColor="Blue"
FontSize="Large" VerticalOptions="Center" Grid.Row="0" Grid.Column="1" />
            </Grid>
        </StackLayout>
    </ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>

<StackLayout Grid.Row="1" Grid.Column="0" Padding="0">
    <Button Text="Inserir novo garçom" x:Name="BtnNovoItem"
Clicked="BtnNovoItemClicked" Image="icone_new.png"/>
</StackLayout>

<StackLayout Grid.Row="2" Grid.Column="0" Padding="0">
    <Button Text="Atualizar Servidor" x:Name="BtnRefreshToUWP"
IsVisible="false" Clicked="BtnRefreshToUWPClicked">
        <Button.IsChecked>
            <OnPlatform x:TypeArguments="x:Boolean">
                <OnPlatform.WinPhone>
                    true
                </OnPlatform.WinPhone>
            </OnPlatform>
        </Button.IsChecked>
    </Button>
</StackLayout>
</ContentPage.Content>
</ContentPage>

```

Ainda não podemos testar a aplicação, pois precisamos implementar os métodos para os eventos de clique dos botões e da atualização (refresh). Entretanto, como na implementação do método de atualização será realizada a invocação para o serviço de atualização que criamos na aplicação servidora, já criaremos esta funcionalidade em nossa aplicação. Sendo assim, na pasta RESTServices , crie a classe GarcomREST , como se segue.

```

public async Task UpdateGarconsToServerAsync(IEnumerable<Garcom> garcons)
{
    var uri = new Uri(string.Format("https://ccfoods.azurewebsites.net/garcom/insert"));
    var garcomDAL = new GarcomDAL();

    foreach (var garcom in garcons)
    {
        //garcom.Foto = null; --> Lembre-se que o serviço precisa de web. Dá para enviar a foto, mas cuidado com a performance
        var json = JsonConvert.SerializeObject(garcom);
        var content = new StringContent(json, Encoding.UTF8, "application/json");

        HttpResponseMessage response = await client.PostAsync(uri, content);

        if (response.IsSuccessStatusCode) {
            garcom.OperacaoSincronismo = Modelo.Enums.OperacaoSincronismo.Sincronizado;
            garcomDAL.Update(garcom);
        }
    }
}

```

Após a chamada ao método anterior, será interessante obtermos do servidor a relação atualizada de garçons, para que possamos atualizar nosso dispositivo. Lembre-se de que garçons é um dado comum entre os dispositivos, ao menos em nossa aplicação. :-). Desta maneira, na sequência, é trazido o código para um método que consuma o serviço da aplicação servidora para esta situação.

```

public async Task<List<Garcom>> GetGarconsAsync()
{
    var uri = new Uri(string.Format("https://ccfoods.azurewebsites.net/garcons/todos"));
    var response = await client.GetAsync(uri);
    if (response.IsSuccessStatusCode)
    {
        var content = await response.Content.ReadAsStringAsync();
        return JsonConvert.DeserializeObject<List<Garcom>>(content);
    }
    return null;
}

```

Agora sim, vamos à implementação da classe referente a página `GarconsPage`. Veja o código seguinte.

Note que há a implementação sobreescrita para evento `OnAppearing`. A aplicação para atualização se dá no método `UpdateToServer()`, que tem sua chamada também no método do clique do botão exclusivo para o Windows, o `BtnRefreshToUWPClicked()`. Agora, a inserção de um novo garçom se dará pelo clique no botão que existe após o `ListView`.

Veja também o método `updateDispositivo()`, que obtém todos os garçons existentes no servidor e retira deles os existentes no dispositivo, ou seja, os inseridos em outros dispositivos e ainda não atualizados no dispositivo onde a aplicação está sendo executada. Em seguida, estes novos garçons são inseridos no dispositivo.

```
using Modulo1.Dal;
using Modulo1.RESTServices;
using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace Modulo1.Pages.Garcons
{
    public partial class GarconsPage : ContentPage
    {
        private GarcomDAL garcomDAL = new GarcomDAL();

        public GarconsPage()
        {
            InitializeComponent();
        }

        protected override void OnAppearing()
        {
            base.OnAppearing();
            lvGarcons.ItemsSource = garcomDAL.GetAll();
        }

        private async void lvGarconsRefreshing(object sender, EventArgs e)
```

```

    {
        await UpdateToServer();
        await UpdateDispositivo();
        lvGarcons.ItemsSource = garcomDAL.GetAll();
        lvGarcons.IsRefreshing = false;
    }

    private async Task UpdateToServer()
    {
        var services = new GarcomREST();
        await services.UpdateGarconsToServerAsync(garcomDAL.Ge
tAllInseridoDispositivo());
    }

    private async Task UpdateDispositivo()
    {
        var garconsServer = await services.GetGarconsAsync();
        var garconsDispositivo = garcomDAL.GetAll();
        var garconsAtualizado = garconsServer.Except(garconsDi
spositivo);
        foreach (var garcomNovo in garconsAtualizado)
        {
            garcomDAL.Add(garcomNovo);
        }
    }

    private async void BtnNovoItemClicked(object sender, Event
Args e)
{
    await Navigation.PushAsync(new GarconsNewPage());
}

    private async void BtnRefreshToUWPClicked(object sender, E
ventArgs e)
{
    await UpdateToServer();
}
}

```

Para que possamos testar nossa aplicação e a sincronização, execute sua aplicação nos emuladores e nos dispositivos que possuir, adicione garçons em cada uma destas instâncias e realize o sincronismo (acesse a listagem e arraste-a para baixo).

É para tudo funcionar perfeitamente! :-) Agora, fica o desafio

para você realizar os processos de remoção e atualização de um garçom, já que os recursos você já conhece.

7.9 CONCLUSÃO

Este capítulo foi trabalhoso. Tivemos de criar uma aplicação servidora e adaptar nossa aplicação mobile para consumir os serviços implementados. Você conheceu o Windows Azure como plataforma para publicação de um projeto Web API; viu mais um recurso oferecido pelo ListView , o Pull To Request; e aprendeu como consumir serviços RESTful de sua aplicação.

Foi bastante conteúdo. Tivemos muita implementação antes de podermos testar nossa aplicação e ver o resultado. Mas foi compensador ao final.

No próximo capítulo, apresentarei e implementaremos o MVVM (Model-View-ViewModel), um padrão para desenvolvimento que facilita a separação da interface com o usuário (nossa XAML) da lógica de negócio, o modelo. Será um capítulo legal, no qual implementarei também uma nova funcionalidade para nossa aplicação e veremos uma aplicação de mapas.

CAPÍTULO 8

APLICAÇÃO DO MVVM E O USO DE MAPAS

Como trabalhamos com uma aplicação que poderá registrar o pedido de pratos e bebidas a serem entregues em domicílio, é interessante (e até necessário) que nossa aplicação disponibilize um mapa, apontando onde é a residência do cliente. E é com isso que trabalharemos neste capítulo.

É um capítulo objetivo, mas a recomendação é que você faça uso de todos os recursos e técnicas já apresentados nos capítulos anteriores. Deixarei a funcionalidade pronta para você. :-) A partir dela, você poderá personalizar a interface com o usuário como preferir.

Para a implementação do mapa proposto, se faz necessário informações de localização e endereço do cliente. Teremos uma área de conteúdo para informação dos dados do cliente e outra para visualização do mapa.

Para a ligação dos controles visuais com as propriedades do modelo de negócio, faremos uso de uma arquitetura conhecida como `Model-View-View Model` (Modelo-Visão-Modelo Visão). Com esta técnica, não precisaremos mais buscar pelo nome do controle para obter o valor informado nele, pois já o teremos ligados, por meio de `Databinding`.

8.1 A CLASSE DE NEGÓCIO E O DAL

Como dito na introdução do capítulo, trabalharemos com dados de clientes. Para isso, crie uma classe chamada `Clientes` na pasta `Modelo`, e procure deixá-la de acordo com o código a seguir.

Observe no código que poderíamos ter uma classe associativa para o endereço do cliente, e reutilizá-la, por exemplo, em garçons e entregadores também. Também poderíamos ter classes para cidades e estados, associadas entre si e, consequentemente, como os endereços.

Você poderia utilizar o `Picker` para estado e, com base nele, abrir um `Search Bar` para a cidade. O que acha de você mesmo fazer esta implementação?

```
using SQLite.Net.Attributes;

namespace Modulo1.Modelo
{
    public class Cliente
    {
        [PrimaryKey, AutoIncrement]
        public long? ClienteId { get; set; }
        public string Nome { get; set; }
        public string Telefone { get; set; }
        public string Endereco { get; set; }
        public string Numero { get; set; }
        public string Bairro { get; set; }
        public string Cidade { get; set; }
        public string Estado { get; set; }
    }
}
```

Agora, passamos à classe DAL. Veja no código da sequência que implementei os métodos básicos para o CRUD com `Clients`. Procure deixar sua implementação (`ClienteDAL`, na pasta `Dal`) semelhante à minha.

```
using Modulo1.Infraestructure;
using Modulo1.Modelo;
using SQLite.Net;
```

```

using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Dal
{
    public class ClienteDAL
    {
        private SQLiteConnection sqlConnection;

        public ClienteDAL()
        {
            this.sqlConnection = DependencyService.Get<IDatabaseConnection>().DbConnection();
            this.sqlConnection.CreateTable<Cliente>();
        }

        public void Add(Cliente cliente)
        {
            sqlConnection.Insert(cliente);
        }

        public void DeleteById(long id)
        {
            sqlConnection.Delete<Cliente>(id);
        }

        public void Update(Cliente cliente)
        {
            sqlConnection.Update(cliente);
        }

        public IEnumerable<Cliente> GetAll()
        {
            return (from t in sqlConnection.Table<Cliente>() select t).OrderBy(i => i.Nome).ToList();
        }

        public Cliente GetClienteById(long id)
        {
            return sqlConnection.Table<Cliente>().FirstOrDefault(t => t.ClienteId == id);
        }
    }
}

```

8.2 O MVVM — MODEL-VIEW-VIEW MODEL

O padrão de projeto MVVM, criado por John Goshmann, busca propiciar a separação de responsabilidades, possibilitando tornar um aplicativo fácil de ser mantido. Inicialmente, este padrão foi usado em aplicações WPF e Silverlight, não mudando sua essência para o uso por aplicações Xamarin Forms. Ele se assemelha em algumas situações com o MVC (*Model-View-Controller*) e ao MVP (*Model-View-Presenter*).

O MVVM mantém uma espécie de Façade (outro padrão de projeto) entre o modelo de negócios e a visão, que é a interface com o usuário. Veja a figura a seguir. Note que há uma ligação (binding) entre a view e a view-model, em que a view não tem acesso direto à camada de negócio, ficando isolada. Na camada view-model, ainda é possível implementar *commands*, que são também ligados e reagem a interações do usuário, o que chamamos de lógica da camada de apresentação.

MVC — MODEL-VIEW-CONTROLLER OU MODELO-VISÃO-CONTROLADOR

O padrão MVC busca dividir a aplicação em responsabilidades relativas à definição de sua sigla. A parte do Modelo trata as regras de negócio, o domínio do problema; a Visão busca levar ao usuário final informações a cerca do modelo, ou solicitar dados para registros. Desta maneira, o ASP.NET MVC busca estar próximo a este padrão.

MVP — MODEL-VIEW-PRESENTER OU MODELO-VISÃO-APRESENTAÇÃO

O MVP é uma derivação do MVC, usado também para construir principalmente interfaces gráficas. É projetado para facilitar os testes unitários automatizados e melhorar a separação de interesses em lógica de apresentação. O Presenter atua sobre Modelo e Visão, recuperando os dados dos repositórios e formatando-os para exibir na Visão.

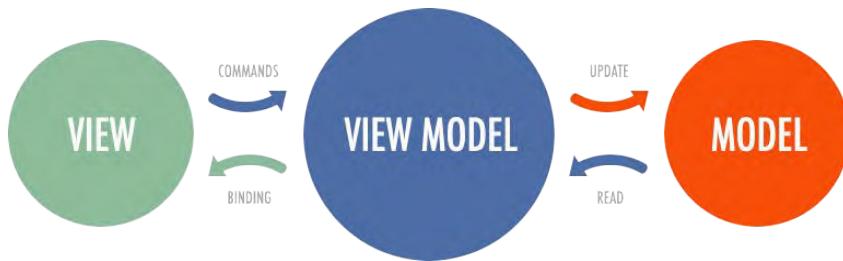


Figura 8.1: Representação do MVVM —
<https://erazerbrecht.wordpress.com/2015/10/13/mvvm-entityframework/>

De início, ao implementarmos a classe que representa a camada de modelo de visão para clientes, você notará que as propriedades de nossa classe de negócio estão presentes na classe de negócio da visão. Isso se deve ao fato de termos todas as propriedades nessa visão que estamos implementando.

Em situações diferentes e específicas, poderíamos ter uma classe de visão de negócio com menos propriedades que a de negócio, ou ainda composta por mais de uma classe de negócio. Vamos lá. Crie uma pasta chamada `ViewModel` em seu projeto, e nela uma classe chamada `ClienteViewModel`, deixando-a com o código semelhante à listagem a seguir.

Veja que a classe implementa `INotifyPropertyChanged`, o que implica na implementação do evento `PropertyChangedEventHandler`. Na sequência, verifique a definição dos campos e a implementação das propriedades. Observe que nos métodos `set()` é chamado `OnPropertyChanged()`, que atualiza a propriedade que sofreu alteração.

Ao final, tem-se a implementação de um `ICommand`, o método `Gravar()`, que deveria fazer a validação dos dados informados, mas isso fica contigo, ok? Note a chamada ao método `GetObjectFromView()`, que cria e retorna um objeto de nosso modelo, `Cliente`. Com a obtenção do objeto, ele é então submetido ao DAL para persistência. Observe também a criação do construtor, que recebe um objeto da classe `Cliente`.

