Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Mobilní klient informačního systému

Bakalářská práce

Autor: Tomáš Podmanický
Studijní obor: Informační management

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Odborný konzultant: Ing. Rostislav Podmanický, vývojář informačních systémů

Hradec Králové duben 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedených zdrojů.

V Hradci Králové dne 28.4.2022 Tomáš Podmanický

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce, poskytnuté znalosti a podporu.

ANOTACE

Bakalářská práce se věnuje popisu vývoje mobilního klienta pro informační systém ve frameworku *Flutter*. V teoretické části je rozepsáno porovnání s alternativními programovacími jazyky nebo frameworky, dále jsou rozebrány technologie, které byly pro vývoj použity. V praktické části je popsán proces vývoje, konkrétní příklady a proces publikace mobilní aplikace.

ANNOTATION

TITLE: MOBILE CLIENT OF INFORMATION SYSTEM

The bachelors thesis deals with the description of the development of a mobile client for an information system in the *Flutter* framework. The theoretical part describes the comparsion with alternative programming languages or frameworks and then discusses the technologies, that have been included for development. The practical part describes the development process, specific examples and the process of publishing a mobile application.

Obsah

Seznam obrázků

SEZNAM TABULEK

SEZNAM POUŽITÝCH ZKRATEK

Vysvětlivky zkratek, pokud není uvedeno jinak, jsou dostupné na *allacronyms.com*. (1)

ERP – Enterprise Resource Planning

API – Application Programming Interface

IDE – Integrated Development Environment

PWA – Progressive Web App

JSON – JavaScript Object Notation

IP – Internet Protocol

VPN – Virtual Private Network

HTTPS(S) – Hypertext Transfer Protocol (Secure)

XML – Extensible Markup Language

PDF – Portable Document Format

ARES – Administrativní Registr Ekonomických Subjektů (2)

Úvod

V době, kdy je mobilní telefon každodenní součástí našich životů, je žádoucí, aby nám poskytnul to, co potřebujeme – ideálně ihned a přehledně. Tato práce se věnuje popisu vývoje multiplatformního mobilního klienta *ERP* systému *Signys*, který je určen výhradně pro *Windows*.

V teoretické části jsou popsány technologie, které byly pro vývoj použity. Nejdříve je tedy uveden framework *Flutter*, dále programovací jazyk *Dart*. Následně jsou rozepsány některé alternativy a jejich výhody nebo nevýhody oproti použitým technologiím. Dále se práce věnuje syntaxi frameworku *Flutter* a programovacího jazyku *Dart*. Jsou zde uvedeny některé jednoduché příklady, ze kterých je možné si odvodit, jak se s technologiemi pracuje.

Přechodem mezi teoretickou a praktickou částí je kapitola, která se věnuje problematice napojení na *ERP* systém. Je zde uvedeno, jak je tento problém řešitelný a co to obnáší. Následně je popsáno, jak bylo napojení na *ERP* řešeno při vývoji této mobilní aplikace.

Praktická část pokračuje popisem procesu vývoje a následné publikace do obchodů *Google Play* a *App Store*. V poslední kapitole je představena výsledná aplikace a některé její zásadní funkce pomocí popsaných screenshotů.

1 CÍL PRÁCE

Cílem této bakalářské práce je vytvoření mobilní aplikace pro operační systémy *Android* a *iOS* za použití programovacího jazyku *Dart* a frameworku *Flutter*, která bude komunikovat s *ERP* systémem *Signys*.

2 FLUTTER & DART, POROVNÁNÍ S JINÝMI NÁSTROJI

2.1 FLUTTER

Flutter je open-source framework vydaný společností *Google* (první vydání květen 2017) pro vývoj multiplatformních aplikací. (3) S frameworkem se pracuje za pomoci programovacího jazyku *Dart*. Vývoj pro všechny platformy probíhá v rámci jednoho zdrojového kódu – odpadá potřeba správy více projektů v případě vývoje pro více operačních systémů. Jádro frameworku je napsané v *C++* (3).

Při práci s *Flutterem* se využívají takzvané widgety. Znamená to, že uživatel vytváří své vlastní widgety vnořováním již existujících (např. *Container*, *Column*, *Row*). (3) Tyto widgety se následně vykreslují. Veškerou práci s frameworkem usnadňuje takzvaný *hot reload* (3), díky kterému se změny v emulátoru nebo na fyzickém zařízení projeví v řádu sekund a není tak zapotřebí celou aplikaci znovu sestavovat.

Flutter je podporován několika vývojovými prostředími. Mezi nejznámější a zároveň nejvyužívanější se řadí *Android Studio*, nebo například *Visual Studio Code*.

Podporované platformy (3):

- Mobilní zařízení – *iOS* a *Android*
- Web
- Desktop – *Windows*, *macOS*, *Linux*
- Embedded (vestavěné) systémy – chytrá zařízení, auta atd.

Výhody frameworku *Flutter*:

- Stejný kód pro více platforem
- Vysoký výkon ve vykreslování animací
- Vlastní widgety zajišťují, že aplikace bude mít shodný vzhled na novém a starém operačním systému

2.2 DART

Dart je objektově orientovaný open-source programovací jazyk vyvíjený společností *Google*. (4) *Dart* je optimalizovaný pro vývoj klientských aplikací a zejména pro práci s frameworkem *Flutter*.

Syntaxe programovacího jazyku se podobá jazykům *Java, JavaScript* nebo *C*. Více informací o syntaxi programovacího jazyku je rozebráno v kapitole č. 3.2.

2.3 POROVNÁNÍ S JINÝMI VÝVOJOVÝMI NÁSTROJI

2.3.1 REACT NATIVE

React Native je stejně jako *Flutter* open-source multiplatformní framework pro vývoj mobilních aplikací. Jedná se o framework vytvořený společností *Meta* a pro práci s ním se využívá programovací jazyk *JavaScript*. První oficiální vydání bylo veřejně uvedeno v roce 2015. (5)

Shodně jako *Flutter,* i *React Native* má jeden společný kód pro *iOS* a *Android*. Framework disponuje funkcí *fast refresh* (u *Flutteru hot reload*), díky které vidí vývojář změny okamžitě bez nutnosti znovu kompilovat projekt.

Mezi mobilní aplikace, které jsou napsané ve frameworku *React Native,* patří například *Facebook, Instagram, Pinterest.* (5)

Výhody *React Native* (5):

- Stejný kód pro více platforem
- Rozšířenější než *Flutter* – má větší komunitu vývojářů, více návodů apod.
- Jednoduchý na naučení

2.3.2 IONIC

Ionic je open-source sada nástrojů pro hybridní vývoj. Pro práci s frameworkem *Ionic* se využívá programovací jazyk *JavaScript*, konkrétně si vývojář může vybrat mezi některými knihovnami, například *React* nebo *Angular*. (6) Nevýhodou je, že na rozdíl od frameworků *Flutter* a *React Native* využívá *Ionic* k vykreslování tzv. *WebView*[1].

[1] Integrovaný webový prohlížeč (18)

Framework exceluje ve výkonu pro webové prohlížeče. Používá se proto také pro tvorbu *PWA* aplikací.

Mezi aplikace, pro jejichž vývoj byl využit framework Ionic, patří například *JustWatch* nebo *Nationwide.* (6)

Výhody *Ionic* (6):

- Framework je nezávislý na platformě
- Využívá rozšířený programovací jazyk *JavaScript*
- Vhodný pro tvorbu *PWA* aplikací

2.3.3 NATIVNÍ TECHNOLOGIE

Pro nativní vývoj na platformě *Android* se využívají jazyky *Kotlin* a *Java* a na *iOS* jazyk *Swift*. Aplikace vytvořené pomocí nativních technologií poskytují nejvyšší možný výkon a rychlost.

Vývoj aplikací s použitím nativních technologií je vhodný, pokud je zapotřebí se úzce vázat na daný operační systém – například komunikace s hardwarem, verze operačního systému apod. (7)

Nativní vývoj je finančně náročnější, jelikož zabere více času, případně vyžaduje vyšší počet vývojářů. Kódy pro každou platformu musí být odděleny.

Výhody nativního vývoje (7):

- Vyšší stabilita a vyšší výkon
- Vhodnější pro aplikace, které vyžadují vyšší výpočetní výkon
- Lepší komunikace se specifickým hardwarem zařízení
- Design aplikace odpovídající dané platformě

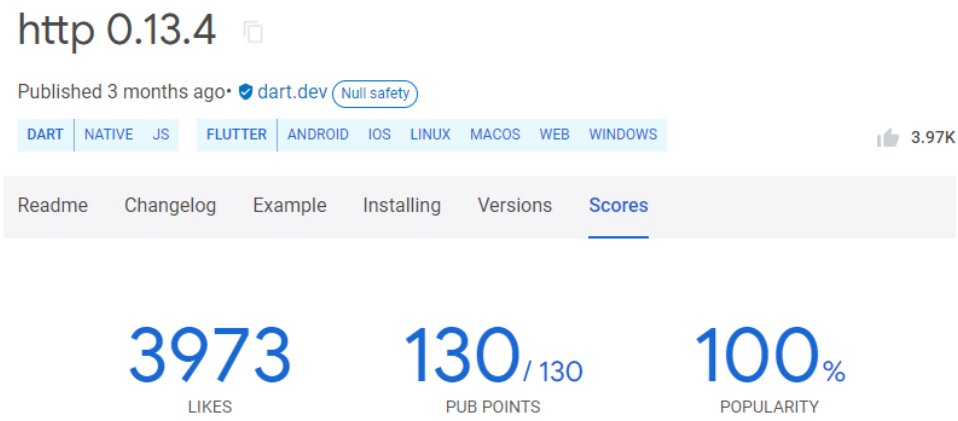
3 Způsob práce s Flutterem, syntaxe a konkrétní příklady

3.1 Balíčky

Flutter již v základu nabízí velké množství balíčků. Ve většině případů ale již zakomponovaný obsah není dostačující, a je proto zapotřebí jej rozšířit.

Všechny dostupné a kompatibilní balíčky, o které lze projekt rozšířit, jsou k dispozici na oficiální webové stránce *pub.dev*. Na stránce lze nalézt balíčky jak přímo od vývojářů *Flutter* nebo *Dart*, tak od vývojářů třetích stran.

Každý balíček má své hodnocení, podle kterého lze poznat, zdali je opravdu funkční a stabilní. Jsou k dispozici dva typy hodnocení – uživatelské a tzv. *pub points*, které vyjadřují, do jaké míry balíček vyhovuje nastaveným pravidlům a standardům.



