Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Decision support backend module for commercial ERP system

Richard Sadloň

Supervisor: RNDr. Štefan Dušík, Ph.D
May 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	Sadloň	Jméno: Richard	Osobní číslo: 453455
Fakulta/ústav:	Fakulta elektrotechnická		
Zadávající katedra/ústav:	Katedra počítačů		
Studijní program:	Otevřená informatika		
Studijní obor:	Softwarové systémy		

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Modul podpory rozhodování pro komerční ERP systém

Název bakalářské práce anglicky:

Decission support backend module for commercial ERP system

Pokyny pro vypracování:

1. Get acquainted with ERP systems and provided DB export
2. Design a solution with the following features:
1. Imports data from the MS SQL database
2. Provides endpoint for basic statistic information of the imported data
3. Converts the imported data to dataset, usable for machine learning
4. Provides at least 3 various classification or regression methods
5. Allows crossvalidation and 2:1 test/train split
6. Allows asynchronous (non-blocking) training of the classifier
7. Provides basic health (utilization) information
8. Allows export of the trained model
9. Allows classification using the trained model
3. Implement the solution (C# programming language is preferred)
4. Create small testing dataset and verify the solution works as expected
5. Unit test and document the project
6. Test the solution with another provided DB

Seznam doporučené literatury:

[1] Marianne Bradford - Modern ERP: Select, Implement, and Use Today&#39;s Advanced Business Systems, 2008
[2] Bret Wagner and Ellen Monk - Concepts in Enterprise Resource Planning, 2001
[3] Kamel Nidal, Aamir Saeed Malik - EEG/ERP Analysis: Methods and Applications, 2014

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Štefan Dušík, Ph.D., Asseco Solutions a.s., Zelený Pruh 1560/99, 140 00 Praha 4, Braník

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: ______

Platnost zadání bakalářské práce: **20.09.2020**

[redacted]	[redacted]	[redacted]
RNDr. Štefan Dušík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

6. 5. 2019	[redacted]
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank my supervisor RNDr. Štefan Dušík, Ph.D. for his guidance during this work. I would also like to thank my parents for their support during this study.

Declaration

I declare that I have created this project independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague on May 13, 2019

....

Abstract

The goal of this project was to create a decision support module for the company Asseco Solutions, which develops ERP systems. This module allows importing data from a database, creating a dataset suitable for training of classifiers, training of classifiers and then using the trained model to classify new data. The application was developed mainly using the programming language C# and application framework Asp.Net Core 2.2. There were three different factory data on which functionality was tested.

Keywords: Classifier, C#, ERP, Asp.Net Core, Machine Learning

Supervisor: RNDr. Štefan Dušík, Ph.D

Abstrakt

Cílem tohoto projektu bylo vytvořit modul podpory rozhodování pro firmu Asseco Solutions, která se zabývá vývojem ERP systémů. Tento modul umožnuje import dát z databáze, vytvoření datasetu vhodného na trénování klasifikátorů, trénování klasifikátorů a následné použití natrénovaného modelu na klasifikaci nových dat. Aplikace byla vyvinuta hlavně využitím programovacího jazyku C# a aplikačního frameworku Asp.Net Core 2.2. K dispozici byly tři různé data z výroby na kterých byla funkčnost otestována.

Klíčová slova: Klasifikátor, C#, ERP, Asp.Net Core, Strojové učení

Překlad názvu: Modul podpory rozhodování pro komerční ERP systém

Contents

Figures

Tables

Chapter 1
Introduction

1.1 ERP

The acronym ERP stands for enterprise resource planning. It is a process used by companies to manage and integrate the important parts of their businesses. ERP systems tie together, define a plethora of business processes and enable the flow of data between them. What primarily distinguishes ERP software from stand-alone targeted software is a common central database from which the various ERP software modules access information, some of which is shared with the other modules involved in a given business process.

An ERP software system can integrate planning, purchasing inventory, sales, marketing, finance, human resources, and more.

1.2 Financial forecast

Financial forecast is an estimate of a future financial outcome for an organization. It estimates future income and expenses for a business over a period of time. By analysing an organization's current financial position and other factors, a properly developed financial forecast presents an assumption of how the organization will perform in the future.

1.3 Motivation

One of the problems of manufacture companies is production optimization. They can not expect that the same number of pieces will be sold from each product. Because of this reason some materials and semi-products lies in warehouse for long period of time, that can mean significant fixation of finances and inefficient use of the storage capacities.

According to [5] the production planner who makes decisions about what products and when they will be produced, does not focus on warehouse status but makes decision based on:

- leadership priority
- current orders
- available resources
- estimated needs
- own experience

However using historical production data and statistical methods, we can create a certain assumption about warehouse status for the future. This assumption can help production planner with making a decision whether to add a semi-product to the manufacturing chain. This can improve productivity and ultimately increase profit of the company.

1.4 Current situation

According to [5] and [6] in the current state there is already functional application which was created by Victoria Eykhmann and Michal Bouška. They implemented the application which can assign a numerical value to the semi-products which are expensive and lies in warehouse for long time. This value is called a penalty, and it represents the value of a semi-product in a warehouse accumulated over time [1] . Subsequently, value of the penalty is transformed. During this transformation, the penalty is multiplied by weight for each day, where weight gradually increases from 0 to 1. It reflects a fact, that company does not mind products stored for a short time, instead it minds long-term stored product.

Then the products can be divided into three classes according to their penalty:

- products with low priority
- products with middle priority
- products with high priority

In this project, Michal Bouška created python scripts which can import data from database and transforms them to a dataset suitable for training of classifiers then training and using of classifiers.

The part which was done by Victoria Eykhmann provides graphical interfaces(see the figure 1.1).

[1]

$$penalty = \sum_{days} number\_of\_total \times price\_of\_one$$

In this section, I would like to notice that this project was developed based on their work.

1.5 The goal of this project

The goal of this project is to solve the problem from the motivation section by implementing WEB API using C# language which should help a responsible person to make decisions about manufacture orders.

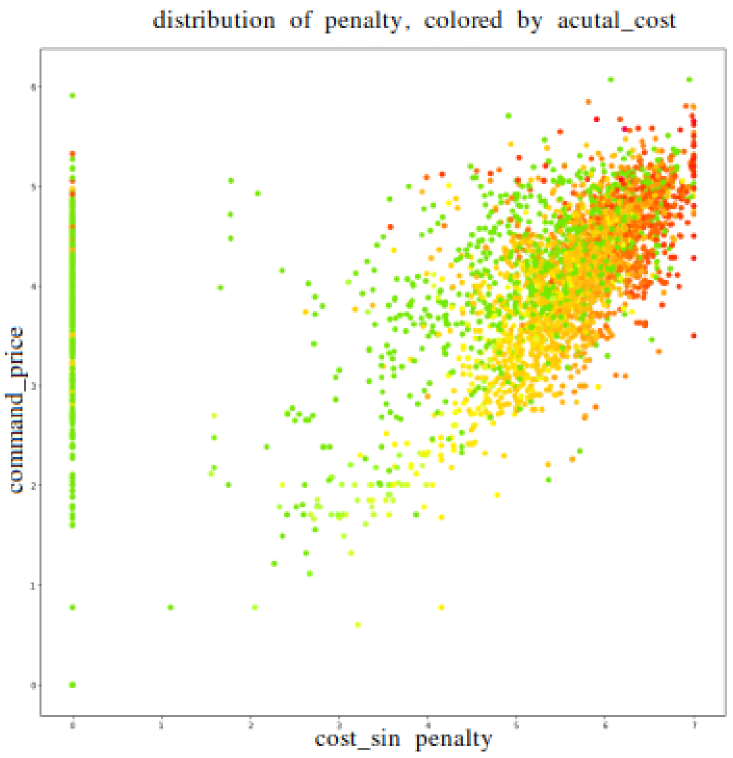


Figure 1.1: " The figure shows the dependence of the parameter price on stock after production vs. penalty value. Colouring is based on price in stock before production (of entered semi-product)." Adapted from [6].

Chapter 2
Theoretical part

2.1 Data preparation

In the beginning, it is necessary to prepare a suitable dataset (ETL [1]) for the training of classifiers. In this dataset, the first row has to contain names of attributes set for training and each additional row contains exactly one instance (manufacture order). The following five files (description from [5]) are needed:

- manufacture_command.csv - contains raw data from Helios database
- material.csv - represents the tree of dependencies of semi-finished products on materials
- slozitost.csv - contains TAC [2] for materials
- stock_load.csv - contains information about movements of the semi-finished products in the warehouse
- stock_price.csv - contains the price of the semi-product.

After adding statistical/cumulative data and collecting only important columns the final dataset which is suitable for training will be created:

- manufacture_command_ex.csv - contains columns needed for training, evaluating and using of the classifiers, the order of the columns can not be changed(see 2.1 for description and desired structure)

[1]Extract-Transform-Load
[2]The time required to perform an operation on one piece of product in minutes.

order	column	description
1	index	number of the row
2	command_price	Order price (amount * manufacture_cost)
3	start_month	Month from date of assignment
4	nrow_TAC	Number of TAC entries needed to create the product (number of steps)
5	sum_TAC	Amount of TAC needed to create the product
6	material_sum	Sum of the materials needed to create the product
7	material_nrow	Number of materials needed to create the product
8	parent_pieces_order_count	Number of parent orders at start_date
9	parent_pieces_order_mean	
10	parent_pieces_order_max	
11	parent_pieces_order_min	
12	parent_depth	Maximum depth of parents in the semi-product tree
13	parent_count	Number of unique parents in the semi-product tree
14	end_stock_price	Reflects the target money, that are blocked in the storage after finishing the manufacture order
15	actual_cost	Actual price
16	ongoing_same_pieces	Sum of products on manufacturing order from ongoing_same_command
17	ongoing_same_command	Number of running manufacturing order on the day of the production order with the same product
18	ongoing_all_command	Number of running manufacturing order on the day of the production order entry
19	ongoing_all_pieces	Sum of pieces from ongoing_all_command
20	cost_sin_2_6	value of the penalty
21	start_date	Actual date of start of manufacture command

Table 2.1: Structure of the prepared dataset (suitable for training). Description is adapted from [5].

