

Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Computer Science and Engineering



Bachelor's Project

**Developing an accounting component for a car-sharing  
support information system**

*Jakub Ječmínek*

Supervisor: Ing. Martin Komárek

Study Programme: Software Engineering and Management

Field of Study: Software Engineering

April 6, 2013



## Aknowledgements

I would like to thank my parents and grandparents for support, then I'd like to thank my supervisor Ing. Marin Komárek for accepting my request to join Metrocar team. Also I have to mention Bc. Petr Pokorný who gave me a lot of advices and recommendations about Django/Python development and my girlfriend Zdenka who helped me with language correction of this thesis.



## Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Zábřeh na Moravě on December 21, 2012 .....



# Abstract

The goal of this work is to continue in development of web application Metrocar, which will be used for operating carsharing company. It addresses analysis, design and implementation of application which connects system Metrocar and accounting system Flexibee. Information system Metrocar is developed on Python platform and framework Django.

As a part of implementation of application which connects system Metrocar and system Flexibee is developed library Flexipy in Python. This library is using REST API which is a part of system Flexibee.

# Abstrakt

Cílem práce je pokračovat ve vývoji webové aplikace Metrocar, která bude sloužit k provozování carsharingové společnosti. Zabývá se analýzou, návrhem a implementací aplikace, která propojuje systém Metrocar s účetním systémem Flexibee. Informační systém Metrocar je vyvíjen na platformě Python za použití frameworku Django.

V rámci implementace aplikace pro komunikaci mezi systémem Metrocar a systémem Flexibee je vyvíjena i knihovna Flexipy pro komunikaci se systémem Flexibee v jazyce Python. Tato knihovna využívá REST API, které je součástí systému Flexibee.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Carsharing definition . . . . .	1
1.2	History of the project . . . . .	1
<b>2</b>	<b>Problem description, goal specification</b>	<b>3</b>
2.1	Problem description . . . . .	3
2.2	Goal specification . . . . .	3
2.2.1	On-line banking . . . . .	4
2.2.2	Accounting system . . . . .	5
<b>3</b>	<b>Analysis and Solution Proposal</b>	<b>7</b>
3.1	Invoices application . . . . .	7
3.2	Flexipy . . . . .	8
3.3	Accounting module . . . . .	8
<b>4</b>	<b>Implementation</b>	<b>11</b>
4.1	Iteration 1 . . . . .	11
4.1.0.1	Introduction . . . . .	11
4.1.1	Structure of flexipy library . . . . .	11
<b>5</b>	<b>Testing</b>	<b>13</b>
5.1	Flexipy . . . . .	13
<b>6</b>	<b>Summary</b>	<b>15</b>
6.1	Summary of semestral project . . . . .	15
<b>A</b>	<b>Seznam použitých zkratek</b>	<b>19</b>
<b>B</b>	<b>UML diagramy</b>	<b>21</b>



# List of Figures

B.1 Seznam přiloženého CD — příklad . . . . .	21
---	----



# List of Tables



# Chapter 1

## Introduction

### 1.1 Carsharing definition

Carsharing is model of renting cars where group of people is sharing car for a short periods of time. This is especially useful for people who use car only occasionally. The organization that provides carsharing service don't have to be necessarily commercial company, it can be cooperative, public agency or ad hoc grouping[15].

Carsharing model of renting cars is today common in North and Western Europe. In Czech Republic there is one company that provides this kind of services for city Brno[14]. Therefore there is still marketplace for another company that would provide carsharing services.

### 1.2 History of the project

Project Metrocar is under development since year 2008. Original implementation of the project was part of a bachelor thesis by Bc. Ondřej Nebeský. His solution was implemented in PHP and CMS Drupal. This implementation allowed users to view cars on the map, create reservation for cars and monitor their journeys. It also provided mechanisms for creating and managing carsharing subsidiaries[7].