Obrázek 1 - Hodnocení balíčku http (8)

Aby bylo možné balíček využívat, je zapotřebí jej do projektu stáhnout. Stažení probíhá zadáním následujícího příkazu do konzole vývojového prostředí, kde poslední slovo určuje, o který balíček se jedná:

```
$ flutter pub add http
```

Obrázek 2 - Přidání balíčku (8)

Jakmile je balíček stažen, je přidán záznam do souboru *pubspec.yaml*, konkrétně do kategorie *dependencies*. Tento soubor je nezbytnou součástí každého *Flutter* projektu.

```
dependencies:
  http: ^0.13.4
```

Obrázek 3 - Záznam v souboru pubspec.yaml (8)

3.2 SYNTAXE

Práce s *Flutterem* je poměrně specifická, ale zároveň velmi jednoduchá. Vše se odvíjí od klíčového slova *widget. Widget* je komponenta, která má různé vlastnosti a slouží k vykreslení obsahu na plátně. Jednotlivé widgety se do sebe mohou vnořovat, a tím tvořit komplexní widget dle potřeb vývojáře.

Pokud se jedná o definování parametrů, tvorbu návratových funkcí apod., tak se vše odehrává v programovacím jazyku *Dart*. Jak již bylo zmíněno, *Dart* má syntaxi velice podobnou jazykům *Java* nebo *JavaScript.*

K jazyku *Dart* je k dispozici velice rozsáhlá dokumentace, proto zde bude uvedeno několik základních, popřípadě zajímavých prvků, které tento jazyk nabízí.

Pro tuto kapitolu byly čerpány informace z oficiální stránek/dokumentace programovacího jazyku *Dart, dart.dev.* (4)

3.2.1 DATOVÉ TYPY A PROMĚNNÉ

Jazyk *Dart* nabízí všechny základní datové typy (*int, String, bool, List,* atd.). Některé datové typy jsou však z důvodu zpracování odpovědí z *API* serveru důležité a stojí za zmínku, například:

- ***Map* {}** – jedná se o typ, který spojuje klíče s hodnotami (*keys* and *values*). Každý klíč se může vyskytovat pouze jednou. Struktura je velice podobná souboru *JSON,* a proto se tento typ využívá pro přiřazení odpovědi *API* serveru v kombinaci s typem *List* (několik *Map* se může vnořit do jednoho *Listu,* klíč může mít hodnotu typu *List* apod.).
    - zajímavými operátory tohoto typu může být například *containsKey* (zjistí, zdali *Mapa* obsahuje daný klíč a vrátí *true*/*false*) nebo *forEach* (projde všechny klíče a jejich hodnoty)
- ***List* []** – jedná se o typ, který umožňuje shromažďovat několik jiných nebo stejných datových typů. V aplikaci se často využívá v kombinaci s *Map.* Používá se ve widgetu *ListView*, kde obsahuje několik elementů typu *Map*, čímž je dosaženo, že každá karta v seznamu může obsahovat velké množství informací.

- operátory, které se v aplikaci často vyskytují jsou například *where* (vrátí seznam elementů, které odpovídají zadané podmínce) nebo *removeAt* (smaže element na konkrétním indexu)

```
Map gifts = {
  'jmeno': 'Tomáš',
  'prijmeni': 'Podmanický',
  'titul': 'Bc.'
};
```

Obrázek 4 - Příklad typu Map (zdroj: autor)

Proměnné se v jazyku *Dart* definují podobně jako například v jazyku *JavaScript*. Je možné využít operátor *var* pro vytvoření proměnné, jejíž datový typ se určí až při prvním přiřazení, nebo datový typ určí již při vytváření proměnné (*String*, *int*, aj.). Je také možné využít operátor *final* (proměnná může být nastavena pouze jednou a je neměnná).

3.2.2 Podmínky

Zápis podmínek je opět velice podobný jazyku *JavaScript* (pro rovnost se využívají 2 rovnítka namísto 3).

```
if(cislo == 2){
   print(„Číslo je rovno“);
}else if(cislo < 2){
   print(„Číslo je menší“);
}else{
   print(„Číslo je větší“);
}
```

Obrázek 5 - Zápis podmínky (zdroj: autor)

Dart umožňuje zapsat podmínku pomocí ternárního operátoru. Tento zápis lze využít i přímo ve widgetech pro nastavení barvy boxu atd. Na obrázku 6 je následující podmínka: pokud je proměnná *polozky* prázdná, přiřaď do proměnné *value* hodnotu 0, jinak 1.

```
int value = polozky.isEmpty ? 0 : 1;
```

Obrázek 6 - Jednořádková podmínka (zdroj: autor)

3.2.3 Funkce

Zápis funkce je obdobný s mnoha jinými programovacími jazyky. Zajímavostí ale může být využití složených závorek nebo klíčového slova *@required*.

Složené závorky určují, že se jedná o jmenné vstupní parametry. To znamená, že při zápisu volání metody je možné vstupní parametry zpřeházet a bude vidět, jak se jmenují.

```
void vyhledejCenikovouKartu ({
    @required String nazev,
    int kategorie
}) async {
    // provedení dotazu
}
```

Obrázek 7 - Vstupní parametry funkce (zdroj: autor)

Klíčové slovo *@required* se využívá při práci s jmennými parametry a určuje, zdali je parametr vyžadován nebo ne. Pokud ne, je výchozí hodnota, není-li uvedeno jinak, *null*.

3.2.4 ASYNCHRONNÍ OPERACE

Jazyk *Dart* umožňuje asynchronní operace. V případě *ERP* aplikace s napojením na *API* a komplexní funkčností se jedná o nezbytnost. K implementaci poskytuje *Dart* klíčová slova *async* (určuje, že funkce je asynchronní) a *await* (umisťuje se před volání asynchronní funkce).

```
Future<void> overVerziAPI() async {
      var verze = await getVerze();
      // Porovnání verzí atd.
}
```

Obrázek 8 - Asynchronní funkce (zdroj: autor)

Pokud se jedná o návratovou funkci, která má datový typ *String* a je převedena na asynchronní, bude její datový typ *Future<String>* a pro získání řetězce bude nutné přidat před volání klíčové slovo *await*.

3.3 WIDGETY

Jelikož je *Flutter* multiplatformní a jeden a ten stejný kód lze bez úprav spustit na *Android* a *iOS* zařízení zároveň, obsahuje i specifické widgety. Všechny *widgety* je možné použít napříč platformami – to znamená, že je možné vytvořit aplikaci pro *Android*, která bude vypadat jako aplikace pro *iOS* a opačně.

Widgety, které jsou designem, ale mnohdy i specifickou funkčností určeny pro *iOS*, obsahují ve svém názvu slovo *Cupertino*. Bohužel ne vždy jsou widgety pro *iOS* v rámci parametrů shodné s těmi pro *Android*, a proto se může zdrojový kód protáhnout.

Pro tuto kapitolu byly čerpány informace z oficiální stránek/dokumentace frameworku *Flutter, docs.flutter.dev.* (3)

```
class Kontakt extends StatelessWidget {
  final String jmeno;
  Kontakt({
    @required this.jmeno
  });

  @override
  Widget build(BuildContext context) {
    return Container(
      child: Text(jmeno),
    );  // Container
  }
}
```

Obrázek 9 - Stateless widget (zdroj: autor)

3.3.1 Rozdíl mezi Stateful a Stateless widgetem

Ve *Flutteru* se *widgety* rozdělují na dva typy – *Stateful a Stateless*. Vytvoření těchto *widgetů* znamená definovat třídu požadovaného typu, jejímž základem je metoda *build*, která je nutnou součástí, protože její obsah se při vytvoření instance vykreslí.

Vlastnosti *Stateless widgetu* jsou neměnné – nelze je v průběhu změnit. Tento druh *widgetů* je vhodné využívat na některé méně komplikované *widgety*, u kterých se neočekává dynamická změna obsahu.

Naproti tomu *Stateful widget* svůj stav, a tím pádem i své parametry, dokáže v průběhu měnit. Z toho důvodu je nejvhodnější tento *widget* využít jako základní kámen každé stránky. Obnova zobrazovaných dat, což je v případě *ERP* aplikace zásadní, tak nebude problém. *Stateful widget* je z důvodu zachování/obnovení stavu nutné definovat pomocí dvou tříd.

- *StatefulWidget* rozšiřuje *Widget*, to znamená, že je také neměnný. Pokud se ale deklarace rozdělí do dvou tříd, je možné, aby byl *StatefulWidget* neměnný a *State* proměnlivý.
- V případě, že by byly třídy sloučeny, instance třídy by obnovila všechny vlastnosti aktuálního stavu kdykoliv, kdy by se rodič obnovil.

```dart
class Kontakt extends StatefulWidget {
  final String jmeno;
  Kontakt({
    @required this.jmeno
  });

  @override
  _KontaktState createState() => _KontaktState();
}

class _KontaktState extends State<Kontakt>{
  @override
  Widget build(BuildContext context) {
    return Container(
      child: Text(widget.jmeno),
    );  // Container
  }
}
```

Obrázek 10 - Stateful widget (zdroj: autor)

Z uvedených příkladů je patrné, že *Stateless* a *Stateful widget* může mít vstupní parametry. V obou případech je nutné, aby parametry byly neměnné, to znamená *final*. U vstupních parametrů lze určit, zdali jsou vyžadované - *@required*. V případě, že je parametr vyžadován, bude *IDE* hlásit varování, pokud parametr zadaný není. Výhodou je, že v případě více parametrů nemusí být každý vyžadovaný a lze tak zajistit vysokou míru přizpůsobitelnosti. Pro příklad *widget Column* vyžaduje jediný parametr, *children*, ostatní jsou volitelné.

```dart
return Kontakt(
    jmeno: "Tomáš Podmanický"
);
```

Obrázek 11 - Užití Stateless/Stateful widgetu (zdroj: autor)

3.3.2 MaterialApp

Základem každé *Flutter* aplikace je soubor s názvem *main.dart*. Tento soubor je v podstatě hlavní „rozcestník“ a je zde definována funkčnost, která platí v rámci celé aplikace. Obsahem souboru je ideálně *Stateful widget*, jehož *build* metoda vrací důležitý *widget MaterialApp* nebo *CupertinoApp*.

Důležité parametry (pro případ, že je vedena evidence *Routes*):

- *builder(BuildContext context, Widget child)* – vrací *child*, který může být vnořený do jiného rozšiřujícího *widgetu*
- *initialRoute, navigatorObservers, onGenerateRoute* – parametry, které slouží k definování práce s *Routes*
- *theme* – definuje font a barvy, které má aplikace používat pro ikony, tlačítka apod.

3.3.3 Container

Container je základní *widget*, který se nejčastěji používá pro „obalení“ ostatních *widgetů* a je jim tak díky němu možné definovat různé vlastnosti. Do *widgetu Container* lze vnořit jeden další *widget* (parametr *child*).

Příklad některých vlastností:

- ***decoration: BoxDecoration()*** – definuje barvy, okraje a jiné dekorace
- ***padding: EdgeInsets.all(10)*** – definuje odsazení. Je možné určit pro každý okraj zvlášť. Je to jeden z nejvyužívanějších parametrů *widgetu Container* a zároveň častý důvod využívání tohoto *widgetu*.
- ***width: double.infinity*** – definuje šířku *widgetu*. Často využívanou hodnotou je *double.infinity* – určí, že *Container* má zabrat veškerou dostupnou šířku.

