CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Computer Science

RESTAURANT SYSTEM ANDROID CLIENT

BACHELOR THESIS

Study Program: **Software Technology and Management**

Branch of Study: **Software Engineering**

Thesis Advisor: **MEng. Martin Komárek**

Tomáš Hogenauer

Prague 2016

DECLARATION

I hereby declare that this bachelor thesis is the product of my own independent work and that I have clearly stated all information sources used in the thesis.

Date Signature

ABSTRACT

The goal of this thesis is to develop an Android application on tablet devices for waiters in restaurants. The application will provide waiters the ability to manage accounts and orders and perform activities related to it . It will have user-friendly GUI and offer fast way to complete user requirements. The application will communicate with a server over REST API.

Keywords: Android, application, restaurant, waiter, tablet

ABSTRAKT

Cílem bakalářské práce je vytvořit Android aplikaci na tablety pro číšníky v restauraci. Aplikace bude číšníkům umožňovat práci s zakaznickými účty, objednávkami a vykonávat aktivity s nimi spojené. Aplikace nabidne uživatelsky přívětivé grafické rozhraní a možnost rychlého splnění požadavků uživatele. Aplikace bude komunikovat se serverem přes REST API.

Klíčová slova: Android, aplikace, restaurace, číšník, tablet

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF USED TERMS

Account	A collection of information stored about restaurant customer and his activities in the restaurant (e.g. orders).
Customer	A person visiting the restaurant.
Order	A list of items ordered by customer.
Role	An attribute of users in the database. Roles helping us identify user who can use our application.
User	A person using the application.
Waiter	A person working in a restaurant. User with a role Waiter.
Quick account	Type of an account created without filling account name and selecting restaurant table. The name of the account is date and time when the account has been created.

LIST OF ACRONYMS

API	Application Program Interface
GUI	Graphical User Interface
OS	Operation System
SLOC	Source Line of Code
WBS	Work Breakdown Structure

1 INTRODUCTION

We are living in the era of information technologies. It has been changing our lives and making it easier. Since we have found the power of technology and embraced it as a helping tool, we are trying to make it more useful. Technology is not staying the same, it is changing and improving almost every day. We are preoccupied with the wish to make new technology better.

After the era of desktop PCs and laptops, new interest in mobile devices and tablets emerged. In 2014, the amount of mobile users reached the same level as desktop users (almost 1,700 million) and that number is still increasing [1]. According to the forecast of the statistic portal Statista, the number of shipped tablets will be higher in upcoming year than the number of shipped tablets and laptops together since 2015 [2]. Companies realized this new trend and started to focus on the mobile industry more than ever before. Nowadays, it is a rapidly growing industry that offers many opportunities. It is also a huge opportunity for software engineers on the one hand, on the other hand it also present big problem for them, because their products have to satisfy a huge amount of customers. There are many devices on a market with different software and hardware, making the job of the software engineers more complicated. They have to look for the best solution for how to satisfy the market in the most convenient way. One of these steps is to replace desktop applications by web applications.

An advantage of web applications is that they do not depend on the platform where they run. It also offers an easy way to update these applications for all users. Another advantage is that all computations are performed on a server, and hence they can run on devices with outdated hardware as well. According to an increasing number of mobile and tablet users, mobile applications are becoming more convenient. They mostly communicate with servers using shared database or solve more complex calculations. Nowadays, mobile and web applications are the most convenient application and it is necessary to go with this trend if we wish to satisfy the market.

1.1 RESTAURANT SYSTEM CASHBOB

CashBob is a solution for restaurants management to bring a new and easier way of managing restaurants and carrying out all activities related to it [3]. CashBob is a long-lived project focused on offering the best solution, because there are many similar solutions on the market.

Developing CashBob applications was the focus of many student projects and theses [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14]. The first CashBob application was developed as a desktop application with many modules. It was used by restaurant management (to manage employees, orders and restaurant storages) or by waiters (to manage accounts and orders, make payments and print receipts).

CashBob was later developed as a mobile application [14]. It worked on devices with Android [15] and iOS [16] operation systems. This application offered waiters the ability to manage restaurant accounts, order items, charge items and move items between accounts.

This year CashBob is being developed as a web application. It will replace the desktop version, because of all the advantages that web applications have to offer in comparison to desktop applications. It will also offer communication with a database for other CashBob applications over REST API interface.

1.2 ANDROID CLIENT CASHBOB

The goal of the thesis is to make CashBob as a mobile application for tablets (devices with screen 7” and more) with Android operation system. Target users are waiters in restaurants and the application will provide waiters the ability to manage accounts and orders and perform activities related to it.

CashBob for mobile devices has been developed once before [14]. It worked on devices with Android and iOS operation systems and offered same functionality as the application, which is an aim of this thesis. The problem of that version was, that it did not satisfy a submitter and it did not fit the expectations. Therefore, there opportunity to develop a new version.

My application should meet all requirements in the best possible way to satisfy the submitter and users. GUI will be logically designed and offer the ability to accomplish any task quickly and properly. It will be also offer intuitive for users. My version will be offered only for Android OS devices and focused exclusively to tablets. The application will communicate with a server over REST API. The result of this thesis is the application, which will be ready to be deployed in restaurants.

2 ANALYSIS

In this chapter we will focus more on our application. First, we will talk about functional and non-functional requirements of our application. Secondly, we will look to application use cases, because they will show interaction between user and application. Finally, we will analyze Android with aim to different Android versions and environment where we develop our application. Important activity for our project is a project planing. It will help us to make the structure of the project, so that we can see what is done and track our progress. Our project planning is attaches as appendix.

2.1 FUNCTIONAL REQUIREMENTS

Functional requirements show what the application should do [17]. Table 1 displays functional requirements of our application. These requirements are given by stakeholder at first meeting about thesis (main requirements to our application functionality are mentioned in the thesis assignment).

#	**Functional Requirement**	**Description**
1	Login	Only user with an account with role can log into the application.
2	Create an account	A user can create a customer account. This account can have a name and is attached to particular restaurant table.
3	Make an order	User can create a customer order.
4	Move items between accounts	From one account it is possible to move items to other existing accounts.
5	Make a payment	A user can select items from an account and make a payment.
6	Close an account	If a user makes a payment of all the items in the account, it will be automatically closed.
7	Print a receipts on a server printer	If a user makes the payment, application can print a receipt on a network printer.
8	Print a receipts on a mobile printer	If a user makes the payment, application can print a receipt on a network printer.
9	Set application language	User can use the application in Czech or in English language.

Table 1: Functional Requirements

2.2 NON-FUNCTIONAL REQUIREMENTS

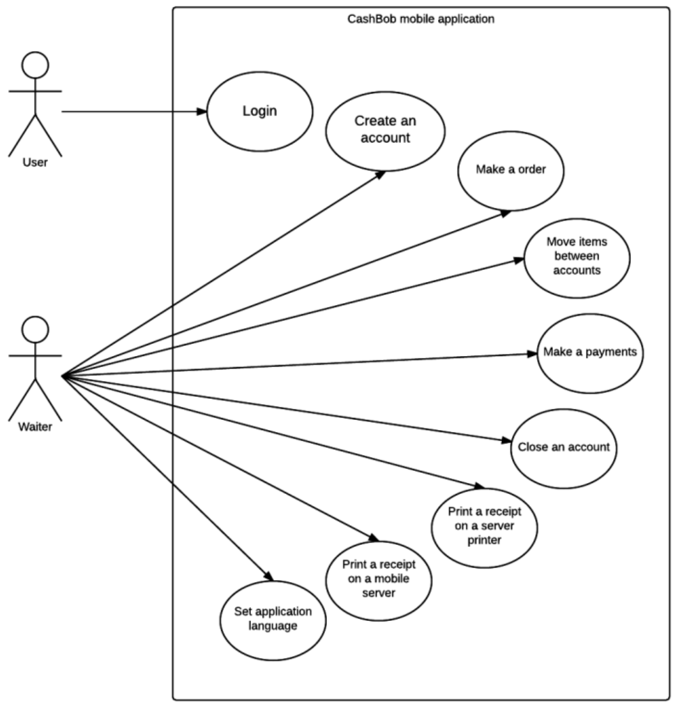
Non-functional requirements show application attributes [17]. Table 2 shows non-functional requirements of our project. Difference between functional and non-functional requirements is, that functional requirements show what an application should do, but non-functional requirements show how an application should work [18]. These requirements, like functional requirements, are also given by stakeholder at first project meeting (main requirements to our application functionality are mentioned in the thesis assignment).

#	**Non-Functional Requirement**	**Description**
1	Design	Application has similar GUI as CashBob desktop application.
2	Platform	Application is available for tablets (devices with screen 7” and more) with Android as an operation system.
3	Communication	Application communicates with a server using REST API

Table 2: Non-Functional Requirements

2.3 USE CASES

Use cases are collections of scenarios showing how the application is used by users [19]. Figure 1 shows us use cases offered to a user. Each use case describes interaction between user and application.



CashBob mobile application
User
Waiter
Login
Create an account
Make a order
Move items between accounts
Make a payments
Close an account
Print a receipt on a server printer
Print a receipt on a mobile server
Set application language

Figure 1: Use Case Model

Detailed use cases:

Use Case 1	Login
Actor	User
Basic Flow *Successful login*	1. User fills a login name, password, server IP (under button Settings) and clicks to Login button. 2. System validates server IP, then correctness of login name and password. System then checks in a database if login is assigned to a waiter. 3. System logins the user, because the validation was successful. 4. User is redirected to screen with the list of open accounts.
Alternative Flow 1 *Unsuccessful login*	3. System denies login into the system, because the validation has not been successful. System shows information that login has been unsuccessful.

Use Case 2	Create an account
Actor	Waiter
Basic Flow *Account with name and table*	1. Waiter selects tab New Account. 2. System loads New Account screen. 3. Waiter fills an account name, selects a restaurant table and confirms (button Create account). 4. System creates the account with given name and attaches the restaurant table in a database. System shows information that account has been created. System loads Order screen.
Alternative Flow 1 *Account without name and with a table*	3. Waiter selects only a restaurant table. 4. System creates the account with name and attaches a restaurant table in a database. The name of the account will be selected table. System shows information that account has been created. System loads Order screen.
Alternative Flow 2 *Account with a name and without table*	3. Waiter fills only a name. 4. System creates the account with given name in the database. System shows information that account has been created. System loads Order screen.
Alternative Flow 3 *Account without name and table*	3. Waiter does not fill name and does not select a restaurant table. 4. System creates the account with a name in a database. The name of the account is date and time when the account has been created. System shows information that account has been created. System loads Order screen.

Use Case 3	Make an order
Actor	Waiter
Precondition 1	An account is created
Basic Flow *Order from a menu card*	1. Waiter clicks to tab Accounts. 2. System loads Accounts screen. 3. Waiter chooses an account. 4. System loads Order screen. 5. Waiter selects items from a menu card, then confirms (button Order). 6. System creates new order, attaches it to the account and stores in a database. System shows information that order has been created.
Alternative Flow 1 *Order from previous account order*	5. Waiter selects items from a previous order for this account, then confirm (button Order). 6. System creates new order, attach it to the account and store in a database. System shows information that order was created.

Use Case 4	Move items between accounts
Actor	Waiter
Precondition 1	Original account is created
Precondition 2	Original account has at least one ordered item
Basic Flow *Move to existing account*	1. Waiter clicks to tab Accounts 2. System loads Accounts screen. 3. Waiter chooses an account. 4. System loads Order screen. 5. Waiter clicks to tab Transfer. 6. System loads Transfer screen. 7. Waiter selects items to move. 8. Waiter selects target account and confirms (button Move). 9. System moves selected items from original account to target account in a database. System shows information that items have been moved.
Alternative Flow 1 *Move to new account*	8. Waiter clicks to "Create New Account". 9. System displays dialog window. 10. Waiter fills new account name and confirms (button Create Account). 11. System creates new account with selected name in a database. 12. Waiter selects target account and confirms (button Move). 13. System moves selected items from the original account to the target account in a database. System shows information that items have been moved.
Alternative Flow 2 *Move all items*	7. Waiter selects target account and confirms (button Move All). 8. System moves all items from the original account to the target account in a database. System shows information that items have been moved.

Use Case 5	Make a payment
Actor	Waiter
Precondition 1	An account is created
Precondition 2	An account has at least one ordered item
Basic Flow *Pay for selected items*	1. Waiter clicks to tab Accounts. 2. System loads Accounts screen. 3. Waiter chooses an account. 4. System loads Order screen. 5. Waiter clicks to tab Pay. 6. System loads Payment screen. 7. Waiter selects items to pay for and confirms. 8. System shows dialog with selected items and price. 9. Waiter confirms. 10. System marks selected items as paid in the database. System shows information that payment has been successful. Paid items are not displayed in account item list.
Alternative Flow 1 *Pay for all items*	7. Waiter clicks to button Pay All and confirms. 8. System shows dialog with selected items and price. 9. Waiter confirms. 10. System marks all items as paid in a database and account as closed. System shows information that payment has been successful. Account is not showed in an account list.