2.2 Classifiers preparation

2.2.1 Introduction

Machine learning is a scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. In supervised learning, each example is a pair consisting of an input object and the desired output value. A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. Classification is a supervised learning in which data are used to predict a category. The case where the output is a real value is called a regression.

In spite of the fact that we want to classify the priority (low/mid/high) of new production order which is actually a category, first we need to "predict" the value of the penalty and because of this reason, we will use regression methods. After estimating the value, we will classify the priority into categories based on the given boundaries:

penalty < 250 000 = low priority

250 000 <= penalty < 1 000 000 = medium priority

penalty >= 1 000 000 = high priority

Before we can start with training and evaluating of classifiers we need to create train and test datasets. This is done by posting parameters: start_year_train, end_year_train, start_year_test and end_year_test.

2.2.2 Used methods

These methods are used from [8]. The reason why I used mainly decision tree based methods lies in the previous work of Michal Bouška[6].

1. FastTree (Boosted Trees) Regression
Trains gradient boosted decision trees to fit target values using least-squares.

2. FastTree (Boosted Trees) Tweedie Regression
Trains gradient boosted decision trees to fit target values using a Tweedie loss function. This learner is a generalization of Poisson, compound Poisson, and gamma regression.

3. Fast Forest Regression
Trains a random forest to fit target values using least-squares.

4. Generalized Additive Model(GAM) for Regression
This GAM trainer is implemented using shallow gradient boosted trees to learn nonparametric shape functions

The accuracy of the optimal model for three different companies: "hudba", "pribory" and "letadla" can be seen in the following table 2.2:

	hudba	**letadla**	**pribory**
train years	2014, 2015	2013, 2014	2016
test years	2016, 2017	2015	2017
accuracy	**0.7**	**0.73**	**0.75**

Table 2.2: The accuracy of the optimal model for three different companies.

2.2.3 Evaluation

Evaluation is divided into two parts:

- regression metrics
- percentage of correct predictions (of priority) on the test dataset

For every model we will count the following regression metrics:

- L1 - the absolute loss of the model
- L2 - the squared loss of the model
- Rms - root mean square loss which is the square root of the L2
- RSquared - the coefficient of determination

Then we will classify the priority of the penalty for the test dataset based on real value and based on the predicted value for every instance in the test dataset. From this information, we will determine a percentage of the correct predictions. Based on this percentage is chosen an optimal model but in the export of classifiers are all models including their evaluation.

2.2.4 k-Fold Cross-Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called "k" that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

In our case, there is an option to evaluate the model using cross-validation by posting num_of_folds parameter, if num_of_folds is less or equals 1 then the cross-validation will not be performed.

2.3 Classification

New manufacture commands which priority we want to classify have to be in a CSV [3] file containing the first 19 columns (order 1-19) from the table 2.1. But in this case, they do not have to be in an exact order.

The resultant file is a CSV file which contains two columns: "index" (to identify the manufacture order) and "priority".

[3]Comma Separated Value(s) (database export/import format and file extension)

Chapter 3

Analytical part

3.1 Architecture

Software architecture refers to the high-level structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.

This project implements REST API which is an architecture style for designing loosely coupled applications over HTTP, that is often used in the development of web services. A web service is a software system destined to support interoperable machine-to-machine interaction over a network.

3.2 Requirements

The software requirements are a description of the features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from a client's point of view.

3.2.1 Functional Requirements

Requirements, which are related to functional aspect of software fall into this category. They define functions and functionality within and from the software system.

Functional requirements for this project are:

1. import data from database (Microsoft SQL Server)

2. prepare dataset suitable for training of the classifiers

3. create train and test datasets

4. training of the classifiers to create a classification model
5. evaluation of the trained models
6. choose optimal model
7. export models
8. use model to classify new data

3.2.2 Non-Functional Requirements

Requirements, which are related to the functional aspect of software fall into this category. They define functions and functionality within and from the software system.

Non-Functional requirements for this project are:

1. training of the classifiers must not blocking the application
2. give the user feedback if the request was successful or not
3. implementation will be based on the C# language

4. delete files once they are redundant from the storage (filesystem in our case)

3.3 Technologies

All technologies used to create this project are open-source.

C#[14] is a programming language developed by Microsoft which is designed for Microsoft's .NET Framework. This allows developers to take advantage of all the features offered by the .NET API.

The application was developed using **ASP.Net Core 2.2**[15] which is a free and open-source managed computer software framework for the Windows, Linux, and macOS operating systems. It fully supports C#. The reason for version 2.2 is that ASP.NET Core 2.x can target both .NET Core and .NET Framework. Besides that .NET Core is cross-platform and open-source there are more advantages to use it instead of .Net Framewrok as improved performance, side-by-side versioning and new APIs.

For posting parameters such as "start_year_train", "num_of_folds", etc. was used **JSON**[16] which is a lightweight format for storing and transporting data. The big advantage of it is that JSON is a "self-describing" and easy to understand.

For easier use of this API was used **Swagger**[17] which is tooling ecosystem for developing APIs with the OpenAPI Specification. It gives us option to call web services, shows basic description and input JSON structure. However, swagger cannot generate full documentation for the project. Because of this reason documentation was generated by Doxygen tool.

Doxygen[18] is a tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages including C#. Documentation is generated from the source code (includes xml comments) as html pages where **XML**[19] is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable and **HTML**[20] is the standard markup language for creating web pages and web applications.

Based on personal sympathies, for versioning the code I chose **Git**[21] which is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

In one case (described below) there is a need to call **Python 3** script. "Python is an interpreted, high-level, general-purpose programming language. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python features a comprehensive standard library, and is referred to as "batteries included"."[24]

3.4 Python part

One of the functional requirements is from the imported data prepare dataset suitable for training of classifiers. In the Michal's Bouška work[6] already exists python script "prepare_data.py" which prepares three files : "days.txt", "data_cont.csv" and "manufacture_command_ex.csv" from the imported data (CSV files). First two files are needed for the preparation of the last one, when the "manufacture_command_ex.csv" contains all needed values for training.

I tried to implement similar functionality in C#. However, for creating this dataset the python script uses **Pandas** which is: "a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python."[23]. After reviewing this python script I found Pandas very powerful tool for this job. Unfortunately, I could not find something with similar functionality for C#. Regardless of that, I have implemented preparation of "days.txt" and "data_cont.csv". On the same machine, preparation of these files lasted about

2 minutes in Python and about 16 minutes in C#. For this reason, I decided for external calling of python script mentioned above.

The following figure shows how calling of python works in C#(3.1).

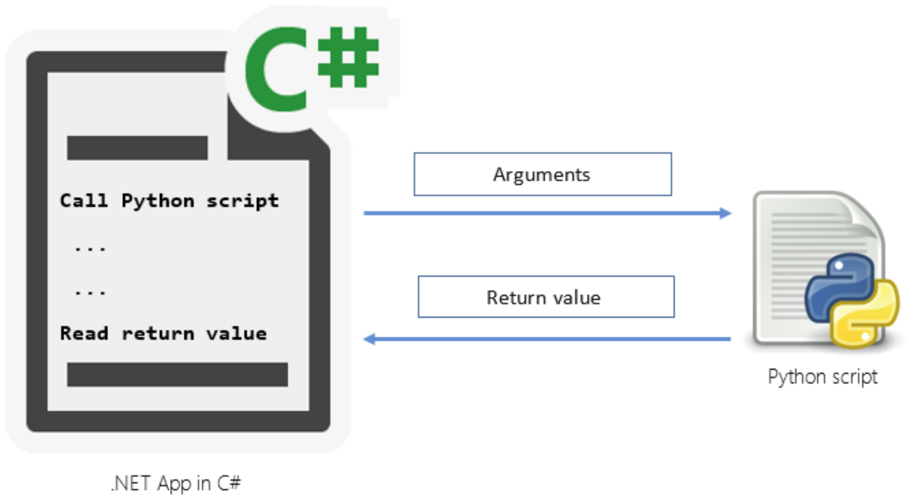


Figure 3.1: C# calls python script. Adapted from[13].

3.5 Use cases

A use case diagram is a dynamic or behaviour diagram in UML[1]. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. For use case diagram see the figure 3.2.

[1]UML is a way of visualizing a software program using a collection of diagrams.