3.3.4 Column, Row

Column a *Row* jsou základní *widgety*, které slouží pro uspořádání více vnořených *widgetů* vedle sebe. *Column* uspořádá vnořené *widgety* vertikálně, zatímco *Row* horizontálně. Vnoření více *widgetů* je možné díky parametru *children*, který je datového typu *List<Widget>*.

I přesto, že se jedná o základní a jedny z nejvyužívanějších *widgetů*, nedisponují zásadními parametry jako třeba *padding*. Proto je vhodné *Column* nebo *Row* vnořit například do *widgetu Container*, čímž se rozšíří možnosti přizpůsobitelnosti.

3.3.5 ListView

ERP systém obsahuje obrovské množství dat, jehož zobrazení je v drtivé většině případů jednodušší na obrazovkách počítače nežli na mobilním zařízení. Nejpřehlednější možností, jak vykreslit velké množství dat, je použití seznamu.

Flutter nabízí *widget ListView*, který umožňuje velkou míru přizpůsobitelnosti a práce s ním je velice jednoduchá.

K dispozici je několik druhů *ListView*:

- ***ListView*** – základní *widget,* který disponuje parametrem *children*. Jeho obsah není dynamický, ale *widget* nabízí všechny parametry, pro nastavení rolování, odsazení apod.
- ***ListView.builder*** – rozšiřuje základní *widget*. Položky jsou tvořeny dynamicky díky parametrům *itemBuilder* a *itemCount.*
- ***ListView.separated*** – rozšiřuje *widget ListView.builder* o takzvaný separátor. Separátor je určen k oddělení položek seznamu a definuje se pomocí parametru *separatorBuilder*. Nejčastěji se jako separátor mezi položkami využívá *widget Divider*, který vytvoří jednoduchou horizontální čáru.

```
List firmy = [
  {
    "nazev": "ABC s.r.o.",
    "adresa": "Květinová 123, Praha",
    "status": 0
  },
  {
    "nazev": "A Servis",
    "adresa": "Průmyslová 34, Hradec Králové",
    "status": 1
  }
];

@override
Widget build(BuildContext context) {
  return ListView.builder(
    itemCount: firmy.length,
    itemBuilder: (BuildContext context, int index){
      return Container(
        padding: EdgeInsets.symmetric(horizontal: 5), // odsazení 5px
        child: Card(
          elevation: 10, // efekt stínu
          child: Container(
            padding: EdgeInsets.all(7.5), // odsazení od okraje Card
            child: Column(
              children: [
                Text(firmy[index]["nazev"], style: TextStyle(fontSize: 18)),
                Text(firmy[index]["adresa"]),
                if(firmy[index]["status"] == 1)
                  Text("Aktivní", style: TextStyle(color: Colors.green)),
                if(firmy[index]["status"] == 0)
                  Text("Neaktivní", style: TextStyle(color: Colors.red)),
              ],
            ),  // Column
          ),  // Container
        ),  // Card
      );  // Container
    }
  );  // ListView.builder
}
```

Obrázek 12 - ListView – jednoduchý seznam firem (zdroj: autor)

4 PROBLEMATIKA NAPOJENÍ NA ERP

ERP systémy využívají pro uchování dat své vlastní databáze. Každá databáze se liší svou strukturou, často technologií a zabezpečením. Přímé napojení mobilní aplikace na databázi proto není ideální hned z několika důvodů uvedených níže.

1. Veškerá datová logika by se nacházela v aplikaci, což může způsobit nepřehlednost, nespolehlivost a špatnou udržitelnost.
2. Náročná změna chybných nebo nevyhovujících dotazů, z důvodu způsobu, kterým jsou aplikace publikovány (čas, který vyžadují služby *Google* a *Apple* k validaci aplikace, se může pohybovat v řádu dnů, až týdnů).
3. Snížená bezpečnost přenosu dat.

4.1 ŘEŠENÍ ZÍSKÁNÍ DAT

Ideálním řešením pro napojení mobilní aplikace na *ERP* systém je využití *API*. *API* běží jako služba na serveru a obsahuje přímé napojení na databázi konkrétního *ERP* systému. Přístupnost *API* serveru určuje zabezpečení firemní infrastruktury a případné zpřístupnění veřejné *IP* adresy pro připojení bez nutnosti využití *VPN*.

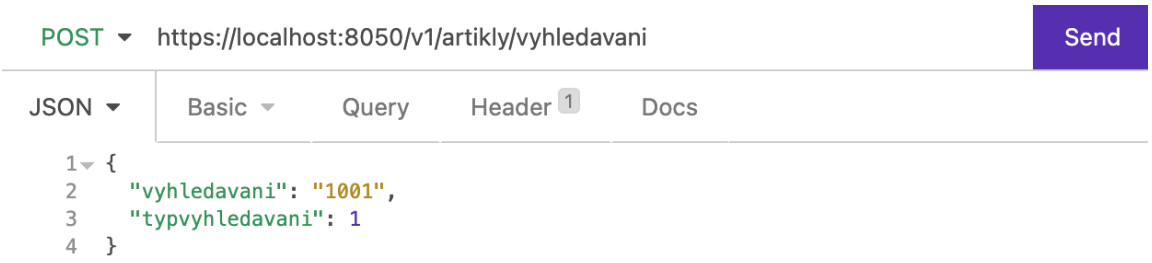
4.1.1 AUTENTIZACE

Dotazy na *API* server mohou být zasílány bez ověření autentizačních údajů. Jelikož ale aplikace, která je napojená na *ERP* systém, pracuje s citlivými firemními daty, je naprosto nezbytné, aby provedení dotazu vyžadovalo ověření.

Nejběžnějším způsobem ověření je tzv. *Basic access authentication*. Znamená to, že autentizační údaje (jméno a heslo) jsou z aplikace zasílány jako jeden textový řetězec, který je zakódován pomocí kódování *Base64*. Nezbytnou nutností při použití tohoto typu ověření je, aby spojení mezi serverem a klientem bylo zabezpečené. K tomu lze využít protokol *HTTPS*.

4.1.2 Tělo dotazu

V případě dotazů, které jsou například typu *POST* nebo *PUT*, očekává *API* server takzvané tělo dotazu (*request body*), které je určeno pro zaslání dat nebo specifikujících parametrů na *API* server. Na základě těchto parametrů se provede požadovaná operace. Tělo dotazu je ve formátu *JSON* a je předem určeno, které parametry jsou vyžadovány.



Obrázek 13 - Příklad dotazu POST a jeho těla (zdroj: autor)

Na obrázku č. 13 je zobrazena adresa pro dotaz, který je určen pro vyhledávání ceníkových artiklů. Základními parametry jsou *vyhledavani* a *typvyhledavani*, kde první parametr je pro vložení textového řetězce a druhý upřesňuje, v jakých databázových polích má *API* server vyhledávat. Výsledkem tohoto dotazu je odpověď obsahující tolik artiklů, kolik jich bylo v databázi nalezeno.

4.1.3 Odesílání dotazů z aplikace

Pro zasílání dotazů na *API* z aplikace je k dispozici knihovna, kterou poskytují vývojáři *Dart*. Nese název *http.dart* a do zdrojového kódu ji po instalaci lze importovat následovně:

```
import 'package:http/http.dart' as http;
```

Obrázek 14 - Import knihovny http (8)

Z důvodu, že knihovna obsahuje stejné názvy některých tříd jako knihovny, které jsou součástí *Flutteru* již od instalace, je za importem definováno *as http*.

Knihovna umožňuje využívat *HTTP* metody *POST, PUT, GET, DELETE* aj. Je také možné zasílat několik různých typů takzvaných *headers*, ty jsou určeny například pro autorizační řetězec, pro upřesnění požadovaného formátu odpovědi nebo zde může být *ID* zařízení a jiné údaje, které se zasílají při každém dotazu.

```
String username = Credentials.username;
String password = Credentials.password;
String basicAuth = "Basic " + base64Encode(utf8.encode("$username:$password"));
```

Obrázek 15 - Autentizační údaje, potřebné pro zaslání dotazu (zdroj: autor)

```
Uri adresa = Uri.parse("https://localhost:8050/v1" + uri);
http.Response response = await http.post(
    adresa,
    headers: {
      HttpHeaders.authorizationHeader: basicAuth,
      "Content-type" : "application/json",
      "Accept" : "application/json",
      "Device-ID": Platform.isAndroid ? SystemConf.androidDeviceInfo.androidId :
                  Platform.isIOS ? SystemConf.iosDeviceInfo.identifierForVendor : ""
    },
    body: body
).timeout(Duration(seconds: SystemConf.TIMEOUT));
```

Obrázek 16 - Funkce z knihovny http (zdroj: autor)

Na obrázku č. 16 je zobrazena funkce *post* z knihovny *http.dart* (8), která po úspěšném dokončení získá odpověď z *API* server ve formátu *JSON*.

Z hlediska přehlednosti zdrojového kódu je vhodné, aby byly funkce, které pracují s knihovnou *http.dart*, oddělené a přístupné v rámci celé aplikace. Volání této funkce může tedy vypadat například takto:

```
final body = json.encoder.convert({
  "vyhledavani": vyhledavani,
  "typvyhledavani": SystemConf.typVyhledavaniCenik
});
return await requestPost(
    uri: "/artikly/vyhledavani",
    body: body,
    onSuccess: (response, int statusCode){
      return response;
    },
    onError: (error){
      return error;
    },
    onTimeout: (timeoutError){
      return timeoutError;
    }
);
```

Obrázek 17 - Volání globální funkce pro zaslání dotazu na API (zdroj: autor)

4.1.4 Odpověď API serveru

Výsledkem dotazu je odpověď, *API response*. Formát, který může odpověď mít, určuje architekt *API* serveru. V některých případech je možné, že *API* nabízí více formátů a klientská strana si může vybrat jí vyhovující. Nejběžnějšími formáty odpovědi jsou *XML* a *JSON*. V případě aplikace vytvořené pro *ERP Signys* je využíváno odpovědi ve formátu *JSON* z důvodů přehlednosti a jednoduchosti.


```
[
  {
    "prirazka7": 0,
    "velikost": "",
    "barva": "",
    "prodejvaluty": 0,
    "opravil": "Administrátor",
    "ktgref": "2600810",
    "ktginterni": "1001",
    "nakup": 370,
    "bj": 1,
    "delka": 0,
    "objem": 0,
    "mj": "Ks",
    "cas": 0,
    "sazbadph": 21,
    "prumernynakup": 815.5421011166588,
    "mena": "CZK",
    "kurz": 0,
    "menamnozstvi": 0,
    "jcd": "",
    "pakcnicena": 0,
    "pbody": 0,
    "bodyhodnota": 0,
    "ktgalt": "8590111000102",
    "delkazaruky": 0,
    "pzdrojnakup": 0,
    "pspec1": 0,
    "pspec2": 1,
    "pbonus": 0,
    "pobjstatus": 0,
    "produktmng": 0,
    "rbkod": 0,
    "kodeod": "",
    "aunaklady": "",
    "auvynosy": "",
    "auzasoby": "",
    "stredisko": 0,
    "kodzbozi": "",
    "statisznak": 0,
    "kodzemepuvodu": "",
```


Obrázek 18 - Příklad odpovědi API serveru (zdroj: autor)

Odpověď může být buď úspěšná, nebo neúspěšná. Úspěšnost doručení nebo vyřízení dotazu lze určit pomocí třímístných stavových kódů. Stavových kódů je celá řada, ale pro vyloženě úspěšné vykonání dotazu se používají kódy, která začínají číslem **2**. Příklady kódů z každé skupiny jsou uvedeny v Tabulce 1.