The final result didn't met all the requirements and also chosen architecture wasn't optimal for the planned deployment of the system. These shortcomings were addressed in the master's thesis of Ing. Filip Vařecha. His goal was to improve and transform this system. The main goal was to create public API for the system. As the result, the whole system was rewritten from scratch. Python and Django web framework were used as primary technologies for the resulting system. The complete list of changes and new features can be found in the thesis of Filip Vařecha[13].

Next development on the project was done as part of bachelor thesis by Bc. Jan Wágner. His main goal was to finish the system for reservations and user management. He was also responsible for selecting hosting for the application. Further details about his work on the project can be found in the resulting thesis[17]. There were also some small contributions to the project by students of the subject Y36SI2. The complete list of their work and changes can be found on the website of the project in the final reports[3].

Last contributor to the project was Bc. Petr Pokorný. He was primarily focused on the communication between the server side of the application and car units. He was also responsible for extreme improvement in documentation of the project and updating project's dependencies. Thanks to his vast knowledge of Python and Django he has become important figure of the project and great advisor.



## Chapter 2

# Problem description, goal specification

### 2.1 Problem description

When I joined the Metrocar project, there was only partial solution for accounting in the system. The solution was capable only to create invoices and invoice items. For printing to PDF files and sending invoices to users, there was small Python script that was supposed to be running periodically by cron<sup>1</sup> and check if some invoices are supposed to be sent to the users. Also there was no way for users to gain credit, which is basically currency in our system. There was also no solution for checking state of company's account. Therefore the state(active, overdue and paid) of every invoice would have to be changed manually by manager of the company. This would force manager to spend too much time doing just checking statuses of invoices. Also this approach would probably fail for a larger group of users. Therefore it was decided, that system for accounting in the project has to be improved.

### 2.2 Goal specification

Design and implement application that provides connection of system Metrocar[1] to accounting software Flexibee[2]. Systems Metrocar and Flexibee will use this application for communication and exchange of data. This application will use REST API, which is provided by Flexibee[3] Official goal specification is as follows:

- Identify requirements for the application.
- Create UML models of proposed application.
- Implement the application.
- Create unit tests, functional tests and performance tests).
- Create documentation for the application.

---

<sup>1</sup>Cron is the time-based job scheduler in Unix-like computer operating systems.

Before we decided to use for our project Flexibee as an accounting system, I was doing as a part of the subject A7B16PRO a research about possibilities how to connect system Metrocar to homebanking solution and accounting system. As a result of this research, it was decided that we will use Flexibee. I decided to attach the output of my research about homebanking systems and accounting systems on Czech market to this report, because it served as a foundation for a decision to choose Flexibee.

### 2.2.1 On-line banking

Originally was system Metrocar without any kind of connection to company's bank account. Therefore, manager of company would have to manually check incoming payments and then change the status of appropriate invoice. This is very bad solution and it would be very time consuming for manager. The team that was working on the project in the year 2010 started to work on the solution, that would require to create python script that would manually parse the informations about payment from email or sms that would be sent from bank[6]. As a part of my research I found only this article about their ideas for this solution, but I wasn't capable to found any source code. Therefore I suppose they didn't implement this idea.

At first I thought that this solution would be great for system Metrocar. I also found out that owner and founder of the Roští.cz<sup>2</sup> Adam Štrauch created Python project that provides similar solution. His module is capable to log in to on-line banking system of mBank company and from there download informations about incoming payments in CSV format and then parse them into Python data structures. His project is released under BSD license and is publicly available on github[2]. Article about the development of this module and the reasons for its creation can be found on his personal website[1]. This solution is not optimal for our project, because as even author has stated, if mBank change some part of their online banking system, the script will simply stop to work. This is also not exactly recommended approach to obtain this kind of informations. Typically every bank institution have their own system, which allows clients to download certain informations about payments, account status etc. These systems are called homebanking systems. There are several standard formats which these systems usually use. The complete list can be found here[? ]. From my experience in these days the most used ones are Gemini, MultiCash and ABO. The basic functionality of these systems is usually same, therefore I will further describe only what Gemini in version 5.0 provides, other format support this too. Banking operations:

- Domestic payments and direct debit orders.
- International payments.
- Domestic and international payments with future value date.
- Urgent payments.
- Permanent payment orders with automatic creation of a pre-set payment calendar.
- Time deposits.

