

UNIVERSITY OF TARTU
FACULTY OF SOCIAL SCIENCES

NARVA COLLEGE
INFORMATION TECHNOLOGY SYSTEMS DEVELOPMENT

Maksim Vassiljev
DEVELOPMENT OF A WEB APPLICATION API FOR ADJUSTMENT
OF SMALL TRADING COMPANY WORKING PROCESSES

Diploma thesis

Supervisor: Assistant Andre Sääsk
Co-supervisor: Assistant Aleksandr Sokhin

NARVA 2018

Olen koostanud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, põhimõttelised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

...../töö autori allkiri/

CONTENTS

CONTENTS.....	3
TERMS AND ABBREVIATIONS	6
REST	6
AJAX	6
JSON	6
PDO.....	6
Web application	6
Adjustment	6
Access token.....	7
API	7
IDE	7
HTTP.....	7
HTML	7
CSS.....	7
URI.....	7
GUID.....	7
INTRODUCTION	8
The problem	8
The solution.....	9
Aim.....	9
Tasks	10
Outline.....	10
1 CHOSEN TECHNOLOGIES AND SIMILAR SOLUTIONS	11
1.1 Chosen technologies	11
1.1.1 PhpStorm.....	11

1.1.2	XAMPP	11
1.1.3	PDO	11
1.2	Previous and similar solutions	12
1.2.1	Mobi-C	12
1.2.2	Inventory, Purchase, Sales Order	13
1.2.3	Stock manager	13
1.2.4	Conclusion	15
2	DESIGN AND ARCHITECHTURE OF API	16
2.1	Functional requirements	16
2.2	Non-functional requirements	16
2.3	Context level 0 diagram	16
2.4	Database development	18
2.5	API kernel development	20
2.5.1	Database connector class	20
2.5.2	Container class	20
2.5.3	Helper classes	20
2.5.4	Request class	21
2.5.5	Token class	25
2.5.6	Language translation class	25
2.5.7	Validator class	25
2.6	API development	25
2.6.1	API configuration	25
2.6.2	API controllers	25
2.6.3	Translation	26
2.6.4	ApiRoot	26
2.6.5	AppController	26

2.6.6	Index.php.....	27
2.7	Client application front-end development.....	27
2.8	API features.....	28
2.8.1	User identification.....	28
2.8.2	Token based authentication.....	28
2.8.3	User role permissions.....	29
3	RESULT OF DEVELOPMENT.....	31
3.1	Get identifier by sending device GUID.....	31
3.2	Login for registered user	31
3.3	Token refresh for online user	32
3.4	Error message if token has expired	33
	CONCLUSION.....	34
	RESÜMEE.....	35
	REFERENCES	37
	APPENDICES	38
	The source code.....	38
	Licence	38

TERMS AND ABBREVIATIONS

REST

Representational State Transfer is an architectural style that defines a set of constraints and properties based on HTTP. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. (REST ... 2018)

AJAX

Asynchronous JavaScript and XML. AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page. (Asynchronous JavaScript ... 2018)

JSON

JSON (JavaScript Object Notation) is a text syntax that facilitates structured data interchange between all programming languages. JSON is a syntax of braces, brackets, colons, and commas that is useful in many contexts, profiles, and applications. (JavaScript Object Notation ... 2017)

PDO

The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. PDO provides a data-access abstraction layer, which means that, regardless of which database you are using, you use the same functions to issue queries and fetch data. (PDO ... n.d.)

Web application

Web application – is an application, which is initiated by user in web browser. Application can run one or more websites, which communicate to one another and may exchange data or start processes. (Kemp; Appelquist; Malhotra; Raman 2010)

Adjustment

Adjustment is a slight change made to something to make it fit, work better, or be more suitable, or the act of making such a change (Adjustment ... n.d.)