Tabulka 1 - Příklad stavových kódů (9)

Kód	Popis
100	**Continue** – vše proběhlo v pořádku a klient může pokračovat
202	**Accepted** – žádost byla přijata ke zpracování, ale zpracování nebylo dokončeno
300	**Multiple Choices** – žádost má více, než jednu možnou odpověď
401	**Unauthorized** – neplatné autentizační údaje, které jsou potřebné pro provedení žádosti
500	**Internal Server Error** – chyba na straně serveru/*API* rozhraní. Neočekávaná výjimka.

4.1.5 Zpracování odpovědi v aplikaci

V případě, že byl dotaz úspěšně zpracován a do aplikace dorazila odpověď v požadovaném formátu, je zapotřebí odpověď rozkódovat a převést do korektního modelu.

V případě aplikace vytvořené pro tuto práci je využíván balíček *json_serializable*. Tento balíček slouží k vygenerování samostatných tříd, které obsahují metody pro převod *JSON* odpovědi do námi vytvořeného objektu/modelu. Balíček umožňuje vnořování dalších objektů, jejichž parametry jsou následně přes hlavní objekt přístupné.

```dart
part 'kontakt.g.dart';

@JsonSerializable()
class Kontakt {
  @JsonKey(defaultValue: [])
  List<KontaktAdresa> adresy;
  String oidadrktgm01;
  String poridil;
  String dporizeno;
  String popis;
  int poradi;
  int pdelete;
  String kancelar;
  String daktualizace;
  String oidadrktgd01;
  String kontakt;
  String jmeno;
  String titul;
  String funkce;
  String kategorie;
  String oddeleni;
  Kontakt();

  factory Kontakt.fromJson(Map<String, dynamic> json) => _$KontaktFromJson(json);
  Map<String, dynamic> toJson() => _$KontaktToJson(this);
}
```

Obrázek 19 - Zápis modelu s využitím json_serializable (17)

Využívání těchto modelů výrazně usnadňuje práci s kódem a snižuje možnou chybovost. Metoda pro převod *JSON* do modelu očekává datový typ *Map*, proto je zapotřebí odpověď dekódovat.

```dart
var response = json.decode(utf8.decode(response.bodyBytes));
```

Obrázek 20 - Dekódování JSON odpovědi (zdroj: autor)

Po dekódování je možné naplnit objekt pomocí metody *fromJson*. K parametrům objektu lze přistupovat pomocí tečky za jeho instancí.

```dart
Kontakt _$KontaktFromJson(Map<String, dynamic> json) {
  return Kontakt()
    ..adresy = (json['adresy'] as List)
          ?.map((e) => e == null
              ? null
              : KontaktAdresa.fromJson(e as Map<String, dynamic>))
          ?.toList() ??
      []
    ..oidadrktgm01 = json['oidadrktgm01'] as String
    ..poridil = json['poridil'] as String
    ..dporizeno = json['dporizeno'] as String
    ..popis = json['popis'] as String
    ..poradi = json['poradi'] as int
    ..pdelete = json['pdelete'] as int
    ..kancelar = json['kancelar'] as String
    ..daktualizace = json['daktualizace'] as String
    ..oidadrktgd01 = json['oidadrktgd01'] as String
    ..kontakt = json['kontakt'] as String
    ..jmeno = json['jmeno'] as String
    ..titul = json['titul'] as String
    ..funkce = json['funkce'] as String
    ..kategorie = json['kategorie'] as String
    ..oddeleni = json['oddeleni'] as String;
}
```

Obrázek 21 - Automaticky vygenerovaná metoda (17)

5 POPIS PROCESU VÝVOJE

Vývoj mobilní aplikace probíhá v několika fázích a je nutné zdůraznit, že se počítá s tím, aby byla aplikace i do budoucna plně funkční a dostupná.

Výsledkem vývoje je aplikace s názvem *Signys Mobile*, která je napojena výhradně na *ERP Signys*. Vývoj mobilní aplikace je neustále v procesu a do budoucna se počítá s dalšími rozšířeními.

5.1 ANALÝZA

Před začátkem vývoje bylo nutné se zadavatelem společně projít požadavky na mobilní aplikaci a provést analýzu. Některými z klíčových požadavků byly například dostupnost na *iOS* a *Android*, stejná logika práce jako v *ERP Signys* a přehlednost při práci s daty.

Účastnil jsem se několika školení, na kterých jsem nabyl poměrně solidní znalosti *ERP Signys* ze všech pohledů. Nezbytností bylo pochopení architektury databáze, ze které aplikace čerpá veškerá data a *API* serveru, který data zprostředkovává.

Při volbě způsobu vývoje připadaly v úvahu dvě možnosti. Konkrétně vývoj nativních aplikací pro *Android* a *iOS* v programovacích jazycích *Swift* a *Kotlin*, nebo využití multiplatformního frameworku. Z kapacitních a časových důvodů byl pro vývoj zvolen multiplatformní framework *Flutter*.

5.2 VÝVOJ

Při vývoji byla nejprve vytvořena základní kostra aplikace, která je nezbytná pro celkovou funkčnost. Na tuto kostru se dále navazovalo a přidávala se funkcionalita. Jelikož je aplikace modulární, neznamená implementace nového modulu zásah do již fungujících částí. V průběhu vývoje jsem úzce spolupracoval s klienty, kteří se účastnili testovacího provozu. Přinášeli mi svůj pohled na problematiku a aktivně aplikaci testovali. Tím jsem docílil toho, že jsem dokázal některé kroky zoptimalizovat a zároveň eliminovat chyby.

K vývoji aplikace je zapotřebí *IDE*, které podporuje framework *Flutter*. V mém případě se jednalo o *Android Studio*. Projekt je zálohovaný službou *GitHub*, kde je z důvodu přehlednosti a organizaci vydání rozdělen na tři větve.

5.2.1 UI DESIGN

„UI (User Interface) design je návrh prvků, celku, animací a interakcí.“ (10)

Cílem bylo vytvořit moderní, přehlednou a efektivní aplikaci. V případě aplikace informačního systému se může jednat o nelehký úkol, jelikož existuje nepřeberné množství dat, která lze zobrazit, a způsobů, jakými je lze nakombinovat.

Snažil jsem se proto, aby byla aplikace co nejvíce uživatelsky jednoduchá a zaměřená na základní operace, které jsou od ní očekávány. Se zadavatelem jsme se shodli, že náročnější práce s daty by měla být vždy prováděna v desktopové verzi *ERP Signys*.

5.3 IMPLEMENTACE

Aplikace je volně dostupná každému ke stažení v obchodě *Google Play* a *App Store*. Bez informačního systému *Signys* a jeho *API* serveru je však nepoužitelná.

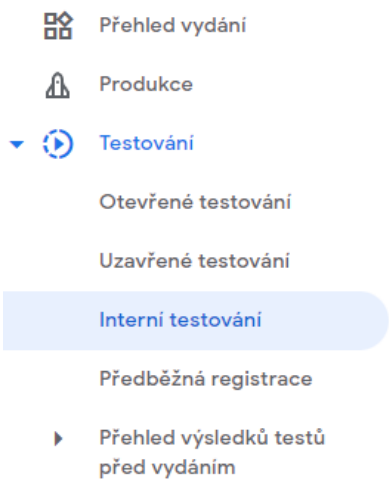
Pro nahrání nového vydání na výše zmiňované služby je zapotřebí, aby měla aplikace vyšší číslo verze nežli vydání předchozí. Služba *Google* umožňuje vždy zvyšovat pouze číslo buildu, to znamená označení za znakem +, zatímco *Apple* požaduje, aby vždy v případě aktualizace produkčního kanálu měla aplikace minimálně o číslo vyšší verzi.

5.3.1 PROCES VYPUBLIKOVÁNÍ APLIKACE

Proces vypublikování nové verze aplikace začíná korektním zvýšením verze a *commitem*[2]. Samotný proces je rozdílný pro platformu *Android* a *iOS*.

5.3.1.1 ANDROID

Ke správě aplikace ve službě *Google Play* se využívá webový nástroj pro vývojáře zvaný *Google Play Console*. Založení vývojářského účtu stojí jednorázově **25 dolarů** (2Q 2022). (11) V tomto nástroji je možné tvořit oddělená vydání, konkrétně *interní testování*, *otevřené testování*, *uzavřené testování* a *produkce*. (12) Pokud se chce uživatel zapojit do některého z testovacích kanálů, je nutné přidat *Google* účet, pod kterým je přihlášený na svém zařízení k danému testovacímu kanálu. Následně musí uživatel