```
using Modulo1.Dal;
using Modulo1.Modelo;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using Xamarin.Forms;

namespace Modulo1.ViewModel
{
    public class ClienteViewModel : INotifyPropertyChanged
    {
        private string _nome;
        private string _telefone;
        private string _endereco;
        private string _numero;
        private string _bairro;
        private string _cidade;
        private string _estado;

        public ClienteViewModel(Cliente cliente)
        {
            this._clienteId = cliente.ClienteId;
            this._nome = cliente.Nome;
            this._endereco = cliente.Endereco;
            this._numero = cliente.Numero;
            this._bairro = cliente.Bairro;
            this._cidade = cliente.Cidade;
            this._estado = cliente.Estado;
        }
    }
}
```

```

}

public string Nome
{
    get { return _nome; }
    set { _nome = value; OnPropertyChanged(); }
}

public string Telefone
{
    get { return _telefone; }
    set { _telefone = value; OnPropertyChanged(); }
}

public string Endereco
{
    get { return _endereco; }
    set { _endereco = value; OnPropertyChanged(); }
}

public string Numero
{
    get { return _numero; }
    set { _numero = value; OnPropertyChanged(); }
}

public string Bairro
{
    get { return _bairro; }
    set { _bairro = value; OnPropertyChanged(); }
}

public string Cidade
{
    get { return _cidade; }
    set { _cidade = value; OnPropertyChanged(); }
}

public string Estado
{
    get { return _estado; }
    set { _estado = value; OnPropertyChanged(); }
}

public ICommand Gravar
{
    get
    {

```

```

        var clienteDAL = new ClienteDAL();
        return new Command(() =>
        {
            clienteDAL.Add(GetObjectFromView());
            App.Current.MainPage.DisplayAlert("Inserção Cl
iente", "Cliente inserido com sucesso", "OK");
        });
    }

    private Cliente GetObjectFromView()
    {
        return new Cliente()
        {
            Nome = this.Nome,
            Telefone = this.Telefone,
            Endereco = this.Endereco,
            Numero = this.Numero,
            Bairro = this.Bairro,
            Cidade = this.Cidade,
            Estado = this.Estado
        };
    }

    public event PropertyChangedEventHandler PropertyChanged;

    private void OnPropertyChanged([CallerMemberName] string p
ropertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(p
ropertyName));
    }
}

```

Existem produtos que podem ser utilizados para implementar o MVVM em nossos modelos e visões, como o MVVM Cross (<https://mvvmcross.com/>) e o MVVM Light (<http://www.mvvmlight.net/>), dentre outros. Vale a pena uma investigação no que eles oferecem.

É importante que, para o uso destas ferramentas, você tenha um bom conhecimento de MVVM, embora os materiais disponibilizados nos sites dos produtos possam lhe fornecer conhecimento gradativo. Não faz parte do escopo deste livro um

aprofundamento em MVVM, ou o uso de ferramentas voltadas para este padrão.

8.3 LISTAGEM E INSERÇÃO DE CLIENTES

Para termos o padrão de nossas páginas criadas até este momento, trabalharemos primeiro com a implementação da página de listagem dos clientes registrados, embora na primeira execução não exista ainda nenhum cadastrado.

Sendo assim, na pasta Pages , crie a página ClientesListPage e implemente seu código para ser semelhante ao apresentado a seguir. Verifique que no código já estão previstos: alteração, exclusão e inserção de um novo cliente.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.Clientes.ClientesListPage"
        Title="Relação de clientes"
        BackgroundColor="#e6ffe6">
    <ContentPage.Content>
        <StackLayout HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand">
            <Grid>
                <Grid.Padding HorizontalOptions="FillAndExpand">
                    <OnPlatform x:TypeArguments="Thickness">
                        <OnPlatform.iOS>
                            5, 10, 5, 10
                        </OnPlatform.iOS>
                        <OnPlatform.WinPhone>
                            5, 0, 5, 35
                        </OnPlatform.WinPhone>
                        <OnPlatform.Android>
                            5, 10, 5, 10
                        </OnPlatform.Android>
                    </OnPlatform>
                </Grid.Padding>
                <Grid.RowDefinitions>
                    <RowDefinition Height="*"/>
                    <RowDefinition Height="Auto"/>
                </Grid.RowDefinitions>
```

```

<Grid.ColumnDefinitions>
    <ColumnDefinition Width="1*"/>
</Grid.ColumnDefinitions>

<ListView Grid.Row="0" Grid.Column="0" x:Name="lvClientes"
BackgroundColor="#e6ffe6"
HasUnevenRows="True" HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand">
    <ListView.ItemTemplate BackgroundColor="#e6ffe6">
        <DataTemplate>
            <ViewCell>
                <ViewCell.ContextActions>
                    <MenuItem Clicked="OnAlterarClick" CommandParameter="{Binding .}"
Text="Alterar" />
                    <MenuItem Clicked="OnRemoverClick" CommandParameter="{Binding .}"
Text="Remover" IsDestructive="True" />
                </ViewCell.ContextActions>
                <Grid Padding="5, 5, 20, 5">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="*"/>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="100*"/>
                    </Grid.ColumnDefinitions>

                    <Label Text="{Binding Nome}" TextColor="Blue" FontSize="Large"
Grid.Row="0" Grid.Column="0" />
                    <Label Text="{Binding Telefone}" TextColor="Green" FontSize="Small"
Grid.Row="1" Grid.Column="0"
HorizontalOptions="FillAndExpand"/>
                </Grid>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
<StackLayout Grid.Row="1" Grid.Column="0" Padding="0" BackgroundColor="#ff8c1a">
    <Button Text="Inserir novo cliente" x:Name="BtnNovoItem"
Image="icone_new.png"/>
</StackLayout>
</Grid>
</StackLayout>
</ContentPage.Content>

```

```
</ContentPage>
```

Para que seja possível o teste desta nova página, precisamos implementar os métodos que serão disparados pela interação com o usuário. A princípio apenas a assinatura dos métodos é implementada, o comportamento veremos mais adiante.

A listagem seguinte apresenta o código da classe para a página:

```
using Modulo1.Dal;
using Modulo1.Modelo;
using System;
using Xamarin.Forms;

namespace Modulo1.Pages.Clientes
{
    public partial class ClientesListPage : ContentPage
    {
        private ClienteDAL clienteDAL = new ClienteDAL();

        public ClientesListPage()
        {
            InitializeComponent();
            BtnNovoItem.Clicked += BtnNovoItemClick;
        }

        protected override void OnAppearing()
        {
            base.OnAppearing();
            lvClientes.ItemsSource = clienteDAL.GetAll();
        }

        private async void BtnNovoItemClick(object sender, EventArgs e) { }

        public async void OnAlterarClick(object sender, EventArgs e) { }

        public async void OnRemoverClick(object sender, EventArgs e) { }
    }
}
```

Agora, para que a página possa ser invocada, no menu da aplicação, na página `MenuPage`, é preciso implementar a opção

para acessá-la. Implemente isso em sua classe para que fique semelhante a <Button Text="Clientes" Image="icone_clientes.png" Clicked="ClientesOnClicked"/>. E para finalizar, é preciso implementar o método ClientesOnClicked(), tal como segue.

```
private async void ClientesOnClicked(object sender, EventArgs args)
{
    await Navigation.PushAsync(new ClientesListPage());
}
```

Depois disso implementado, vamos testar a aplicação? Sua página com a listagem dos clientes deve estar semelhante à apresentada na figura a seguir.

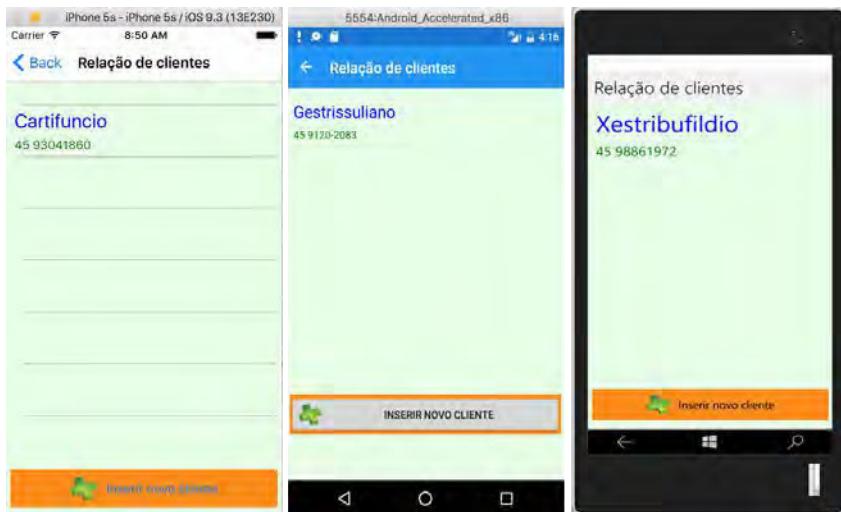


Figura 8.2: Página com a listagem dos clientes registrados

Tudo certo? Agora vamos trabalhar com a inserção de clientes. Teremos duas páginas: uma que servirá tanto para inserção como alteração (e você pode implementar depois a visualização detalhada dos dados, o que acha?); e outra que terá a apresentação da localização do cliente, o mapa.

A listagem a seguir apresenta o código XAML para a página ClientesCRUDPage . Observe, na definição dos controles <Entry> , que fazemos uso do Binding na propriedade Text . Isso significa que teremos um objeto associado a esta página, e que ele possui uma propriedade com o nome que segue a palavra Binding . Já veremos isso.

Veja que, ao final, estão implementados dois Buttons : um para inserir o cliente e outro para visualizar a sua localização no mapa. O que acha de dar uma melhorada na interface desta página? Quem sabe colocando um botão ao lado do outro, utilizando apenas uma imagem. :-)

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.Clientes.ClientesCRUDPage"
        Title="Título alterará de acordo à operação"
        BackgroundColor="#e6f2ff">

    <ContentPage.Content>
        <ScrollView>
            <StackLayout VerticalOptions="Center" Padding="0">
                <Grid Padding="5,10,5,10">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="100*"/>
                    </Grid.ColumnDefinitions>

                    <Entry Placeholder="Nome do Cliente" PlaceholderColor="Gray"
                           Grid.Row="0" Grid.Column="0" Text="{Binding Nome}"/>
                    <Entry Placeholder="Telefone" PlaceholderColor="Gray"
```

```

        Grid.Row="1" Grid.Column="0" Text="{Binding Tel
efone}"/>
    <Entry Placeholder="Endereço" PlaceholderColor="Gray"
        Grid.Row="2" Grid.Column="0" Text="{Binding End
ereco}"/>
    <Entry Placeholder="Número" PlaceholderColor="Gray"
        Grid.Row="3" Grid.Column="0" Text="{Binding Num
ero}"/>
    <Entry Placeholder="Bairro" PlaceholderColor="Gray"
        Grid.Row="4" Grid.Column="0" Text="{Binding Bai
rro}"/>
    <Entry Placeholder="Cidade" PlaceholderColor="Gray"
        Grid.Row="5" Grid.Column="0" Text="{Binding Cid
ade}"/>
    <Entry Placeholder="Estado" PlaceholderColor="Gray"
        Grid.Row="6" Grid.Column="0" Text="{Binding Est
ado}"/>
    <Button Text="Gravar dados do cliente" Command="{Binding
Gravar}"
        Grid.Row="7" Grid.Column="0"/>
    <Button Text="Visualizar no mapa" Command="{Binding Mapa
}"/>
        Grid.Row="8" Grid.Column="0"/>
    </Grid>
</StackLayout>
</ScrollView>
</ContentPage.Content>
</ContentPage>
```

Precisamos implementar agora a classe para esta página. Lembro de que esta página atenderá tanto ao processo de inserção como de alteração dos dados de um cliente. Desta maneira, codifique a sua para que fique igual ao código apresentado na sequência.

Observe que substituí o construtor padrão por um construtor que receberá o objeto que será manipulado pela visão. É no momento de sua execução, quando a classe for instanciada, que saberemos se ela deverá inserir os dados ou alterá-los. Sendo assim, note que o Id do cliente é utilizado como um flag para alterar o título da página.

```
using Modulo1.Modelo;
using Modulo1.ViewModel;
using Xamarin.Forms;
```

```

namespace Modulo1.Pages.Clientes
{
    public partial class ClientesCRUDPage : ContentPage {
        private ClienteViewModel clienteViewModel = new ClienteView
        Model();
        public ClientesCRUDPage(Cliente cliente)
        {
            InitializeComponent();
            if (cliente.ClienteId == null)
            {
                Title = "Inserção de um novo cliente";
            } else {
                Title = "Alteração de dados do cliente";
            }
            clienteViewModel = new ClienteViewModel(cliente);
            BindingContext = clienteViewModel;
        }
    }
}

```

Para testarmos, precisamos agora implementar o comportamento para o botão de inserção de um novo cliente, presente na página de listagem. Codifique para que seu método fique semelhante ao apresentado na sequência.

É neste comportamento que instanciaremos a página implementada anteriormente, para que o usuário possa inserir um novo cliente. Veja que o objeto enviado como argumento é apenas a instanciação da classe, um objeto novo, sem dados.

```

private async void BtnNovoItemClick(object sender, EventArgs e)
{
    await Navigation.PushAsync(new ClientesCRUDPage(new Cliente()))
};

```

Se tudo estiver certo, sua aplicação deverá estar semelhante à apresentada na figura a seguir. Ao inserir clientes, informe o telefone de maneira completa, como +55[OPERADORA][DDD] [NÚMERO], pois usaremos isso no próximo capítulo e seria legal que os números dos clientes estejam bem informados.

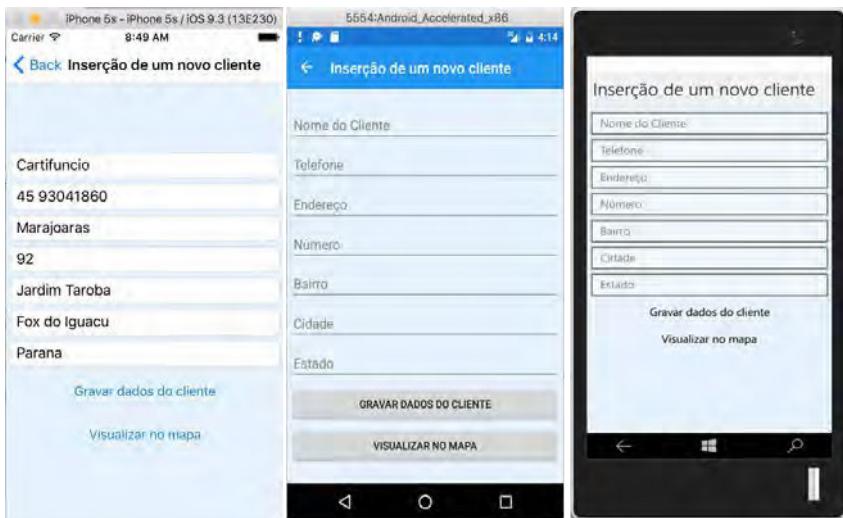


Figura 8.3: Página para inserção de um cliente

8.4 LOCALIZAÇÃO DO CLIENTE EM UM MAPA

Para que possamos trabalhar com mapas no Xamarin Forms de maneira simplificada, sem precisar trabalharmos com especificidades de cada plataforma, faremos uso de mais um componente, o Xamarin Forms Maps. Vamos instalá-lo para a solução, o que levará a instalação para todos os projetos contidos nela.

Clique com o botão direito no nome da solução, e depois na opção `Manage Nuget Packages for Solution`. Verifique isso na figura a seguir.



Figura 8.4: Instalação do componente Xamarin Forms Maps

Após a instalação deste componente, tive problemas no build da aplicação Android no Windows. Ele acusava não encontrar um determinado pacote, o `Xamarin.Android.Support.Vector.Drawable`, versão 23.3.0 . Precisei remover todas as pastas `Xamarin.Android*` e a zips de `C:\Users\USERNAME\AppData\Local\Xamarin` . No Mac, este problema não ocorreu. Fica a dica.

Vendo o mapa no iOS

Mesmo instalando o componente, precisamos realizar uma alteração nos projetos específicos das plataformas. Começaremos no projeto iOS, que é o mais simples de todos. No arquivo `AppDelegate` , no método `FinishedLaunching()` , após a instrução `global::Xamarin.Forms.Forms.Init();` , insira a `global::Xamarin.FormsMaps.Init();` .

Com isso, precisamos apenas implementar a interface com o usuário agora. Sendo assim, na pasta `Clientes` , crie uma nova página, dê a ela o nome `ClientesMapPage` e implemente-a de acordo com o código seguinte. Observe a declaração do tipo `maps` na tag `<ContentPage>` e a definição do mapa dentro de um `StackLayout` .

```
<?xml version="1.0" encoding="utf-8" ?>
```

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.Clientes.ClientesMapPage"
        xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=
=Xamarin.Forms.Maps"
        Title="Localização cliente"
        BackgroundColor="#ffd9b3">
    <StackLayout VerticalOptions="FillAndExpand" HorizontalOptions=-
"FillAndExpand" Padding="10, 20, 10, 10">
        <maps:Map x:Name="MyMap"
            IsShowingUser="true"
            MapType="Street"
            />
    </StackLayout>
</ContentPage>

```

Precisamos agora implementar a classe para esta página, pois é ela que terá a responsabilidade de apresentar, no mapa, a localização do cliente, com base no endereço informado na página de inclusão. Desta maneira, é importante que, para funcionamento, você saiba que deverá digitar todos os dados do cliente para uma localização precisa.