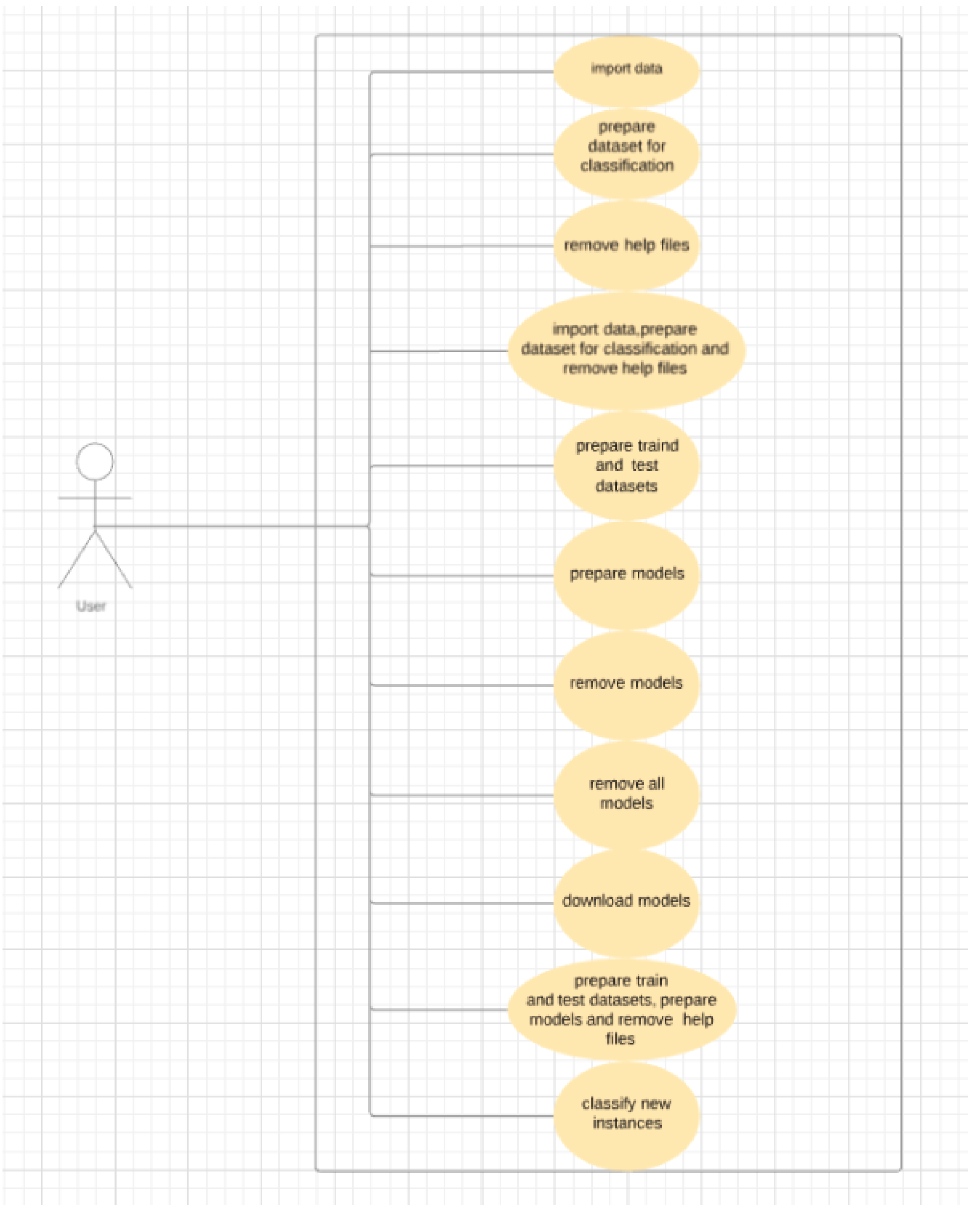

import data
prepare dataset for classification
remove help files
import data,prepare dataset for classification and remove help files
prepare traind and test datasets
prepare models
remove models
remove all models
download models
prepare train and test datasets, prepare models and remove help files
classify new instances
User

Figure 3.2: Use case diagram

Chapter 4

Implementation part

4.1 Structure of the project

This application consists of two projects AssecoWebApp.csproj and AssecoWebAppTestProject.csproj. These projects are in the solution AssecoWebApp.sln which is simply a container for one or more related projects. In the AssecoWebApp.csproj is the implementation of the API and AssecoWebAppTestProject.csproj contains unit tests.

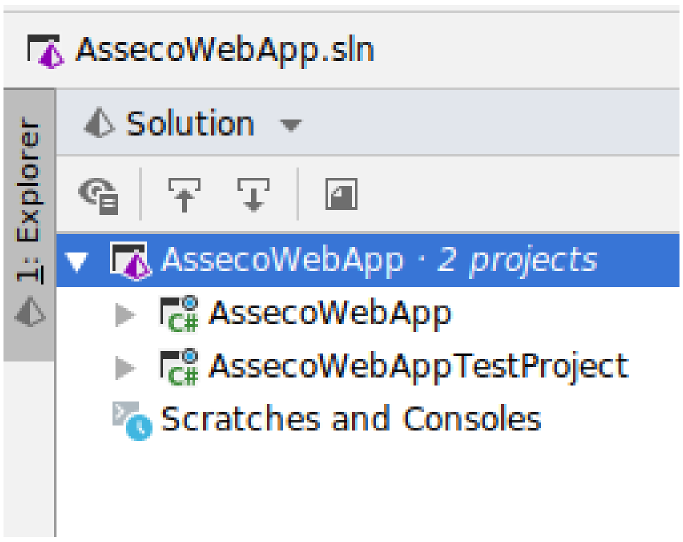
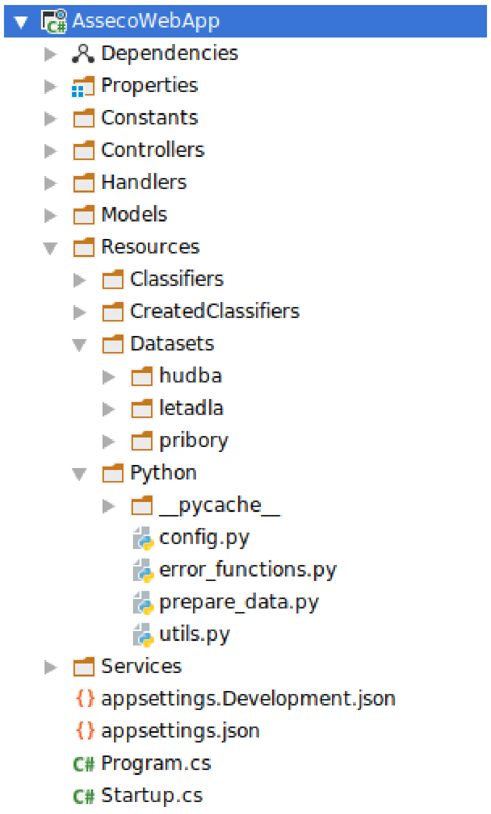


Figure 4.1: Solution's projects

The following text describes implementation part, unit tests are described in the section 5.2 .

Classes in C# have extension ".cs". On the figure 4.2 can be seen two important classes of this application Startup and Program.

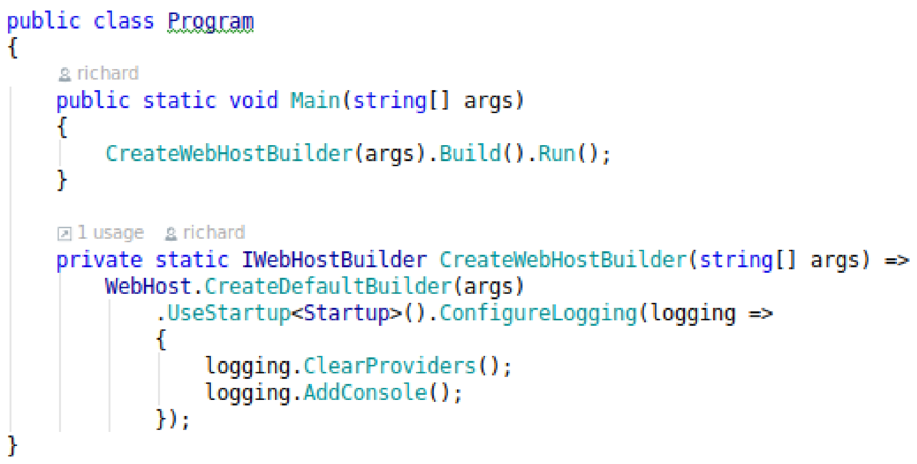


```
▼ AssecoWebApp
  ▶ Dependencies
  ▶ Properties
  ▶ Constants
  ▶ Controllers
  ▶ Handlers
  ▶ Models
  ▼ Resources
    ▶ Classifiers
    ▶ CreatedClassifiers
    ▼ Datasets
      ▶ hudba
      ▶ letadla
      ▶ pribory
    ▼ Python
      ▶ __pycache__
        config.py
        error_functions.py
        prepare_data.py
        utils.py
  ▶ Services
    appsettings.Development.json
    appsettings.json
    Program.cs
    Startup.cs
```

Figure 4.2: Impementation structure

The web application starts executing from the entry point "public static void Main(string[] args)" in the Program class where a host for the web application is created.

The Startup class is specified to the app when the app's host is built. The app's host is built when Build is called on the host builder in the Program class (see the figure 4.3).



```
public class Program
{
    richard
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    1 usage   richard
    private static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>().ConfigureLogging(logging =>
            {
                logging.ClearProviders();
                logging.AddConsole();
            });
}
```

Figure 4.3: Program.cs class

The Startup class:

• Includes a ConfigureServices method to configure the app's services. A service is a reusable component that provides app functionality. Services are configured in ConfigureServices and consumed across the app via dependency injection (DI) or ApplicationServices.

• Includes a Configure method to create the app's request processing pipeline.

ConfigureServices and Configure are called by the runtime when the app starts.

Dependency injection is a software design pattern, which is a technique for achieving Inversion of Control (IoC) between classes and their dependencies.
Inversion of Control is a design principle. It is used to invert different kinds of controls in object-oriented design to achieve loose coupling. Here, controls refer to any additional responsibilities a class has, other than its main responsibility. This include control over the flow of an application, and control over the flow of an object creation or dependent object creation and binding.

You can see on the Figure 4.2 that the application is divided into several folders.
The Controllers folder contains controllers which are classes that handle http requests. They take input values from requests and pass these values to services which do the expected job. After the job is finished, the controller shows the response code and needed information.
The Services folder contains services which are classes doing the business logic of the application and are directly called in controllers.
The Handlers folder contains classes which implement functionality used in multiple service's classes.
Among others classes in the Models folder, there is also the Manufacture class. This class maps the "manufacture_command_ex.csv" (see the table 2.1) as C# class, where column's names in CSV file reflect attributes in the class.