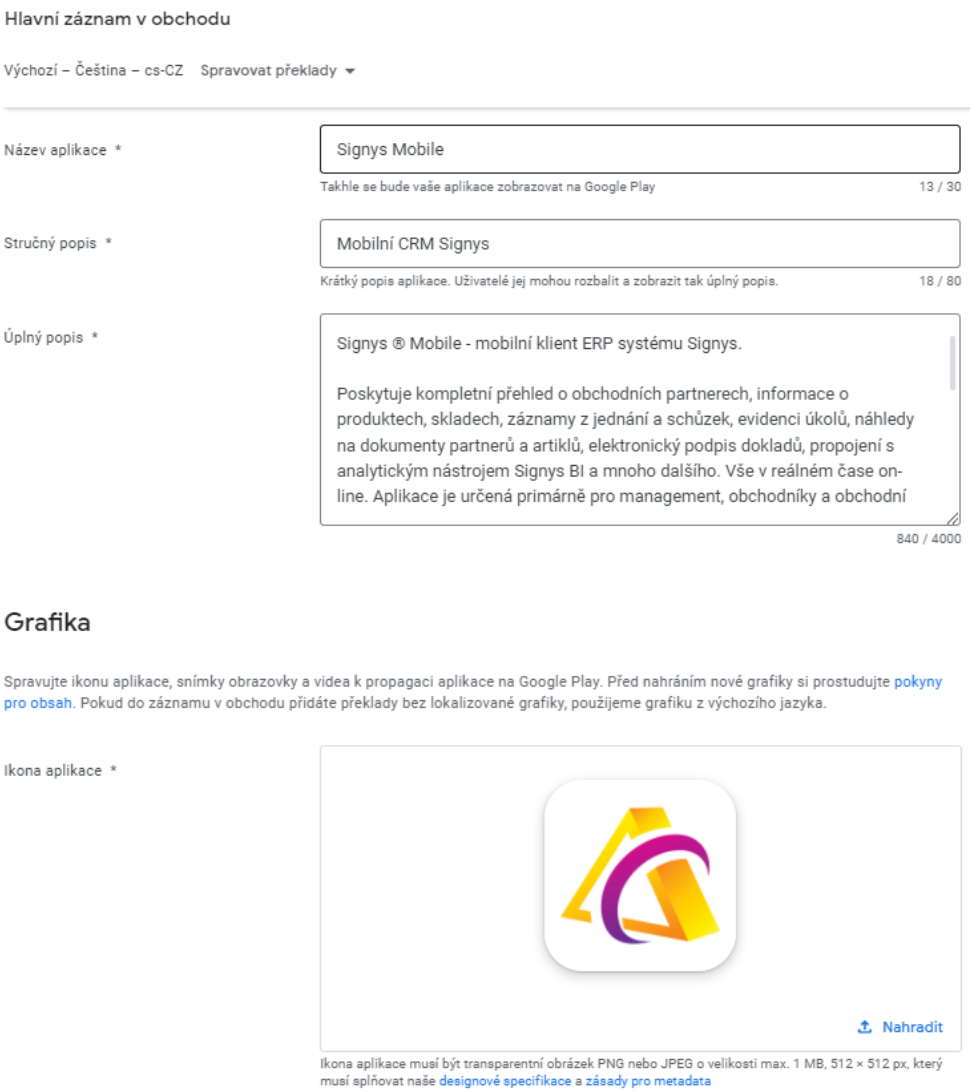


Obrázek 22 - Rozcestník kanálů (12)

[2] Odeslání změn do repositáře (16)

potvrdit vstup do testování (přes odkaz nebo v aplikaci *Google Play*). Po zprocesování a obnovení uvidí v obchodě *Google Play* jen tu verzi, která je aktuálně v testovacím kanálu, ke kterému je přihlášený.

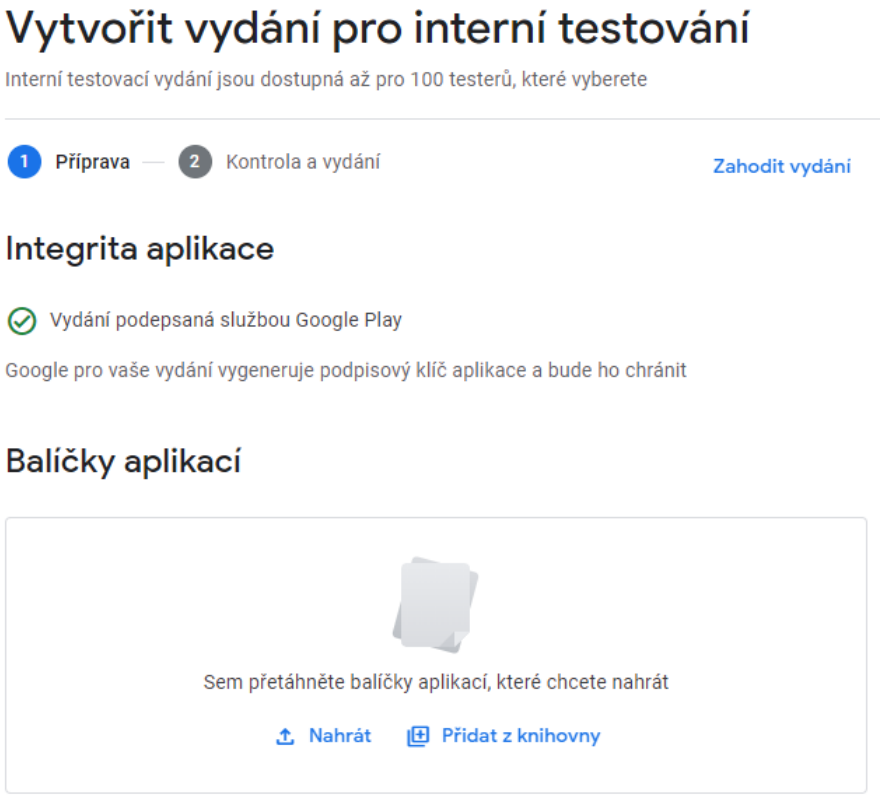
Aby byla aplikace veřejně přístupná v obchodě *Google Play*, je nutné v nástroji *Console* nastavit některé základní parametry, které určují, jak se bude aplikace zobrazovat. Nezbytně nutnými jsou grafické prvky (logo aplikace apod.), snímky obrazovky, název a popis.



Obrázek 23 - Ukázka nastavení záznamu v Google Play (12)

Aby bylo možné novou verzi do *Google Play Console* přidat, je zapotřebí v *IDE* zadat do terminálu příkaz *flutter build appbundle*. Tento příkaz vygeneruje soubor s příponou *.aab*, který je následně možné do *Console* nahrát. *Console* si převezme informace z balíčku a automaticky vydání přiřadí verzi, která byla uvedená v projektu. Následně je vhodné k vydání přiřadit novinky a potvrdit jej.

V případě kanálu *interní testování* (je pouze jeden) je verze schválena okamžitě a pro uživatele přihlášené k testování v obchodě *Google Play* dostupná v řádu minut. Pokud se ale jedná o kanály *uzavřeného testování* (může jich být více) nebo kanál *produkce*, prochází vydání kontrolou. Délka kontroly není předem známa a může trvat v rozmezí minut, až dnů.



Obrázek 24 - Tvorba nového vydání (12)

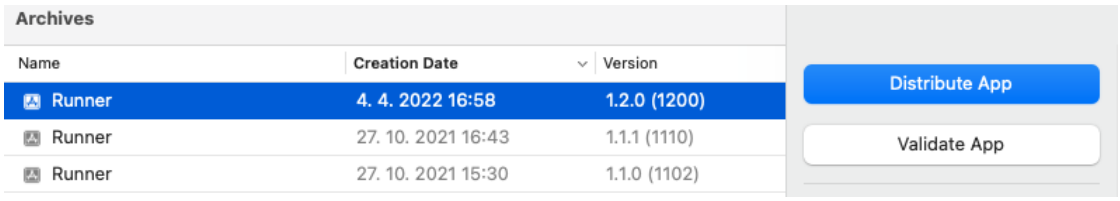
Pro všechny kanály, kromě *interního testování* je nutné, aby byl uveden postup, jak se do aplikace přihlásit a všechny pro to potřebné údaje. Aplikace jinak nemůže být schválena.

5.3.1.2 IOS

Ke správě vydání pro iOS se využívá webový nástroj *App Store Connect* (dále jen *APS*). Pro přístup do tohoto nástroje je nutné mít *Apple Developer* účet a zaplacené roční předplatné, které stojí **99 dolarů** (2Q 2022). (13)

Aby bylo možné aplikaci publikovat, je nutné mít k dispozici počítač s operačním systémem *macOS*, na kterém musí být nainstalovaný program *XCode*. Jakmile jsou tyto podmínky splněny, je možné projekt, konkrétně část projektu, která je určena pro *iOS* (v projektu v podsložce */ios*), otevřít v programu *XCode*. Po otevření projektu v nástroji *XCode* je zapotřebí zacílit build na *„Any iOS Device (arm64, armv7)“* – díky tomu se zpřístupní možnost projekt archivovat.

Archivace projektu provede validaci kódu, balíčků a kompatibilitu verzí. Pokud je úspěšná, vznikne nový záznam v tzv. organizéru archivů. Z tohoto organizéru je možné aplikaci distribuovat přímo do nástroje *APS*.

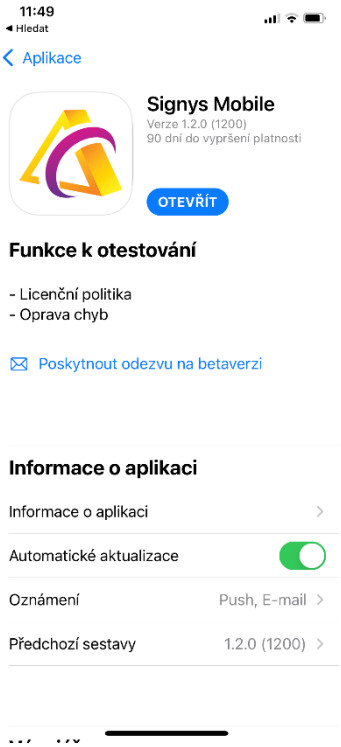


Obrázek 25 - Seznam vytvořených archivů v XCode (zdroj: autor)

Po potvrzení distribuce aplikace se provede ověření licencí a následně se balíček objeví v testovacím rozhraní *TestFlight*. Zde je možné vydání přiřadit testery, novinky a potvrdit jeho testování – vydání se přesune do stavu *„Waiting for Review“*, což znamená, že se musí počkat na ověření a schválení.

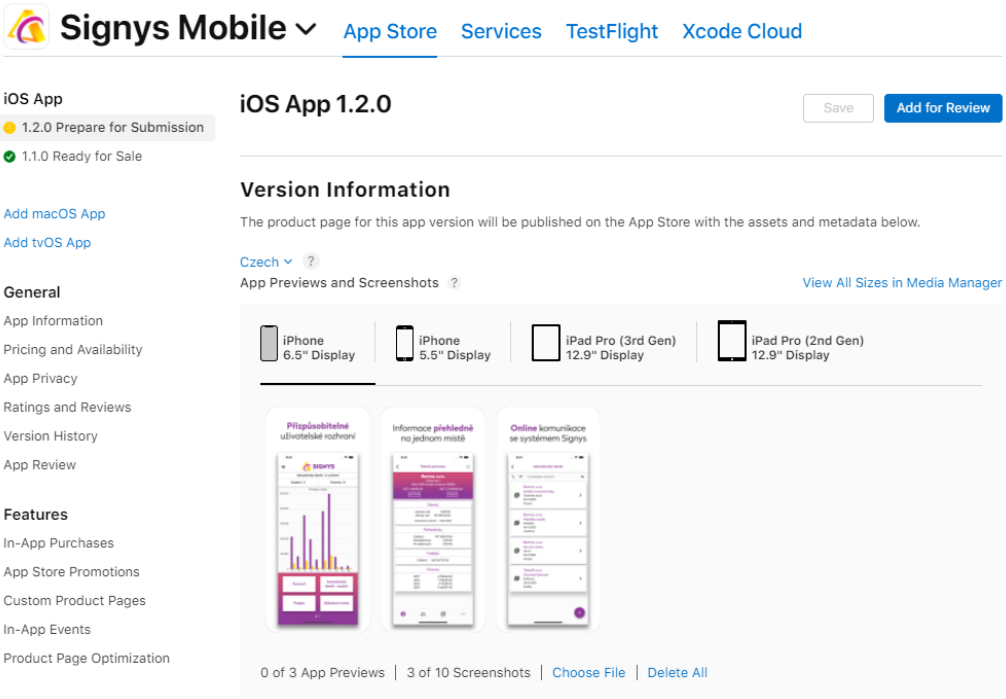
Pro produkční a testovací (oproti *Play Console* navíc) verzi je zapotřebí, aby byl u aplikace uveden postup, jak se do ní přihlásit, a přihlašovací údaje.