Testei apenas com o campo endereço e número, e já foi suficiente em meu caso, mas fica a recomendação. Se você for limpar os dados do cliente após a gravação, precisa clicar no botão de visualização no mapa antes de gravar. Veja esta classe no código a seguir.

Observe que o construtor recebe o endereço como argumento e o atribui a um campo da classe. O método do evento `OnAppearing()` realiza a chamada para o método implementado `PutAddressInTheMap()`, que faz uso do endereço recebido na inicialização da classe.

```

using System.Linq;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Maps;

namespace Modulo1.Pages.Clientes

```

```

{
    public partial class ClientesMapPage : ContentPage
    {
        private string endereco;

        public ClientesMapPage(string endereco)
        {
            InitializeComponent();
            this.endereco = endereco;
        }

        protected async override void OnAppearing()
        {
            base.OnAppearing();
            await PutAddressInTheMap();
        }

        private async Task PutAddressInTheMap()
        {
            var geoCoder = new Geocoder();
            var position = await geoCoder.GetPositionsForAddressAs
ync(endereco);
            MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(position
.First(), Distance.FromKilometers(0.3f)));

            var pin = new Pin()
            {
                Position = position.First(),
                Label = "Residência cliente",
                Address = endereco
            };

            MyMap.Pins.Add(pin);
        }
    }
}

```

Execute sua aplicação iOS, informe os dados para o cliente e clique no botão referente ao mapa. Se tudo estiver certo, seu emulador deverá estar semelhante à figura apresentada na sequência.

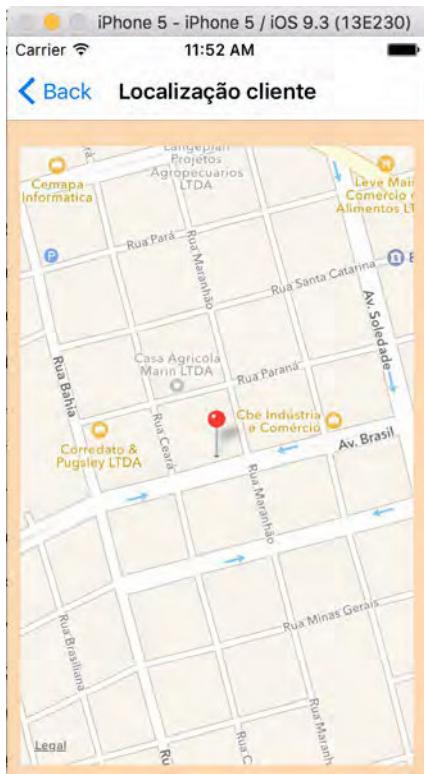


Figura 8.5: Visualização do mapa no iOS

Vendo o mapa no Android

A preparação do projeto para visualização do mapa em um dispositivo Android é um pouco mais trabalhosa, pois existe a necessidade de obtenção de uma chave de API da Google. O procedimento para isso está detalhado no link oficial do Xamarin, em

https://developer.xamarin.com/guides/android/platform_features/maps_and_location/maps/obtaining_a_google_maps_api_key/.

Embora exista esta documentação, passarei aqui as etapas. A primeira delas é obter a `Signing Key Fingerprint` com a execução da instrução:

- No Mac:

```
keytool -list -v -keystore /Users/[USERNAME]/.local/share/Xamarin/Mono\ for\ Android/debug.keystore -alias androiddebugkey -storepass android -keypass android
```

- No Windows:

```
keytool -list -v -keystore "C:\Users\[USERNAME]\AppData\Local\Xamarin\Mono for Android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

Substitua o [USERNAME]. A figura a seguir apresenta o resultado da execução do comando anterior. Para o Mac, você precisará guardar o valor para o SHA1.

```
Everton — bash — 80x31
MacBook-Pro-de-Everton:~ Everton$ keytool -list -v -keystore /Users/Everton/.local/share/Xamarin/Mono\ for\ Android/debug.keystore -alias androiddebugkey -storepass android -keypass android
Nome do alias: androiddebugkey
Data de criação: 05/07/2016
Tipo de entrada: PrivateKeyEntry
Comprimento da cadeia de certificados: 1
Certificado[1]:
Proprietário: CN=Android Debug, O=Android, C=US
Emissor: CN=Android Debug, O=Android, C=US
Número de série: 56d85bc5
Válido de: Tue Jul 05 14:15:33 BRT 2016 a: Thu Jun 28 14:15:33 BRT 2046
Fingerprints do certificado:
    05: [REDACTED]
    SHA1: [REDACTED]
    SHA256: [REDACTED]
D1:14:02:DF:A0:51:C7:5F:58:0A:E5
    Nome do algoritmo de assinatura: SHA256withRSA
    Versão: 3

Extensões:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: [REDACTED] .....U.
0010: 6D 60 EE 4C ] ] m'.L

MacBook-Pro-de-Everton:~ Everton$
```

Figura 8.6: Geração da Signing Key FingerPrint

Com a chave criada, precisamos da criação da API para o

projeto na Google, e isso pode ser feito em <https://console.developers.google.com>, que solicitará os dados de sua conta Google. Com o acesso realizado, é preciso criar o projeto. Veja os passos na figura a seguir.

Ao clicar em **Criar projeto**, uma janela solicitará o nome que o projeto terá. Eu informei **ccfoods** como nome da minha aplicação.

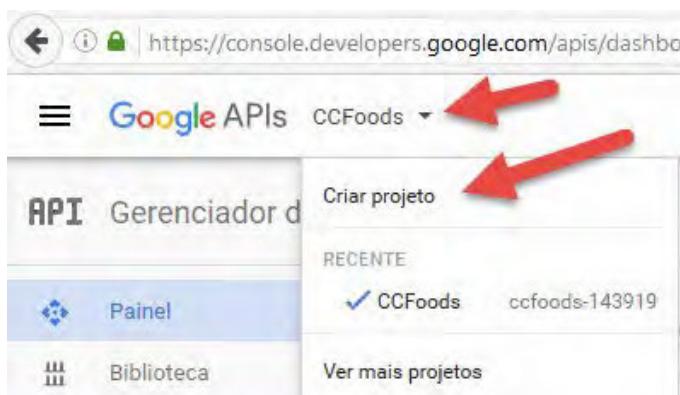


Figura 8.7: Criação de um projeto no Google API

Após criar sua aplicação, o Google a disponibilizará no mesmo controle combobox que você acessou para criar o projeto. Ela deverá estar ativa para você. No dashboard, em **API do Google Maps**, clique em **Google Maps Android API**, tal qual exibe a figura a seguir.

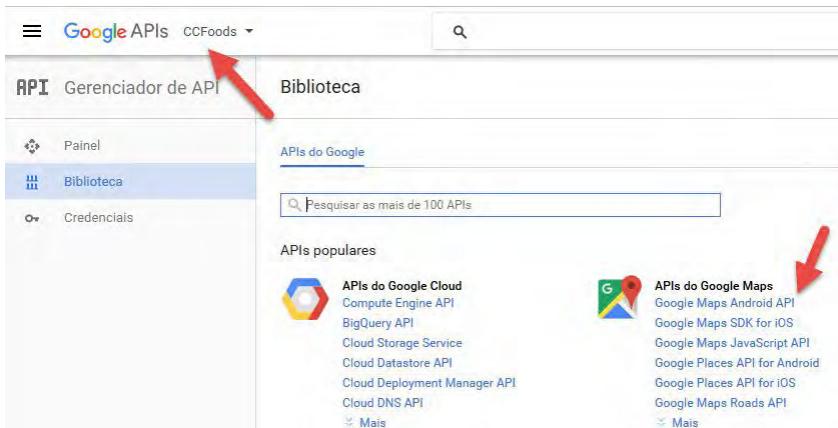


Figura 8.8: Seleção da Google Maps Android API

No topo da página que se abre, você deve encontrar um link chamado **Ativar**, clique nele. Após a ativação, é exibido um botão, no topo direito da página, chamado **Ir para Credenciais**. Precisamos destas credenciais para que a API seja habilitada.

Você pode também acessar as credenciais por meio do menu do lado esquerdo da página, clicando em **Credenciais**, e depois em **Criar credenciais**. Escolha a opção **Chave de API**. Veja a figura que se segue.

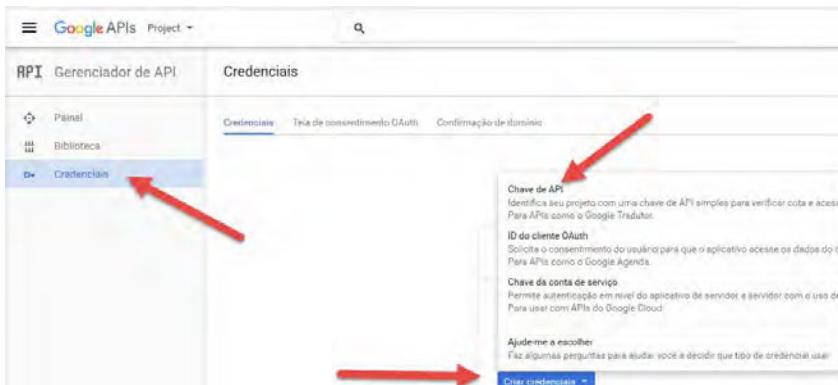


Figura 8.9: Criação das credenciais para a Google Maps Android API

Ao confirmar a opção, uma janela com a chave da API será exibida. Copie-a, pois será usada no nosso projeto Android. Clique no botão **Restringir chave**. Na página que se abre, você precisa marcar a opção **Apps para Android** e, ao final da página, clicar no botão **Adicionar nome do pacote e impressão digital**.

O nome do pacote você encontrará no arquivo **AndroidManifest.xml**, que está na pasta **Properties**, logo na primeira linha do arquivo, ao final dela. A **Impressão digital para certificação SHA-1** é a que criamos no início desta sessão. Fornecidas estas informações, salve sua configuração de credenciais.

Agora, com tudo configurado, precisamos fazer uso da API Key. Para isso, no arquivo **AndroidManifest.xml** que comentei no parágrafo anterior, insira a implementação a seguir. Verifique se você já não tem um elemento **<application>** em seu arquivo. Se tiver, apenas insira as tags **<meta-data>**. É óbvio que o valor para a **API_KEY** que está indicado não é um valor real. :-)

```
<application android:label="CC Foods" android:name="CC Foods" android:icon="@drawable/icon">
    <meta-data android:name="com.google.android.geo.API_KEY" android:value="B1zaSyBK-5LQbv1X8RVGxC29c1l4f2_rx-DPRTB" />
    <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version" />
</application>
```

Estamos quase terminando, mais alguns passos apenas. Precisamos configurar a aplicação para que ela tenha acesso a determinados recursos do dispositivo Android. Podemos fazer isso clicando com o botão direito do mouse sobre o nome do projeto, escolhendo **Properties** e depois a opção **Android Manifest**.

No final da janela, existe um conjunto de opções. As que você precisa marcar como habilitadas são:

- **ACCESS_COARSE_LOCATION**
- **ACCESS_FINE_LOCATION**

- ACCESS_LOCATION_EXTRA_COMMANDS
- ACCESS_MOCK_LOCATION
- ACCESS_WIFI_STATE
- INTERNET
- ACCESS_NETWORK_STATE

Esta configuração se faz necessária para que seja possível acessar os recursos de localização do dispositivo.

Agora só precisamos configurar o projeto Android para que faça uso do `Xamarin.Forms.Maps`, e isso deve ser feito na classe `MainActivity`. Logo após a instrução `global::Xamarin.Forms.Forms.Init(this, bundle);`, que está no método `OnCreate()`, insira `global::Xamarin.FormsMaps.Init(this, bundle);`.

Agora sim, podemos testar nossa aplicação Android. Veja se funciona para você. Se tudo estiver certinho, sua aplicação deverá estar semelhante à apresentada na figura a seguir.

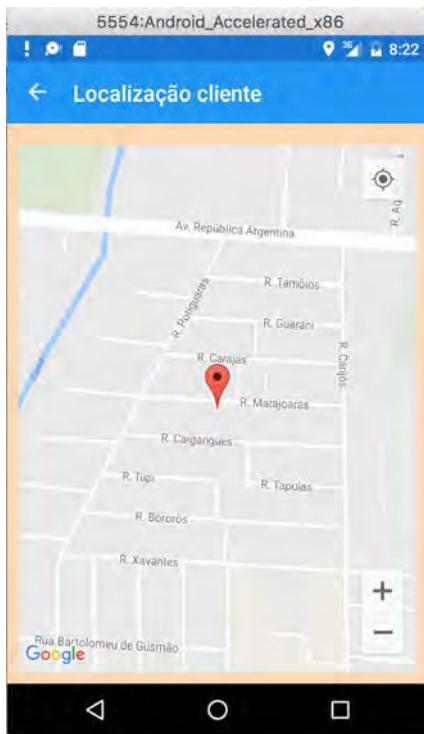


Figura 8.10: Visualização do mapa no Android

Vendo o mapa no Windows

O processo necessário para visualizar uma localização no mapa de um dispositivo Windows é mais simples do que o requerido pelo Android. Você precisa acessar o link <https://www.bingmapsportal.com>, realizar o login com uma conta Microsoft, depois no menu superior, clique em My Accounts e em seguida em My Keys .

Na página que é exibida, clique no link para criar uma nova chave e uma página com o formulário apresentado na figura a seguir é mostrada. O campo Application URL é opcional. Clique no botão Create para a geração de sua chave.

Create key

Application name *

CCFoods

Application URL

Enter application URL

Key type *

Basic

What's This

Application type *

Universal Windows App

Create **Cancel**

* Required field

The screenshot shows a 'Create key' dialog box. At the top is the title 'Create key'. Below it is a field labeled 'Application name *' with the value 'CCFoods' highlighted in yellow. There is a 'What's This?' link next to the 'Key type *' label. The 'Key type' dropdown is set to 'Basic'. The 'Application type *' dropdown is set to 'Universal Windows App'. At the bottom are two buttons: a blue 'Create' button and a white 'Cancel' button. A note at the bottom left indicates that the application name field is required.

Figura 8.11: Formulário de criação de chaves para aplicações Microsoft

Com a chave criada, precisamos informá-la no projeto UWP. Abra o arquivo da classe `MainPage` e, no construtor, após a instrução `InitializeComponent();`, insira a instrução `Xamarin.FormsMaps.Init("INSERT_MAP_KEY_HERE");`, informando sua chave criada.

Precisamos agora dar permissão à aplicação para fazer uso de um recurso do dispositivo. Dê duplo clique no arquivo `Package.appmanifest`, selecione a opção `Capabilities` e marque a opção `Location`. Veja a figura a seguir.

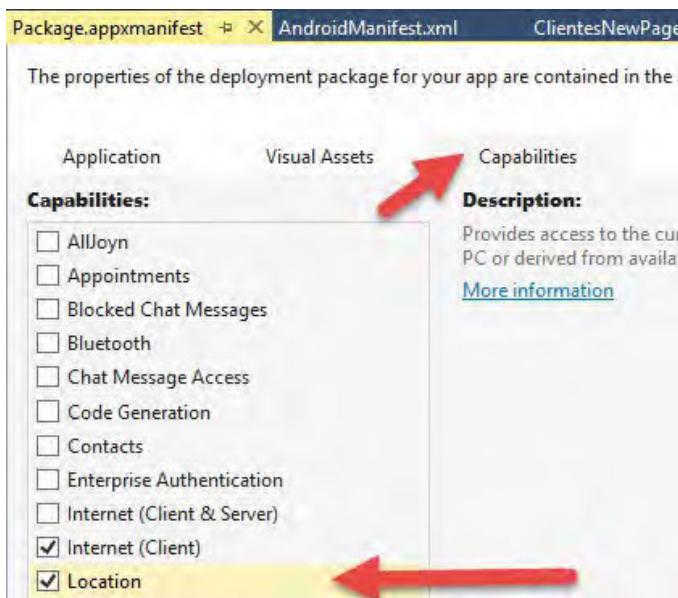


Figura 8.12: Configurando recursos para a aplicação UWP

Execute sua aplicação. Se tudo estiver certo, a visualização do mapa deve estar semelhante a apresentada na figura a seguir.



Figura 8.13: Visualização do mapa no Windows

8.5 ALTERAÇÃO E REMOÇÃO DE UM CLIENTE JÁ INSERIDO

Para implementarmos a funcionalidade de alteração dos dados de um cliente, precisamos implementar o método que será disparado quando o usuário clicar no botão `Alterar`, que aparece quando ele desliza uma linha para a esquerda, ou pressiona por um determinado tempo a linha em questão, dependendo da plataforma, como já vimos anteriormente.

O método já existe na classe, falta apenas o comportamento e o teste. Implemente-o para que fique semelhante ao apresentado na

sequência. Teste sua aplicação e tente alterar um cliente, é para todos os dados informados para ele já aparecerem no formulário. Muito legal, não é?