To understand the way I was working with constants in this project, see following arrangements:

• if the constant is used only in one class then it is defined in the beginning of this class

• constants for saving/loading data on/from filesystem are defined in the PathConstants.cs (Constants folder)

• column's names of the manufacuture_command_ex.csv are defined in the ManufactureAttributes.cs (Constants folder)

Resources subfolders:

• Classifiers - helps with training, evaluating of classifiers and defines how the result folder with trained models will look like

• CreatedClassifiers - storage for the trained models

• Datasets - storage for company data

• Python - contains python scripts used in this project, only the script "prepare_data.py" is called directly even though it requires functions from other scripts

4.2 Run the application

To run the application open AssecoWebApp project in console and run the command: "dotnet run". Subsequently, go to the:
"https://localhost:5001/swagger/index.html"
(see the figure 4.4)

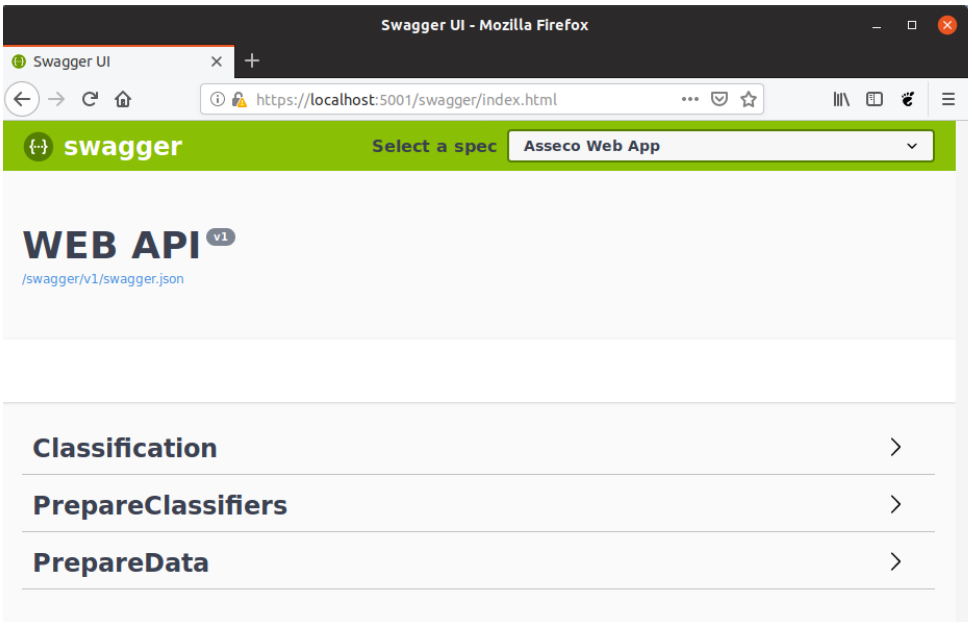


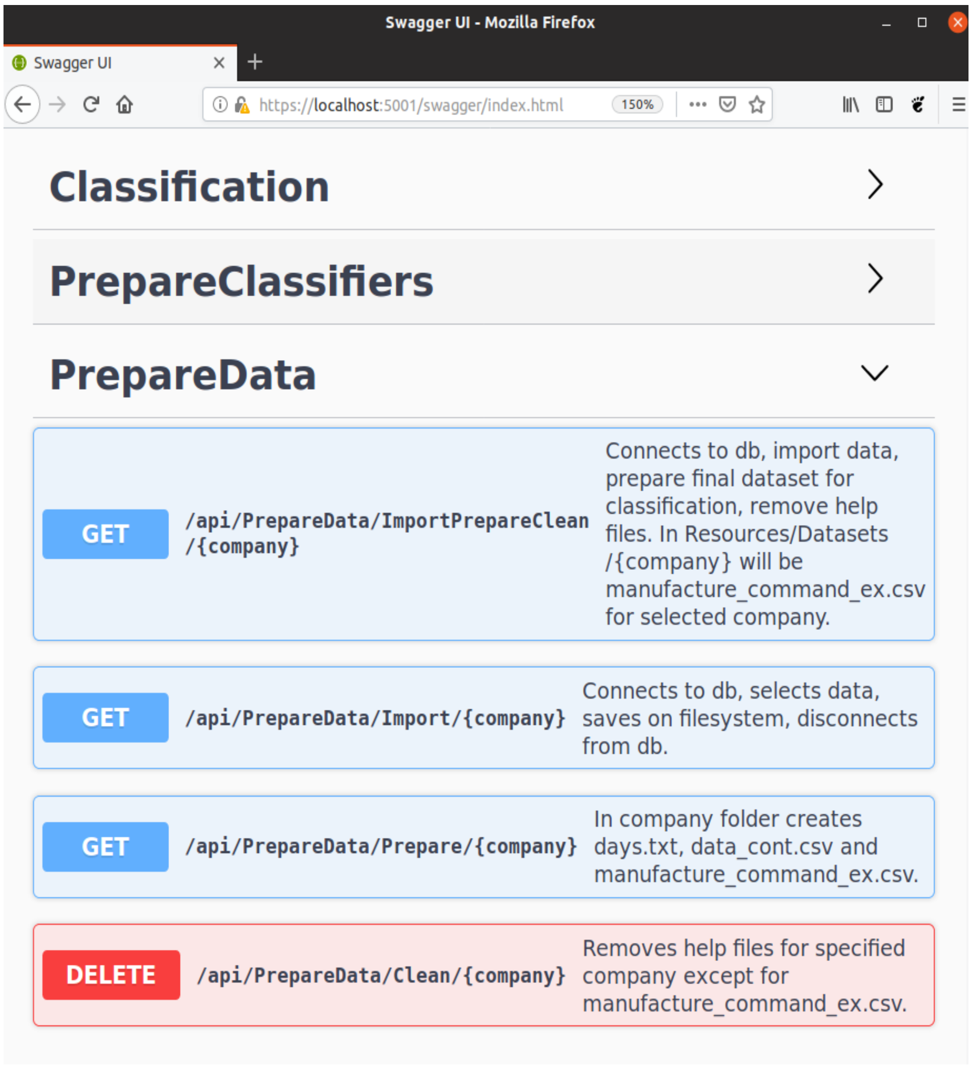
Figure 4.4: Swagger start page

On the bottom of the figure 4.4, there are three sections: Classification, PrepareClaasifiers and PrepareData. Each of them stands for controller class, e.g. Classification stands for the ClassificationController.cs .

4.3 Data preparation

The purpose of this part is preparation of dataset suitable for training of classifiers (manufacture_command_ex.csv). This job is divided into three steps (see the figure 4.5):

1. import data from the database
2. use imported data to create needed dataset
3. remove "help files"[1]


Swagger UI - Mozilla Firefox
Swagger UI
https://localhost:5001/swagger/index.html
150%
Classification
PrepareClassifiers
PrepareData

GET /api/PrepareData/ImportPrepareClean/{company} — Connects to db, import data, prepare final dataset for classification, remove help files. In Resources/Datasets/{company} will be manufacture_command_ex.csv for selected company.

GET /api/PrepareData/Import/{company} — Connects to db, selects data, saves on filesystem, disconnects from db.

GET /api/PrepareData/Prepare/{company} — In company folder creates days.txt, data_cont.csv and manufacture_command_ex.csv.

DELETE /api/PrepareData/Clean/{company} — Removes help files for specified company except for manufacture_command_ex.csv.

Figure 4.5: PrepareDataController's methods

[1]files imported from the db + days.txt, data_cont.csv

However, specific situations may arise. One of them is that a data specialist may want to check help files so it is not advisable to delete them immediately. Another situation that happened in our case, when for the companies "hudba" and "letadla" imported data from the databases were already available, but connection to databases worked no longer. For these reasons there are two possible scenarios. The first is to do the whole job from importing the data to cleaning the help files, the second is to call these web services separately according to your need.

The following subsections describe how to use these web services properly and what are the their prerequisites.

4.3.1 Import data

To import data from the database there are two prerequisites. First is to setup connection string to the database in the file "appsetings.json". This connection string has to be written in the section "ConnectionStrings". The connection string has to have its own unique name, the connection string key which identifies it among the others.

Secondly is that there has to exist a folder where the imported data will be saved. This folder must be placed in the "Resources/Datasets/{folderName}". To make it work properly, the "folderName" has to be the same as the connection string key. In the current state, this works only for the company: "pribory". There is already connection string for the company: "pribory" in the appsetings.json and also in the "Resources/Datasets" exists folder with the name: "pribory".

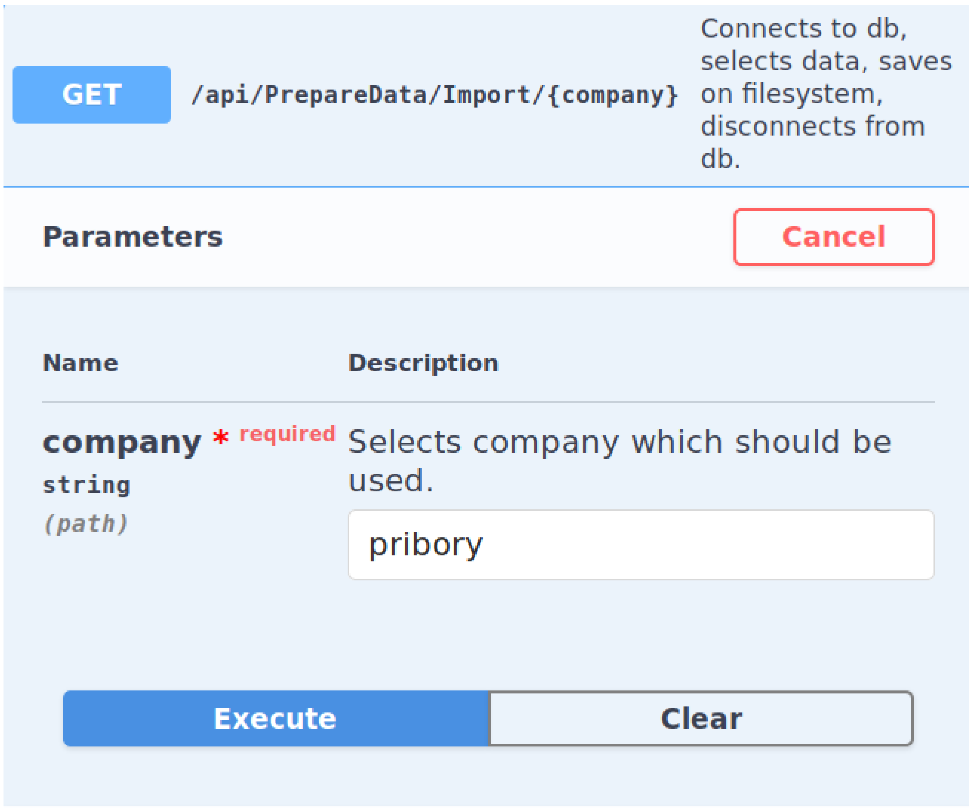
To execute data import, we need to call http GET method on the following route:

"api/PrepareData/Import/{company}"

where parameter company equals to connection string key and folder name described above. This can be done using swagger as follows (see the figure 4.6):

1. click on: GET /api/PrepareData/Import/company
2. click: try it out
3. set parameter company to: pribory
4. click execute

Once the job is finished, the response code should be 201 (see the figure 4.7) likewise the "pribory" folder should contain the following files: "material.csv", "slozitost.csv", "manufacture_command.csv", "stock_price.csv" and "stock_load.csv".