Use Case 6	Close an account
Actor	Waiter
Precondition 1	An account is created
Precondition 2	An account has at least one ordered item
Basic Flow *Close an account*	1. Waiter clicks to tab Accounts. 2. System loads Accounts screen. 3. Waiter chooses an account. 4. System loads Order screen. 5. Waiter clicks to tab Pay. 6. System loads Payment screen. 7. Waiter clicks to button Pay All. 8. System shows dialog with selected items and price. 9. Waiter confirms. 10. System marks all items as paid in a database and account as closed. System shows information that payment has been successful. Account is not showed in an account list.

Use Case 7	Print a receipts on a server printer
Actor	Waiter
Precondition 1	An account is created
Precondition 2	An account has at least one ordered item
Basic Flow *Print on a server printer*	1. Waiter clicks to tab Accounts. 2. System loads Accounts screen. 3. Waiter chooses an account. 4. System loads Order screen. 5. Waiter clicks to tab Pay. 6. System loads Payment screen. 7. Waiter selects items to pay for and confirms (button Pay). 8. System shows dialog with selected items and price. 9. Waiter selects print a receipt on a server printer and confirms (button Pay). 10. System marks selected items as paid in a database. System shows information that payment has been successful. Paid items are not showed in account item list. System prints receipt on a server printer.

Use Case 8	Print a receipts on a mobile printer
Actor	Waiter
Precondition 2	An account is created
Precondition 3	An account has at least one ordered item
Basic Flow *Print on a mobile printer*	1. Waiter clicks to tab Accounts. 2. System loads Accounts screen. 3. Waiter chooses an account. 4. System loads Order screen. 5. Waiter clicks to tab Pay. 6. System loads Payment screen. 7. Waiter selects items to pay for and confirms (button Pay). 8. System shows dialog with selected items and price. 9. Waiter selects print a receipt on a mobile printer and confirms (button Pay). 10. System marks selected items as paid in a database. System shows information that payment has been successful. Paid items are not showed in account item list. System prints receipt on a mobile printer.

Use Case 9	Set application language
Actor	Waiter
Basic Flow *Set Czech language*	1. Waiter sets Czech language in android setting. 2. Application is available in Czech language.
Alternative Flow 1 *Set English Language*	1. Waiter sets English language in android setting. 2. Application is available in English language.

2.4 ANDROID

In this chapter we will look at Android versions and deployment environment. Every new Android version brings new features, so we have to find oldest Android version which satisfy our needs. That will make our application available to bigger amount of devices. Then, we will cover development environment.

2.4.1 ANDROID VERSIONS

The initial release of first Android operating system was in 2008 and since then many updated versions have been released. Each version has brought new features and new possibilities for developers. Android developers have to be careful about considering which version of Android their application will work with.

Our purpose is to develop an application that will work on the largest range of devices. According to the amount of devices running particular version of Android (table 3 and figure 2), we want to develop an application for version 4.1.x (Jelly Bean) at least, because Jelly Bean is still used by 29% of devices [20]. However, one requirement is to be able to print receipts on a printer and this feature is available since version 4.4 (KitKat) [21], hence our application can be available for only 63.7% of current devices.

Codename	Version	API	Distribution
Froyo	2.2	8	0.2%
Gingerbread	2.3.3 -2.3.7	10	3.8%
Ice Cream Sandwich	4.0.3 -4.0.4	15	3.3%
Jelly Bean	4.1.x	16	11.0%
	4.2.x	17	13.9%
	4.3	18	4.1%
KitKat	4.4	19	37.8%
Lollipop	5	21	15.5%
	5.1	22	10.1%
Marshmallow	6	23	0.3%

Table 3: Number of devices running a given Android version

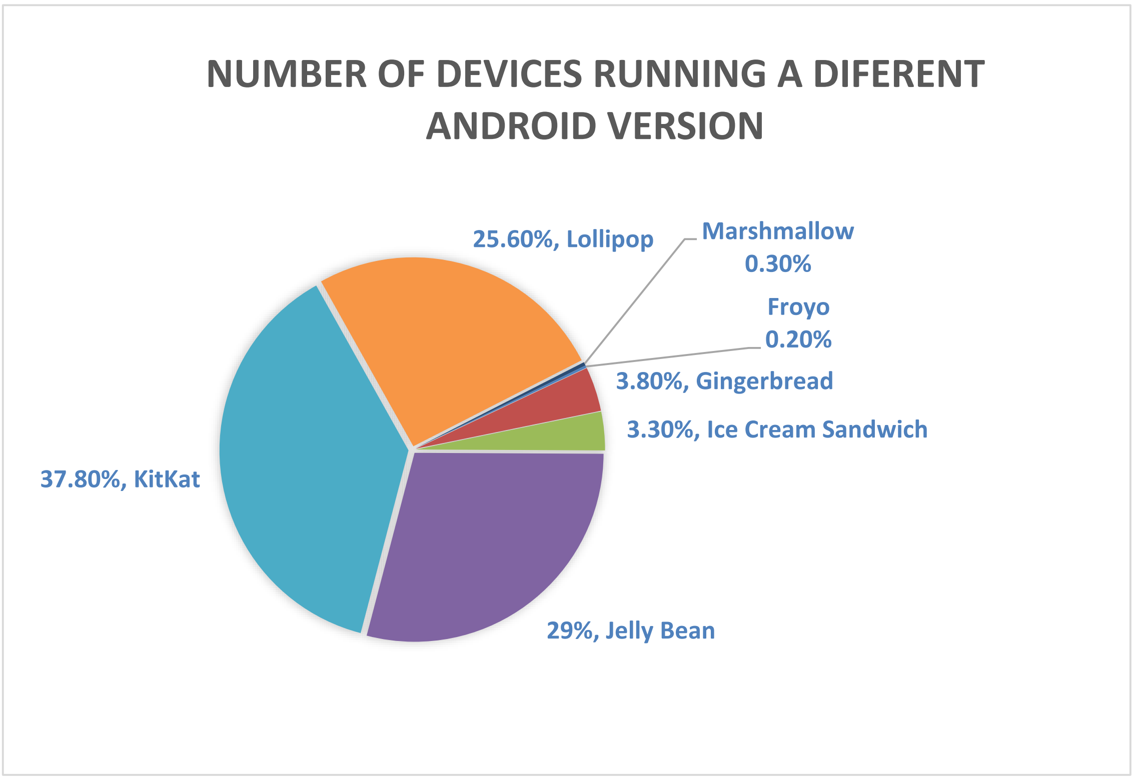


Figure 2: Number of devices running a different Android version

2.4.2 ANDROID STUDIO

For our project, we will use Android Studio as the official IDE for Android Application development [22]. Android studio offers a good environment with logical structuring. GUI can be written in form of XML file. The application during development will be running in Android emulator or tested on a device with Android OS connected to our computer. Android Studio offers many features and makes the development of applications more intuitive.

3 DESIGN AND IMPLEMENTATION

At this chapter we focus at design and implementation of our application. First of all, we focus how to design our application. Secondly, we show a configuration, which we have to do to communicate with a server. Next, we describe REST API used by our application. Then, we will write about, layouts and localizations. Finally, we will show how class which get list of accounts from a server is implemented.

3.1 USER INTERFACE DESIGN

Important part of our application is the GUI design. It focuses on interaction between the user and the system. If design is constructed poorly, it will have a negative effect to a user. User will not understand our application it will decrease his involvement in application experience. Our goal is to satisfy user needs and offer perfectly designed application.

From non-functional requirements we can see that we need to make GUI of our application similar to desktop version. This requirement gives a user opportunity to easily work with our application if he has previous experiences with the desktop version. Waiters can use both applications in a restaurant, so different GUI can confuse them and extend time to finish of any given task.

3.1.1 APPLICATION MENU

One of most important part of GUI is a menu (figure 3). It offers switching between tabs and makes application more intuitive.

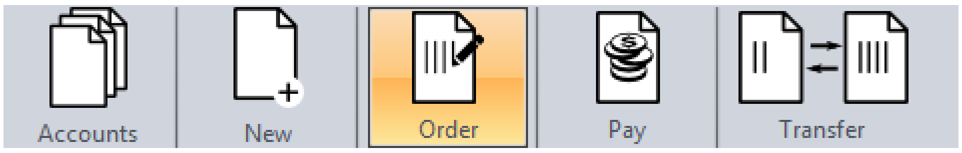


Figure 3: Menu from Desktop Version

Menu tabs are:

- **Accounts**: Show list of accounts
- **New**: Create new account
- **Order**: Make an order for current account
- **Pay**: Make payment
- **Transfer**: Transfer items from one account to another

The menu will be displayed in every screen after user login.

3.1.2 LOGIN SCREEN

Only waiters in a restaurant will use our application. For login to our application, we need:

- Login name of waiter: It is necessary to distinguish who is using the application, especially if the account is marked as account of waiter.
- Password: Access to our system has to be secure.
- Server IP: CashBob database has to run on a server, so it is necessary to identify the server by IP.

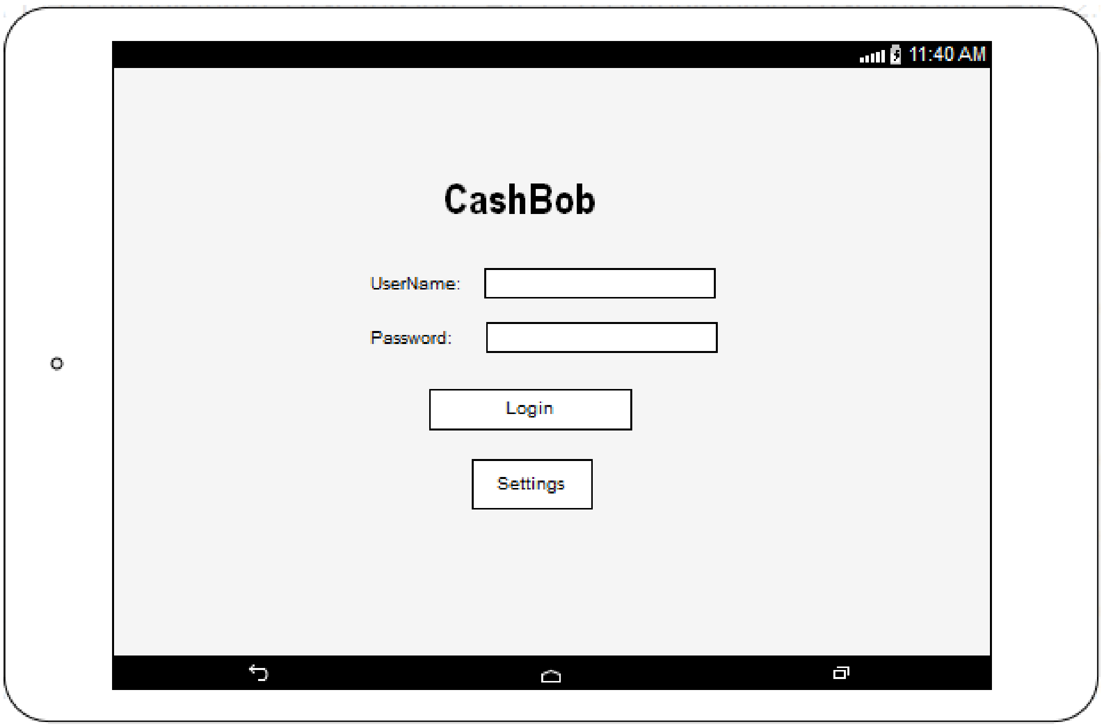


Figure 4: Login Screen

On this screen user could fill in their user name and password (figure 4). To fill an IP address of server or choose a printing server he has to click the “Setting” button.

3.1.3 ACCOUNTS SCREEN

First tab shows a list of all accounts (figure 5). This tab is also the first window that is displayed after login. User can choose an account by clicking on it.

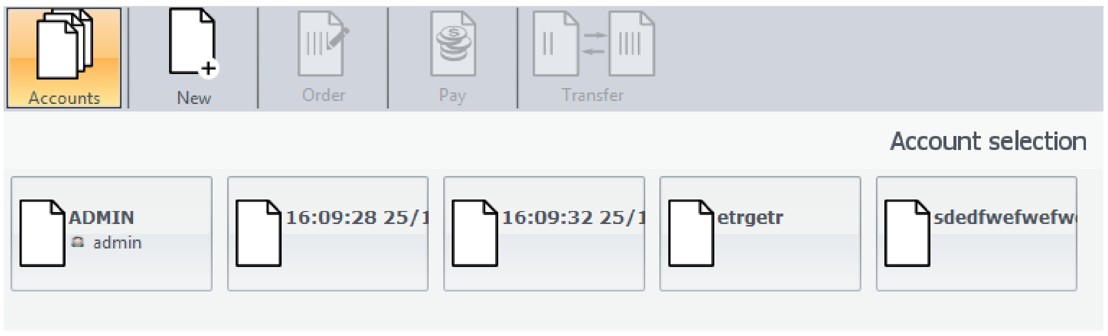


Figure 5: Desktop Accounts Screen

We will display accounts in form of a scroll list, where you can click on the account you want to work with (figure 6). After choosing an account, activities “Order”, “Pay”, and “Transfer Items” will be available to work with.