TestFlight je prostředí pro vývojáře a testery aplikací, do kterého vývojář nahrává testovací verze aplikací. Z pohledu vývojáře stačí vytvořit skupiny testerů a ty přiřadit požadovaným aplikacím. Z pohledu testera stačí v obchodě *App Store* stáhnout stejnojmennou aplikaci *TestFlight*, potvrdit pozvánku do testovacího programu, kterou zasílá vývojář, a aplikace určená k testování se v testovacím rozhraní zpřístupní. Z testu je možné kdykoliv odejít, případně přeinstalovat na stabilní verzi aplikace, která je dostupná v *App Store*.



Obrázek 26 - Záznam v aplikaci TestFlight (zdroj: autor)

Aby bylo možné produkční verzi zpřístupnit na *App Store*, vyžaduje *APS* vložení screenshotů v několika rozměrech (rozděleno podle velikostí displejů Apple zařízení) a základní popis aplikace. (14)

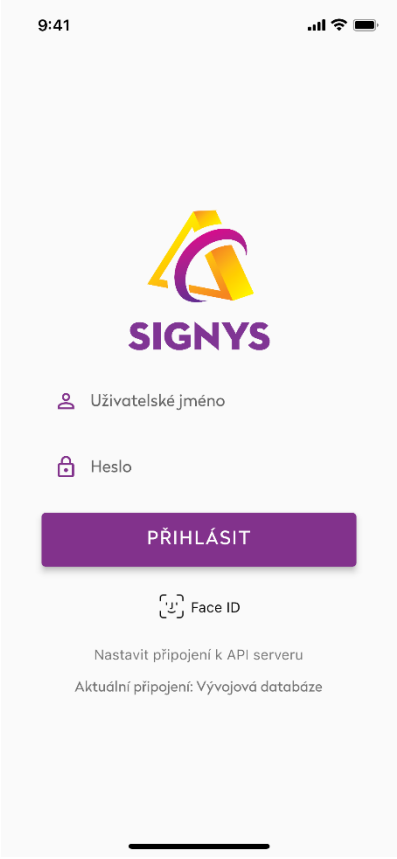


Obrázek 27 - Ukázka nastavení záznamu v App Store Connect (14)

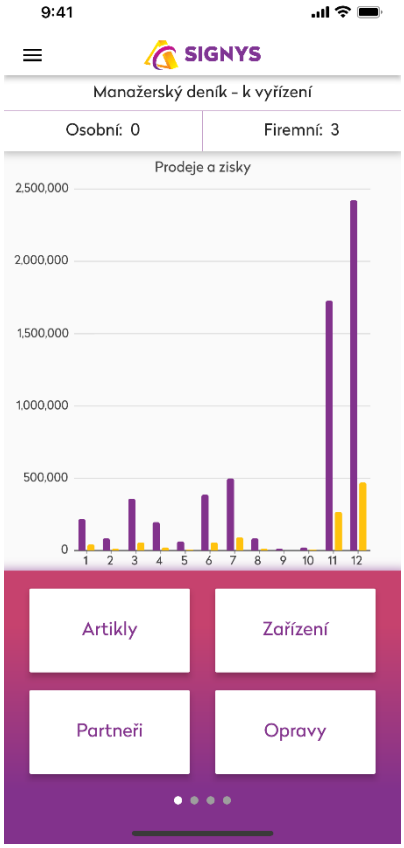
6 VÝSLEDKY

Výsledkem vývoje je modulární mobilní aplikace, která je pod názvem *Signys Mobile* veřejně dostupná v obchodech *Google Play* a *App Store*.

Aplikace je konstituovaná tak, aby nebylo možné provést takové akce, které by mohly narušit základní funkčnost a provozuschopnost informačního systému *Signys*. Aplikace je tedy určena z větší části pro získávání a následné zobrazovaní dat, ale v některých případech lze záznamy i tvořit.



Obrázek 28 - Přihlašovací stránka aplikace (zdroj: autor)



Obrázek 29 - Domovská stránka aplikace (zdroj: autor)

6.1 FUNKCE

Níže je uveden soupis funkcí, které jsou v aplikaci implementovány a stojí za zmínku.

- Biometrické přihlašování – pokud je na zařízení k dispozici, lze se díky balíčku *local_auth* přihlásit na *iOS* pomocí *TouchID/FaceID* a na *Android* pomocí dostupných možností.
- Přihlášení pod vlastním uživatelským účtem – do aplikace se přihlašuje pomocí uživatelského účtu, který je vytvořen v systému *Signys*. Zde je také možné uživatelskému účtu nastavit konkrétní parametry, které ovlivňují chování aplikace.

- Systém konkurenčních připojení – aplikaci může v jeden moment používat pouze tolik uživatelů, kolik licencí má klient zakoupených. V případě překročení tohoto limitu aplikace zobrazí upozornění a zamítne přihlášení. Zároveň je implementována funkčnost, která zajišťuje, že jeden účet může být ve stejnou chvíli přihlášen pouze na jednom zařízení.
- Odesílání e-mailů s dodacími listy – *API* server na základě tiskové sestavy vygeneruje *PDF* soubor.
- Seznam připojení – v aplikaci lze vytvořit několik připojení, mezi kterými se lze jednoduše přepojovat. Jednoduchým příkladem užití může být produkční/testovací databáze nebo firma, která má více nežli jednu pobočku.
- Napojení na *API* rozhraní *Google* a *ARES.*
- Pořizování fotografií – lze přidat fotografii k ceníkové položce nebo k dokladu.
- Snímání čárových kódů prostřednictvím fotoaparátu:
    - Vyhledávání – seznam ceníkových/skladových karet, tvorba nabídky.
    - Přidávání *API* připojení (dle předurčeného formátu snímaného kódu).

6.2 Moduly

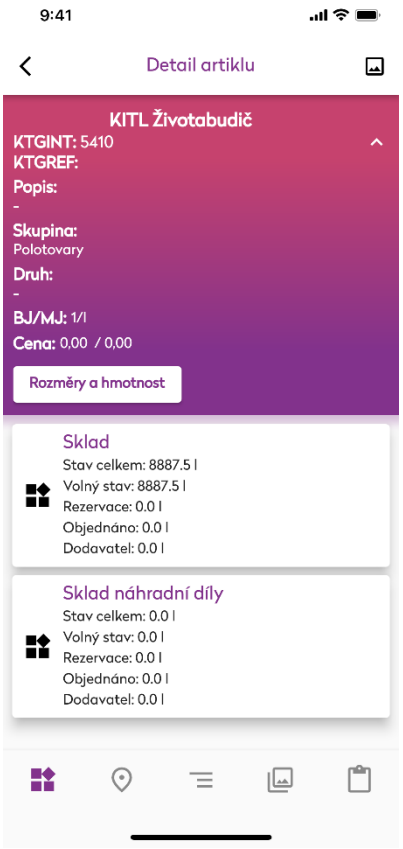
Moduly se nastavují v systému *Signys* přímo uživateli, nebo je lze nastavit globálně pomocí takzvané role, kterých může být několik. Roli je možné přiřadit více uživatelům a lze tak dosáhnout přehledného nastavení.

Na moduly je aplikována licenční politika a v aplikaci jsou proto přístupné pouze ty moduly, ke kterým v systému existuje licence. Pokud licence neexistuje, nelze do modulu vstoupit a zobrazí se u něj informační status.

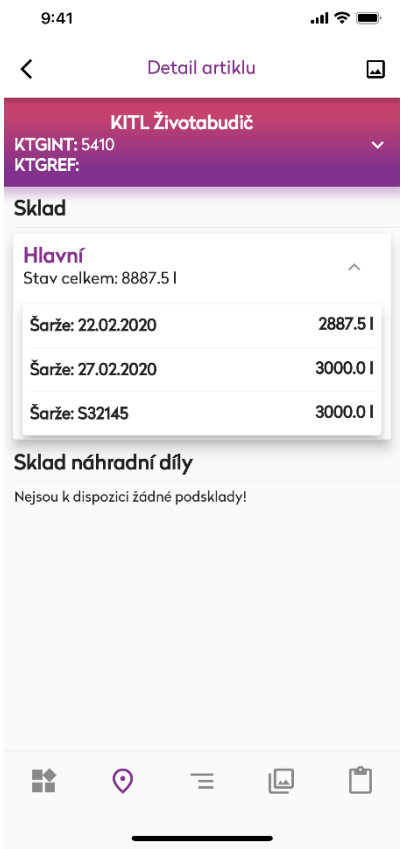
6.2.1 Ceník

Modul poskytuje seznam ceníkových karet, ve kterém je možné vyhledávat. Vyhledávání je pro vyšší rychlost možné nastavit na konkrétní parametry (např.: čárový kód, název, interní číslo apod.), ve výchozím stavu se vyhledává podle několika základních parametrů. Také lze filtrovat podle číselníků.

Aplikace umožňuje v seznamu ceníkových karet přehledně zobrazit kusovník – zobrazí se stromeček všech podřazených karet dané ceníkové karty. Stromeček může mít několik úrovní.



Obrázek 30 - Ceníková karta – sklady (zdroj: autor)



Obrázek 31 - Ceníková karta – stavy na skladech a podskladech (zdroj: autor)

Po rozkliknutí konkrétní ceníkové karty se zobrazí její detail, který se může na základě parametrů dynamicky měnit. Konkrétně se na základě nastavení můžou měnit zobrazené parametry ve sbalovací „hlavičce“ stránky. Obsah dolní lišty a tím pádem počet stránek se mění na základě příznaků artiklu nebo globálních nastavení.

Ikonkou v pravém horním rohu je možné rychle otevřít primární fotografii artiklu. Primární fotografie se zobrazuje v systému *Signys* v rychlém náhledu položky, případně se zobrazuje na e-shopu apod.