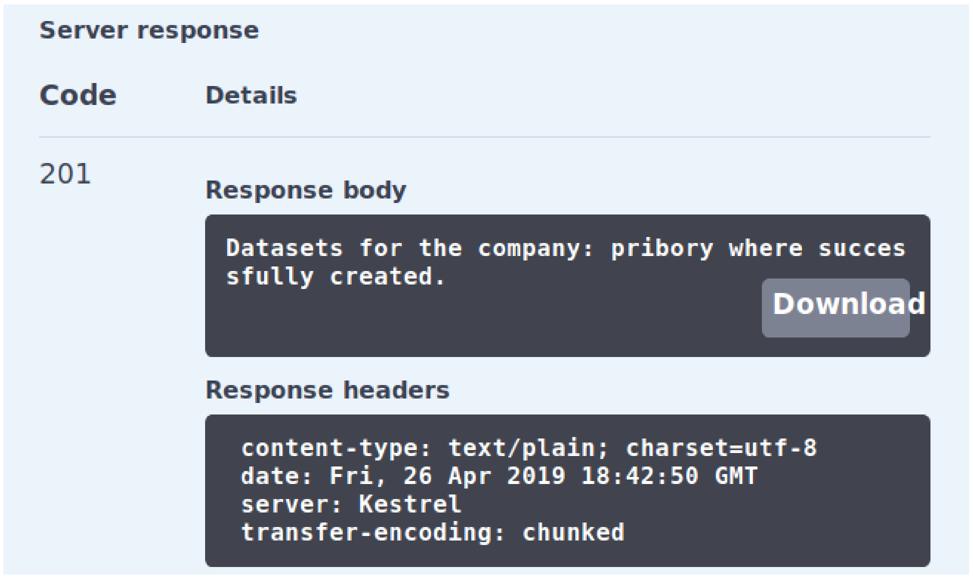
/api/PrepareData/Import/{company}

Connects to db, selects data, saves on filesystem, disconnects from db.

Parameters

Name	Description
company * required string (path)	Selects company which should be used. pribory

Figure 4.6: Import of data from the db.



Server response

Code	Details
201	Response body

```
Datasets for the company: pribory where succes
sfully created.
```

Response headers

```
content-type: text/plain; charset=utf-8
date: Fri, 26 Apr 2019 18:42:50 GMT
server: Kestrel
transfer-encoding: chunked
```

Figure 4.7: Successful response for import of data from the db.

4.3.2 Prepare dataset

The purpose of this part is to create dataset suitable for training of classifiers (manufacture_command_ex.csv). The company folder must exist in the "/Resources/Datasets/" and it has to contain all five needed CSV files (see the section 2.1). This can be done in two ways. The first is to call Import method (see the section 4.3.1). The second is to put files in this folder manually. In the current state, this can be done for the companies "hudba" and "pribory", the needed files are in the folder "NewDbData".

In this case the application calls external python script, for this reason the python 3 and a few of its modules must be installed (see the Appendix C).

To do the job we need to call http GET method on the following route:

"api/PrepareData/Prepare/{company}"

where parameter company equals to company folder (described in the section 4.3.1).

If the job is finished successfully, the response code will be 201 and to the company folder will be added following files: "days.txt", "data_cont.csv" and "manufacture_command_ex.csv".

4.3.3 Remove help files

The purpose of this part is to remove files which are not needed anymore (described in the 4.3). There is only one requirement to do that, in the "Resources/Datasets/" must be company folder from which the files should be removed.

To do so, we need to call http DELETE method on the following route:

"api/PrepareData/Clean/{company}"

where parameter company has same meaning as in the previous two sections.

If the job finished successfully, the response code will be 200 and in the company folder will remain only one file: "manufacture_command_ex.csv".

4.3.4 Import, prepare, clean

Following web service merges work done by previous three sections in the one. Because we are importing data from the database, this works only for the company: "pribory" (same as in the section 4.3.1).

To carry out the job, we need to call http GET method on the following route:

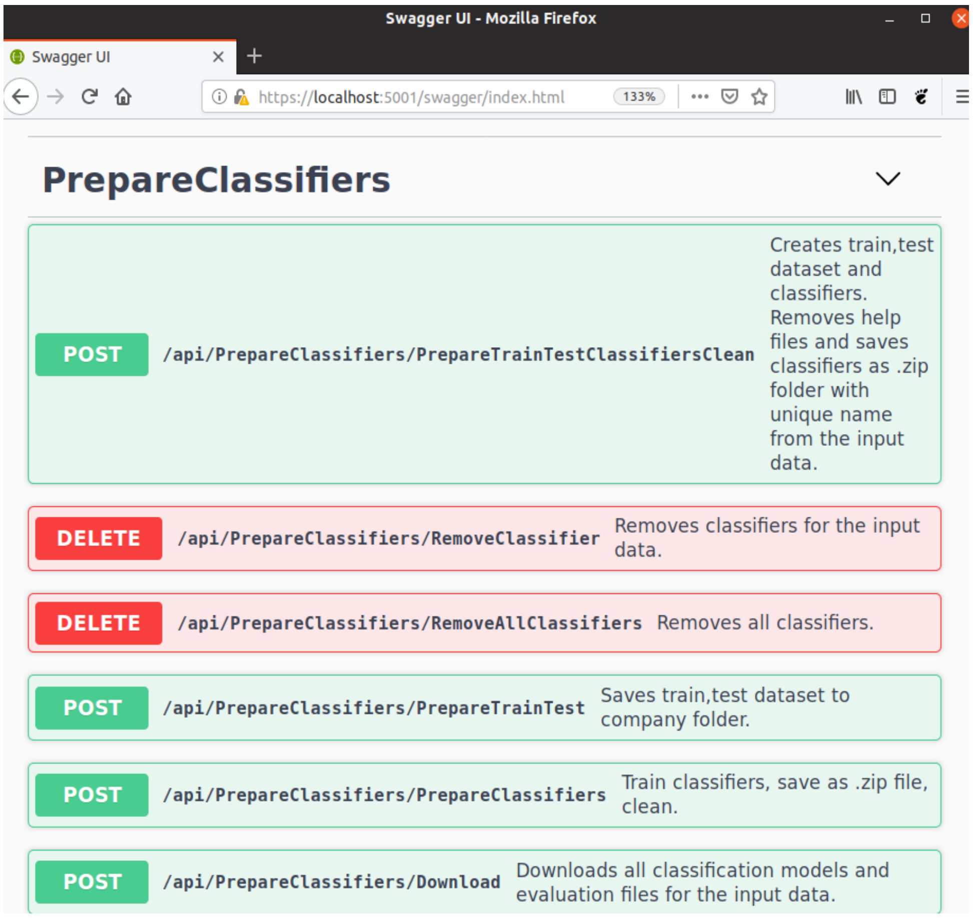
"api/PrepareData/ImportPrepareClean/{company}"

where parameter company has the same meaning as in the previous sections.

If the job finished successfully, the response code will be 201 and in the company folder will be only "manufacture_command_ex.csv" file.

4.4 Training and evaluation of classifiers

This section describes methods of the PrepareClassifiersController (see the figure 4.8). Before the training can start, train and test datasets must be created. Once the training and evaluating is finished, there will be availability to download classifiers or to remove them from the storage. How to do this is described in the subsections below.


Swagger UI - Mozilla Firefox
Swagger UI
https://localhost:5001/swagger/index.html
133%

PrepareClassifiers

POST /api/PrepareClassifiers/PrepareTrainTestClassifiersClean Creates train,test dataset and classifiers. Removes help files and saves classifiers as .zip folder with unique name from the input data.

DELETE /api/PrepareClassifiers/RemoveClassifier Removes classifiers for the input data.

DELETE /api/PrepareClassifiers/RemoveAllClassifiers Removes all classifiers.

POST /api/PrepareClassifiers/PrepareTrainTest Saves train,test dataset to company folder.

POST /api/PrepareClassifiers/PrepareClassifiers Train classifiers, save as .zip file, clean.

POST /api/PrepareClassifiers/Download Downloads all classification models and evaluation files for the input data.

Figure 4.8: PrepareClassifiersController's methods.

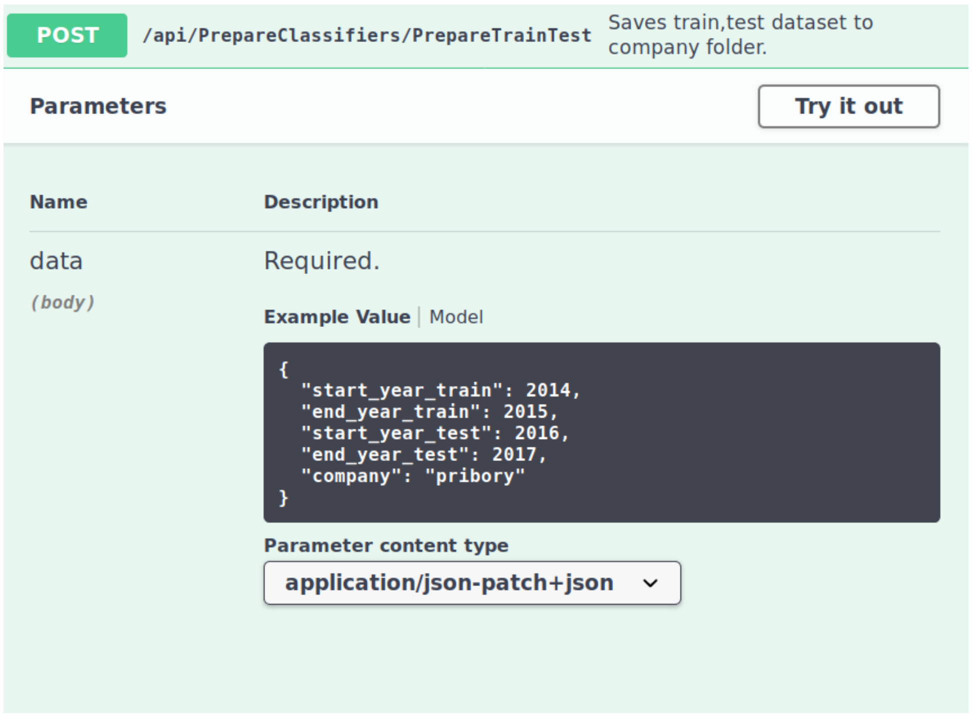
4.4.1 Create train and test datasets

The purpose of this web service is to create train and test datasets. Both of these datasets are subsets of the file manufacture_command_ex.csv, for this reason in the company folder this file must exist.