---

<sup>2</sup>company that is providing hosting for project Metrocar(autonapul.cz)

Security features:

- Password for electronic signature.
- Individual user passwords and access rights.
- Authorization before sending payment orders to the bank.
- Private and public RSA key.
- Electronic signatures are automatically checked by bank institution.

Gemini has other useful features but not all of them are supported by every homebanking system. The most resourceful documentation about Gemini has Raiffeisen Bank[10] from which I learned about this format. Therefore these other special features might be implemented only in Raiffeisen Bank homebanking system.

Other features of Gemini:

- Export/import of payment orders to/from accounting system.
- Template of payment orders.
- Automatic communication with bank.
- Current exchange rates Raiffeisenbank Inc. and ČNB.

There are also other features, for complete list visit[10]. When I joined the Metrocar team as a part of the A7B16PRO subject, it was not yet decided which bank company will be used for Metrocar's company bank account. Therefore it was decided that my next assignment will be to select suitable accounting system for system Metrocar, which will already support communication and information exchange with homebanking system and that this communication will be using Gemini format and other formats too if possible.

### 2.2.2 Accounting system

I was responsible for selecting appropriate accounting system for our project by the supervisor Ing. Martin Komárek. I created a list of requirements which we expected from the accounting system.

- Capability to communicate and exchange data with homebanking system.
- The accounting system should have interface(for access from other systems).
- This interface should be open and free if possible.
- The accounting system should have sufficient documentation for developers of information systems.
- The aforementioned documentation should be free of charge.

After I had this specification about what kind of system we were looking for, I started to search on Czech market. A lot of people were telling me about system Pohoda[8]. After small research I found out that system Pohoda allows imports and exports of data through exchange of XML files[9]. XSD files and example XML files are publicly available and free. There is also short documentation for developers, but I found it to be too simple. The biggest problem with system Pohoda was the fact that for testing of my component, I would have to purchase a license for the system. Also a lot of people who are working in companies that are using system Pohoda told me that they have negative experience with the system.

So I started to look for other systems, until I spoke with teammate Petr Pokrný, who recommended me accounting system Flexibee[12]. After some research I found out that Flexibee is the only accounting system on Czech market that offers completely open REST API for developers with extensive documentation, examples and tutorials. They also provide publicly available cloud version of Flexibee which developers can use for testing of their applications. Other possibility of testing is to register on their website as a developer and then they will provide full version of their system for free for period of three months. They also have mailing list which they use to announce new releases and changes in their system. This is perfect ecosystem for developers. The REST API supports both XML import/export or JSON. Strongest side of Flexibee is documentation for developers.

After I familiarized myself with Flexibee adequately I presented it to the supervisor as a solution that I would recommend and that seemed to me as the most suitable to our project and our needs. Also after discussion with Petr Pokorný who already had experience with connecting information system to Flexibee, we suggested to use Flexibee in cloud instead of deployment on our server. This was approved by our supervisor Ing. Martin Komárek and I began to analyse existing application<sup>3</sup>Invoices and what would be the best approach for connection of accounting system Flexibee.

---

<sup>3</sup>Application in terms of Django application <https://docs.djangoproject.com/en/dev/intro/tutorial01/#creating-models>

## Chapter 3

# Analysis and Solution Proposal

In this chapter I will provide analysis of my solution on how to connect existing system Metrocar to an accounting system Flexibee. I will describe reasons for chosen approach and technologies.