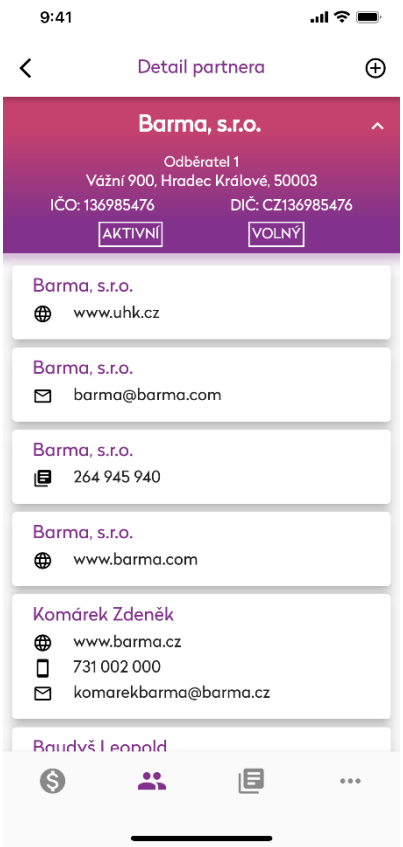
Tabulka 2 - Ikony navigační lišty ceníku (15)

	Stavy na skladech. Záložka se zobrazuje ve všech případech.
	Skladová místa/šarže a stavy na nich. Zobrazí se, pokud je v systému *Signys* zapnuta evidence skladových míst nebo pokud artikl používá šarže/expirace.
	Položky kusovníku daného artiklu. Mohou být nadřazené nebo podřazené. Zobrazí se pouze, když položka používá kusovník.
	Galerie fotografií daného artiklu. Zobrazuje se vždy.
	Dokumenty daného artiklu. Zobrazuje se vždy.

6.2.2 Adresář

Modul adresář poskytuje seznam partnerů. V seznamu lze vyhledávat podle názvu, *IČO* ad. Mezi partnery se řadí odběratel, dodavatel a také koncový zákazník. Jelikož se jedná o číselníky, je možné v seznamu dle těchto hodnot přehledně filtrovat.

Obrázek 32 - Detail partnera – finance (zdroj: autor)



Obrázek 33 - Detail partnera – kontakty (zdroj: autor)

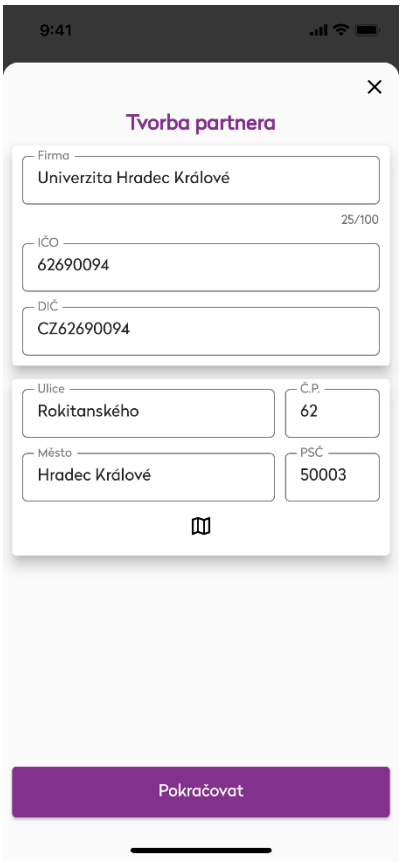
Po otevření konkrétního partnera se zobrazí jeho detail. Počet podstránek je na rozdíl od ceníkové karty pevně daný a nijak se nemění. V „hlavičce“ se zobrazují některé ze základních parametrů. Dále se na základě příznaků partnera zobrazuje, zdali je partner aktivní, blokovaný nebo pokud je dlužníkem.

Tabulka 3 - Ikony navigační lišty detailu partnera (15)

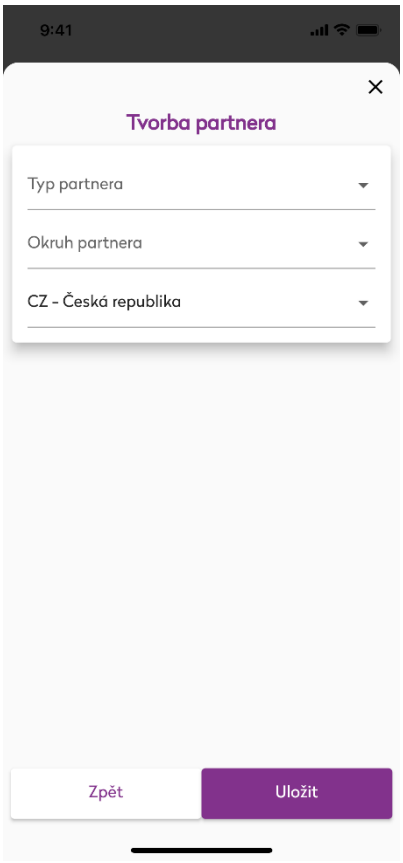
	Finanční informace o partnerovi ve vztahu k firmě, která aplikaci využívá.
	Kontakty partnera. Kontakty lze stiskem tlačítka v pravém horním rohu vytvářet, případně podržením záznamu upravovat. Kliknutím na konkrétní adresu se vyvolá systémová akce – otevření webové stránky/otevření emailového klienta atd.
	Na základě parametru se na této stránce zobrazují deníky partnera – velice jednoduché záznamy nebo manažerské deníky vázané na daného partnera. Deníky lze pod touto záložkou tvořit nebo upravovat.
	Stránka, která obsahuje další možné akce, které se týkají zvoleného partnera. Je to například zobrazení pozice na mapě použitím *Google Maps*, zobrazení faktur vydaných, nedodaného zboží, dokumentů aj.

Další funkcionalitou tohoto modulu je možnost založení nového partnera, a to dvěma způsoby. Prvním způsobem je ruční zadání všech údajů, druhým způsobem je import dat z databáze *ARES* zadáním *IČO*. Po importu se do příslušných polí vyplní všechna data, která jsou v databázi *ARES* dostupná.

Aplikace také umožňuje při zakládání partnera automatické doplnění adresních údajů díky komunikaci s *Google Places API*. Tato funkce je dostupná pod ikonkou mapy v adresní sekci tvorby partnera.



Obrázek 34 - Tvorba partnera – import z ARES (zdroj: autor)



Obrázek 35 - Tvorba partnera – systémové číselníky (zdroj: autor)

6.2.3 Manažerský deník

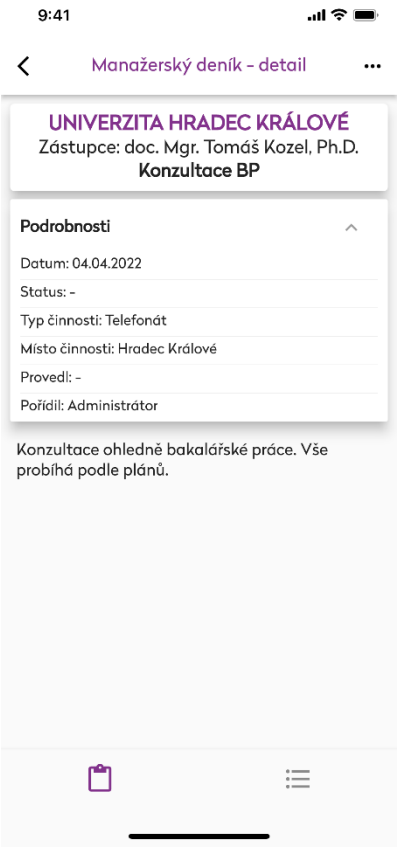
Manažerský deník je v systému *Signys* modul, který umožňuje tvořit deníky k danému partnerovi. Jedná se o zápisy z jednání nebo o evidenci komunikace mezi oběma stranami. Každá firma může modul využívat k trochu jiným účelům.

V aplikaci *Signys Mobile* nabízí tento modul náhled na seznam manažerských deníků (podle datumového rozmezí). Aplikace umožňuje nový manažerský deník vytvořit. V seznamu existujících záznamů je poté možné konkrétní manažerský deník upravit, odstranit nebo otevřít jeho detail.

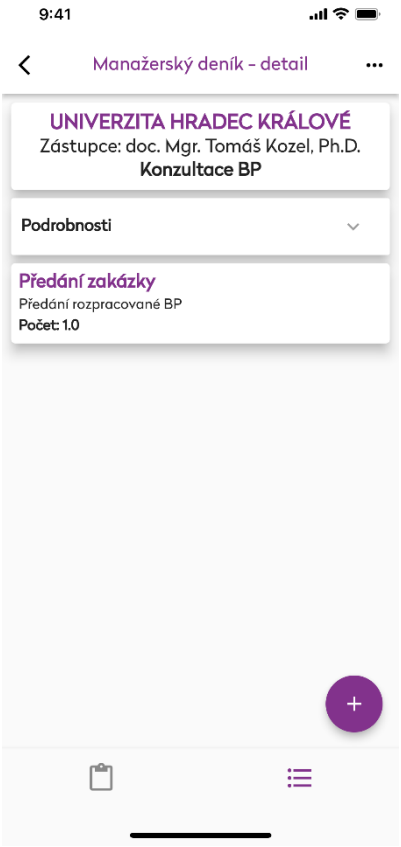
Všechny informace, které jsou v detailu zobrazeny, je možné upravit. V detailu je opět navigační lišta, díky které se dá přepnout na takzvané položky manažerského deníku. To, jaký má položka význam, určuje několik parametrů, které se nastavují v číselníku těchto položek. Celkem jsou tyto možnosti následující tři:

- Položka s pevnou vazbou na ceníkovou kartu. V tomto případě se z ceníkové karty převezme prodejní a nákupní cena a následně se k položce vyplní. To znamená, že k manažerskému deníku lze přidat odkaz na konkrétní ceníkovou kartu – aplikace v tomto případě umožňuje zobrazit její detail.
- Položka s příznakem, že uživatel musí ceníkovou kartu přiřadit. Jedná se o obdobnou záležitost jako v první případě s tím rozdílem, že uživatel může vždy zvolit jinou ceníkovou kartu. Jednoduchým příkladem může být položka s názvem „Zboží, materiál“. Uživatel po jejím zvolení vybere konkrétní ceníkovou kartu a ta se k záznamu připojí.
- Položka bez vazby na ceníkovou kartu. Položka má svou vlastní předem určenou cenu a nemá vazbu na ceníkovou kartu.

Ve všech případech položek manažerského deníku je možné vložené ceny upravit.