To do the job we need to call http POST method on the following route:

"api/PrepareClassifiers/PrepareTrainTest"

by the POST method must be send JSON with the name: "data" containing following attributes: "start_year_train","end_year_train", "start_year_test", "end_year_test", "company". See the following figure 4.9 for better understanding.



POST /api/PrepareClassifiers/PrepareTrainTest Saves train,test dataset to company folder.

Parameters

Name	Description
data *(body)*	Required.

Example Value | Model

```
{
  "start_year_train": 2014,
  "end_year_train": 2015,
  "start_year_test": 2016,
  "end_year_test": 2017,
  "company": "pribory"
}
```

Parameter content type: application/json-patch+json

Figure 4.9: Prepare train and test datasets. In this case, train dataset will contain years 2014 and 2015, test dataset will contain years 2016 and 2017 and they will be created from the manufacture_command_ex.csv which is in the Resources/Datasets/pribory.

If the call finished successfully, the response code will be 201 and to the company folder will be added "TrainDataset.csv" and "TestDataset.csv".

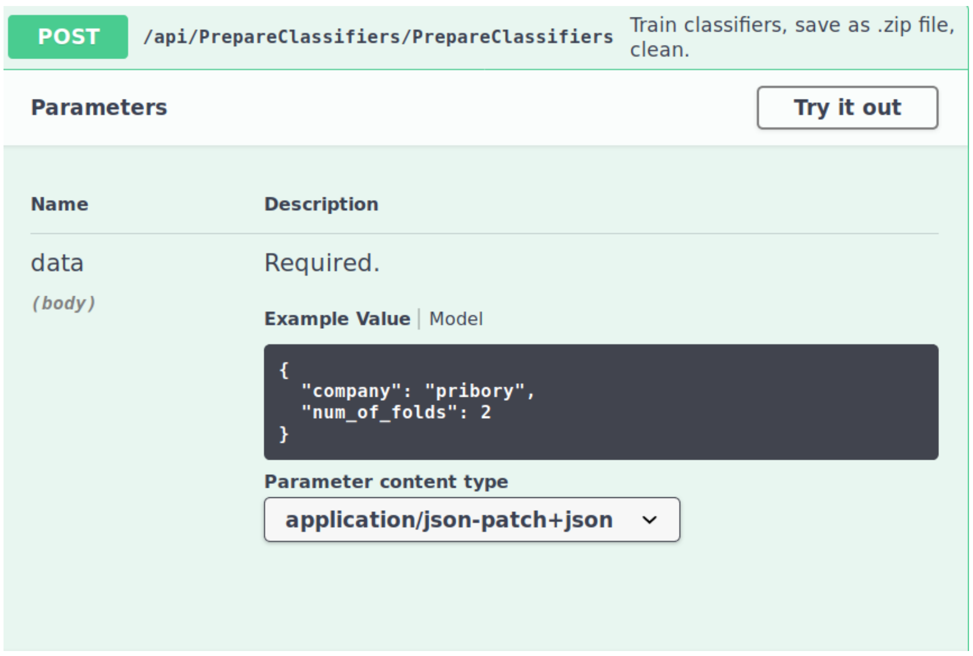
4.4.2 Prepare classifiers

This web service will train classifiers, evaluate them and choose the optimal model. Once the job is finished, all models will be saved as zip file to the folder "Resources/CreatedClassifiers". To carry this out, the must be datasets for training and testing in the company folder (see the section 4.4.1).

To make it happen, we need to call http POST method on the following route:

"api/PrepareClassifiers/PrepareClassifiers"

by the POST method must be send JSON with the name: "data" containing following attributes: "company" and "num_of_folds". See the following figure 4.10 for better understanding.


POST
/api/PrepareClassifiers/PrepareClassifiers
Train classifiers, save as .zip file, clean.
Parameters
Try it out
Name
Description
data
(body)
Required.
Example Value | Model

```
{
  "company": "pribory",
  "num_of_folds": 2
}
```

Parameter content type
application/json-patch+json

Figure 4.10: Prepare classifiers. In this case, for training/testing it expects train and test datasets in the Resources/Datasets/pribory and 2-fold Cross-Validation will be performed.

If the job finished successfully, the response code should be 201. To the "Resources/CreatedClassifiers" will be added zip file (name from the inputted JSON) with all trained models.

4.4.3 Prepare train,test datasets and classifiers

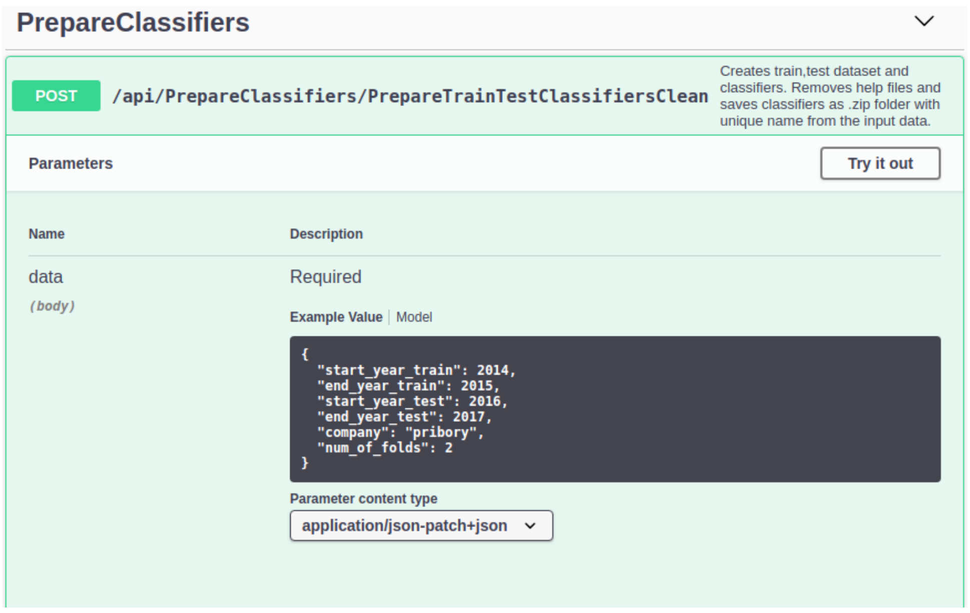
This part combines the work of two previous sections (4.4.1 and 4.4.2) plus it will remove train and test datasets at the end. In this case there is only one prerequisite, the "manufacture_command_ex.csv" should be existent in the company folder.

To make it happen, we need to call http POST method on the following route:

"api/PrepareClassifiers/PrepareTrainTestClassifiersClean"

by the POST method must be send JSON with the name: "data" and with the following attributes: "start_year_train", "end_year_train", "start_year_test", "end_year_test", "company", "num_of_folds".

For example see the following figure:



Creates train,test dataset and classifiers. Removes help files and saves classifiers as .zip folder with unique name from the input data.

```
{
  "start_year_train": 2014,
  "end_year_train": 2015,
  "start_year_test": 2016,
  "end_year_test": 2017,
  "company": "pribory",
  "num_of_folds": 2
}
```

.

Figure 4.11: Creates train and test datasets for training of classifiers, train classifiers and at the end removes train,test datasets. Explanation to the attributes can be seen in the figures 4.9 and 4.10

Once the job finished successfully, the response code will be 200. To the "Resources/CreatedClassifiers" will be added zip file (name from the inputted JSON) with all trained models and from the company folder train and test datasets will be removed.

4.4.4 Download trained models

This part describes how to download trained models. The result of this is zip file which contains all trained models, evaluation and optimal model.

To execute this we need to call http POST method on the following route:

"api/PrepareClassifiers/Download"

by the POST method must be send JSON with the name: "data". The

JSON attributes must be the same as in the training part, that means if for training was used web service from the section 4.4.2 then the exactly same JSON has to be send for downloading and so on for the web service from the section 4.4.3 .

4.4.5 Remove trained models from the storage

In this section two web services will be described, the first will remove all files with trained models from the folder "Resources/CreatedModels". The second will remove zip file with models only for the inputted JSON which identifies it.

To use the first option, call http DELETE method(without parameters) on the following route:

"/api/PrepareClassifiers/RemoveAllClassifiers"

Once the job is finished successfully, the response code will be 200 and "Resources/CreatedModels" will be empty.

To use the second option, call http DELETE method on the following route:

"/api/PrepareClassifiers/RemoveAllClassifiers"

by the POST method must be send JSON with the name: "data". For JSON attributes applies the same rules as in the section 4.4.4 .

Once the job is finished successfully the response code will be 200 and a zip file with models will be removed from the "Resources/CreatedModels".

4.5 Use of trained model

This section describes how to use downloaded models to classify new data. For that the ClassificationController class will be used.