### 3.1 Invoices application

Application invoices in project Metrocar has responsibilities for CRUD<sup>1</sup> operations with invoices and other accounting documents. It also contains tools for printing invoices and sending them to users(customers). This application was originally created by Filip Vařecha and then maintained by Jan Wágner. As I mentioned before, this solution has many shortcomings. Therefore as a part of this project I am supposed to connect system Metrocar to system Flexibee. At first I planned to remove application Invoices and replaced it by new application that would be directly connected to Flexibee. However as it turned out this application is very integrated into the Metrocar system. Also the standard things like managing invoices is well handled by the application. Flexibee is very good system and has a lot of satisfied customers, but even systems like this can experience from time to time problems. Petr Pokorný experienced a few server crashes while he was working on the project that involved Flexibee as an accounting system. At that point the whole system is inaccessible for users. because they can't create invoices or issue their bills. At this point keeping an old Invoices application could help. When a user creates invoice, or issue a bill or just do some action that involves accounting and at this point Flexibee crashes, accounting module(later described) will notice this and set certain actions into motion. Accounting module will use old Invoices application and store the changes or new data in local database in appropriate tables. Also he notifies administrator that the server on which is Flexibee deployed has crashed. After server with Flexibee is running again, accounting module will resolve inconsistencies between local database and Flexibee's records. This way we can say that old Invoices application will serve as some kind of fail-safe mechanism, which will be primarily used to serve users' requests while Flexibee is down. For this reasons I've decided to keep old application Invoices in the project Metrocar. It will be slightly modified and

---

<sup>1</sup>Create, read, update and delete.

certain functionalities will be transferred on to Flexibee (for example printing of invoices and sending them to users).

## 3.2 Flexipy

Flexipy is a Python library that is being created as a part of this project by myself. The main goal of this library is to provide simple interface for other Python/Django projects that wants to use Flexibee as their accounting system. Therefore flexipy is not being developed just as a component of Metrocar system, but as a standalone library that can be easily used in other projects. Hopefully the library will be available from PyPI<sup>2</sup> when the state of library will be optimal. Thanks to the fact that PyPI is central source for all third party Python libraries, flexipy would become available to other developers that want to use REST API of Flexibee.

Design goal of flexipy is to be as simple as possible. It is powered by Requests library<sup>[5]</sup> which provides tools for working with HTTP protocol. The main advantage of Requests over traditional urllib2 is simplicity. With Requests I am capable to send request with just one line of code. Also it simplifies work with HTTP status codes, which are used in Flexibee's REST API as identifiers of operation results.

Flexipy is using the fact that most of the fields of every evidence<sup>3</sup> item<sup>4</sup> are automatically populated by Flexibee during their creation, thanks to the mechanism of internal dependencies<sup>[11]</sup>. Therefore the number of fields that user of the flexipy has to specifically insert is not so big. Without this mechanism, creating new items in Flexibee would be very hard, for example if user would have to specify all attributes of invoice, he would have to manually insert about 100 values. Flexibee is also very customizable when it comes to specific company's requirements. For example, user can define custom relations and types of customer and then use these features to simplify the process of creation of new evidence items. This aspect of Flexibee is also part of flexipy as there is configuration file that can be customized to specific needs of the user and state of Flexibee. This mechanism will be described in further details in the implementation part.

## 3.3 Accounting module

As was mentioned above, the accounting module is responsible for handling situations in which Flexibee has crashed, but this will be only secondary purpose of this module. The main purpose of this module is to act as intermediate between Metrocar system and Flexibee. It defines interface for Metrocar system through which Metrocar interacts with Flexibee. Thanks to the nature of Python, this module can be easily replaced in the future with different implementation of the same interface, so on the side of Metrocar system, it will look like that nothing has changed, but actually Metrocar system will now be communicating with

---

<sup>2</sup>PyPI -The Python Package Index is a repository of software for the Python programming language.

<sup>3</sup>In Flexibee the word evidence corresponds to one specific part of the system, for example issued invoices, or address book of companies etc.

<sup>4</sup>By the word item is meant one specific document that belong to certain evidence, for example one concrete invoice.

completely different accounting system on the other side. Therefore there will be no coupling between Metrocar system and Flexibee. This is very important because if the interface of accounting module was coupled with Flexibee it would be extremely hard and tedious to change accounting system.