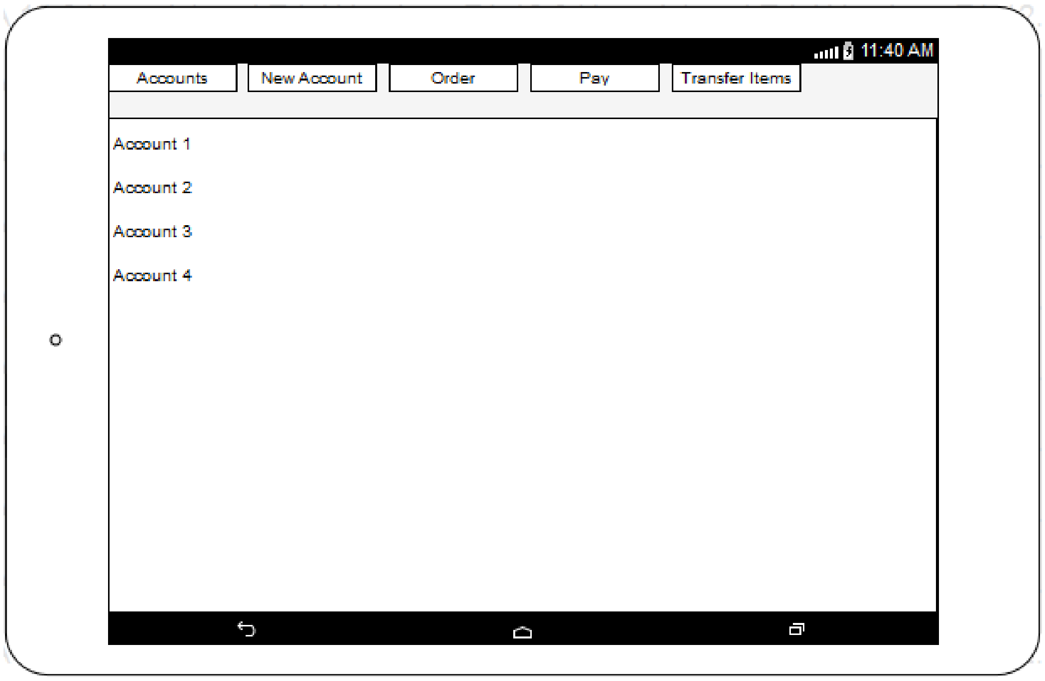


Figure 6: Accounts Screen

3.1.4 NEW ACCOUNT SCREEN

At this screen user can create a new account. Figure 7 displays a desktop version of the tab. User can fill a name of the account, select table and add a note. Figure 8 shows our design of the new account tab.

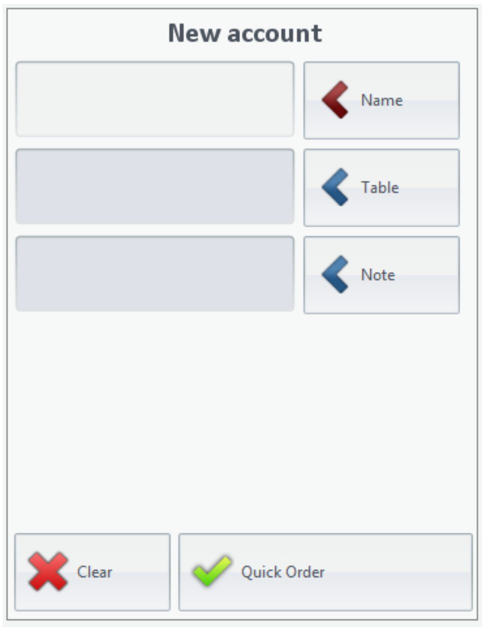


Figure 7: Desktop New Account

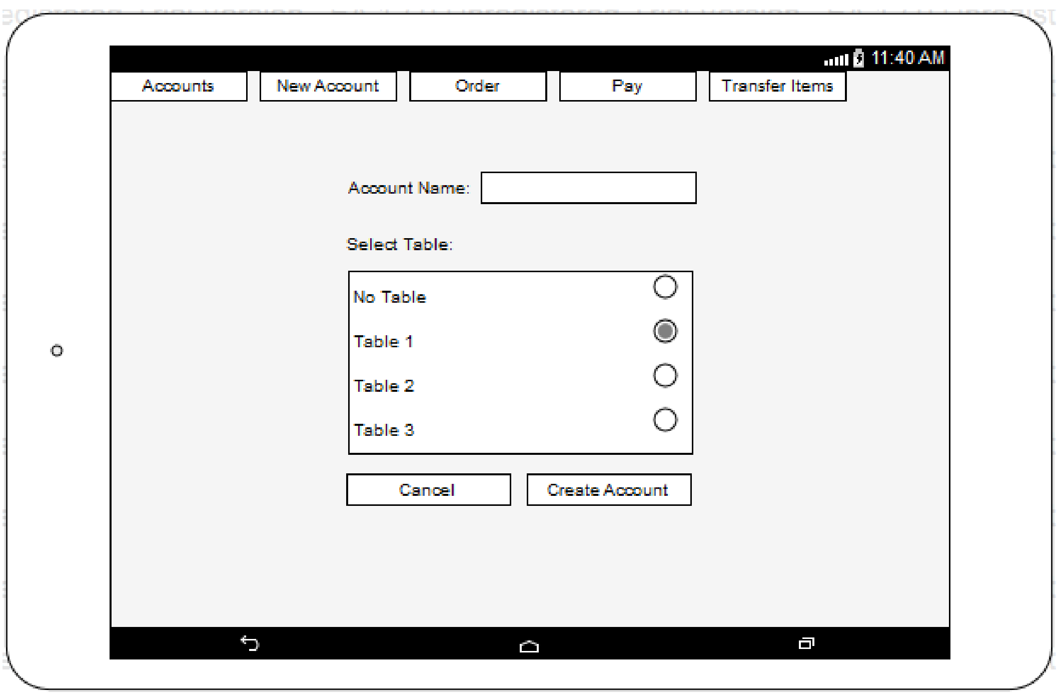


Figure 8: New Account Screen

3.1.5 ORDER SCREEN

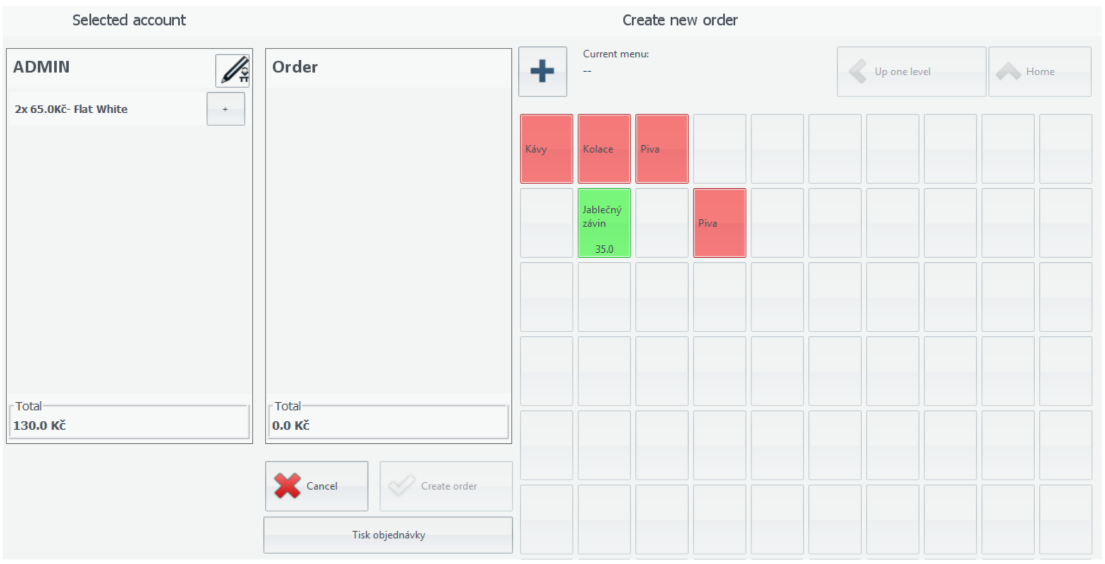


Figure 9: Desktop Order Screen

Desktop version is divided into three columns (figure 9). First column shows a list of items in currently selected account. Second column is a list where a current order is shown (unfinished, to finish it user has to click to “Create Order” button). Third column is a menu card.

In our application, the first column will be designed as selectable list. If one item is selected, the item will be added to an order list. Under the list will be displayed information with the sum of all prices of account items.

The second column (order list) will be also designed as selectable list, but after selecting an item, the item will be removed from that list. Under the order list will be also information about price and three buttons: “Cancel”, “Create Order” and “Print Order”. We decided to remove the last button “Print Order”. We think that printing a receipt should be available only after making payment. If a user wants to print the receipt independently, he should create a quick account, add items to it and make the payment.. This button was connected to currently selected account, which would appear confusing also for the user. The application will offer the option of printing the receipt only under the tab “Pay”.

The third column is a menu card. In the desktop version it is designed as a button area. However, we are trying to avoid this, because we think it would look complicated. The design of the structure of buttons would make it unreadable on a device. In our design it is shown as an expandable list and it will look as a paper menu card offered to restaurant customers. It is possible that this part could be changed in our development process. Figure 10 displays our design of the order tab.

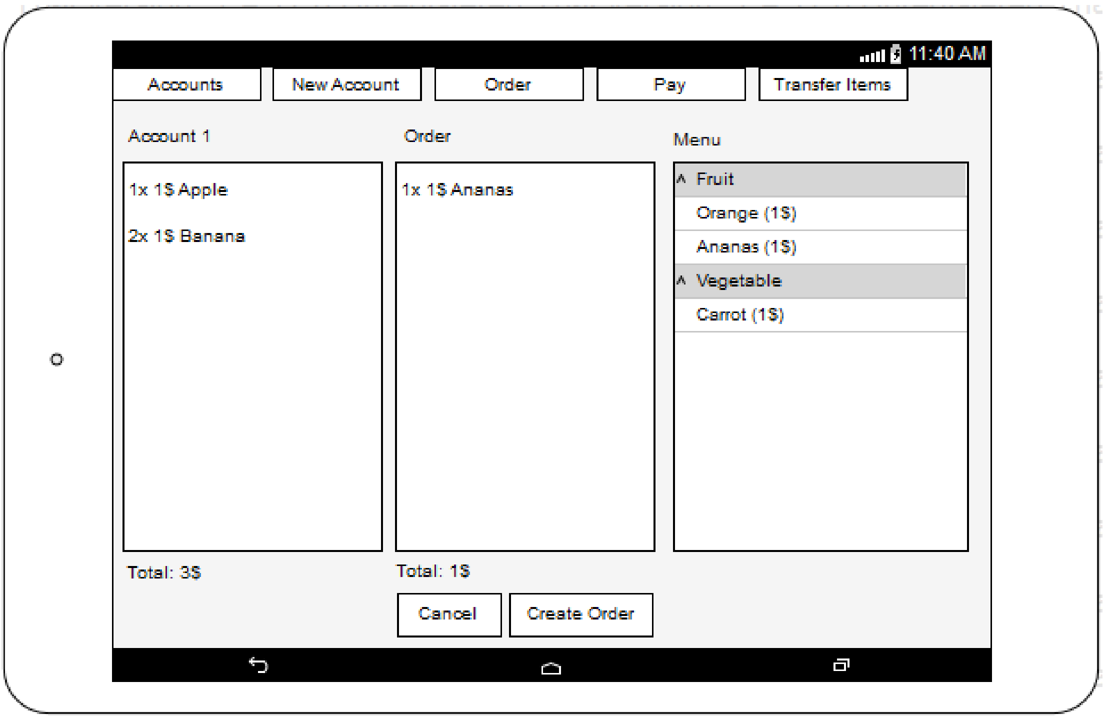


Figure 10: Order Screen

3.1.6 PAYMENT SCREEN

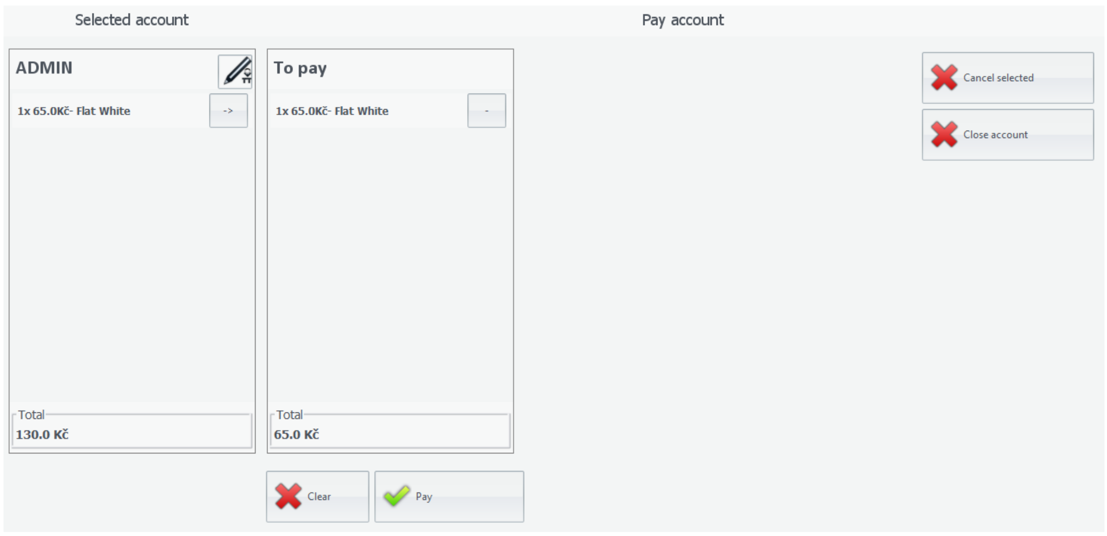


Figure 11: Desktop Payment Screen

Figure 11 displays payment screen of desktop version. Screen is divided to three parts:

- List of items from currently selected account (click on item to move item to pay list)
- List of items to pay + confirm button + cancel button
- "Cancel Selected" button (delete item from account item list) + "Cancel Account" button (delete account)

Figure 12 displays design of the payment screen for our application.