```
public async void OnAlterarClick(object sender, EventArgs e)
{
    var mi = ((MenuItem)sender);
    var cliente = mi.CommandParameter as Cliente;
    await Navigation.PushAsync(new ClientesCRUDPage(cliente));
}
```

Finalizando nossa atividade para este capítulo, implementaremos o comportamento para o método de exclusão para os itens da listagem. Procure deixá-lo semelhante ao apresentado na sequência.

```
public async void OnRemoverClick(object sender, EventArgs e)
{
    var mi = ((MenuItem)sender);
    var cliente = mi.CommandParameter as Cliente;
    var opcao = await DisplayAlert("Confirmação de exclusão",
        "Confirma excluir o cliente " + cliente.Nome.ToUpper() + "?",
        "Sim", "Não");
    if (opcao)
    {
        clienteDAL.DeleteById((long)cliente.ClienteId);
        this.lvClientes.ItemsSource = clienteDAL.GetAll();
    }
}
```

8.6 CONCLUSÃO

Foi mais leve o trabalho neste capítulo. Nele fizemos uso de mais um recurso específico dos dispositivos, utilizando mais um componente. Isso sem precisar implementarmos especificidades em cada plataforma, apenas a inicialização do componente.

O uso de mapas pode trazer mais recursos do que foi apresentado. É claro que, para implementarmos isso, precisamos estudar mais o componente e os recursos oferecidos por ele. Sugiro que você leia a documentação disponibilizada pelo Xamarin sobre

os mapas. Ela traz tudo o que precisa.

Também foi introduzido o conceito e a implementação de MVVM, o que possibilita um isolamento entre camadas de visão e negócio. No próximo capítulo, trabalharemos os pedidos realizados pelos clientes, e buscaremos traçar uma rota para a entrega, além de buscarmos a localização do dispositivo móvel que está executando a aplicação, o que possibilitará ao cliente o acompanhamento da entrega do pedido.

Faremos também uso do envio de mensagens para informar o andamento do pedido, para o cliente, até a entrega. Respire fundo e vamos lá.

CAPÍTULO 9

PEDIDO DE VENDA, ROTAS EM MAPAS E SMS

No capítulo anterior, para uso dos mapas, foi utilizado um componente que permite a localização de um determinado endereço ou posição em um mapa. Neste capítulo, faremos uso, em relação a mapas, dos serviços específicos de cada plataforma: o Mapas para iOS, o Google Maps para o Android e o Bing Maps para Microsoft. Isso nos permitirá traçar rotas para a entrega de pedidos.

Usaremos também outro componente que permite obter a localização atual do dispositivo, o que nos possibilitará o registro das rotas e, com isso, o acompanhamento do deslocamento do entregador até seu destino, por parte do cliente.

Em relação a lógica de negócio aplicada neste capítulo, será a de Pedidos. Registraremos o pedido do cliente e as etapas dele, desde sua abertura e, a cada mudança no processo, uma mensagem será enviada a ele. Utilizaremos também um componente para o uso destas mensagens (SMS).

Tudo isso gerará muito código, mas nada assustador. Lembre-se de sempre procurar inovar em suas interfaces com o usuário com base nas técnicas que apresentei nos capítulos anteriores. Vamos ao trabalho.

9.1 CLASSES DE MODELO PARA REGISTRO DE

PEDIDOS

Para o registro de um pedido, precisaremos de uma classe `Pedido` e uma `ItemPedido`. A classe `Pedido` estará associada com a classe `Cliente` e a `ItemPedido`. Já a classe `ItemPedido` estará associada com `Pedido` e `ItemCardapio`. Com estas associações, temos que um cliente realiza um pedido, com um ou mais itens, onde cada item de pedido está associado a um item do cardápio.

Veja na sequência a implementação para a classe `Pedido`. Observe a definição de quatro propriedades relacionadas à data e à hora. Elas serão flags para identificação da situação do pedido, e este estado poderá ser recuperado pela propriedade `Situacao`, que é apenas de leitura.

Veja os atributos para a chave primária, estrangeira e relacionamentos. Veja o cascamenteamento para a propriedade `Itens`. Ele permitirá que, ao inserirmos um pedido na base de dados, todos os itens relacionados a ele sejam também inseridos no mesmo processo. Como temos uma associação com `Entregador` e que será mapeada para um relacionamento, você precisa, na classe `Entregador`, inserir a configuração para chave primária, pois ainda não temos isso nela.

```
using SQLite.Net.Attributes;
using SQLiteNetExtensions.Attributes;
using System;
using System.Collections.Generic;

namespace Modulo1.Modelo
{
    public class Pedido
    {
        [PrimaryKey, AutoIncrement]
        public long? PedidoId { get; set; }
        public DateTime DataEHoraPedido { get; set; }
        public DateTime? DataEHoraProducao { get; set; }
        public DateTime? DataEHoraEntrega { get; set; }
```

```

public DateTime? DataEHoraEntregue { get; set; }
public string Situacao { get
{
    if (DataEHoraProducao == null)
        return "Aberto";
    else if (DataEHoraEntrega == null)
        return "Produção";
    else if (DataEHoraEntregue == null)
        return "Em entrega";
    else
        return "Fechado";
} }

public double Total { get
{
    return this.Itens.Sum(i => i.ValorUnitario * i.Quantidade);
} }

[ForeignKey(typeof(Cliente))]
public long? ClienteId { get; set; }

[ManyToOne(CascadeOperations = CascadeOperation.CascadeRead)]
public Cliente Cliente { get; set; }

[ForeignKey(typeof(Entregador))]
public long? EntregadorId { get; set; }

[ManyToOne(CascadeOperations = CascadeOperation.CascadeRead)]
public Entregador Entregador { get; set; }

[OneToMany(CascadeOperations = CascadeOperation.CascadeInsert)]
public List<ItemPedido> Itens { get; set; }
}
}

```

Agora, vamos implementar a classe `ItemPedido`, que tem seu código apresentado na sequência. Note que implementei a sobrescrita dos métodos `Equals()` e `GetHashCode()`, para que possamos trabalhar a identidade do objeto (não estou falando da chave primária, ok?). Lembro de que esta implementação é importante sempre que formos trabalhar com coleções, mas nada

impede de você fazer isso em todas as suas classes. :-)

O código relacionado à chave primária, estrangeira e associações segue a mesma lógica dos já apresentados até aqui. Não há nada de novo nesta classe.

```
using SQLite.Net.Attributes;
using SQLiteNetExtensions.Attributes;

namespace Modulo1.Modelo
{
    public class ItemPedido
    {
        [PrimaryKey, AutoIncrement]
        public long? ItemPedidoId { get; set; }
        public double Quantidade { get; set; }
        public double ValorUnitario { get; set; }

        [ForeignKey(typeof(Pedido))]
        public long? PedidoId { get; set; }

        [ForeignKey(typeof(ItemCardapio))]
        public long? ItemCardapioId { get; set; }

        [ManyToOne(CascadeOperations = CascadeOperation.CascadeRea
d)]
        public Pedido Pedido { get; set; }

        [ManyToOne(CascadeOperations = CascadeOperation.CascadeRea
d)]
        public ItemCardapio ItemCardapio { get; set; }

        public override bool Equals(object obj)
        {
            ItemPedido itemPedido = obj as ItemPedido;
            if (itemPedido == null)
            {
                return false;
            }

            return (ItemPedidoId.Equals(itemPedido.ItemPedidoId));
        }

        public override int GetHashCode()
        {
            return ItemPedidoId.GetHashCode();
        }
    }
}
```

```
    }  
}
```

9.2 DAL PARA PEDIDOS

Com o modelo implementando, já prevendo o uso do SQLite, precisamos agora implementar a classe `PedidoDAL`. Crie-a na pasta `Dal` de seu projeto, com o código apresentado na sequência.

Veja o método `Add()` que invoca o `InsertWithChildren()` para que, ao se inserir um pedido, seus itens também o sejam. Depois, verifique a criação de diversos métodos `GetAll...()`, que retornam registros de acordo com o estado dos pedidos, conforme o nome de cada método. Os demais métodos são semelhantes aos vistos anteriormente.

```
using Modulo1.Infraestructure;  
using Modulo1.Modelo;  
using SQLite.Net;  
using SQLiteNetExtensions.Extensions;  
using System.Collections.Generic;  
using System.Linq;  
using Xamarin.Forms;  
  
namespace Modulo1.Dal  
{  
    public class PedidoDAL  
    {  
        private SQLiteConnection sqlConnection;  
  
        public PedidoDAL()  
        {  
            this.sqlConnection = DependencyService.Get<IDatabaseCo  
nnexion>().DbConnection();  
            this.sqlConnection.CreateTable<Pedido>();  
            this.sqlConnection.CreateTable<ItemPedido>();  
        }  
  
        public void Add(Pedido pedido)  
        {  
            sqlConnection.InsertWithChildren(pedido);  
        }  
    }  
}
```

```

public void DeleteById(long id)
{
    sqlConnection.Delete<Pedido>(id);
}

public IEnumerable<Pedido> GetAllWithChildren()
{
    return sqlConnection.GetAllWithChildren<Pedido>().OrderBy(i => i.Cliente.Nome).ToList();
}

public IEnumerable<Pedido> GetAbertosWithChildren()
{
    return sqlConnection.GetAllWithChildren<Pedido>().Where(p => p.DataEHoraProducao == null).OrderBy(i => i.Cliente.Nome).ToList();
}

public IEnumerable<Pedido> GetEmProducaoWithChildren()
{
    return sqlConnection.GetAllWithChildren<Pedido>().Where(p => p.DataEHoraProducao != null && p.DataEHoraEntrega == null).OrderBy(i => i.Cliente.Nome).ToList();
}

public IEnumerable<Pedido> GetEmEntregaWithChildren()
{
    return sqlConnection.GetAllWithChildren<Pedido>().Where(p => p.DataEHoraEntrega != null && p.DataEHoraEntregue == null).OrderBy(i => i.Cliente.Nome).ToList();
}

public IEnumerable<Pedido> GetFechadosWithChildren()
{
    return sqlConnection.GetAllWithChildren<Pedido>().Where(p => p.DataEHoraEntregue != null).OrderBy(i => i.Cliente.Nome).ToList();
}

public void Update(Pedido pedido)
{
    sqlConnection.Update(pedido);
}

```

```
    }  
}
```

9.3 A LISTAGEM DOS PEDIDOS

Como foi o padrão que adotei em todas as nossas páginas, começarei a implementar a interface com o usuário pela página de listagem dos pedidos e, a partir dela, dispararmos os demais serviços. Na sequência está o código desta página. Tem bastante código, mas procurei separá-los por meio de comentários, para facilitar sua leitura e compreensão.

Crie a página `PedidosPage` dentro da pasta `Pedidos` , que deverá ser criada em `Pages` . A visualização desta página só será possível após a implementação dos métodos invocados por ela na classe. Logo veremos isso.

Note, na listagem, a criação de um layout principal e um layout para cada uma das quatro linhas configuradas. Veja os comentários antes de cada um destes layouts. Observe o uso de imagens, você precisará dos arquivos delas.

```
<?xml version="1.0" encoding="utf-8" ?>  
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
  
             x:Class="Modulo1.Pages.Pedidos.PedidosPage"  
             Title="Relação de pedidos"  
             BackgroundColor="#e6ffe6">  
  
    <ContentPage.Content>  
        <!-- Layout inicial para a página -->  
        <StackLayout HorizontalOptions="FillAndExpand" VerticalOptions:  
                    "FillAndExpand">  
            <Grid>  
                <Grid.RowDefinitions>  
                    <RowDefinition Height="Auto"/>  
                    <RowDefinition Height="Auto"/>  
                    <RowDefinition Height="*"/>  
                    <RowDefinition Height="Auto"/>  
                </Grid.RowDefinitions>  
                <Grid.ColumnDefinitions>
```

```

        <ColumnDefinition Width="1*" />
    </Grid.ColumnDefinitions>

<!-- Layout que apresenta o estado dos pedidos que estão sendo exibidos no ListView.
Este Label terá seu valor alterado de acordo ao filtro estabelecido pelos
botões que estão na sequência -->
<StackLayout Grid.Row="0" Grid.Column="0" Padding="0" BackgroundColor="#99ff33">
    <Label Text="Todos os pedidos" TextColor="Blue" FontSize="Large"
x:Name="tituloConsulta" HorizontalOptions="Center" />
</StackLayout>

<!-- Layout com coleção de botões para aplicação de filtros nos pedidos registrado.
Note a existência de imagens para cada um deles e que a interação com eles se dá por meio de Gestures -->
<StackLayout Grid.Row="1" Grid.Column="0" Padding="0" BackgroundColor="#99ff33">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="2*" />
        </Grid.ColumnDefinitions>

        <Image Source="pedidos.todos.png" HorizontalOptions="Center"
Grid.Row="0" Grid.Column="0" >
            <Image.GestureRecognizers>
                <TapGestureRecognizer
                    Tapped="OnTappedTodos"
                    NumberOfTapsRequired="1" />
            </Image.GestureRecognizers>
        </Image>

        <Image Source="pedidos.abertos.png" HorizontalOptions="Center"
Grid.Row="0" Grid.Column="1" >

```

```

<Image.GestureRecognizers>
    <TapGestureRecognizer
        Tapped="OnTappedAbertos"
        NumberOfTapsRequired="1" />
</Image.GestureRecognizers>
</Image>

<Image Source="pedidos_producao.png" HorizontalOptions="Center"
        Grid.Row="0" Grid.Column="2" >
<Image.GestureRecognizers>
    <TapGestureRecognizer
        Tapped="OnTappedEmProducao"
        NumberOfTapsRequired="1" />
</Image.GestureRecognizers>
</Image>

<Image Source="pedidos_ementrega.png" HorizontalOptions="Center"
        Grid.Row="0" Grid.Column="3" >
<Image.GestureRecognizers>
    <TapGestureRecognizer
        Tapped="OnTappedEmEntrega"
        NumberOfTapsRequired="1" />
</Image.GestureRecognizers>
</Image>

<Image Source="pedidos_fechados.png" HorizontalOptions="Center"
        Grid.Row="0" Grid.Column="4" >
<Image.GestureRecognizers>
    <TapGestureRecognizer
        Tapped="OnTappedFechados"
        NumberOfTapsRequired="1" />
</Image.GestureRecognizers>
</Image>
</Grid>
</StackLayout>