On the side of Flexibee system, the accounting module is using library flexipy that was described in the previous section. This will also help to adhere separation of concerns principle[16], because accounting module simply calls function from flexipy and passes data if needed. It does not know anything about Flexibee REST API or format of data in which Flexibee communicates.

The accounting module is responsible for these requirements: Creation of new invoices Cancellation of invoice - this is common use case in accounting systems. Deletion of invalid invoice - sometimes there is a need for deletion of invoice, for example if the Uprava udaju na fakturu Matching of payments - the module, must be capable to assign incoming payment to invoice by certain parameter. Sending reminder for unpaid invoices - this is already partially handled by Flexibee. Printing of invoices to PDF - the module will use build in support in Flexibee for printing of invoices. Sending emails with printed invoices - this is also already possible in Flexibee, therefore module will simply use and call this functionality through flexipy. Handling the situation in which the Flexibee server is unreachable - there is probability (small but still real) that Flexibee server will become unreachable (for example Flexibee server in cloud) then accounting module has to act appropriately. This will be described later on.

For better understanding of interactions between flexipy, accounting module and Metrocar system I created simple UML diagram using Enterprise architect. I've used this case modeling system because it was used before by previous teams.





## Chapter 4

# Implementation

In this part of thesis are described the implementation details of both flexipy, accounting module and all other parts of Metrocar system that were either created or modified by me. For this chapter it was decided to use the iteration model after discussion with Martin Komárek. This model allows to plan specific goals and problems for iteration and was also used in thesis of Jan Wágner. Another reason for this is the fact that after Petr Pokorný finished his work on the project I have become only person actively involved, that was familiar with server side of the project, therefore I took the role of project maintainer. So as a maintainer I had to solve problems that were not necessarily important for my thesis or part of my task description but were important for further continuation of the project as a whole. I also worked closely with Roman Kubů who is working on the car unit solution for the Metrocar project. All of these tasks are later on described in this chapter.

### 4.1 Iteration 1

#### 4.1.0.1 Introduction

The main goal of the first iteration was to implement flexipy library to the point where it could be used for basic accounting requirements of Metrocar system. This goal was partially achieved but the library was not completely finished due to certain tasks on server side of Metrocar system that were required by member of team who is working on the car unit solution.

#### 4.1.1 Structure of flexipy library

Structure of flexipy library is trying to follow best practices at python development described at The Hitchhiker's Guide to Python![\[4\]](#) which is written by important python community figure Kenneth Reitz who is also author of Requests library that is used in flexipy for handling HTTP requests and responses. The folder structure of flexipy library:

```
flexipy/  
├── docs/  
├── flexipy/  
│   └── __init__.py
```

```
├── addressbook.py
├── api.py
├── bank.py
├── config.py
├── config_example.py
├── invoice.py
├── main.py
└── test_flexipy/
```

## Chapter 5

# Testing

### 5.1 Flexipy

For Flexipy I used pytest library, which is standar testing tool for python applications. It is one of the most used testing libraries in the Python world.



## Chapter 6

# Summary

### 6.1 Summary of semestral project

I've joined the team Metrocar as a part of semestral project, at first I started to familiarize myself with the newest changes on the project made by Petr Pokorný. He started to apply best practices of Python development on the project, including thorough documentation, to which I added a tutorial on how to setup a local database on the developer's machine. At first I didn't know what exactly will be the topic of my the bachelor thesis or which part of the system will become my responsibility. Therefore at first I worked on fixing bugs and I also was tutoring other teammates, who never before programmed in Python or used Django framework. After discussion with supervisor, it was decided that my bachelor thesis and further work during semestral project will be focused on selecting an accounting system, that would also provide a way how to communicate with homebanking system. I also was helping other teammate, Petr Holeček who was writing functional and unit tests for the Metrocar system. This was great experience for me, same as the research of accounting systems and homebanking systems.