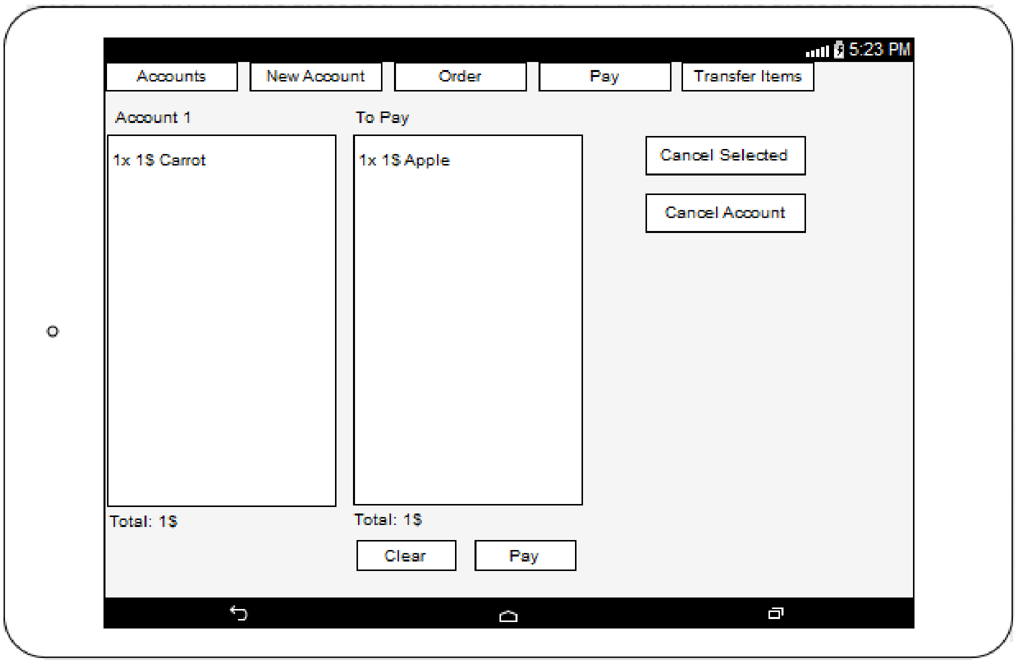


Figure 12: Payment Screen

3.1.7 TRANSFER SCREEN

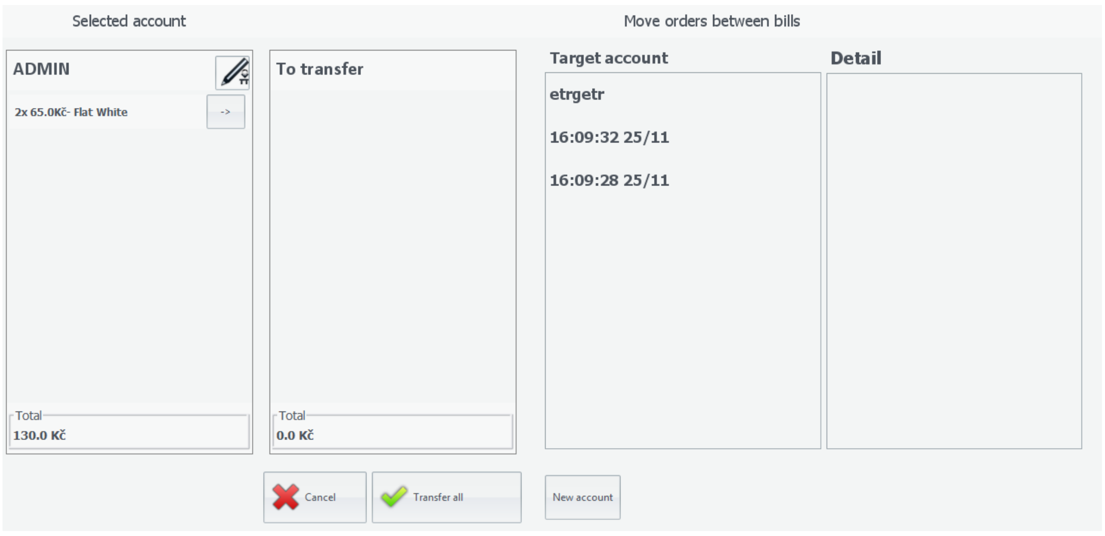


Figure 13: Desktop Transfer Screen

Figure 13 displays desktop version divided to three parts:

- List of items from currently selected account (click on item to move item to transfer list)
- List of items to transfer + confirm button + cancel button
- List of target account where items will be transferred with detail (account item list) + button to create quick account

The problem is that the last column has two lists, which means we will have four lists next to each other. Tablet devices are devices with screen from 7" and that means there is possibility it will make the screen not readable (too low width of each list to see content properly). To solve this problem, we can:

- Remove list of accounts and exchange it to spinner (only selected name of account will be shown and by clicking on it, the list of accounts will appear). In that case, it will take longer to find target account if there are too many open accounts.
- Do not show details of target account. In that case, a user will not see which items are on target account.

With stakeholder we decided to choose second solution (figure 14).

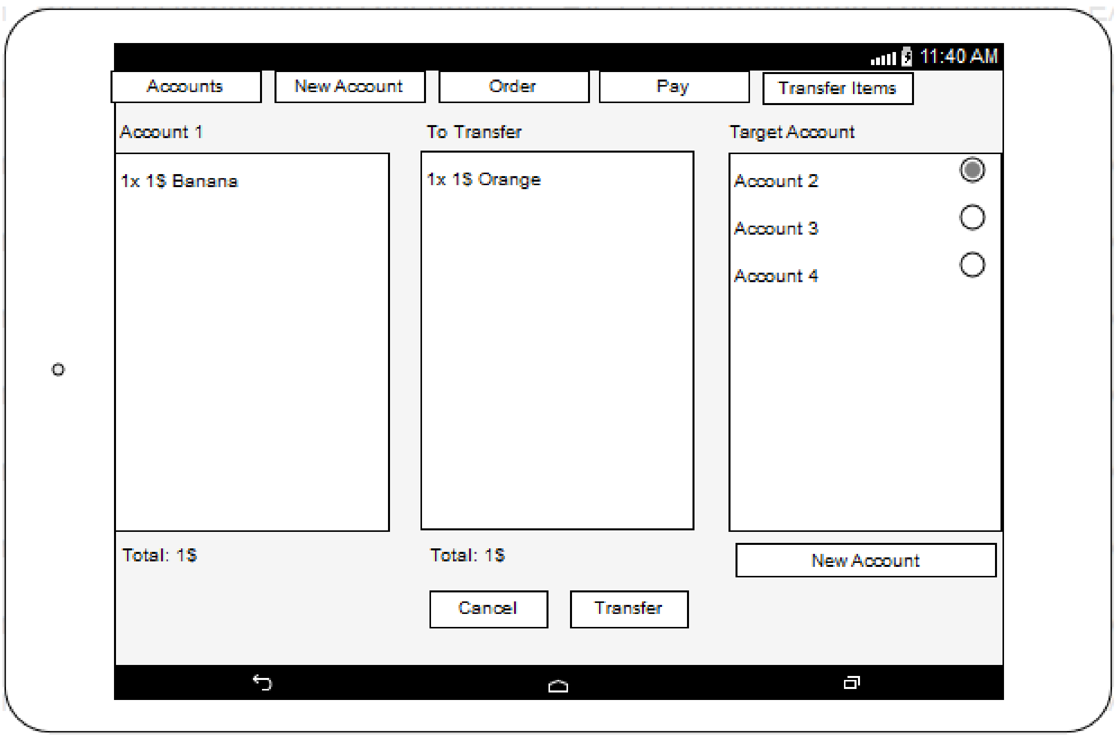


Figure 14: Transfer Screen

3.1.8 DISCUSSION ABOUT DESIGN WITH STAKEHOLDER

Over the time, when we were designing the application we had a few meeting with a stakeholder about our application. These discussions bring new changes to our application, mostly to GUI.

3.1.8.1 PREDESIGN MEETING

- We suggested deleting the attribute “note” in entity “account”, because after creating an account “note” is not shown anywhere, so it is useless. The stakeholder agreed.

3.1.8.2 FIRST DESIGN MEETING

- Login Screen: We placed a button for settings before Login button. Stakeholder wanted the button for Login before the button for Settings and makes the Settings button smaller.
- Tab Accounts: We designed one list of accounts over the whole screen, but stakeholder suggested to create more lists accounts in that screen to make accounts more accessible to click. This stayed as open point to a discussion.
- Tab Order: Stakeholder was not sure about expandable list for a menu card - it would not be accessible as a menu card in desktop version. We agreed to look for a better solution.
- Stakeholder was interested in adding an icon to each tab to make it more understandable. We agreed to add it if it will be possible.
- Tab Order: After discussion with stakeholder about two possibilities of showing target account in this screen, stakeholder agreed that we would show only the list of the account, not the items.
- Stakeholder agreed with removing “print receipt” button.

3.1.8.3 SECOND DESIGN MEETING

- Tab New Account: Stakeholder suggested that if user chooses a table for an account and an account name is not yet created, it would be automatically named according to the table name.
- Tab Pay: Stakeholder decided to remove buttons “Delete Account” and “Delete items”. This feature will be available only in web application. The account will be deleted automatically if an account item list is empty after making a payment.
- Tab Order: We presented a new design for the menu card. Stakeholder agreed with the solution. Figure 15 shows new design.

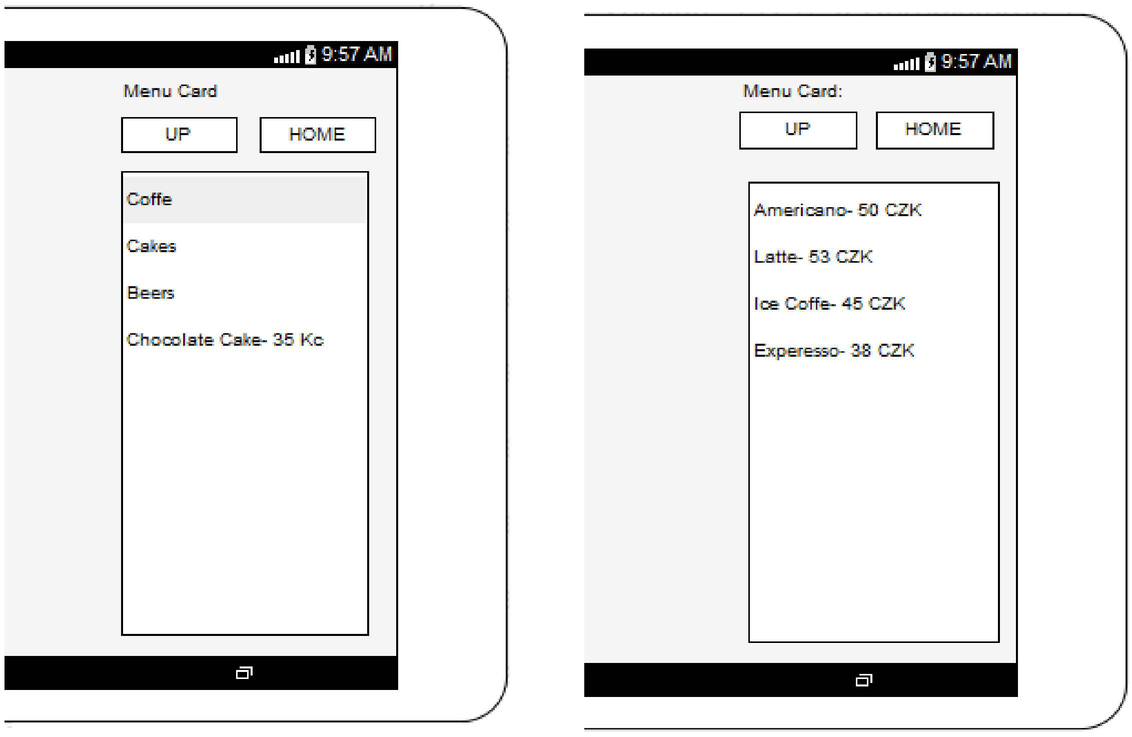


Figure 15: New Menu Card

The menu card contains two buttons and simple list. If the user clicks on the menu item and its category, the category items will show in this list. By clicking on a product, the item will be added to an order list. Button "Up" will show previous category and button "Home" shows first level list of menu card.