<!-- Layout para o ListView. As interações são dadas pelo Tapped nos itens e pelas ContextActions definidas, as quais alterarão o estado do pedido. O Tapped no item abrirá o mapa para visualização da rota.-->

<StackLayout Grid.Row="2" Grid.Column="0" Padding="0, 0, 0, 0" BackgroundColor="#99ff33">
    <ListView x:Name="lvClientes" BackgroundColor="#e6ffe6">

```

```

        HasUnevenRows="True" HorizontalOptions="FillAndE
xpand"
        VerticalOptions="FillAndExpand" ItemTapped="OnIT
emTapped">
    <ListView.ItemTemplate BackgroundColor="#e6ffe6" >
        <DataTemplate>
            <ViewCell>
                <ViewCell.ContextActions>
                    <MenuItem Clicked="OnProduzirClick" CommandParam
eter="{Binding .}"
                        Text="Produzir" />
                    <MenuItem Clicked="OnEntregarClick" CommandParam
eter="{Binding .}"
                        Text="Entregar" />
                    <MenuItem Clicked="OnFcharClick" CommandParamet
er="{Binding .}"
                        Text="Fchar" />
                    <MenuItem Clicked="OnVerificarRotaClick" Command
Parameter="{Binding .}"
                        Text="Rota" />
                </ViewCell.ContextActions>
                <Grid Padding="5, 5, 20, 5">
                    <Grid.RowDefinitions >
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="*"/>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="70*"/>
                        <ColumnDefinition Width="30*"/>
                    </Grid.ColumnDefinitions>

                    <Label Text="{Binding Cliente.Nome}" TextColor="
Blue" FontSize="Large"
                        Grid.Row="0" Grid.Column="0" />
                    <Label Text="{Binding PedidoId}" TextColor="Red"
FontSize="Small"
                        Grid.Row="0" Grid.Column="1" HorizontalO
ptions="EndAndExpand"/>
                    <Label Text="{Binding Cliente.Telefone}" TextCol
or="Green" FontSize="Small"
                        Grid.Row="1" Grid.Column="0"
                        HorizontalOptions="FillAndExpand"/>
                    <Label Text="{Binding Situacao}" TextColor="Red"
FontSize="Small"
                        Grid.Row="1" Grid.Column="1"
                        HorizontalOptions="EndAndExpand"/>
                </Grid>
            </ViewCell>

```

```

        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
</StackLayout>

<!-- Layout com botão responsável pela abertura da página de inserção de um novo pedido-->
<StackLayout Grid.Row="3" Grid.Column="0" Padding="0" BackgroundColor="#99ff33">
    <Button Text="Inserir novo pedido" x:Name="BtnNovoPedido" Image="icone_new.png"/>
</StackLayout>
</Grid>
</StackLayout>
</ContentPage.Content>
</ContentPage>
```

Se você verificou o código anterior atentamente (e sei que o fez), notou as chamadas a diversos métodos, e estes estão implementados na classe da página, que tem sua representação na sequência, também com comentários. Observe os métodos que capturam os gestos de interação com as imagens e ListView (Tapped).

Para que você possa testar sua aplicação neste momento e possa ver se está tudo bem, mesmo que sem dados, deixei o comportamento para os métodos relacionados a novo item, verificação da rota e acompanhamento da entrega sem implementação. Além disso, você precisará criar um item na página MenuPage para a abertura desta página.

```

using Modulo1.Dal;
using Modulo1.Modelo;
using System;
using Xamarin.Forms;

namespace Modulo1.Pages.Pedidos
{
    public partial class PedidosPage : ContentPage
    {
        private PedidoDAL pedidoDAL = new PedidoDAL();

        public PedidosPage()
        {
```

```

        InitializeComponent();
        BtnNovoPedido.Clicked += BtnNovoPedidoClick;
    }

    // Captura do clique do botão de novo Pedido
    private async void BtnNovoPedidoClick(object sender, EventArgs e)
    {
        // Mais adiante será pedido para você implementar este
        comportamento
    }

    // Captura do gesture da imagem de selecionar todos os ped
    idos
    private async void OnTappedTodos(object sender, EventArgs args)
    {
        lvClientes.ItemsSource = pedidoDAL.GetAllWithChildren();
        tituloConsulta.Text = "Todos os pedidos";
    }

    // Captura do gesture da imagem de selecionar todos os ped
    idos
    // ABERTOS. Como a chamada pode ocorrer em mais de um méto
    do,
    // foi implementado um específico para isso e aqui ele é c
    hamado
    private async void OnTappedAbertos(object sender, EventArgs args)
    {
        SelecionarPedidosEmAberto();
    }

    // Captura do gesture da imagem de selecionar todos os ped
    idos
    // EM PRODUÇÃO
    private async void OnTappedEmProducao(object sender, EventArgs args)
    {
        lvClientes.ItemsSource = pedidoDAL.GetEmProducaoWithCh
        ildren();
        tituloConsulta.Text = "Pedidos em produção";
    }

    // Captura do gesture da imagem de selecionar todos os ped
    idos
    // EM ENTREGA

```

```

        private async void OnTappedEmEntrega(object sender, EventArgs args)
        {
            lvClientes.ItemsSource = pedidoDAL.GetEmEntregaWithChildren();
            tituloConsulta.Text = "Pedidos em entrega";
        }

        // Captura do gestore da imagem de selecionar todos os pedidos
        // JÁ ENTREGUES E FECHADOS
        private async void OnTappedFechados(object sender, EventArgs args)
        {
            lvClientes.ItemsSource = pedidoDAL.GetFechadosWithChildren();
            tituloConsulta.Text = "Pedidos fechados";
        }

        // Captura da Action Context para encaminhar pedido para PRODUÇÃO
        private async void OnProduzirClick(object sender, EventArgs e)
        {
            var mi = ((MenuItem)sender);
            var pedido = mi.CommandParameter as Pedido;
            if (!pedido.Situacao.Equals("Aberto"))
            {
                DisplayAlert("Atenção", "Pedido precisa estar aberto", "Ok");
            }
            else
            {
                pedido.DataEHoraProducao = DateTime.Now;
                pedidoDAL.Update(pedido);
                DisplayAlert("Atualização", "Pedido enviado para a produção", "Ok");
                SelecionarPedidosEmAberto();
            }
        }

        // Captura da Action Context para encaminhar pedido para ENTREGA
        private async void OnEntregarClick(object sender, EventArgs e)
        {
            var mi = ((MenuItem)sender);
            var pedido = mi.CommandParameter as Pedido;

```

```

        if (!pedido.Situacao.Equals("Produção"))
        {
            DisplayAlert("Atenção", "Pedido precisa estar em p
rodução", "Ok");
        }
        else
        {
            pedido.DataEHoraEntrega = DateTime.Now;
            pedidoDAL.Update(pedido);
            DisplayAlert("Atualização", "Pedido enviado para a
entrega", "OK");
            SeleccionarPedidosEmAberto();
        }
    }

    // Captura da Action Context para FECHAR Pedido
    private async void OnFecharClick(object sender, EventArgs
e)
{
    var mi = ((MenuItem)sender);
    var pedido = mi.CommandParameter as Pedido;
    if (!pedido.Situacao.Equals("Em entrega"))
    {
        DisplayAlert("Atenção", "Pedido precisa estar em e
ntrega", "Ok");
    }
    else
    {
        pedido.DataEHoraEntregue = DateTime.Now;
        pedidoDAL.Update(pedido);
        DisplayAlert("Atualização", "Pedido finalizado com
sucesso", "OK");
        SeleccionarPedidosEmAberto();
    }
}

// Sobrescrita de método
protected override void OnAppearing()
{
    base.OnAppearing();
    SeleccionarPedidosEmAberto();
}

// Método criado para selecionar todos os pedidos em abert
o
private void SeleccionarPedidosEmAberto()
{
    lvClientes.ItemsSource = pedidoDAL.GetAbertosWithChild

```

```

ren();
    tituloConsulta.Text = "Pedidos abertos";
}

// Método que captura o clique da action context para verificação de rota
// de entrega
public async void OnVerificarRotaClick(object o, ItemTappedEventArgs e)
{
}

// Método que captura a gesture no item da ListView
public async void OnItemTapped(object o, ItemTappedEventArgs e)
{
}
}
}
}

```

9.4 INSERÇÃO DE NOVOS PEDIDOS

Precisamos agora popular esta listagem que criamos, e isso só pode ser feito se implementarmos a página de inserção de novos pedidos. Vamos fazer isso, mas como esta página oferecerá recursos para pesquisa de clientes e produtos, implementaremos estas duas antes.

Na listagem a seguir está o código para a página `ClientesSearchPage`, que deve ser criada na pasta `Clientes`, dentro de `Pages`. O código é simples e semelhante ao que fizemos para tipos de itens do cardápio, quando inserimos um item.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        x:Class="Modulo1.Pages.Clientes.ClientesSearchPage"
        Title="Relação de clientes"
        BackgroundColor="#fffff66">
<StackLayout Orientation="Vertical" HorizontalOptions="FillAndEx

```

```

pand" Padding="5,20,5,0">
    <SearchBar Placeholder="Digite o nome do cliente ..."
        TextColor="Black"TextChanged="OnTextChanged"/>
    <ListView x:Name="lvClientes" HasUnevenRows="True" ItemTapped=
"OnItemTapped" BackgroundColor="#ffff66">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <StackLayout Orientation="Vertical">
                        <Label Text="{Binding ClienteId}" TextColor="Blue" F
ontSize="0"/>
                        <Label Text="{Binding Nome}" TextColor="Blue" FontSi
ze="Large"/>
                        <Label Text="{Binding Telefone}" TextColor="Black" F
ontSize="Small"/>
                    </StackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</StackLayout>
</ContentPage>

```

Agora, vamos implementar o código para a classe da página. Veja-o na sequência. Todo o código é também semelhante com o que vimos para itens de pedido. O que acha de você começar a pensar em componentização para reutilização de código? Nós já vimos isso.

```

using Modulo1.Dal;
using Modulo1.Modelo;
using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Pages.Clientes
{
    public partial class ClientesSearchPage : ContentPage
    {
        private ClienteDAL dalClientes = new ClienteDAL();
        private Label displayValue;
        private Label keyValue;
        private IEnumerable<Cliente> clientes;

        public ClientesSearchPage(Label keyValue, Label displayVal
ue)

```

```

    {
        InitializeComponent();
        clientes = dalClientes.GetAll();
        this.keyValue = keyValue;
        this.displayValue = displayValue;
    }

    private void OnTextChanged(object sender, TextChangedEventArgs e)
    {
        lvClientes.BeginRefresh();

        if (string.IsNullOrWhiteSpace(e.NewTextValue))
            lvClientes.ItemsSource = clientes;
        else
            lvClientes.ItemsSource = clientes.Where(i => i.Nome.Contains(e.NewTextValue));

        lvClientes.EndRefresh();
    }

    public async void OnItemTapped(object o, ItemTappedEventArgs e)
    {
        var cliente = (o as ListView).SelectedItem as Cliente;
        this.keyValue.Text = cliente.ClienteId.ToString();
        this.displayValue.Text = cliente.Nome;
        await Navigation.PopAsync();
    }
}

```

Agora, faremos implementação semelhante para a busca de itens, pois na tela de registro de pedido possibilitaremos a pesquisa de cada item que o cliente solicita. Crie na pasta `ItensCardapio` a página `ItensCardapioSearchPage`. Veja o código na sequência e verifique a semelhança com o código anterior.

Logo após a apresentação do código da página, já apresentarei o código da classe desta página. Procure comparar os códigos e identificar onde é possível a generalização.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

```

```

        x:Class="Modulo1.Pages.ItensCardapio.ItensCardapioSearchPage"
        Title="Relação de itens de cardápio"
        BackgroundColor="#fffff66">
    <StackLayout Orientation="Vertical" HorizontalOptions="FillAndExpand" Padding="5,20,5,0">
        <SearchBar Placeholder="Digite o nome do item"
            TextColor="Black"TextChanged="OnTextChanged"/>
        <ListView x:Name="lvItens" HasUnevenRows="True" ItemTapped="OnItemTapped" BackgroundColor="#fffff66">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout Orientation="Vertical">
                            <Label Text="{Binding ItemCardapioId}" TextColor="Blue" FontSize="0"/>
                            <Label Text="{Binding Nome}" TextColor="Blue" FontSize="Large"/>
                            <Label Text="{Binding TipoItemCardapio.Nome}" TextColor="Black" FontSize="Small"/>
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>

```

Com o código da página mostrado, vamos ao código da classe para esta página.

```

using Modulo1.Dal;
using Modulo1.Modelo;
using System.Collections.Generic;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Pages.ItensCardapio
{
    public partial class ItensCardapioSearchPage : ContentPage
    {
        private ItemCardapioDAL dalItensCardapio = new ItemCardapioDAL();
        private Label displayValue;
        private Label keyValue;
        private IEnumerable<ItemCardapio> itens;

```

```

        public ItensCardapioSearchPage(Label keyValue, Label displayValue)
    {
        InitializeComponent();
        itens = dalItensCardapio.GetAllWithChildren();
        this.keyValue = keyValue;
        this.displayValue = displayValue;
    }

    private void OnTextChanged(object sender, TextChangedEventArgs e)
    {
        lvItens.BeginRefresh();

        if (string.IsNullOrWhiteSpace(e.NewTextValue))
            lvItens.ItemsSource = itens;
        else
            lvItens.ItemsSource = itens.Where(i => i.Nome.Contains(e.NewTextValue));