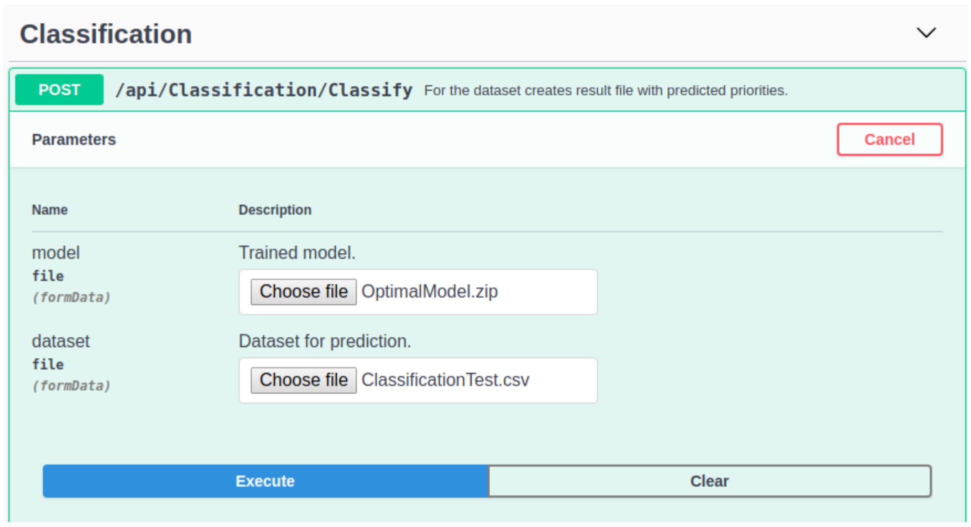
In this part there is only one web service which will carry out the job. To do so, you need to call http POST method on the following route:

"/api/Classification/Classify"

by this method, two parameters must be send, the first has name: "model" and it has to be one of the downloaded models in zip format. The second has name: "dataset" and it has to be a CSV file which should be classified (see the section 2.3 for the description).

To try this out, you can use as a model "OptimalModel.zip" from the section 4.4.4 (or any other). As the dataset which should be classified you can use "ClassificationTest.csv" which is in the "Resources/Datasets". See the figure 4.12 for better understanding.

After execution, a result file with the predicted priorities should be produced. To download it click on the "Download File" (see the figure 4.13).

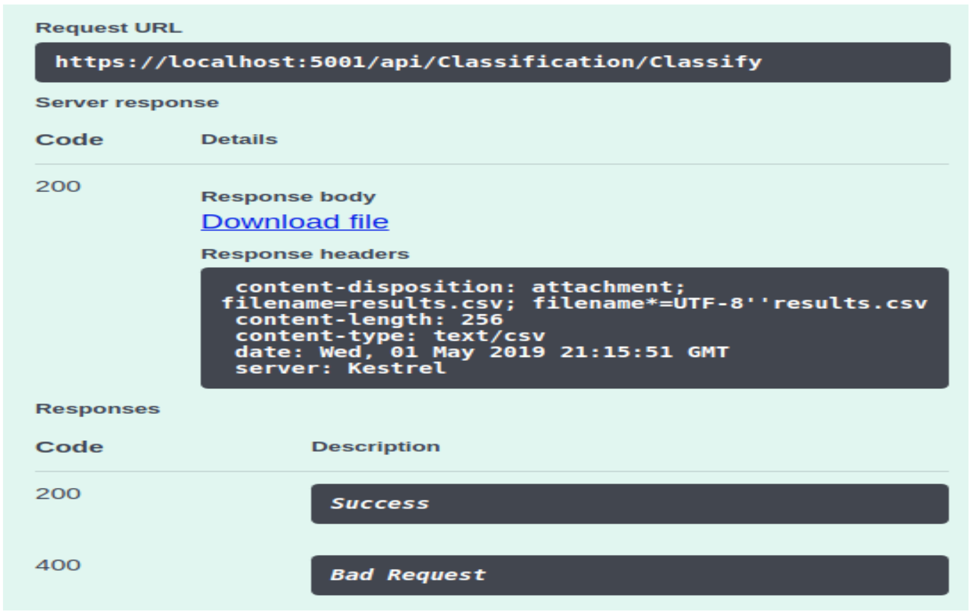


/api/Classification/Classify For the dataset creates result file with predicted priorities.

Parameters

Name	Description
model file (formData)	Trained model. Choose file OptimalModel.zip
dataset file (formData)	Dataset for prediction. Choose file ClassificationTest.csv

Figure 4.12: Classification of new data.



Request URL

```
https://localhost:5001/api/Classification/Classify
```

Server response

Code	Details
200	Response body: Download file

Response headers

```
 content-disposition: attachment;
filename=results.csv; filename*=UTF-8''results.csv
 content-length: 256
 content-type: text/csv
 date: Wed, 01 May 2019 21:15:51 GMT
 server: Kestrel
```

Responses

Code	Description
200	*Success*
400	*Bad Request*

Figure 4.13: Classification's result.

Chapter 5
Testing

5.1 Manual testing

In this section, I have tested the required functionality manually. For every web service which was tested I asked the following questions:

1. Does the web service work as expected with correct input given?

2. Is there a suitable feedback if the input is not correct?

The results can be see in the following table 5.1 :

row	method	route	Q1	Q2
1	GET	/api/PrepareData/ImportPrepareClean/{company}	yes	yes
2	GET	/api/PrepareData/Import/{company}	yes	yes
3	GET	/api/PrepareData/Prepare/{company}	yes	yes
4	DELETE	/api/PrepareData/Clean/{company}	yes	yes
5	POST	/api/PrepareClassifiers/PrepareTrainTestClassifiersClean	yes	yes
6	DELETE	/api/PrepareClassifiers/RemoveClassifier	yes	-
7	DELETE	/api/PrepareClassifiers/RemoveAllClassifiers	yes	-
8	POST	/api/PrepareClassifiers/PrepareTrainTest	yes	yes
9	POST	/api/PrepareClassifiers/PrepareClassifiers	yes	yes
10	POST	/api/PrepareClassifiers/Download	yes	yes
11	POST	/api/Classification/Classify	yes	yes

Table 5.1: Manual testing results.

There are a few notes to the table above:

1. In the current state, web services can be clearly identified. For this reason post/delete parameters are not mentioned in the table.

2. In the row 6, the web service checks if a file exists and if it does then removes it, there is no bad input here. For this reason, the second question was not answered.

3. In the row 7, there is no input at all. For this reason, the second question was not answered.

5.2 Unit testing

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. In unit testing we want to test methods of one class in isolation. But classes are not isolated. They are using services and methods from other classes. So in that situation, we mock the services and methods from other classes and simulate the real behaviour of them using some mocking framework.

Writing of unit tests was carried out by Xunit which is an open-source unit testing tool for the .NET framework. As the Mocking framework I have used Moq which is mocking library for .NET.

Unit tests can be found in the AssecoWebAppTestProject. To run them, open AssecoWebAppTestProject in console and run the command: "dotnet test". There are 20 unit tests and all should pass.

Unit tests were written for public methods of Controllers, Services and Handlers. The following table 5.2 shows the coverage of that:

name	total	tested
ClassificationHandler	3	3
CsvHandler	2	1
DataTableHandler	2	2
ManufactureHandler	2	1
ClassificationController	1	0
PrepareClassifiersController	6	5
ClassificationService	1	1
CleanService	4	3
PrepareClassifiersService	1	1
PrepareTrainTestDatasetService	1	1
ImportDatasetService	1	0
PrepareClassificationDatasetService	1	0

Table 5.2: Table showing coverage of tested methods.

I believe that unit tests should be fast, otherwise it could lead to be skipped by developers. Importing of data and preparation of the dataset suitable for training of classifiers takes a lot of time, for this reason unit tests were not written for this part.

Chapter 6

Conclusion

During this work, I became acquainted with the functioning of ERP systems, two already existing projects that served as a decision support tool for production planner. I also learned about machine learning, cross validation, decision tree method and a few other classification and regression methods. I got acquainted with C# language and ASP.Net Core web framework. Based on the acquired knowledge and instructions for this project, I analysed the problem. Subsequently, I implemented functional REST API which allows importing data from database, prepare a dataset suitable for training of classifiers, create training and testing datasets, train classifiers and then evaluate them, select an optimal model, export models and then use the model to classify new data. I also created documentation for the code. At the end, I tested the required functionality manually and for suitable parts of code I have also written unit tests. I see the application as fully functional and suitable for use.

Additionally, I have to mention one requirement which I failed to fully meet. Although the application is able to create a dataset from imported data which is suitable for training of classifiers for preparation of this dataset there is called python script which was not done by me.

Chapter 7
Further Work

The application is functional and ready to use, yet there are few things to be improved in near future:

1. removing python dependency in data preparation part
2. providing wider range of tests, e.g. integration tests
3. using database as storage instead of filesystem

Bibliography

[1] Marianne Bradford: *Modern ERP: Select, Implement, and Use Today &#39;s Advanced Business Systems, 2008*

[2] Kamel Nidal, Aamir Saeed Malik: *EEG/ERP Analysis: Methods and Applications, 2014*

[3] *The definition of 'Financial forecast'*[online, 2018-4-15] from `https://www.syndicateroom.com/learn/glossary/financial-forecast`

[4] *Financial Forecasting*[online, 2018-4-15] from `https://corporatefinanceinstitute.com/resources/knowledge/modeling/financial-forecasting-guide/`

[5] Victoria Eykhmann: *Forecasting in Finance (ERP)* [online, 2018-4-15] from `https://dspace.cvut.cz/handle/10467/76448`, Bachelor Project, Open Informatics, Czech Technical University in Prague

[6] Michal Bouška: *EclubAsseco Predikce* [online, 2018-4-15] from `https://www.overleaf.com/14980802pxsxccmdyhth#/56877630/`

[7] *What is Machine Learning? A definition*[online, 2018-4-15] from `https://www.expertsystem.com/machine-learning-definition/`

[8] Microsoft: *ML.NET*[online, 2018-4-15] from `https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet`

[9] Wikipedia contributors: *Decision tree learning*[online, 2018-4-15] from `https://en.wikipedia.org/wiki/Decision_tree_learning`

[10] *Difference Between Classification and Regression in Machine Learning*[online, 2018-4-15] from `https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/`

[11] Microsoft: *ML.NET API reference*[online, 2018-4-15] from `https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.data.regressionmetrics?view=ml-dotnet`

[12] Todd Fredrich, Pearson eCollege: *Intro to REST*[online, 2018-4-15] from `https://www.restapitutorial.com/lessons/whatisrest.html`

[13] *Microsoft Developer Network Samples*[online, 2018-4-15] from `http://microsoftdevelopernetworksamples.blogspot.com/2017/03/windowsdesktopc-and-python-interprocess.html`