- Stakeholder decided to remove the ability to print receipts from a mobile application list of requirements. It will be developed only if developer has enough time to implement it.

3.1.8.4 THIRD DESIGN MEETING

- Tab Transfer: Button to create a new account was implemented to create quick account. Stakeholder would like to create an account with a given name.
- Stakeholder wanted all prices with currencies be showed with a blank space between them.
- Tab New Account: Stakeholder would like buttons to be moved to the right side of the field for a new account name.

3.2 CONFIGURATION

At this chapter we will show how we configure the device, where our server is installed, to be able to communicate with it. First, we decided to test our application against a server provided by a student developing CashBob as a web application. Reason was, that we wanted test our application against the server that will be used in the future, but many mistakes appeared during that time on the server side:

- Server did not reply to a query (for example method GET for http://192.168.1.100:9000/rest/table)
- We cannot connect to the server from another device (from device where there is not a server)
- Method POST did not work
- Get a menu card did not work
- Proper documentation is not available (problem in communication with the server)
- Server does not show ordered items of an account
- Tree structure of menu card was implemented with mistakes (wrong pointers between items)

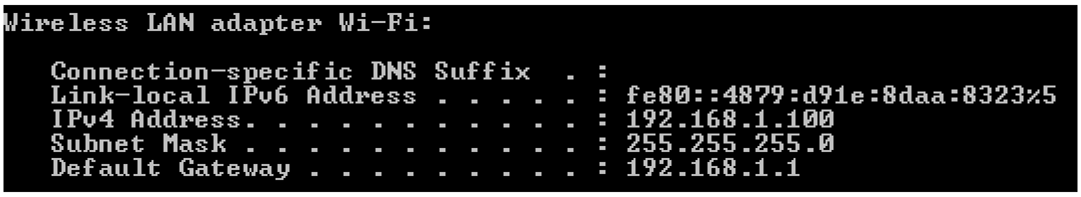
After all these problems with the new server REST API we decided change the server and test against old server provided by CashBob desktop version. The desktop version was downloaded from website of CashBob [3]. Desktop version works only with older version of Java, so we decided run it in in our virtual machine using VirtualBox [23].

For virtual machine we had to setup two network adapters in settings:

- NAT (Adapter Type: Intel PRO/1000 MT Desktop 82540EM, check Cable Connected)
- Bridged Adapter (Name: Intel(R) WiFi Link 5100 AGN, Adapter Type: Intel PRO/1000 MT Desktop 82540EM, Promiscuous Mode: Allow All, check Cable Connected)

Settings may be different in other machines. With this configuration any client application will be available connect to the server, if it is connected to same network like server (desktop application).

To connect to our computer server (localhost: 9000) we need to get the IP address of our PC. In Windows, we can get this information over Command Prompt by command "ipconfig"



```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::4879:d91e:8daa:8323%5
   IPv4 Address. . . . . . . . . . . : 192.168.1.100
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

Figure 16: Command Prompt

From figure 16 we can see that our IP address is 192.168.1.100, so connection to our server is over link http://192.168.1.100:9000

So, for example, if we want to communicate with a server and we want a list of tables, we will use link http://192.168.1.100:9000/rest/table

3.3 REST API INTERFACE

At this chapter we will show which commands we are using in our application for communication with a server. Server provided by desktop application offers larger amount of commands, but not all of them are necessary for our purposes. All commands are written in CashBob documentation [24].

Our application communicates with the server over REST API by methods GET, POST and PUT. Method GET is to retrieve a resource from server, POST is to create resource on server and PUT is for update a resource on the server. Server is responding to our methods by sending a resource or by response code. Table 4 displays all queries used by our application.

URL path	Method	What we used
user/{name}	GET	roles: [{name}]
account	GET	accs: [{opened, id, name}]
table	GET	tables: [{id, tablenumber}]
account	POST	name,table: {id}
account/{id}	GET	order: [{ispaid, iscanceled, menuItem: [{menuitemid, name, price}]
account/{id}	PUT	id, name, opened
cell	GET	cells: [{parentMenu: {menuId, name}, itemId, name, price, isMenu}]
account/{id}/order	POST	items: [{manuItemId, count}]
account/{id}/payItems	PUT	items: [{manuItemId, count}]
account/{id}/moveItems	PUT	targetAccId, items: [{manuItemId, count}]

Table 4: REST query table

3.4 LAYOUTS AND LOCALIZATIONS

Important part of our application is graphic of user interface. For that we use layouts. In layouts we can define a structure of every part of our application. For example, figure 17 displays structure of GUI for warning dialog which shows if user wants to make an order, make a payment or transfer items between account before he selects account he want to work with. Linear layout with vertical orientation defines that mentioned objects will be sorted vertically. These objects are TextView and Button. Textview displays a text and Button adds an action for the dialog.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:text="@string/textView_no_account_selected" />

    <Button
        android:id="@+id/dialogButtonOk"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/OK" />
</LinearLayout>
```

Figure 17: Layout for warning dialog

In figure 17 you can also see that text content of each object is written as reference. This definition helps us make application available in more languages. For every implemented language is created a file string.xml (Czech and English in our case). According to a language setting of Android environment, the application will decide which file will be used. The file with English sources is set as default, so if selected language is not implemented, the application will use default file with English language. Figure 18 shows how text sources for warning dialog are implemented in file for English language and figure 19 for Czech language.

```
<resources>
    <string name="textView_no_account_selected">No Account Selected</string>
    <string name="OK">OK</string>
</resources>
```

Figure 18: strings.xml (English)

```
<resources>
    <string name="textView_no_account_selected">Nebyl vybrán účet</string>
    <string name="OK">OK</string>
</resources>
```

Figure 19: strings.xml (Czech)

3.5 LIST OF ACCOUNTS

Most important part of our application is communication with the server using Rest API. One of the examples is receiving a list of accounts. For that we were using class RestApiGetAccount, which is displayed in the figure 20. The class extends class AsyncTask [25].

```
class RestApiGetAccounts extends AsyncTask<Void, Void, Void> {

        @Override
        protected void onPreExecute() {
            //code
        }

        @Override
        protected Void doInBackground(Void... param) {
        //code
        }

        @Override
        protected void onPostExecute(Void aVoid) {
        //code
        }

        protected void outputParser (JSONArray jsonArray) {
        \\code
        }
}
```

Figure 20: Class for getting list of accounts

Method doInBackground overrides a method to make a computation on a background thread. Method onPreExecute runs before doInBackground and onPostExecute runs after doInBackground on the UI. In our case, method onPreExecute is used for disable user interaction with application, method doInBackground communicate with a server (send a request and receive a response) and onPostExecute works with received data. Method outputParser is called by method onPostExecute. In the figure 21 is shown that this method is used for parse received JSON data and stores them.

```java
protected void outputParser(JSONArray jsonArray) {

    try {
        for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject = jsonArray.getJSONObject(i);
            if (jsonObject.getBoolean("opened"))
                accounts.add(new Account(jsonObject.getInt("id"),
                 jsonObject.getString("name")));
        }
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Figure 21: Method outputParser

Important part is also to implement a layout to display these accounts. Figure 22 displays xml file for a tab account, which shows all accounts.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="5dp">

    <ListView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/listView_accounts"
        android:padding="5dp" />

</LinearLayout>
```

Figure 22: Account list layout

4 TESTING

In this chapter we will focus on testing our application. First, we will write about unit tests, which are testing smallest parts of our code to make sure they behave as expected. Then, we will write about usability tests, which often reveal problems in our application by monitoring interaction between user and application. Finally, we will do an acceptance testing, which show to a stakeholder that all his requirements were implemented.

4.1 UNIT TESTING

It is necessary to test our application to ensure that everything is implemented properly. We created tests to check the proper functionality of implemented methods. We ran the tests in the application Android Studio [22], which we also used as an environment to develop our application.

To be sure that all our implemented features work properly, during the time of development we tested each new feature in all possible ways that user can use it. If we found any failure, we immediately solved it before developing any new feature. This testing took a lot of time, but gave us valuable information about developed features.

4.2 USABILITY TEST

To achieve better quality of our application, it is also necessary to do a usability test. Usability test is performed with users, who get a set of tasks and have to accomplish them. User activity and interactions are monitored and offer the ability to see application errors, mostly in GUI scope. This testing gives us important feedback from users and allows us to see problems which we could not discovers as developers with too narrow approach.

According to many recommendations, usability testing does not require more than 5 users, because the main problems will be discovered by the first or second user testing of the application. The others discover usually the same problems as the previous testers and do not discover a new ones [26].

Before the test we set the testing environment in application database to:

- Create 5 accounts
- Create 5 tables
- Create menu card with 14 items in 4 submenus

Anyone can use our application, so we were not looking users with any specific characteristic (age, sex or android mobile experiences). We had five users for our usability test: four Android mobile phone user and one iOS mobile user. The tested person was introduced to the idea of CashBob and received a tablet (10.1" tablet, Android version 4.1.1) with installed CashBob Android application and instructions to accomplish given tasks. Before the test, we told them about activities done by a waiter in a restaurant and about our application without showing them the application.

4.2.1 TESTER TASKS

1. Log in with a given user name and password.
2. Customer Tomas came into a restaurant and sat at table two. He wants 2x beer Plzen 12 and 3x red wine.
3. From table one people are divided. Move all beers to account called Petr and cakes to new account Martin.
4. Tomas wants three more beers Plzen 12 and one Gambrinus 10.
5. Michal wants make a payment for two apple pies.
6. Create account for table 5.
7. Tomas wants to pay for all Plzen 12 and one wine.

4.2.2 RESULT

According to our testing we found few faultiness:

Orientation in menu card

- User had problem get up one level in menu card
- Occurrence: high
- Source: User does not connect button "Up one level" and "Home" with menu card or Czech text on buttons does not make sense to a user.
- Solution: Not solved. It is on a discussion with stakeholder, because it will have effect to other CashBob applications.

Looking for create account under tab account

- User wanted create new account and looked for that in tab "Accounts"
- Occurrence: high
- Solution: This problem can be solved by removing tab "New Account" and adding button for new account under tab "Account". We had requirement keep design of desktop version, so we did not repair it. This change would be decision of a stakeholder.

Problem confirm activity

- User had problem confirm activity by clicking to confirm button.
- Occurrence: high
- Source: Buttons were too small and hidden down on a screen
- Solution: Size of buttons was extended and text size highlighted

Problem create new account

- Users had problem recognize he could fill account name and select table
- Occurrence: high
- Source: Activities were described wrongly and too close to each other
- Solution: Added more space and added labels.

Problem recognize fields to fill

- In our Android version user had problem recognize if showed text is normal text or a field to write input.
- Occurrence: high
- Source: Form of hints in these fields was written wrongly
- Solution: Hints in text field were transformed to be more understandable

One useless step

- User was redirected to a tab “Accounts” after creating a new account and then he had to select that account to make an order. Different from desktop version.
- Occurrence: high
- Source: Wrong implementation by developer. Desktop version solved this useless step.
- Solution: Faultiness was removed. After creating account user is redirected to a tab “Order” for this account.

Orientation in tabs

- User had problem recognize tabs and use them to switch to different screen.
- Occurrence: low
- Source: Tabs had same background color as rest of the screen
- Solution: Change background color of tabs

Problem recognize selected account

- User had problem recognize which account is selected and made tasks under different accounts.
- Occurrence: low
- Source: Account name had same size as other text
- Solution: Highlight name of selected account in tabs

Return in activity by using Android return button

- User did something he did not want and tried return it by using android return button. It returned him to previous screen.
- Occurrence: low
- Source: User was using inappropriate way how to cancel something
- Solution: Android return button was blocked

Our usability test was created to find errors within our application. The errors were then analyzed and corrected. A few of them we were unable to correct because our application's GUI was implemented as a desktop version. If we had changed anything, those changes would have affected the GUI of the entire CashBob application. So the final decision to not make those changes was decided by a stakeholder. Final design of our application is in the appendix Final Application Design.

Feedback from users:

- Accounts for tables would be created as default and no need create them every time
- Creating new account under tab accounts

4.3 ACCEPTANCE TEST

Acceptance test is a test conducted to determine that software product meets requirement specification [27]. For this test we will use scenarios from chapter Use Cases to prove it. This testing is done with a stakeholder to show him, that everything is implemented properly. For each scenario flow we tested, that application behavior to user actions match the description.