        lvItens.EndRefresh();
    }

    public async void OnItemTapped(object o, ItemTappedEventArgs e)
    {
        var item = (o as ListView).SelectedItem as ItemCardapio;
        this.keyValue.Text = item.ItemCardapioId.ToString();
        this.displayValue.Text = item.Nome.Trim() + " / " + item.TipoItemCardapio.Nome;
        await Navigation.PopAsync();
    }
}

```

Agora sim, podemos criar a página responsável pela inserção de um item para o pedido. Na pasta de pedidos, crie uma nova página, chamada `PedidosNewPage`. Sua página em execução deverá estar semelhante a mostrada na figura a seguir.

Optei por mostrar as imagens antes aqui, pois é uma tela interessante. Precisamos implementar a chamada a esta página no método que captura a inserção de um novo pedido, que deve ser `await Navigation.PushAsync(new PedidosNewPage());`.

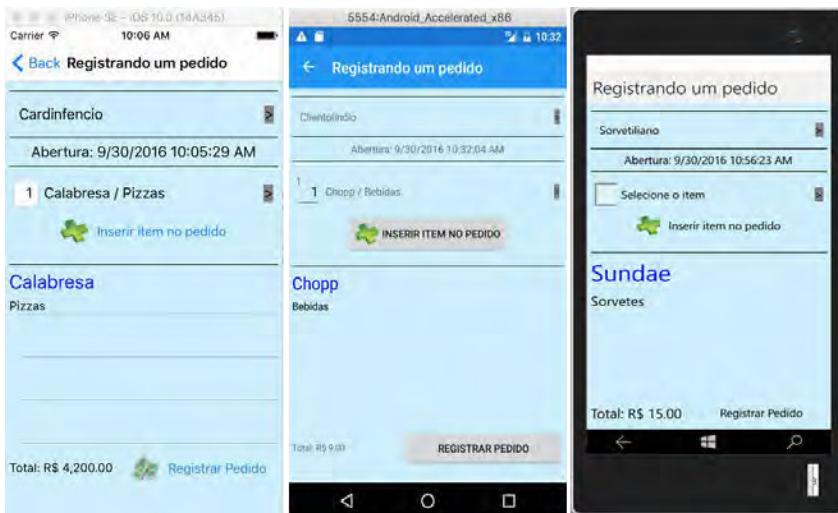


Figura 9.1: Página de inserção de um pedido

O código para a página representada anteriormente é composto pela implementação que se segue. Como pôde ser visto na figura anterior, existe uma área onde, ao realizar um toque, será aberta uma página para localizar um cliente e outra para localizar o item de cardápio. Lembra de que fizemos isso também no cadastro do item de cardápio, para selecionar o tipo do item? Pois bem, a lógica é a mesma e o código desta página também é apresentado na sequência.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

              x:Class="Modulo1.Pages.Pedidos.PedidosNewPage"
              Title="Registrando um pedido"
              BackgroundColor="#cceeff">

<!-- Criação do Layout principal para a página de inserção de pedido -->
<StackLayout VerticalOptions="FillAndExpand" Padding="0">
    <Grid Padding="5,10,5,10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="200"/>
        </Grid.RowDefinitions>
        <Image Source="img/icon-cliente.png" />
        <Label Text="Selecione o cliente" />
        <Entry Placeholder="Nome do cliente" />
        <Image Source="img/icon-item.png" />
        <Label Text="Selecione o item" />
        <Entry Placeholder="Nome do item" />
        <Image Source="img/icon-carrinho.png" />
        <Label Text="Total: R$ 15.00" />
        <Button Text="Registrar Pedido" />
    </Grid>
</StackLayout>
```

```

        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100*"/>
    </Grid.ColumnDefinitions>

<!-- Layout relacionado aos dados do cliente.-->
<StackLayout Padding="0" Grid.Row="0" Grid.Column="0">
    <StackLayout HeightRequest="1" BackgroundColor="Black"
        Padding="0"/>
    <Grid Padding="5,10,5,10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Label Text="" x:Name="idCliente" Grid.Row="0" Grid.Colu
mn="0"
            WidthRequest="0"/>
        <Label Text="Selecione o Cliente" x:Name="nomeCliente"
            Grid.Row="0" Grid.Column="1">
            <Label.GestureRecognizers>
                <TapGestureRecognizer
                    Tapped="OnTapLookForClientes"
                    NumberOfTapsRequired="1" />
            </Label.GestureRecognizers>
        </Label>
        <Label Text=">" HorizontalOptions="End" Grid.Row="0"
            Grid.Column="2" BackgroundColor="Gray">
            <Label.GestureRecognizers>
                <TapGestureRecognizer
                    Tapped="OnTapLookForClientes"
                    NumberOfTapsRequired="1" />
            </Label.GestureRecognizers>
        </Label>
    </Grid>
    <StackLayout HeightRequest="1" BackgroundColor="Black"
        Padding="0"/>
</StackLayout>

<!-- Layout com informações de data e hora da abertura do pedido -
->
<StackLayout Padding="0" Grid.Row="1" Grid.Column="0">
    <Label Text="" x:Name="lblDataEHoraAbertura" HorizontalOpt
ions="Center"/>

```

```

<StackLayout HeightRequest="1" BackgroundColor="Black"
             Padding="0"/>
</StackLayout>

<!-- Layout relacionado aos dados do item de cardápio que será inserido -->
<StackLayout Padding="0" Grid.Row="2" Grid.Column="0">
    <Grid Padding="5,10,5,10">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="10*"/>
            <ColumnDefinition Width="80*"/>
            <ColumnDefinition Width="10*"/>
        </Grid.ColumnDefinitions>
        <Label Text="" x:Name="idItem" Grid.Row="0" Grid.Column="0"
               WidthRequest="0"/>
        <Entry Placeholder="Quantidade" PlaceholderColor="Gray"
               x:Name="quantidadeItem" Grid.Row="0" Grid.Column="0"
               Text="1" HorizontalTextAlignment="End"/>
        <Label Text="Selecione o item" x:Name="nomeItem"
               Grid.Row="0" Grid.Column="1" VerticalOptions="Center" />
        <Label.GestureRecognizers>
            <TapGestureRecognizer
                Tapped="OnTapLookForItens"
                NumberOfTapsRequired="1" />
        </Label.GestureRecognizers>
    </Label>
    <Label Text=">" HorizontalOptions="End" Grid.Row="0"
           Grid.Column="2" BackgroundColor="Gray"
           VerticalOptions="Center">
        <Label.GestureRecognizers>
            <TapGestureRecognizer
                Tapped="OnTapLookForItens"
                NumberOfTapsRequired="1" />
        </Label.GestureRecognizers>
    </Label>
    <StackLayout Grid.Row="1" Grid.Column="0" Padding="0"
                Grid.ColumnSpan="3" HorizontalOptions="Center">
        <Button Text="Inserir item no pedido" x:Name="BtnInserirItem" Image="icone_new.png"/>
    </StackLayout>

```

```

        </Grid>
    <StackLayout HeightRequest="1" BackgroundColor="Black"
                 Padding="0"/>
</StackLayout>

<!-- Layout com ListView exibindo todos os itens inseridos do pedido -->
<StackLayout Padding="0" Grid.Row="3" Grid.Column="0">
    <ListView x:Name="lvItens" HasUnevenRows="True" BackgroundColor="#cceeff">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <StackLayout Orientation="Vertical">
                        <Label Text="{Binding ItemCardapio.Nome}" TextColor="Blue" FontSize="Large"/>
                        <Label Text="{Binding ItemCardapio.TipoItemCardapio.Nome}" TextColor="Black" FontSize="Small"/>
                    </StackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</StackLayout>

<!-- Layout final com total do pedido e botão para registrar o pedido -->
<StackLayout Grid.Row="4" Grid.Column="0" Padding="0">
    <Grid Padding="0">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="40*"/>
            <ColumnDefinition Width="60*"/>
        </Grid.ColumnDefinitions>
        <Label Text="Total: R$ 0,00" VerticalOptions="Center" x:Name="lblTotalPedido"
               FontAttributes="Bold" Grid.Row="0" Grid.Column="0"
               Font="Small"/>
        <Button Text="Registrar Pedido" x:Name="BtnRegistrarPedido" Image="icone_registrar.png"
               Grid.Row="0" Grid.Column="1"/>
    </Grid>
</StackLayout>
</Grid>
</StackLayout>
</ContentPage>

```

Agora, para finalizar o registro de um novo pedido, vamos à classe desta página, que tem seu código apresentado em seguida. Não apresentarei aqui as páginas da pesquisa de clientes e itens de cardápio, ok? Veja no código a seguir a declaração de campos para a classe, pois são recursos que usaremos nos métodos.

Veja que, para os itens, voltamos a usar um `ObservableCollection`, pois queremos que a `ListView` seja atualizada de acordo com a inserção de novos itens no pedido. Deixarei a seu cargo a criação de context actions para remoção de um item do pedido.:-)

```
using Modulo1.Dal;
using Modulo1.Modelo;
using Modulo1.Pages.Clientes;
using Modulo1.Pages.ItensCardapio;
using System;
using System.Collections.ObjectModel;
using System.Linq;
using Xamarin.Forms;

namespace Modulo1.Pages.Pedidos
{
    public partial class PedidosNewPage : ContentPage
    {
        private DateTime DataEHoraAbertura;
        private ObservableCollection<ItemPedido> itensPedido = new
        ObservableCollection<ItemPedido>();
        private ItemCardapioDAL itemCardapioDAL = new ItemCardapio
        DAL();
        private PedidoDAL pedidoDAL = new PedidoDAL();
        private ClienteDAL clienteDAL = new ClienteDAL();
        private Pedido pedido = null;

        public PedidosNewPage()
        {
            InitializeComponent();
            DataEHoraAbertura = DateTime.Now;
            lblDataEHoraAbertura.Text = "Abertura: " + DataEHoraAb
            ertura.ToString();
            BtnInserirItem.Clicked += BtnInserirItemClicked;
            BtnRegistrarPedido.Clicked += BtnRegistrarPedidoClicke
            d;
            lvItens.ItemsSource = itensPedido;
        }
    }
}
```

```

    }

    private async void BtnRegistrarPedidoClicked(object sender
, EventArgs e)
{
    if (itensPedido.Count == 0)
    {
        DisplayAlert("Sem itens", "Informe ao menos um ite
m para o pedido", "Ok");
    }
    else
    {
        PreparePedidoToPersist();
        ClearControls();
        DisplayAlert("Pedido inserido", "Pedido foi regist
rado com sucesso", "Ok");
    }
}

private void ClearControls()
{
    itensPedido.Clear();
    nomeCliente.Text = "Selecione o Cliente";
    nomeItem.Text = "Selecione o item";
    idCliente.Text = "";
    idItem.Text = "";
}

private void PreparePedidoToPersist()
{
    pedido.Itens = itensPedido.ToList();
    pedido.DataEHoraPedido = DateTime.Now;
    pedido.Cliente = clienteDAL.GetClienteById(Convert.ToInt32(idCliente.Text));
    pedido.ClienteId = pedido.Cliente.ClienteId;
    pedidoDAL.Add(pedido);
}

private async void BtnInserirItemClicked(object sender, EventArgs e)
{
    if (idCliente.Text.Trim() == string.Empty || idItem.Te
xt.Trim() == string.Empty)
    {
        DisplayAlert("Dados incompletos", "Informe um clie
nte e um item de cardápio", "Ok");
    }
    else

```

```

    {
        if (pedido == null)
        {
            this.pedido = new Pedido();
        }
        var itemCardapio = itemCardapioDAL.GetItemById(Convert.ToInt32(idItem.Text));
        itensPedido.Add(new ItemPedido()
        {
            ItemCardapio = itemCardapio,
            ItemCardapioId = itemCardapio.ItemCardapioId,
            ValorUnitario = itemCardapio.Preco,
            Quantidade = Convert.ToInt32(quantidadeItem.Text),
            Pedido = this.pedido
        });

        lblTotalPedido.Text = "Total: R$ " + itensPedido.Sum(i => (i.Quantidade * i.ValorUnitario)).ToString("N2");
        nomeItem.Text = "Selecione o item";
        idItem.Text = "";
    }
}

private async void OnTapLookForClientes(object sender, EventArgs args)
{
    await Navigation.PushAsync(new ClientesSearchPage(idCliente, nomeCliente));
}

private async void OnTapLookForItens(object sender, EventArgs args)
{
    await Navigation.PushAsync(new ItensCardapioSearchPage(idItem, nomeItem));
}
}

```

Teste sua aplicação, insira alguns pedidos e volte para a página de listagem de pedidos. Ela deve estar semelhante à figura apresentada na sequência.

Na listagem, clique e arraste (ou pressione por um maior tempo), e veja as opções de trabalho para cada pedido. Mude as

fases, veja os testes. Está tudo implementado já, menos o traçado da rota.

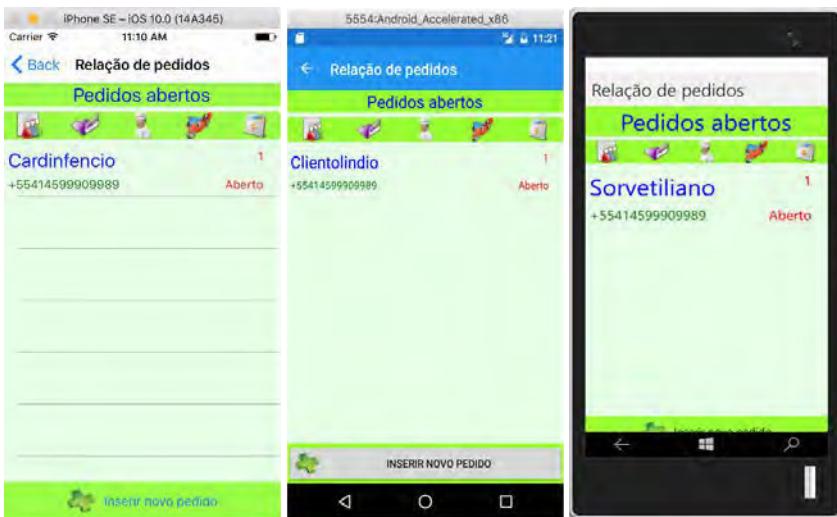


Figura 9.2: Página de listagem de pedidos

9.5 TRANSIÇÃO DE FASES DO PEDIDO, COM ENVIO DE SMS

Espero que você tenha testado toda a interação que implementamos para pedidos. O que acha de, a cada evolução do pedido (produzir, entregar e fechar), um SMS ser enviado ao cliente? Bem, para isso, existe um plugin que usaremos e nos auxiliará. Ele, além de permitir o envio de SMS, permite o envio de e-mails e de realização de chamada telefônica.

Estou falando do *Messaging Plugin for Xamarin and Windows*. Vamos instalá-lo? Clique com o botão direito do mouse sobre o nome da solução, e então em *Manage NuGet Packages for Solution...*. Siga a figura a seguir para proceder com a instalação. Você pode saber mais sobre o plugin em <https://github.com/cjlotz/Xamarin.Plugins/tree/master/Messaging>.

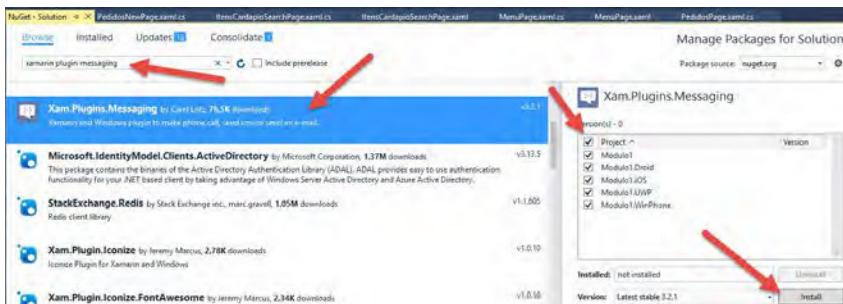


Figura 9.3: Instalando o Messaging Plugin for Xamarin and Windows

Apresentarei agora o uso deste plugin no processo de encaminhar o pedido para a produção. Lembre-se de que, para fazer isso, basta clicar no pedido e arrastar (pressionar e manter), e escolher a opção **Producir**.

O método que realiza isso já foi implementado por você anteriormente. Desta maneira, na listagem que vê na sequência, está a adaptação para o uso deste novo recurso. É importante você saber que a mensagem não é enviada automaticamente. A aplicação invocará o enviaor de mensagens do dispositivo, apresentará o destinatário, o texto da mensagem, e então você precisará confirmar o envio. Após isso, a aplicação retorna para sua janela do dispositivo.

```
private async void OnProducirClick(object sender, EventArgs e)
{
    var mi = ((MenuItem)sender);
    var pedido = mi.CommandParameter as Pedido;
    if (!pedido.Situacao.Equals("Aberto"))
    {
        DisplayAlert("Atenção", "Pedido precisa estar aberto", "Ok");
    }
    else
    {
        pedido.DataEHoraProducao = DateTime.Now;
        pedidoDAL.Update(pedido);
        var smsMessenger = CrossMessaging.Current.SmsMessenger;
        if (smsMessenger.CanSendSms)
```

```

    {
        smsMessenger.SendSms(pedido.Cliente.Telefone, "Pedido "
+ pedido.PedidoId +
        " enviado para a produção. CCFoods, obrigado pela
preferência");
    } else
    {
        DisplayAlert("Ação",
        "O SMS não pôde ser enviado, mas o pedido foi envi
ado para a produção",
        "Ok");
        SelecionarPedidosEmAberto();
    }
}
}

```

Com a lógica apresentada anteriormente, agora é com você. Implemente o envio de SMS para as demais trocas de fase. Quem sabe seja possível até uma reutilização de código. :-)

9.6 VERIFICAÇÃO DA ROTA PARA A ENTREGA DO PEDIDO

O uso de mapas, como visto no capítulo anterior, permite que localizemos em um mapa um determinado endereço, ou uma determinada posição geográfica, por meio da latitude e longitude. Mas e como fica a situação de traçar a rota para a entrega do pedido? Isso não seria tão simples com este plugin que usamos.