# Bibliography

- [1] Adam Štrauch. *Jak jsem si dělal API pro internetové bankovníctví mBank* [online]. 2012. [cit. 23.12.2012]. Dostupné z: <<http://initd.cz/posts/jak-jsem-si-delal-api-pro-internetove-bankovnictvi-mbank/>>.
- [2] Adam Štrauch. *python-mbank - mBank Python parser* [online]. 2012. [cit. 23.12.2012]. Dostupné z: <<https://github.com/creckx/python-mbank>>.
- [3] Assembla. [https://www.assembla.com/spaces/wagnejan\\_metrocar/wiki](https://www.assembla.com/spaces/wagnejan_metrocar/wiki).
- [4] Kenneth Reitz. *The Hitchhiker's Guide to Python!* [online]. 2012. [cit. 1.4.2013]. Dostupné z: <<https://python-guide.readthedocs.org/en/latest/writing/structure.html#structure-is-key>>.
- [5] Kenneth Reitz. *Requests: HTTP for Humans* [online]. 2012. [cit. 30.12.2012]. Dostupné z: <<http://docs.python-requests.org/en/latest/>>.
- [6] Metrocar team 2010. *eBanking a SMS* [online]. 2010. [cit. 26.12.2012]. Dostupné z: <[https://www.assembla.com/spaces/metrocar/wiki/eBanking\\_a\\_SMS](https://www.assembla.com/spaces/metrocar/wiki/eBanking_a_SMS)>.
- [7] NEBESKÝ, O. *Bakalářská práce: Rezervační systém carsharingové společnosti*. KP FEL ČVUT, 2009.
- [8] Pohoda. Accounting system Pohoda, . <http://www.ucto-pohoda.cz/>.
- [9] Pohoda. XML communication with accounting system Pohoda, . <http://www.stormware.cz/xml/>.
- [10] Raiffeisen Bank. *Gemini 5.0* [online]. 2012. [cit. 23.12.2012]. Dostupné z: <<http://www.rb.cz/firemni-finance/velke-podniky/platebni-styk-a-cash-management/elektronicke-bankovnictvi/gemini-5-0/>>.
- [11] S.R.O, F. S. *Flexibee's internal dependencies* [online]. 2012. [cit. 30.12.2012]. Dostupné z: <<http://www.flexibee.eu/api/doc/ref/internal-dependencies>>.
- [12] S.R.O, F. S. *Flexibee accounting system* [online]. 2012. [cit. 30.12.2012]. Dostupné z: <<http://www.flexibee.eu/>>.
- [13] VAŘECHA, F. *Diplomová práce: Transformace informačního systému pro carsharing do webové služby*. KP FEL ČVUT, 2010.

- [14] web:brno. Autonapůl Brno.  
<http://www.autonapul.org/>, stav z 30. 12. 2012.
- [15] Wikipedia contributors. *Carsharing* [online]. 2012. [cit. 23. 12. 2012]. Dostupné z: [<http://en.wikipedia.org/wiki/Carsharing>](http://en.wikipedia.org/wiki/Carsharing).
- [16] Wikipedia contributors. *Separation of concerns* [online]. 2012. [cit. 29(). 12. 2012]. Dostupné z: [<http://en.wikipedia.org/wiki/Separation\\_of\\_concerns>](http://en.wikipedia.org/wiki/Separation_of_concerns).
- [17] WÁGNER, J. *Bakalářská práce: Příprava systému autonapul.cz na reálné nasazení*. KP FEL ČVUT, 2012.



## Appendix A

# Seznam použitých zkratek

**XML** Extensible Markup Language

**JSON** JavaScript Object Notation

**API** Application programming interface

**REST** Representational state transfer

**CSV** Comma-separated values

⋮



## Appendix B

### UML diagramy

Tato příloha není povinná a zřejmě se neobjeví v každé práci. Máte-li ale větší množství podobných diagramů popisujících systém, není nutné všechny umísťovat do hlavního textu, zvláště pokud by to snižovalo jeho čitelnost.



Figure B.1: Seznam příloženého CD — příklad