Obrázek 36 - Manažerský deník – detail (zdroj: autor)



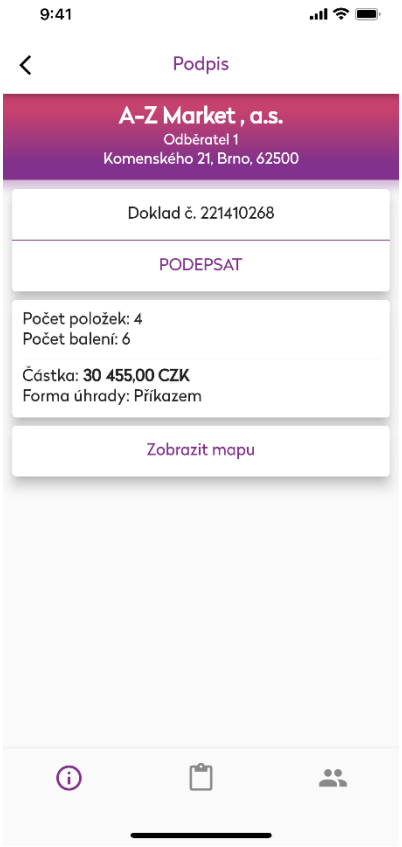
Obrázek 37 - Manažerský deník – položky (zdroj: autor)

6.2.4 Podpis dodacího listu

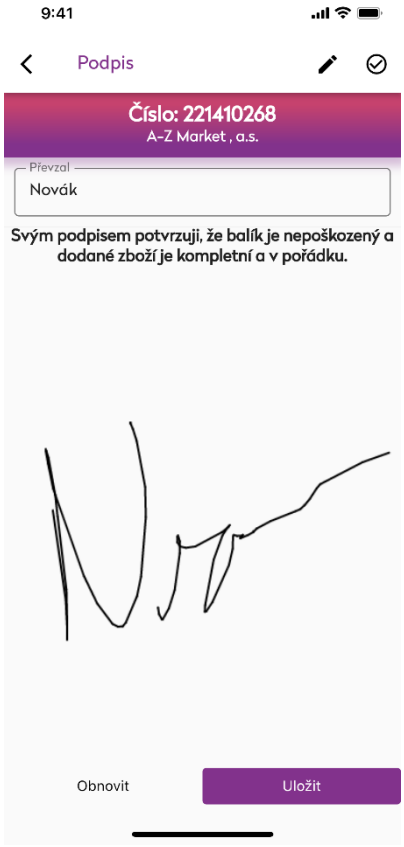
Modul podpis dodacího listu je určen pro firmy, které potřebují potvrzovat převzetí zboží. Konkrétně pro podepisování dodacích listů vydaných u zákazníka na místě doručení.

Modul umožňuje zadat konkrétní číslo dokladu nebo zobrazit seznam. Po otevření dokladu se zobrazí jeho detail a tlačítko pro podpis. V detailu dodacího listu je možné zobrazit položky, které se na dokladu nachází, a seznam všech dostupných kontaktů k danému zákazníkovi. Stiskem „Podepsat“ se otevře nová stránka, na které aplikace čeká na podpis od zákazníka, který přebírá zboží.

Jakmile se zákazník podepíše a řidič vepíše do textového pole jméno zákazníka, je možné podpis potvrdit. Při potvrzení umožňuje *API* server odeslání podepsaného dodacího listu na emailovou adresu zákazníka.



Obrázek 38 - Detail dodacího listu (zdroj: autor)



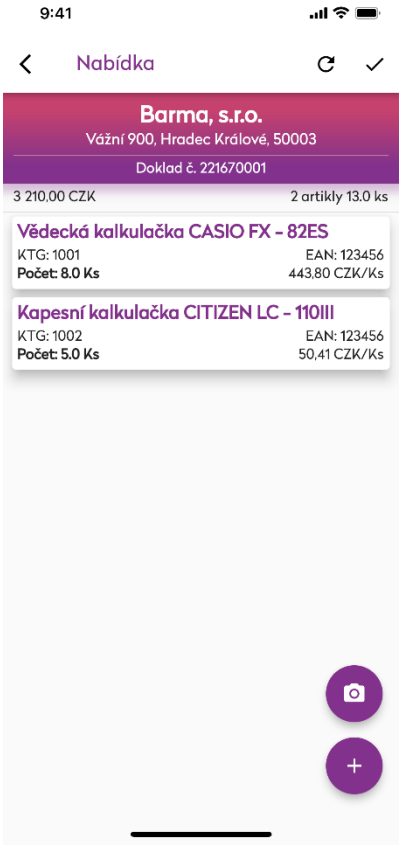
Obrázek 39 - Podpis dodacího listu (zdroj: autor)

6.2.5 Nabídky

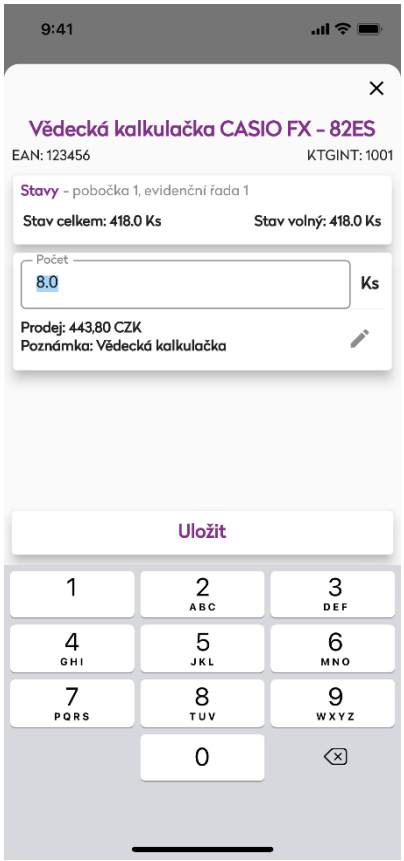
Modul nabídky je určen k pořizování a evidenci nabídek konkrétním zákazníkům – zejména pro práci obchodních zástupců se zákazníkem přímo u partnera (evidovaného nebo lze v modulu adresář založit) nebo například na výstavách, konferencích a podobných akcích.

Nabídka je v informačním systému *Signys* nezávazný obchodní doklad, na kterém se partnerovi uvedou položky, které požaduje, včetně nabízené prodejní ceny. Tento doklad může následně v informačním systému *Signys* sloužit jako podklad pro pořízení objednávky přijaté.

Položky nabídky se vybírají z evidence ceníkových karet – je možné naskenovat čárový kód pomocí fotoaparátu. Při tvorbě položky se automaticky přednastaví prodejní ceny na základě údajů na ceníkové kartě a nastavení partnera. Uživatel mobilní aplikace může tyto ceny upravit.



Obrázek 40 - Detail nabídky (zdroj: autor)



Obrázek 41 - Nabídka – úprava položky (zdroj: autor)

ZÁVĚR

Vývoj mobilního klienta pro *ERP* systém je skvělá zkušenost nejen z pohledu programování, ale i z pohledu práce s klienty. Každý klient se liší tím, pro jaké účely a jak *ERP* ve firmě využívá. Je proto výzvou vyhovět každému tak, aby byl spokojený, a zároveň si systém zachoval svoji logiku.

Jelikož *ERP* systémy obsahují obrovské množství dat, bylo a nadále je velkou výzvou zvládnout zobrazit ta nejdůležitější data, pokud možno co nejvíce srozumitelně a udržet aplikaci uživatelsky přívětivou.

Aplikace *Signys Mobile* se již v několika firmách využívá a úspěšně slouží svému účelu. Ohlasy jsou pozitivní a požadavky jsou řešeny co nejefektivněji a pokud možno v rozumném časovém intervalu. Vývoj aplikace jde hladce, nenastávají žádné omezující problémy ze strany zvolených technologií a v moment odevzdání této bakalářské práce rozhodně není u konce. Cílem je do aplikace implementovat další moduly *ERP* systému *Signys* a nadále tak rozšiřovat její funkčnost.

Framework *Flutter* je dle mého názoru jednoduchý na pochopení, poměrně přehledný, má k dispozici rozsáhlou dokumentaci a velké množství rozšiřujících balíčků, které nadále přibývají. V kombinaci s programovacím jazykem *Dart* je vývoj velice komfortní. Myslím si, že tyto technologie byly dobrou volbou pro vývoj mobilního klienta *ERP* systému a rozhodně je doporučuji.

POUŽITÉ ZDROJE

1. Computing Abbreviations. *AllAcronyms.* [Online] [Citace: 28. duben 2022.] https://www.allacronyms.com/computing/abbreviations.

2. Ares.cz - ekonomické subjekty. *Ares.* [Online] [Citace: 28. duben 2022.] https://ares.cz/.

3. Flutter documentation. *Flutter.* [Online] [Citace: 25. leden 2022.] https://docs.flutter.dev.

4. A tour of the Dart language. *Dart.* [Online] [Citace: 15. březen 2022.] https://dart.dev/guides/language/language-tour.

5. Flutter vs. React Native – What to Choose in 2022? *The Droids on Roids.* [Online] 10. květen 2021. [Citace: 13. duben 2022.] https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2021#f25.

6. Flutter or Ionic: What would you choose for your upcoming project? *Radix.* [Online] 3. leden 2022. [Citace: 13. duben 2022.] https://radixweb.com/blog/flutter-vs-ionic#final.

7. Flutter vs Native: What is Best for Your Project? *Surf.* [Online] [Citace: 13. duben 2022.] https://surf.dev/flutter-vs-native/.

8. http. *pub.dev.* [Online] dart.dev. [Citace: 7. duben 2022.] https://pub.dev/packages/http.

9. HTTP response status codes. *mdn web docs.* [Online] [Citace: 15. leden 2022.] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

10. Co je UX/UI design (webů a aplikací)? *UX/UI design.* [Online] [Citace: 15. březen 2022.] https://www.cojeuxui.cz/.

11. Jak používat službu Play Console. *Nápověda Play Console.* [Online] [Citace: 5. duben 2022.] https://support.google.com/googleplay/android-developer/answer/6112435?hl=cs#zippy=%2Ckrok-zaplacen%C3%AD-registra%C4%8Dn%C3%ADho-poplatku.

12. Google Play Console. [Online] Google. [Citace: 7. duben 2022.] https://play.google.com/console.

13. Choosing a Membership. *Apple Developer.* [Online] [Citace: 5. duben 2022.] https://developer.apple.com/support/compare-memberships/.

14. Apps. *App Store Connect.* [Online] Apple. [Citace: 7. duben 2022.] https://appstoreconnect.apple.com/apps.

15. Icons. *Google Fonts.* [Online] Google. [Citace: 7. duben 2022.] https://fonts.google.com/icons.

16. Git Tower. *Git Commands.* [Online] [Citace: 11. duben 2022.] https://www.git-tower.com/learn/git/commands/git-commit.

17. json_serializable. *pub.dev.* [Online] google.dev. [Citace: 7. duben 2022.] https://pub.dev/packages/json_serializable.

18. Jaký je rozdíl mezi WebView a PWA? *ThinkEasy.* [Online] [Citace: 27. duben 2022.] https://thinkeasy.cz/jaky-je-rozdil-mezi-webview-a-pwa/.

PŘÍLOHY

1. Zdrojový kód aplikace

Dostupný na adrese: https://github.com/tomaspodmanicky/SignysMobileBakalarskaPrace

Přístupové údaje na vyžádání poskytne autor.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Tomáš Podmanický**
Studium:	I1900717
Studijní program:	B0688A140001 Informační management
Studijní obor:	Informační management
Název bakalářské práce:	**Mobilní klient informačního systému**
Název bakalářské práce AJ:	Mobile Client of Information System

Cíl, metody, literatura, předpoklady:

Cíl práce:
Využití programovacího jazyku Dart a frameworku Flutter pro vývoj mobilního klienta ERP systému.

Osnova:

1. Úvod
2. Flutter a Dart, porovnání s jinými nástroji
3. Způsob práce s Flutterem, syntaxe a konkrétní příklady
4. Problematika napojení na ERP
5. Popis procesu vývoje
6. Výsledky
7. Závěr

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Tomáš Kozel, Ph.D.
Datum zadání závěrečné práce:	15.10.2021