Para isso, faremos uso da API específica de mapas disponível em cada plataforma. Veja na sequência a implementação para o método que captura o clique do menu de action context para visualizar a rota. Verifique no corpo do método que são feitas verificações que possibilitem a identificação da plataforma na qual a aplicação está sendo executada, para que assim seja invocada a aplicação correta de mapas.

Observe na instrução `new Uri()` que, para o iOS, é possível enviar o ponto de origem (`saddr`), destino (`daddr`) e a

modalidade de navegação (`dirflag`), que é `drive`. Já para o Android, apenas o destino e a modalidade são possíveis, a localização de seu dispositivo é o ponto de partida para a rota. E, finalizando, para o Windows, também é possível informar os pontos de origem, destino e a modalidade.

Caso você queira se aprofundar nestas APIs, recomendo as leituras específicas a seguir:

- https://developer.apple.com/library/content/featuredarticles/iPhoneURLScheme_Reference/MapLinks/MapLinks.html
- <https://developers.google.com/maps/documentation/static-maps/intro?hl=pt-br>
- <https://msdn.microsoft.com/pt-br/windows/uwp/launch-resume/launch-maps-app>

O Xamarin traz também o seguinte link, que fala sobre isso, mas de maneira superficial:

<https://developer.xamarin.com/recipes/cross-platform/xamarin-forms/maps/map-navigation/>.

```
// Método que captura o clique da action context para verificação
// de rota
// de entrega
public async void OnVerificarRotaClick(object sender, EventArgs e)
{
    var mi = ((MenuItem)sender);
    var pedido = mi.CommandParameter as Pedido;
    var origem = "Av Jorge Schimmelpfeng 600 Centro Foz do Iguaçu
Paraná";
    var endereco = pedido.Cliente.Endereco + " " + pedido.Cliente.
Número + " " +
        pedido.Cliente.Bairro + " " + pedido.Cliente.Cidade + " "
+ pedido.Cliente.Estado;

    switch (Device.OS)
    {
        case TargetPlatform.iOS:
```

```

        Device.OpenUri(
            new Uri(string.Format("http://maps.apple.com/?saddr={0}&daddr={1}&dirflg=d",
            WebUtility.UrlEncode(origem), WebUtility.UrlEncode(endereco)))); 
            break;
        case TargetPlatform.Android:
            Device.OpenUri(
                new Uri(string.Format("google.navigation:q={0}&mode=d",
                WebUtility.UrlEncode(endereco)))); 
            break;
        case TargetPlatform.Windows:
        case TargetPlatform.WinPhone:
            Device.OpenUri(
                new Uri(string.Format("bingmaps:?rtp=adr.{0}-adr.{1}&mode=d",
                Uri.EscapeDataString(origem), Uri.EscapeDataString(endereco)))); 
            break;
    }
}

```

A figura a seguir apresenta a execução dos mapas nos dispositivos. Não foi possível executar esta aplicação no emulador do Windows, mas tanto para iOS como Android isso foi possível. Teste em seus emuladores e nos dispositivos reais.

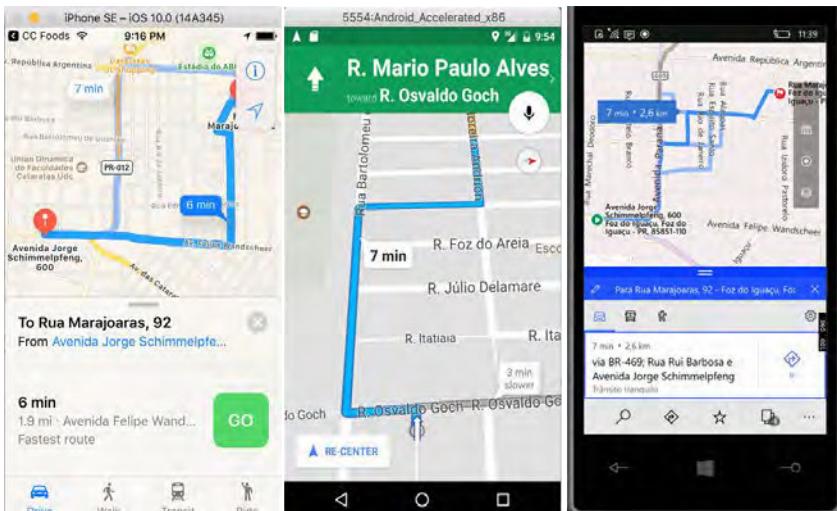


Figura 9.4: Mapas com as rotas de entrega

9.7 REGISTRO DA POSIÇÃO DO ENTREGADOR

Se você lembrar do mapa que trabalhamos no capítulo anterior, ele mostra a posição atual do dispositivo, assim como os mapas que vimos neste capítulo também. Entretanto, o cliente pode querer acompanhar a posição do entregador quando sai para entregar seu pedido. Ou seja, o cliente de seu dispositivo deseja obter a localização de outro dispositivo.

Para atendermos a esta requisição, precisamos que a aplicação obtenha a localização do dispositivo do entregador, e então a registre na base de dados e publique-a na nuvem. Assim, o cliente, por meio de seu dispositivo (ou por um web site), acompanha esta evolução, acessando a aplicação na nuvem.

Eu não implementarei tudo isso aqui, mas já trabalhei como fazer tudo isso funcionar, então é com você. Mas, para recuperar a posição do dispositivo, que é algo que ainda não foi visto, precisamos de outro componente que fará isso, que é o *GeoLocator* (`Xam.Plugin.Geolocator`). Vamos instalá-lo na solução.

Para isso, clique com o botão direito do mouse sobre o nome da solução, e então em `Managed NuGet Packages for Solution...` e siga os passos da figura a seguir.

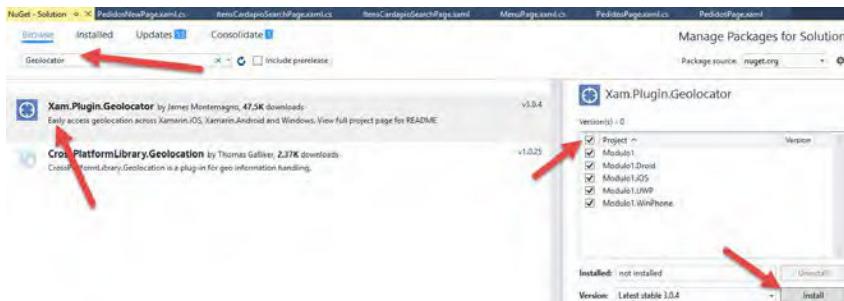


Figura 9.5: Instalação do plugin GeoLocator

Para que a rota possa ser obtida em um iOS, precisamos inserir duas instruções no arquivo `info.plist`, que são as seguintes. Insira-as antes do fechamento da tag `<dict>`. São configurações requeridas para se obter a posição do dispositivo móvel.

```
<key>NSLocationWhenInUseUsageDescription</key>
<string>Get Location</string>
<key>NSLocationAlwaysUsageDescription </key>
<string>Get Location</string>
```

Vamos exibir em um mapa a localização do entregador por meio de pinos. A ideia é que você verifique como recuperar uma localização e fazer uso dela. Crie uma página chamada `PedidosAcompanhamentoPage` e implemente o seguinte XAML nela. Observe que a implementação é semelhante a que fizemos no capítulo anterior, para Clientes.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        xmlns:maps="clr-namespace:Xamarin.Forms.Maps;assembly=
=Xamarin.Forms.Maps"
        x:Class="Modulo1.Pages.Pedidos.AcompanhamentoP
age"
        Title="Localização do entregador"
        BackgroundColor="#ffd9b3">

    <ContentPage.Content>
        <StackLayout VerticalOptions="FillAndExpand" HorizontalOption:
="FillAndExpand"
            Padding="0">
            <maps:Map x:Name="MyMap"
                IsShowingUser="true"
                MapType="Street"
            />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

Agora, vamos implementar a classe. Veja o código seguinte. Verifique, no início do código, a existência de um campo `locator`. Ele é utilizado no construtor para algumas configurações e definição

do método que será disparado quando houver uma alteração de posição, respeitando o tempo de 60 segundos e a distância de 50m.

Neste método, é inserido um pino, com o label tendo a hora em que a posição foi recuperada. Neste momento é que você poderia inserir este dado na base de dados, para consulta por parte do cliente. Se você desejar que a posição seja obtida por um pressionar de botão, poderia ser usada a instrução `locator.GetPositionAsync(timeoutMilliseconds: 10000);`.

Observe ainda a sobrescrita de dois métodos, o `OnAppearing()` e o `OnDisappearing()`. O primeiro é invocado quando a página for exibida, e o segundo quando ela for fechada ou colocada em modo sleep. Maiores informações sobre este plugin podem ser obtidas em <https://github.com/jamesmontemagno/GeolocatorPlugin>.

```
using Plugin.Geolocator;
using Plugin.Geolocator.Abstractions;
using Xamarin.Forms;
using Xamarin.Forms.Maps;

namespace Modulo1.Pages.Pedidos
{
    public partial class PedidosAcompanhamentoPage : ContentPage
    {
        private IGeolocator locator;

        public PedidosAcompanhamentoPage()
        {
            InitializeComponent();
            locator = CrossGeolocator.Current;
            locator.DesiredAccuracy = 50;

            locator.PositionChanged += OnPositionChanged;
        }

        private void OnPositionChanged(object obj, PositionEventArgs e)
        {
            MyMap.MoveToRegion(MapSpan.FromCenterAndRadius(
                new Xamarin.Forms.Maps.Position(e.Position.Latitud
```

```
        e, e.Position.Longitude),
        Distance.FromKilometers(0.3f)));

    var localPin = new Pin()
    {
        Position = new Xamarin.Forms.Maps.Position(
            e.Position.Latitude, e.Position.Longitude),
        Label = "Entregador/" + e.Position.Timestamp.ToLoc
alTime().TimeOfDay
    };

    MyMap.Pins.Add(localPin);
}

protected override void OnAppearing()
{
    base.OnAppearing();
    locator.StartListeningAsync(60000, 50);
}

protected override void OnDisappearing()
{
    base.OnDisappearing();
    locator.StopListeningAsync();
}
}
```

Implemente a chamada a esta página de acompanhamento, na página de listagem, no método que responde ao evento Tapped do ListView. A implementação é referente à chamada a esta página, que é

```
await Navigation.PushAsync(new PedidosAcompanhamentoPage()); .
```

Teste sua aplicação em um dispositivo. O emulador não permite a obtenção da localização. Na primeira execução, será solicitada a permissão para acesso a localização do dispositivo.

Esta atualização que implementamos para a localização do dispositivo do entregador poderia ser feita com o uso de serviços de Background, que é um assunto mais complexo, fora do escopo deste livro. Porém, como estamos terminando-o, recomendo uma investigação sobre isso, tanto para implementação no Xamarin e

Xamarin Forms como para as especificidades de cada plataforma.

9.8 CONCLUSÃO

Chegamos ao final deste capítulo. Ufa! A criação da interface com o usuário foi a mais trabalhosa dos capítulos que já vimos. Precisamos interagir com várias páginas, várias classes e ainda com acesso a mapas e envio de mensagens.

Neste capítulo, você obteve subsídio para criar uma aplicação apenas para registro de pedidos, uma para o entregador registrar a entrega, uma para os setores registrarem a transição do pedido e uma para o cliente acompanhar a entrega de seu pedido. Resta agora implementar tudo isso em módulos separados, fazendo também o registro na base de dados e sincronismo das coordenadas do entregador na nuvem. Isso deixo a você.

O próximo capítulo será o último e trabalharemos com gráficos. Dê uma descansada e se prepare para a nossa última etapa.

CAPÍTULO 10

GRÁFICOS

Chegamos ao último capítulo. Espero que você tenha aprendido muito desde o início de sua leitura. Para finalizar este livro, trago aqui uma apresentação sobre a geração de gráficos.

Não existem controles padrões no Xamarin e Xamarin Forms para isso. Desta maneira, uma vez mais, faremos uso de componentes de terceiros. Agora, de uma coleção muito boa, da empresa *Syncfusion* (<https://www.syncfusion.com/>).

Se você fizer uma rápida pesquisa da internet, além de conhecer a coleção de controles que são oferecidos, certamente verá que é um controle com *custo*. Mas calma, fique tranquilo. Eles têm uma licença comunitária, que permite seu uso, sem custo algum. Agora, se você for realmente ganhar dinheiro com seus produtos, e espero que ganhe, pense em adquirir a licença para uso profissional.

Como a Syncfusion, existem diversas outras empresas que possuem controles que você pode considerar no futuro, mas não falarei delas aqui. Não é o foco.

10.1 INSTALAÇÃO DO SYNCFUSION

O site da Syncfusion traz seus controles para Xamarin disponíveis para instalação por meio de arquivo executável e compactado (para Windows) e pacotes para o Mac, além de possibilitar o download via NuGet.

Tentei fazer o download pelo NuGet, mas não consegui fazer a instalação para a plataforma UWP, o que me levou a realizar o download do executável (Windows) e referenciar as bibliotecas em cada projeto. É mais trabalho do que vimos até agora para os demais componentes, mas vale a pena.

Para isso, acesse (<https://www.syncfusion.com/>), clique Products e localize Free Community License Available , como mostra a figura a seguir.

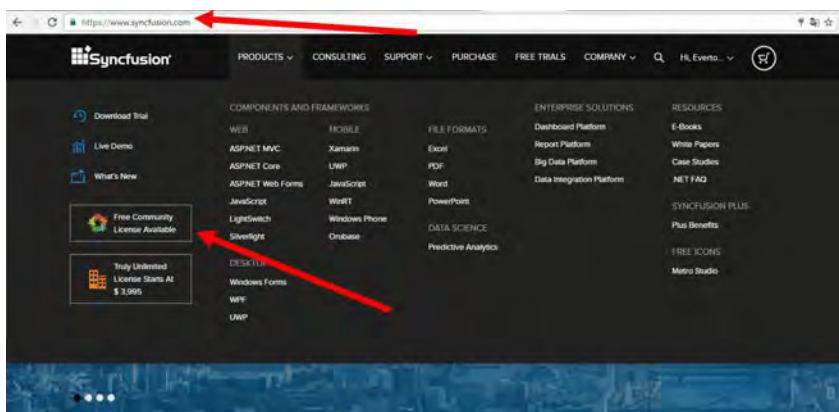


Figura 10.1: Página inicial de seleção de produtos Syncfusion

Na nova página, clique em `Claim License`. Veja a figura desta página na sequência. Você precisará neste momento realizar seu cadastro e, após isso, sua chave para a licença comunitária será enviada ao seu e-mail.



Figura 10.2: Página de solicitação de licença

São várias páginas para chegarmos ao download. Calma. Nessa nova que aparece, clique no link **Proceed to download section**, conforme mostra a seguinte figura.