[14] Wikipedia contributors: *C Sharp (programming language)*[online, 2018-4-15] from `https://en.wikipedia.org/wiki/C_Sharp_(programming_language)`

[15] Microsoft: *ASP.NET Core Documentation*[online, 2018-4-15] from `https://docs.microsoft.com/en-us/aspnet/?view=aspnetcore-2.2#pivot=core)`

[16] *Introducing JSON*[online, 2018-4-15] from `https://www.json.org/`

[17] SmartBear Software: *Swagger*[online, 2018-4-15] from `https://swagger.io/`

[18] Dimitri van Heesch: *Doxygen*[online, 2018-4-15] from `http://www.doxygen.nl/`

[19] *Introduction to XML*[online, 2018-4-15] from `https://www.w3schools.com/xml/xml_whatis.asp`

[20] *HTML5 Tutorial*[online, 2018-4-15] from `https://www.w3schools.com/html/`

[21] Wikipedia contributors: *Git*[online, 2018-4-15] from `https://en.wikipedia.org/wiki/Git`

[22] *Python 3.x documentation*[online, 2018-5-2] from `https://docs.python.org/3/`

[23] AQR Capital Management, LLC, Lambda Foundry, Inc. and PyData Development Team: *Python Data Analysis Library*[online, 2018-5-2] from `https://pandas.pydata.org/`

[24] Wikipedia contributors: *Python (programming language)*[online, 2018-5-2] from `https://en.wikipedia.org/wiki/Python_(programming_language)`

[25] Wikipedia contributors: *xUnit.net)*[online, 2018-5-2] from `https://en.wikipedia.org/wiki/XUnit.net`

Appendix A

List of abbreviations

ERP - Enterprise Resource Planning

API - Application Programming Interface

ETL - Extract-Transform-Load

REST - Representational State Transfer

HTTP - Hypertext Transfer Protocol

GAM - Generalized Additive Model

UML - Unified Modeling Language

JSON - JavaScript Object Notation

HTML - Hypertext Markup Language

XML - Extensible Markup Language

PANDAS - Python Data Analysis Library

DB - Database

DI - Dependency Injection

IoC - Inversion of Control

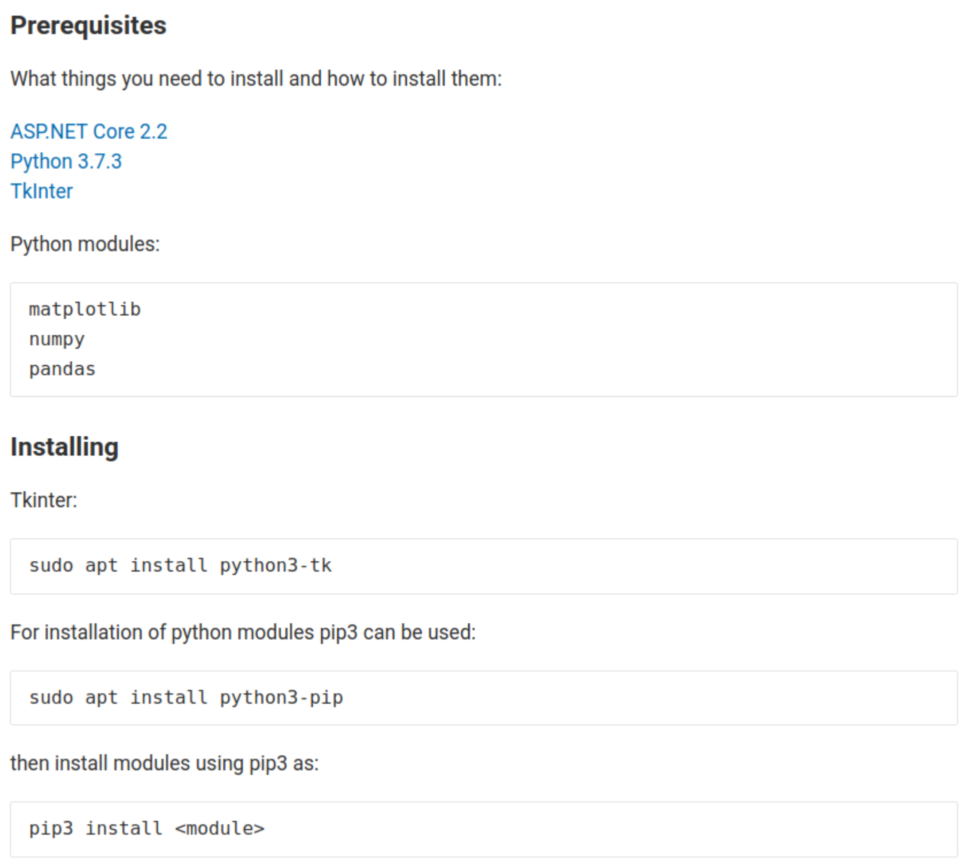
APP - Application

Appendix B
Content of the attached CD

- code
  - AssecoWebApp
  - AssecoWebAppTestProject
  - Documentation
  - NewDbData
  - AssecoWebApp.sln
  - README.md
- text
  - source
  - F3-BP-2019-Sadlon-Richard-Thesis.pdf

Appendix C

Setup of the project



Prerequisites

What things you need to install and how to install them:

ASP.NET Core 2.2
Python 3.7.3
TkInter

Python modules:

```
matplotlib
numpy
pandas
```

Installing

Tkinter:

```
sudo apt install python3-tk
```

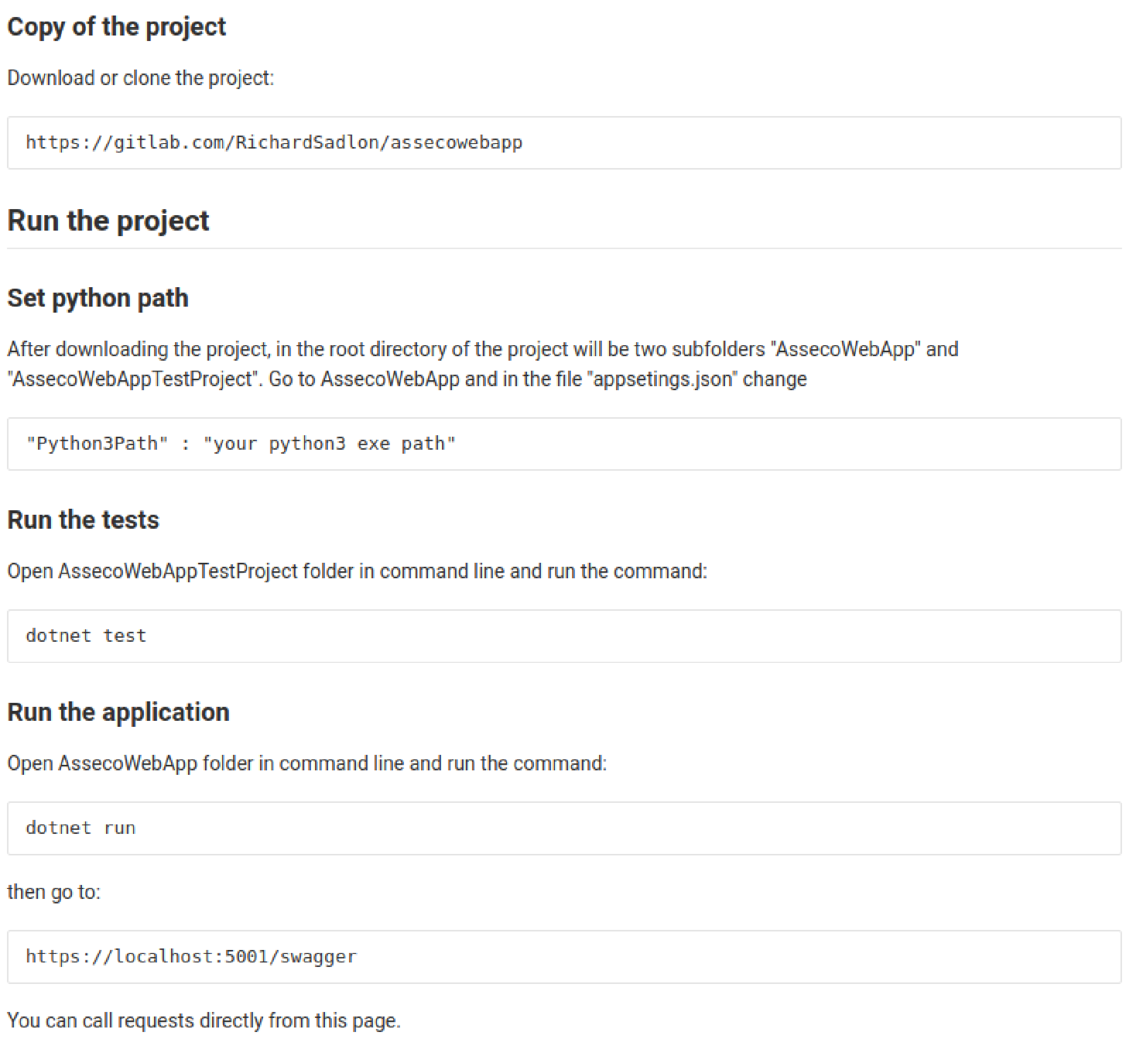
For installation of python modules pip3 can be used:

```
sudo apt install python3-pip
```

then install modules using pip3 as:

```
pip3 install <module>
```

Figure C.1: Setup of a project part 1.



Copy of the project

Download or clone the project:

```
https://gitlab.com/RichardSadlon/assecowebapp
```

Run the project

Set python path

After downloading the project, in the root directory of the project will be two subfolders "AssecoWebApp" and "AssecoWebAppTestProject". Go to AssecoWebApp and in the file "appsetings.json" change

```
"Python3Path" : "your python3 exe path"
```

Run the tests

Open AssecoWebAppTestProject folder in command line and run the command:

```
dotnet test
```

Run the application

Open AssecoWebApp folder in command line and run the command:

```
dotnet run
```

then go to:

```
https://localhost:5001/swagger
```

You can call requests directly from this page.

Figure C.2: Setup of a project part 2.