5 CONCLUSION

The goal of this thesis was to develop a fully functional Android application for waiters in restaurants. This goal was successfully reached. The application provides waiters the ability to manage accounts and orders and perform activities related to it. All requirements were successfully met except printing on a server and mobile printer. Problems appeared during development, which extended the time taken to accomplish our tasks. Because of this, we did not have enough time to implement these last two features of our application. However, according to meetings with a stakeholder, these two requirements have been changed to optional. All other requirements, including changes given at meetings with the stakeholder, were accomplished and satisfies the stakeholder's needs.

The biggest problem we had to solve was dealing with communication with server. We tried to test the application against a server provided by web application developer, but an interface was not implemented properly and it was impossible to test functionality. To solve the issue, we started to work with a server provided by a desktop application. However, the documentation was written poorly and missing parts, so we had to use alternative methods to figure out how to communicate with the server properly.

According to our requirements, the application was implemented on tablet devices and had the ability for the text size and design to be adjustable for each device. On smaller devices, this could cause problems with the control of the application. The application is provided in two languages: Czech and English. A user manual was written for our application and is attached as an appendix.

Every implemented functionality was tested to ensure it works correctly. In my opinion, the application is ready to be used in testing mode because all implemented functionalities have been implemented properly. The only limitation is that the application cannot print receipts. However, if this function is implemented, the application will be prepared to be used in real restaurants and will satisfy the restaurant managers who choose to use the CashBob system for their business.

Developing our application gave us a lot of experience. We learned new methods of how to implement application features as well as improved our problem solving skills. Writing this thesis was part of subject Project Control, so many previously learned topics were used and better understood because of this thesis.

I would like to thank my supervisor for his time spent helping me with my thesis and advice given during that time. I have learned a lot from this experience, all of which will be helpful in my future IT career.

5.1 NEW IDEAS FOR CASHBOB

Throughout the time spent developing this application I had a few ideas of extensions for CashBob:

- Item price is saved in one currency. According to settings, application will automatically convert price to another currency.
- If item will be selected, application will show side dishes for that item (for example: if grilled chicken will be selected, application will offer add potatoes or bread).
- Possibility to list in previous payments and reprint them.
- Make application to show the cook new orders and inform the waiters of the order that items are prepared to be served.
- Add meal menu to a menu card-one item in menu card can order more items (e.g. menu1: select soup and chicken).
- List of ingredients will be available for waiters within the application.
- Ordered items of an account will be paid for from a deposit (money payed to restaurant before any order was done) - good for company events in a restaurant.
- Possibility to select amount of an item (e.g. for red wine, we can select 1dcl or 2dcl).

6 BIBLIOGRAPHY

[1] D. Bosomworth. Mobile Marketing Statistics 2015. 22 July 2015. Smart Insights [Online]. Available: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

[2] Forecast for global shipments of tablets, laptops and desktop PCs from 2010 to 2019 (in million units). 2015. Statista [Online]. Available: http://www.statista.com/statistics/272595/global-shipments-forecast-for-tablets-laptops-and-desktop-pcs/

[3] CashBob [Online]. Available: http://cashbob.cz/

[4] V. Samek. Restaurant System Deployment and Promation. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/61850

[5] M. Kosek. Working Shifts Planning Module for CashBob System. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/9110

[6] Š. Tesař. "Úpravas a Příprava na Reálné Nasazení Systému CashBob. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/10484

[7] T. Apeltauer. Restructuring Storage and Stocktaking Module of CashBob System. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/10529

[8] J. Vrtiška. CashBob - inventory management module. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/16351

[9] L. Vyhlídka. CashBob Server. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/16350

[10] J. Rohlíček. Modification and Preparation of CashBob System for Deployment. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/18510

[11] A. Švec. Extending the mobile client of a restaurant system for OS Android. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/18260

[12] A. Makarič. Modular Architecture of Restaurant System. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/25053

[13] J. Kubíček. Restaurant System Deployment and Maintenance. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/24973

[14] J. Dryk. Mobilní Klient Restauračního Systému. ČVUT DSpace [Online]. Available: https://dspace.cvut.cz/handle/10467/61643

[15] Android [Online]. Available: https://www.android.com/

[16] Apple [Online]. Available: http://www.apple.com/ios/

[17] I. Sommerville. Software Engineering 9th ed. 2009. Pearson [Print]

[18] R. F. Goldsmith. Differentiating between Functional and Nonfunctional Requirements. June 2009. TechTarget [Online]. Available: http://searchsoftwarequality.techtarget.com/answer/Differentiating-between-Functional-and-Nonfunctional-Requirements

[19] Satzinger John, Robert Jackson, Stephen Burd. System Analysis and Design in a Changing World 7th ed. 2015. Cengage Learning [Print]

[20] Dashboards. 2 November 2015. Android [Online]. Available: https://developer.android.com/about/dashboards/index.html

[21] Android 4.4 APIs. Android [Online]. Available: http://developer.android.com/about/versions/android-4.4.html

[22] Android Studio. Android [Online]. Available: http://developer.android.com/sdk/index.html

[23] Oracle. VirtualBox [Online]. Available: https://www.virtualbox.org/

[24] Rest Api. GitLab Fel[Online]. Available: https://redxml.felk.cvut.cz/cashbob/cashbob/wikis/rest

[25] AsyncTask. Android [Online]. Available: http://developer.android.com/reference/android/os/AsyncTask.html

[26] J. Nielsen. Why you only need to test with 5 users. Nielsen Norman Group [Online]. Available: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

[27] Acceptance testing. 31 March 2016. Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Acceptance_testing

[28] Function Point. Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Function_point

APPENDIX: PROJECT PLAN

Project planning is an important activity. It will help us to make the structure of the project, so that we can see what is done and track our progress. Also, we can estimate time to finish project and a budget. In project planning, we also need to focus on risks, which can affect our project plan. By taking risks into an account, we can be prepared for them and reduce their effects.

6.1 WORK BREAKDOWN STRUCTURE

Work breakdown structure (WBS) is a structure, which shows us characteristic of our project. Project is divided to delivery items which are divided to smaller tasks. This division shows us a structure of the project and what have to be done.

WBS for our project:

- Implementation
    - User Login
    - Create Application Structure
    - Manage Accounts
        - Show List of Accounts
        - Create Account
        - Create New Order
        - Move Items between Accounts
        - Make a Payment
        - Print a Receipt on a Server Printer
        - Print a Receipt on a Mobile Printer
    - Usability test
- Thesis
    - Introduction
    - Project Overview Statement
    - Project Plan
    - Analysis
    - Design
    - Implementation
    - Testing
    - Conclusion
    - User Manual
    - Make Final Document
        - Text Correction
        - Graphic Style and Structure

6.2 TIME SCHEDULE AND TIME ESTIMATION

For our project we have to create a time schedule. WBS gives us a list of tasks we have to do. We have to order them and estimate how much time each task can take, so it will show how much time can be spent on the project.

In the table 5 is shown the order of tasks. Each task of the project was estimated (in hours) according to our previous experience and knowledge. Better experience with a task brings more accurate estimation than without it.

Only one person is carrying out this project, so working on more tasks simultaneously is not possible. Estimated time to finish whole project, given by sum of estimations of all tasks, is 346 hours. If we take into consideration working usual 8 hours per day, finishing the project is going to take almost 44 days. If a problem appears and extends estimated time for one single task, it will affect the whole project and extend time for the delivery of the final product.

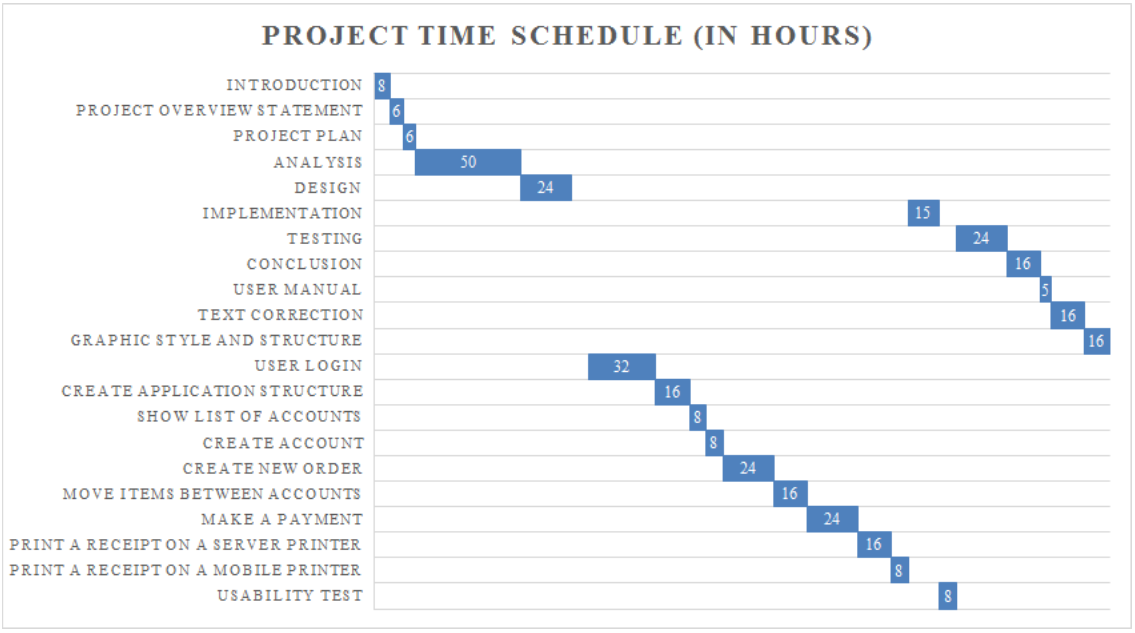


Table 5: Time Schedule (in hours)

6.3 RISK PLANNING

In every project can appear risks, which have an effect a project [17]. We can divide these risks into three categories according to their effect:

- Business risks: Risks that effect an organization developing or promoting the software
- Project risks: Risks that effect a project time schedule or resources
- Products risks: Risks that effect quality or performance of the developed software

There are also risks that can have effects in more than one category. These risks might change project schedule or reduce quality of the software. Our goal is to avoid the possibility, that any risk will appear.

First at all, we have to identify these risks. Then we need to analyze them (what is the probability of their appearance and how big effect they can have on the project) and plan a solution. Finally, we need to monitor if any risks appear during the project.

These are risks, that we identify for our project:

Rest API does not communicate properly

- Probability: moderate
- Effect: serious
- Solution: Developer will study documentation and communicate with developers of other CashBob applications.

Developer does not have enough knowledge

- Probability: moderate
- Effect: tolerable
- Solution: Developer will properly study each requirement/task to find the best solution. Developer will find another developer with more experience to teach him what he needs to know.

Team member is unavailable

- Probability: very low
- Effect: catastrophic
- Solution: Developer will make a timetable for activities of the project with enough time to solve any unexpected problem.

Customer changes requirements

- Probability: low
- Effect: serious
- Solution: Developer will discuss with the customer every part of the application before developing to avoid serious change.

<u>The time required to develop the software is underestimated</u>

- <u>Probability</u>: moderate
- <u>Effect</u>: serious
- <u>Solution</u>: Developer will alter old time schedule to solve every problem/task/requirement on time and monitor work efficiency to make estimate more accurate.

<u>Customer will not communicate</u>

- <u>Probability</u>: low
- <u>Effect</u>: catastrophic
- <u>Solution</u>: All requirements will be written before the project starts in order to minimalize effect when a customer is unavailable.

<u>Correction of thesis will take more time</u>

- <u>Probability</u>: moderate
- <u>Effect</u>: tolerable
- <u>Solution</u>: Split thesis into chapters and make corrections of each chapter after completion.

<u>Printing of thesis will take more time</u>

- <u>Probability</u>: very low
- <u>Effect</u>: tolerable
- <u>Solution</u>: Find best offer for print order document desks in advance.

6.4 PROJECT BUDGET

For our project we would like to estimate the price. We decided to use three types of estimations: From WBS (Work Breakdown Structure), functional points and COCOMO. Estimation from WBS calculates price from time spent on a project, estimation from functional points calculates price from product characteristic and estimation from COCOMO calculates price from project characteristic.

6.4.1 WORK BREAKDOWN STRUCTURE

For the estimation from WBS we calculate a time spent on whole project and using price of work we get final project price. According to a project timetable, we estimated we will spend 346 hours to finish a project. If a salary is 150 CZK per hour, the project cost will be 51 900 CZK.

The estimation covers only cost of time. It does not include price of developer experiences, project complexity or development costs (e.g. energy, internet, etc.). The estimation is then used by developer company to calculate how much money they would spend on work of developers.

6.4.2 FUNCTIONAL POINTS

Functional point is the unit of measurement for software projects to express the amount of business functionality that product provides to its users [28]. From functional points we can count time to develop a software and then its price. For calculation of functional points, we have to divide our software solution to five components:

- External inputs
- External outputs
- External inquiries
- External interface files
- Internal logical files

Table 6 displays counted functional points for our software solution. Each component is classified according to its complexity multiplied by the number of attributes that have to consider.