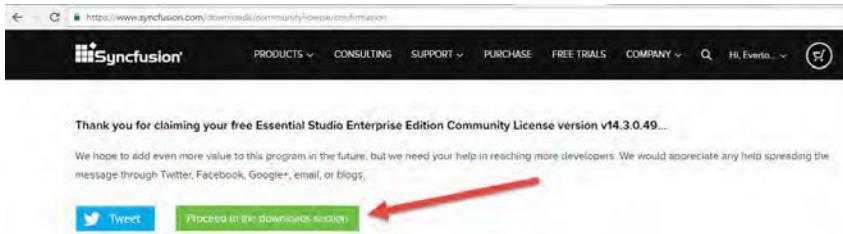


Figura 10.3: Página de agradecimento pela solicitação da licença

Preferi mostrar todas as páginas, justamente por ser longo este caminho. Agora, clique em **Download links and details** da versão desejada. Eu usei a mais recente, a que consta na figura a seguir. Talvez quando você for realizar o download seja outra. Caso isso ocorra e o exemplo que mostro não funcionar, você precisará recorrer à documentação dos controles, que é muito rica.

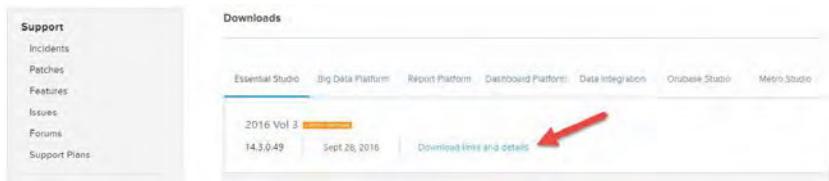


Figura 10.4: Seleção da versão para download

Ufa, finalmente chegou a página para download. Selecione o arquivo e plataforma que deseja, e aguarde para que ele seja baixado.



Figura 10.5: Escolha da plataforma e arquivo para download

Com o arquivo baixado, podemos iniciar nossa instalação. Executando o arquivo, você deverá informar seu nome de usuário e senha, de acordo com o que você usou em seu cadastro no site. Veja a figura na sequência.

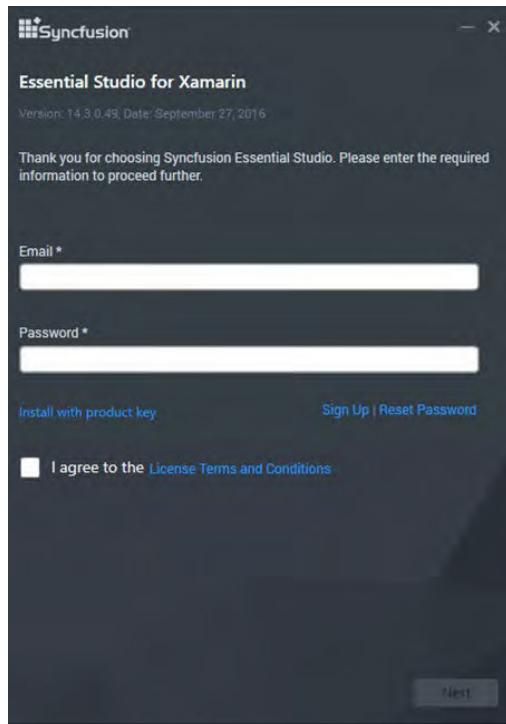


Figura 10.6: Instalação dos controles Syncfusion

Note que, nesta mesma figura (a anterior), tem um link para a informação da chave de produto, aquela que você recebeu em seu e-mail. Ao clicar neste link, uma nova janela será exibida, como você pode ver pela figura a seguir.

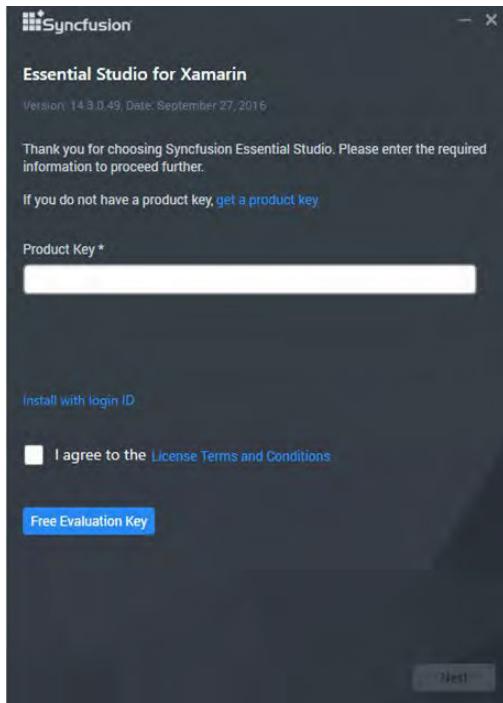


Figura 10.7: Informação da chave do produto

Informe sua chave para a instalação e clique no botão para seguir na instalação. Uma última janela com algumas configurações é exibida. Confirme ou altere o que desejar, e dê sequência no processo. Ele pode demorar um pouco, então seja paciente.

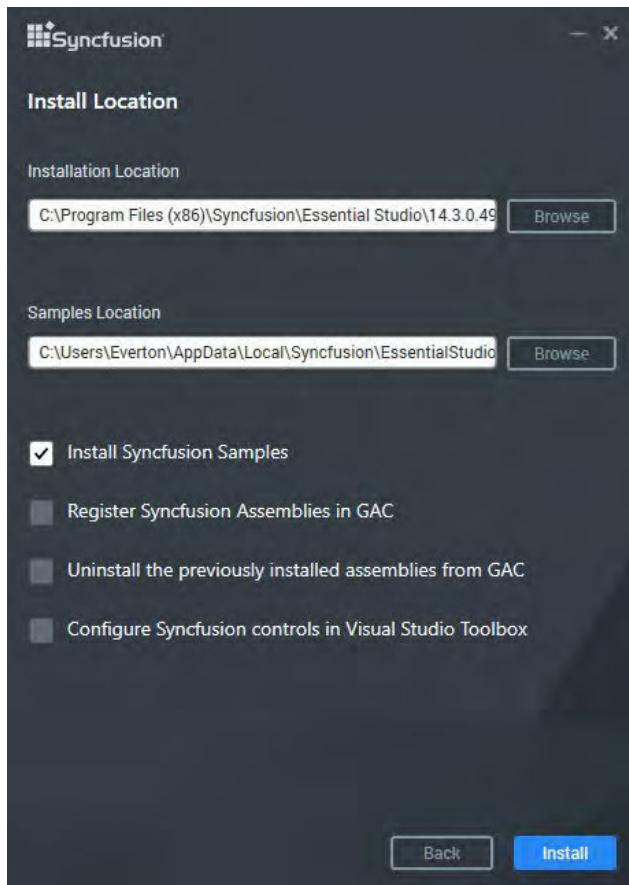


Figura 10.8: Informação da chave do produto

10.2 INSERÇÃO DAS REFERÊNCIAS PARA O USO DE GRÁFICOS DO SYNCFUSION

Após a instalação dos controles ter sido realizada, toda a biblioteca para os controles está disponível em `C:\Program Files (x86)\Syncfusion\Essential Studio\14.3.0.49\Xamarin\lib`, com separações de acordo com a plataforma. Precisamos adicionar em cada projeto as bibliotecas necessárias para o uso do controle de gráfico. Instale-as de acordo com a relação a seguir.

Em cada projeto, clique com o botão direito do mouse sobre References e em Add Reference . Depois em Browse e busque por elas de acordo com o endereço informado anteriormente.

- *PCL project:* pcl\Syncfusion.SfChart.XForm.dll
- *Android project:*
 android\Syncfusion.SfChart.Andriod.dll ,
 android\Syncfusion.SfChart.XForms.Andriod.dll
 e android\Syncfusion.SfChart.XForm.dll
- *iOS project:*
 ios-unified\Syncfusion.SfChart.iOS.dll , ios-unified\Syncfusion.SfChart.XForms.iOS.dll e
 ios-unified\Syncfusion.SfChart.XForm.dll
- *UWP project:* uwp\Syncfusion.SfChart.UWP.dll ,
 uwp\Syncfusion.SfChart.XForms.UWP.dll e
 uwp\Syncfusion.SfChart.XForm.dll

O inconveniente de adicionar as referências seguindo esta metodologia que apresento e não o NuGet é que, em meu caso, eu preciso instalar o Syncfusion em todas as máquinas, e nelas adicionar corretamente as referências aos projetos. Fazendo o uso de NuGet, os pacotes ficam registrados no arquivo packages .

10.3 UM GRÁFICO DE BARRAS

Vamos criar agora a página que conterá um gráfico para apresentar a evolução de vendas entregues dia a dia. Para isso, precisamos criar, em nosso PedidoDAL , um método que atenda a esta demanda. Veja seu código na sequência.

Observe o retorno do método. Precisaremos criar a classe DadoParaGrafico , e logo faremos isso. Note no código que o que

está implementado é um agrupamento pelo dia do mês, mas não me preocupei em filtrar o mês. Você pode pensar em implementar isso por meio de uma página que anteceda a exibição do gráfico, na qual o usuário poderia selecionar o mês e ano para a seleção de dados. As possibilidades são infinitas. No retorno do agrupamento, é feito referência ao tipo de dado do retorno, populando objetos deste tipo.

```
public IEnumerable<DadoParaGrafico> GetFechadosGroupedByDayWithChildren()
{
    return sqlConnection.GetAllWithChildren<Pedido>().Where(
        p => p.DataEHoraEntregue != null).
        GroupBy(p => p.DataEHoraEntregue.Value.Day).
        Select(pedidos => new DadoParaGrafico
    {
        DiaDoMes = pedidos.First().DataEHoraEntregue.Value.Day
    ,
        Total = pedidos.Sum(s => s.Total)
    }).ToList();
}
```

Agora, para que o código anterior possa ser compilado, crie na pasta `ViewModels` a classe a seguir, chamada `DadoParaGrafico`.

```
namespace Modulo1.ViewModel
{
    public class DadoParaGrafico
    {
        public int DiaDoMes { get; set; }
        public double Total { get; set; }
    }
}
```

Na pasta `Pages`, crie uma nova, chamada `Graficos`, e nela crie uma página chamada `GraficoDeBarrasDePedidosPorDiaPage` e implemente nele a seguinte listagem. Veja a linha do `xmlns:chart`, que define o controle que usaremos na página.

As tags dentro de `<chart:SfChar>` definem, respectivamente, a legenda, título e eixo primário para o desenho das barras. Esta é uma codificação simples.

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

        xmlns:chart="clr-namespace:Syncfusion.SfChart.XForms;
assembly=Syncfusion.SfChart.XForms"
        x:Class="Modulo1.Pages.Graficos.GraficoDeBarrasDePedidosPorDiaPage">
    <ContentPage.Content>
        <chart:SfChart x:Name="grafico">
            <chart:SfChart.Legend>
                <chart:ChartLegend/>
            </chart:SfChart.Legend>

            <chart:SfChart.Title>
                <chart:ChartTitle Text="Entregas por dia do mês"/>
            </chart:SfChart.Title>

            <chart:SfChart.PrimaryAxis>
                <chart:CategoryAxis>
                    <chart:CategoryAxis.Title>
                        <chart:ChartAxisTitle Text="Dia"></chart:ChartAxisTitle>
                    </chart:CategoryAxis.Title>
                </chart:CategoryAxis>
            </chart:SfChart.PrimaryAxis>
        </chart:SfChart>
    </ContentPage.Content>
</ContentPage>

```

Com a interface do usuário implementada, precisamos agora codificar o comportamento para esta classe, em que será definida a fonte de dados para o gráfico. Essa listagem pode ser verificada na sequência. Observe a definição dos campos para acessar o DAL e armazenar os dados a serem exibidos no gráfico.

No construtor, após a inicialização dos componentes, os objetos são recuperados e adicionados à coleção, que precisa ser do tipo `ChartDataPoint`, que é do Syncfusion. Em seguida, uma série de dados é adicionada e configurada para ser exibida. Bem simples, não é?

```

using Modulo1.Dal;
using Syncfusion.SfChart.XForms;

```

```

using System.Collections.ObjectModel;
using Xamarin.Forms;

namespace Modulo1.Pages.Graficos
{
    public partial class GraficoDeBarrasDePedidosPorDiaPage : ContentPage
    {
        public PedidoDAL pedidoDAL = new PedidoDAL();
        public ObservableCollection<ChartDataPoint> Dados = new ObservableCollection<ChartDataPoint>();

        public GraficoDeBarrasDePedidosPorDiaPage()
        {
            InitializeComponent();
            foreach (var dado in pedidoDAL.GetFechadosGroupedByDayWithChildren())
            {
                Dados.Add(new ChartDataPoint(dado.DiaDoMes.ToString(), dado.Total));
            }
            Dados.Add(new ChartDataPoint("3", 90));
            Dados.Add(new ChartDataPoint("2", 100));

            grafico.Series.Add(new ColumnSeries()
            {
                ItemsSource = Dados,
                Label = "Vendas",
                YAxis = new NumericalAxis()
                {
                    OpposedPosition = false,
                    ShowMajorGridLines = false
                }
            });
        }
    }
}

```

É preciso configurar nossa página de menus para que ofereça uma opção de acessar nosso gráfico. Além disso, para executar a aplicação no iOS, na classe `AppDelegate` (no projeto iOS), precisamos inserir a instrução `new SfChartRenderer();` no método `FinishedLaunching`.

Para a aplicação UWP, no respectivo projeto, no construtor da

`MainPage`, também é preciso inserir `new SfChartRenderer();`. No Android, não é preciso fazer nada. Fazendo isso, já pode executar sua aplicação e testar.

A figura a seguir apresenta a aplicação com o gráfico renderizado:



Figura 10.9: Aplicação com apresentação do gráfico

10.4 CONCLUSÃO

Chegamos ao final do capítulo. Não fazia parte de meus objetivos e nem deste capítulo explorar os controles da Syncfusion, mas apenas apresentar esta coleção para que você pudesse verificar o quanto simples é seu uso.

Se você optar por utilizar este e os demais controles oferecidos, recomendo esta leitura: <https://help.syncfusion.com/xamarin/introduction/overview>. Você verá que possui documentação detalhada para todos os controles disponíveis. Com isso, fechamos este último capítulo.

CAPÍTULO 11

OS ESTUDOS NÃO PARAM POR AQUI

Os dispositivos móveis já fazem parte do dia a dia da maioria da população. Você, como programador (ou desenvolvedor), não pode perder essa onda.

São três as maiores plataformas móveis atualmente disponíveis (iOS, Android e Windows) e várias são as ferramentas para desenvolvimento para elas, quer seja de forma nativa ou híbrida. Neste livro, você teve acesso à metodologia de implementação de aplicações nativas às plataformas descritas, por meio do Xamarin e Xamarin Forms, utilizando uma única linguagem, o C#.

Fizemos um "passeio" sobre grande parte dos controles disponibilizados pelo Xamarin Forms. Também testamos as aplicações em emuladores e em dispositivos físicos, o que nos possibilitou uso de recursos como câmera, álbum de fotografia e localização, dentre outros.

Como o foco da aplicação desenvolvida no livro foi comercial, não podíamos deixar de trabalhar com acesso a base de dados, o que fizemos a partir do capítulo cinco, por meio do SQLite. Uma característica comum em aplicações móveis é o sincronismo com aplicações na nuvem, e isso também foi trabalhado.

O livro não esgotou os temas trabalhados. É preciso dedicação para investigação e descoberta de novas tecnologias e recursos.

Tenho certeza de que ele foi mais do que um "pontapé" inicial para o seu desenvolvimento, no que se diz respeito a aplicações móveis.

Espero que, com o conteúdo que trabalhei, tenha sido despertada em você uma grande curiosidade e que tenha desmitificado o desenvolvimento de aplicativos para dispositivos móveis. Agora é bola para a frente na evolução.

Obrigado pela companhia, sucesso, e que a força esteja com você! ;-)