Component	Value
External Inputs	12
External Outputs	15
External Inquiries	5
External Interface Files	20
Internal Logical Files	7

Table 6: Estimate from Functional Points

Sum of these values is 59 function points.

The estimation can be done only if we know the structure of the project. Accuracy of the estimation depends on how many hours takes one functional point. Experienced developer spends less hours on a functional point than beginner.

In average, one functional point is similar to 8 working hours, so our estimation is 472 working hours. If out salary is 150CZK per hour, then project budget on salary side will be 70 800 CZK.

Problem could be that functional points are counted only by software characteristic, therefore time spent on documentation is not included.

6.4.3 CONSTRUCTIVE COST MODEL

Constructive Cost Model (COCOMO) calculates the cost of the project according to past, current and future project characteristics. In the table 7 is shown a characteristic of our project divided to software size, software scale drivers and software cost drivers. For calculate COCOMO we used website http://csse.usc.edu/tools/COCOMOII.php. From the calculation we read out that for having month salary 24 000 CZK, we have:

- Effort: 26 person-months
- Schedule: 10.8 months
- Cost: 632 257 CZK (salary 24 000 CZK per month)

Estimated cost of our project from WBS is 51 900 CZK, from functional points is 70 800 CZK and from COCOMO is 632 257 CZK. According to COCOMO estimated schedule 10.8 months, this method looks overestimated. The method uses project characteristic and the project is too simple to gives accurate estimation, especially when team contains only one developer, who effects this project characteristic.

Software Size [SLOC]	New	3000
	Reused	0
	Modified	0
Software Scale Drivers	Precedence	Low
	Development Flexibility	Low
	Architecture / Risk Resolution	Low
	Team Cohesion	Very High
	Process Maturity	Low
Software Cost Drivers		
Product	Required Software Reliability	Very High
	Data Base Size	Nominal
	Product Complexity	Low
	Developed for Reusability	High
	Documentation Match to Lifecycle Needs	Nominal
Personnel	Analyst Capability	Low
	Programmer Capability	Low
	Personnel Continuity	Low
	Application Experience	Low
	Platform Experience	Low
	Language and Toolset Experience	Low
Platform	Time Constraint	Very High
	Storage Constraint	Nominal
	Platform Volatility	Low
Project	Use of Software Tools	High
	Multisite Development	High
	Required Development Schedule	Nominal

Table 7: COCOMO

6.5 EVALUATION OF ESTIMATION

Before working on the thesis we divided our work into smaller tasks and estimated how much time we would spend on each one. We also created a time schedule for our work on these tasks.

The work breakdown structure was correctly estimated and no changes were made (only the tasks for implementing printing functions were skipped). Our estimation of how much time was spent on each task was mostly correct; some tasks were underestimated or overestimated. Figure 23 displays the time differences between our estimations and the real amount of time spent on each task.

We estimated that our project would take 346 hours which would cost 51 900 CZK (at 150 CZK per hour). The real time of our work was 340 hours which costs 51 000 CZK. The total time of this project was not very different compared to the estimated time. However, two tasks were not implemented, so the time difference could have been much bigger. Estimation depends on our experiences. With more experience, we would reach a more precise estimation of tasks and total project time.

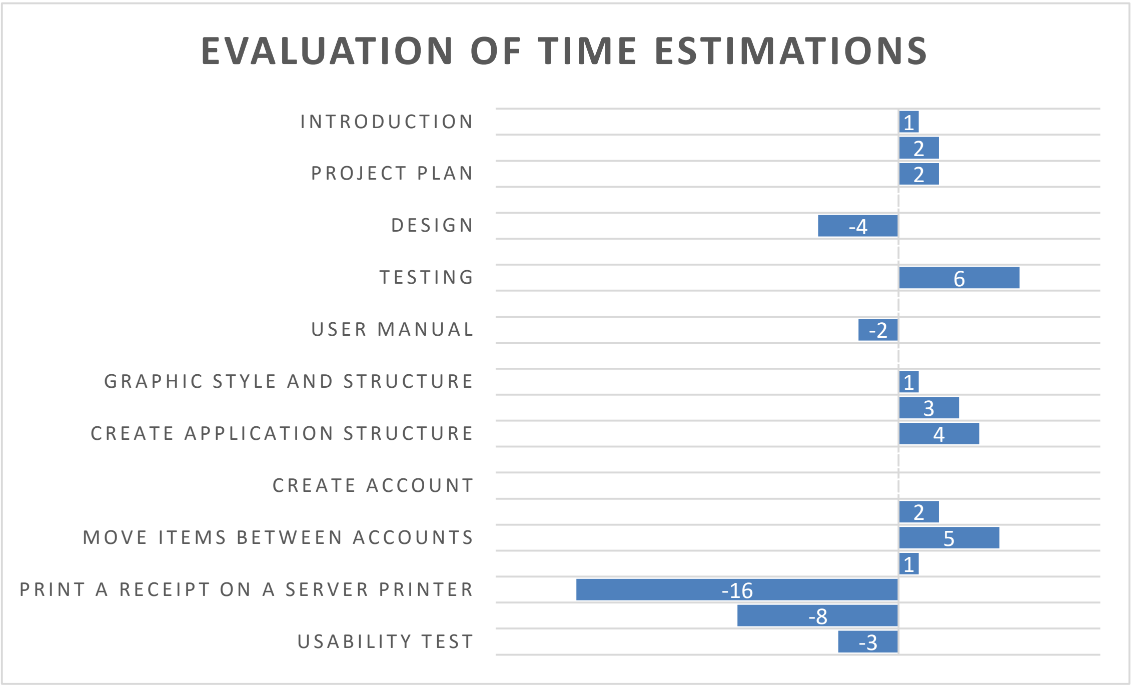


Figure 23: Evaluation of time estimations

7 APPENDIX: FINAL APPLICATION DESIGN

In this chapter we will show final GUI of the application:

- Figure 24 displays a login screen
- Figure 25 displays an account screen
- Figure 26 displays a new account screen
- Figure 27 displays an order screen
- Figure 28 displays a payment screen
- Figure 29 displays a transfer item screen

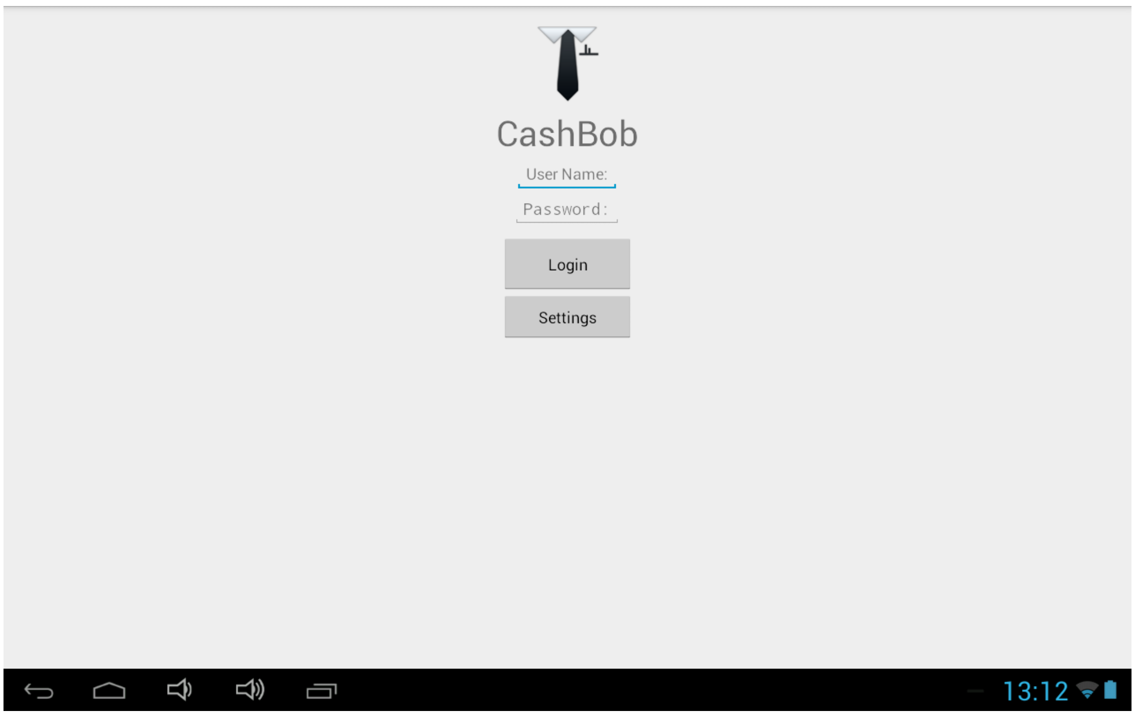
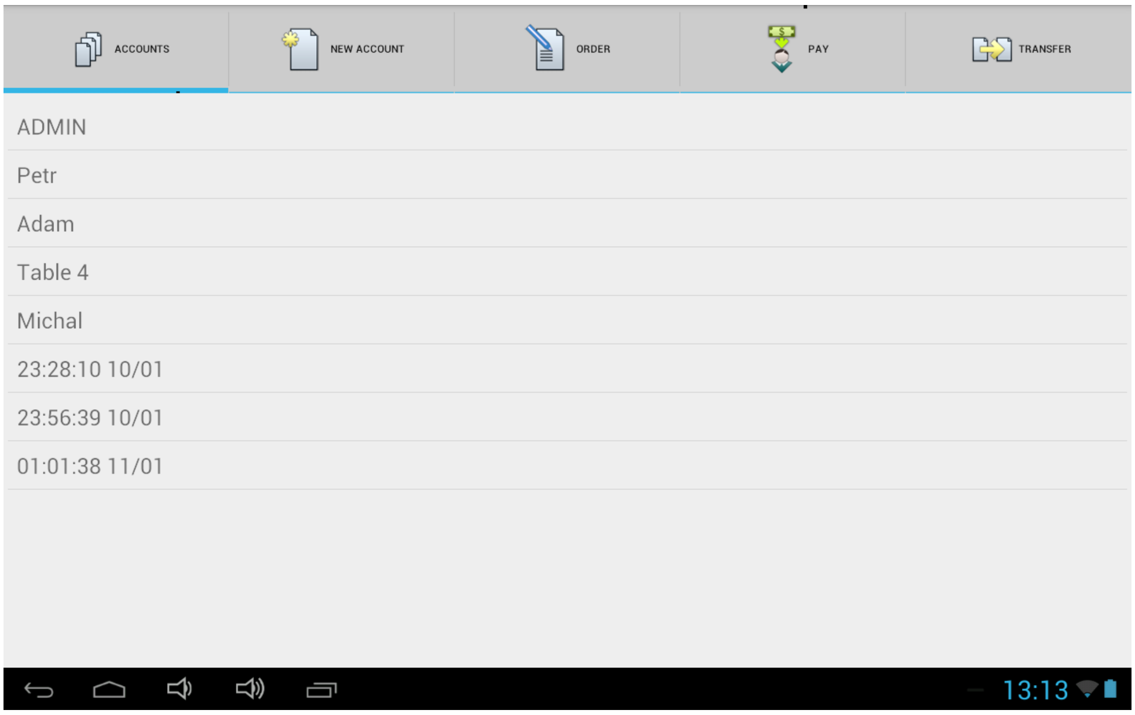


Figure 24: Login Screen



ADMIN

Petr

Adam

Table 4

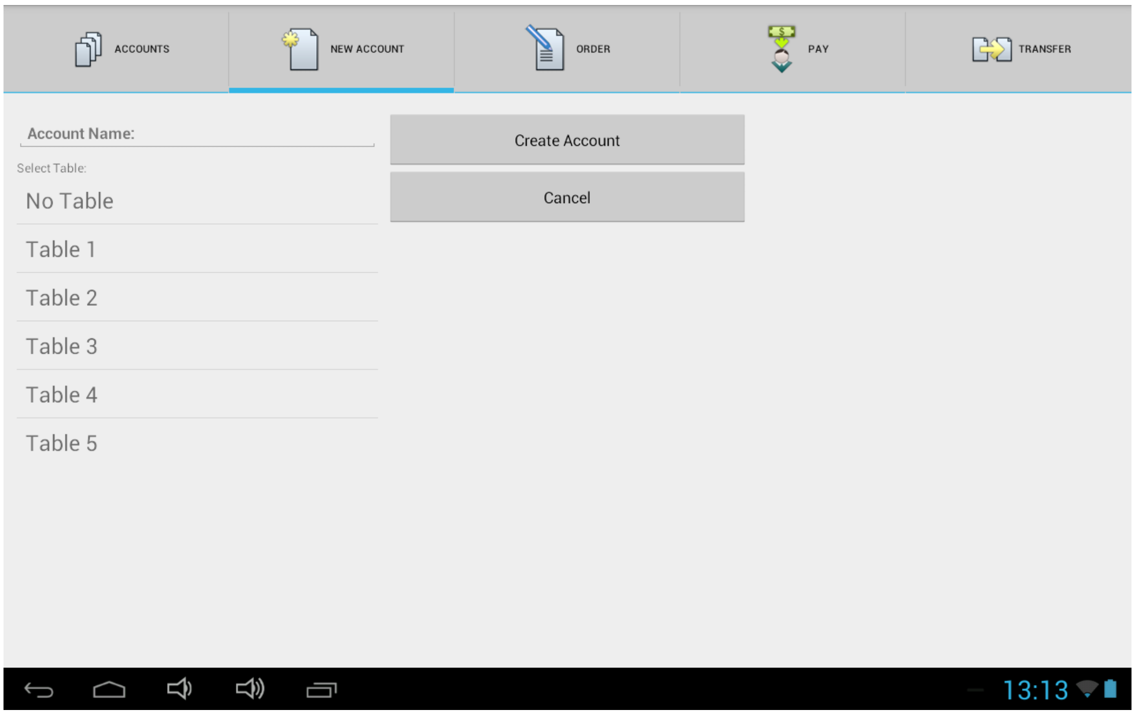
Michal

23:28:10 10/01

23:56:39 10/01

01:01:38 11/01

Figure 25: Account Screen



Account Name:

Select Table:

No Table

Table 1

Table 2

Table 3

Table 4

Table 5

Figure 26: New Account Screen

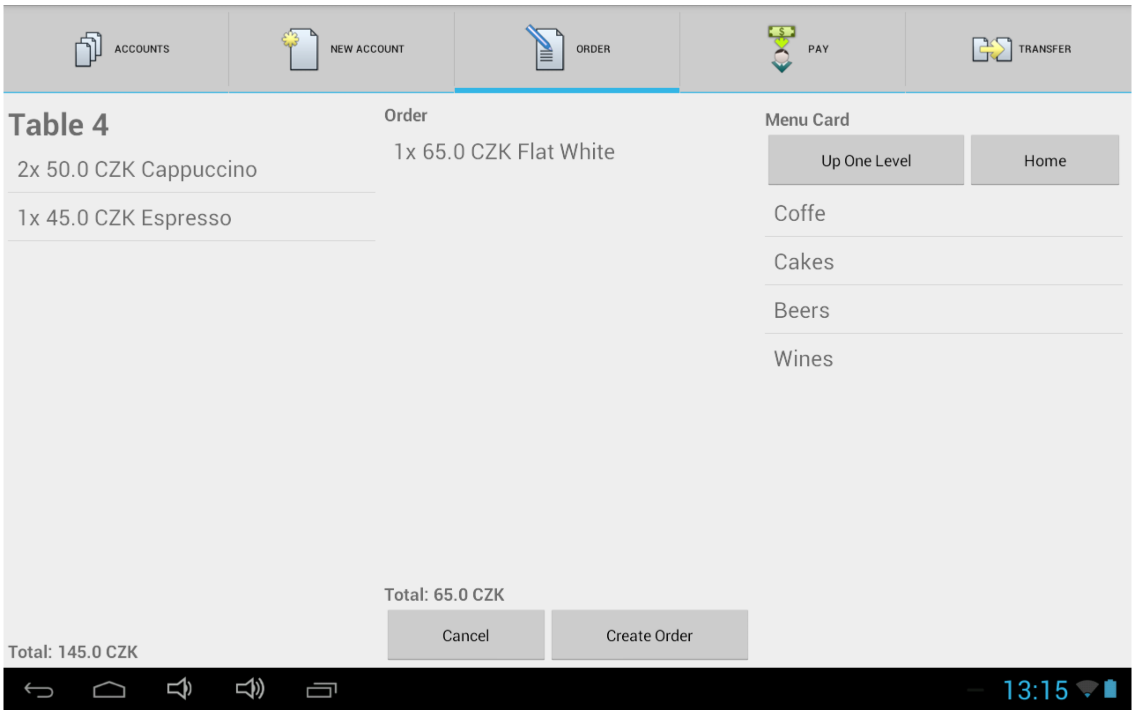


Table 4

2x 50.0 CZK Cappuccino

1x 45.0 CZK Espresso

Total: 145.0 CZK

Order

1x 65.0 CZK Flat White

Total: 65.0 CZK

Menu Card

Coffe

Cakes

Beers

Wines

Figure 27: Order Screen

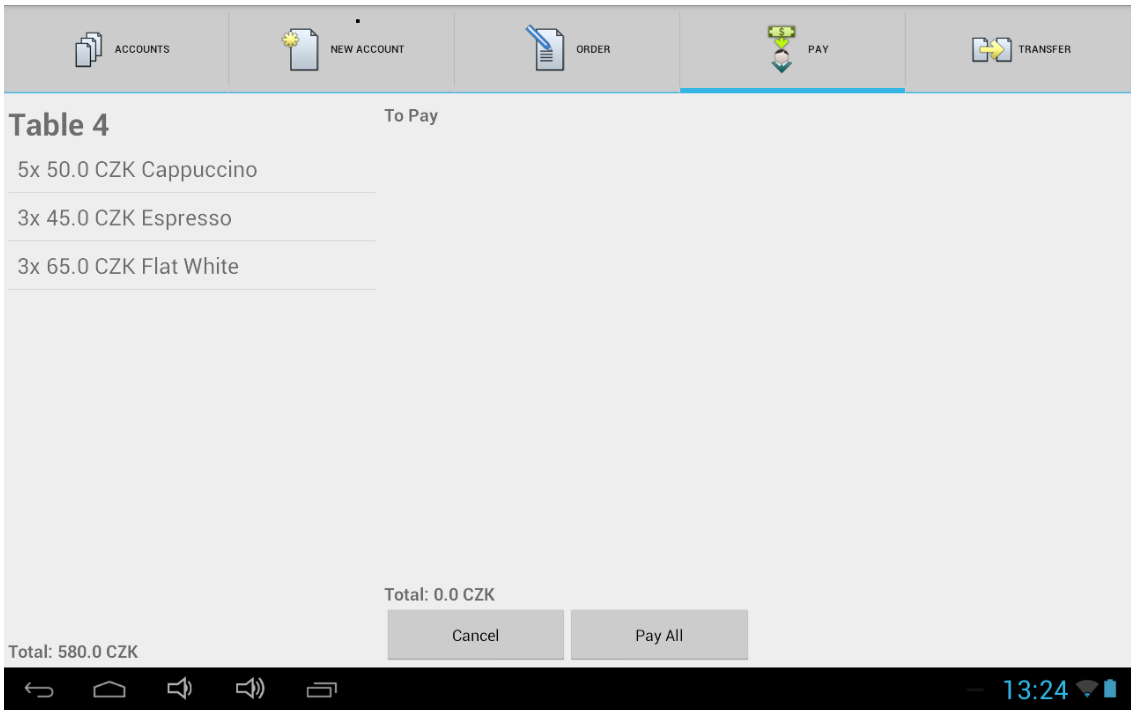


Table 4

5x 50.0 CZK Cappuccino

3x 45.0 CZK Espresso

3x 65.0 CZK Flat White

Total: 580.0 CZK

To Pay

Total: 0.0 CZK

Figure 28: Payment Screen

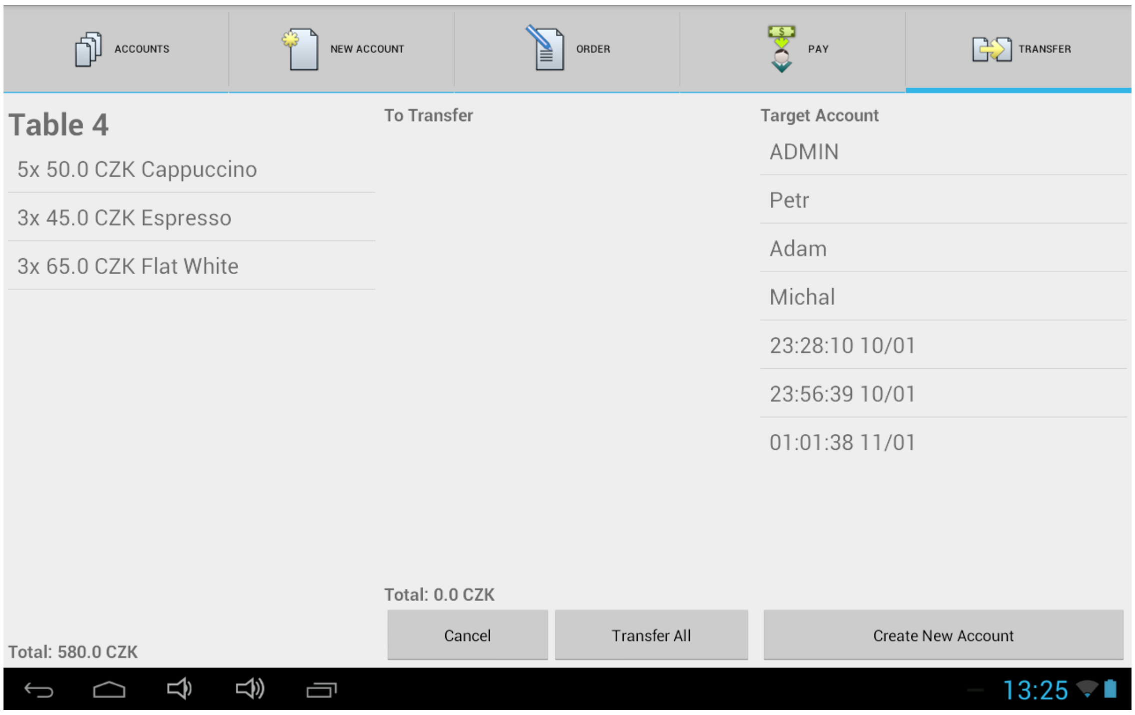


Table 4

5x 50.0 CZK Cappuccino
3x 45.0 CZK Espresso
3x 65.0 CZK Flat White

Total: 580.0 CZK

To Transfer

Total: 0.0 CZK

Target Account

ADMIN
Petr
Adam
Michal
23:28:10 10/01
23:56:39 10/01
01:01:38 11/01

Figure 29: Transfer Items Screen

8 APPENDIX: USER MANUAL

CashBob is a solution for restaurant management to deliver a new and easier way of how to manage their restaurant and carry perform activities related to restaurants. CashBob mobile application provides restaurant waiters the ability to manage accounts and orders and perform activities related to it.

8.1 LOGIN

- Open the application and fill your **User Name** and **Password**. If your account does not have attached role WAITER, application will not be accessible for you.
- Click to button **Settings**, fill **Server IP** (without http:// or https://) and click to **Save**.
- Click to button **Connect**. If validation is successful, you will be redirected to screen with the list of open accounts.
-

8.2 CREATE AN ACCOUNT

- Select tab **New Account**. System will load New Account screen.
- Fill an account name (optional), select a restaurant table (optional) and click to button **Create account**. System will create the account and redirect you to Order screen.

Note: If a name of the account is not filled and a table is selected, the account name will be the selected table.

Note: If a name of the account is not filled and a table is not selected, the account name will be time when account was created.

8.3 MAKE AN ORDER

- Select tab **Accounts**. System will load Account screen.
- Select an account where you want do an order. System will load Order screen.
- If you want to order an item that you ordered for the account before, click on it in an account item list.
- If you want to order a new item, find it in a menu card.
- If you do not want order selected item, click on the item in the list of items to order. Confirm your order by clicking on button **Order**. System will create the order.

8.4 ORIENTATION IN MENU CARD

Menu card contains sub-menus and items, which can be ordered. If you want to show items in submenu, click on that sub-menu.

To get back to previous menu, click on the button **UP.**

To get back to root menu list, click on the button **HOME.**

To add an item to an order list, click on the item.

8.5 MOVE ITEM BETWEEN ACCOUNTS

- Select tab **Accounts**. System will load Account screen.
- Select an original account. System will load Order screen.
- Select tab **Transfer**. System will load Transfer screen.
- Select items to move by clicking on them.
- If you do not want move selected item to target account, click on the item in the list of items to move.
- Select a target account, where you want to move these items. You can create a new account by clicking on the button **Create Quick Account** (fill account name and **Confirm**).
- Click on button **Transfer**. Items will be moved to the target account.

Note: If you want move all item from one account to another, choose target account and click to button **Move All**. All items will be moved to target account.

8.6 MAKE PAYMENT

- Select tab **Accounts**. System will load Account screen.
- Select an account to make a payment. System will load Order screen.
- Select tab **Pay**. System will load Payment screen.
- Select items to pay by clicking on them.
- If you do not want pay for selected item, click on the item in the list of items to pay.

11. Click on **Pay**. System will show dialog with selected items and price.

- Confirm that by clicking on button **Pay**. Payment will be saved.

Note: If you want move all item from one account to another, choose target account and click to button **Move All**. All items will be moved to target account.

8.7 CHANGE APPLICATION LANGUAGE

If you want change application setting to Czech Language, set Android OS to Czech language.

Note: For every another language setting in Android OS, the application will be provided in English language.

9 APPENDIX: COMPACT DISK

To thesis has been attached a compact disk. It contains:

- CashBob (Application source code)
- CashBob.apk (Android application package)
- Thesis_Tomas_Hogenauer.pdf (thesis text in PDF file)