Access token

In computer systems, an access token contains the security credentials for a login session and identifies the user, the user's groups, the user's privileges, and, in some cases, a particular application. (Access token ... 2017)

API

Application Programming Interface

IDE

Integrated Development Environment

HTTP

Hyper Text Transfer Protocol

HTML

Hyper Text Markup Language

CSS

Cascading Style Sheets

URI

Universal Resource Identifier

GUID

Globally Unique Identifier

INTRODUCTION

Nowadays trading companies transfer, process and save a huge amount of data during their work. Employees of a certain department may have a common information system, however in order to communicate to a colleague from the different department they may have to use a telephone, an email, paper or arrange a face-to-face meeting. Each communication method has its own pros and cons, but progress does not stand still. Most employees have personal computers or smartphones at work, which gives an opportunity to build a common information system for the whole company or its several departments and exchange data in real time.

The problem

Two out of every three employees believes that the flow of communication between departments within their organization is poor. Inevitably, this results in a reduction in the quality of the products and services provided by the organization. (Katcher, Bruce L. 2018)

There are several reasons why communication between colleagues may not be effective:

1. Personal Conflict Between Department Managers

When department heads do not speak to each other, it makes it very difficult for others below them to communicate effectively.

2. Communication Can Be Time Consuming

When the pressure for speed and productivity is high, employees do not bother to take the time to share important information.

3. Communication is Not Part of the Standard Operating Procedures

Documented procedures often leave out the critically important step of communicating with other departments.

4. Physical Separation

It is difficult for departments to communicate effectively with each other when they are located on different floors or buildings.

5. Stereotyping/Finger Pointing

Most organizations have their warring Hatfields and McCoys¹. In manufacturing, it is typically Sales versus Production. In publishing, it is Editorial versus Sales. In Education, it is teachers versus the administration. Each side stereotypes the other as being insensitive to the needs of other departments and customers. (Katcher, Bruce L. 2018)

In addition to the list above, for example, email communication is not as fast as telephone, however there may be spelling mistakes while using telephone. Face to face meetings may take too long to reach to the meeting point. Employees may lose papers with task lists or products to proceed, so they have to re-request for another one. Higher management may need some specific statistic data about operations, but it may take some time to collect required data. Status of order is not always available for all involved roles. This demands additional time for order details confirmation. There already exist software solutions, which can negotiate most of the problems given above, but they are designed for large trading companies and may have excessive amount of features or be relatively expensive.

The solution

Development of the platform-independent system allows multiple departments to use convenient ways to cooperate in real time, or when available, while working on their own tasks. This also gives an opportunity for employees to use working time more effectively and avoid misunderstanding in some cases. In addition, it gives more possibilities for management representatives to deal with management operations, track some working processes, collect statistic information and check statuses of specific orders or operations.

Aim

The aim of this thesis is to develop a back-end part of the web application, which would allow users of different roles access to it from desktop web browser, mobile web browser or mobile application. The application would have basis for user request processing, also user login, user profile and user registration functions as well as basis for future development and further features implementation.

¹ Meaning of two belligerent parties

Tasks

To reach aim of this thesis, author has to complete following tasks:

- Choose appropriate tools and technologies to develop and run API
- Develop a database for API
- Develop API kernel containing classes such as db connector, token, request
- Develop API application, containing controllers, configuration, error messages
- Create simple web forms for demonstration of user request and API response
- Create diagram of client and API interaction

The author also provides a very minimalistic front-end interface to show certain functionality of his work, as developing a full-featured front-end of the application was not the purpose of this thesis. Desktop browser client was chosen to demonstrate results of user requests, as it is easier to set up.

Outline

Section 1 describes similar solutions and also tools and technologies chosen by the author.

Section 2 describes the application development process.

Section 3 demonstrates the results of development.

1 CHOSEN TECHNOLOGIES AND SIMILAR SOLUTIONS

1.1 Chosen technologies

1.1.1 PhpStorm

PhpStorm is a PHP IDE that actually ‘gets’ your code. It supports PHP 5.3/5.4/5.5/5.6/7.0/7.1/7.2, provides on-the-fly error prevention, best autocomplete & code refactoring, zero configuration debugging, and an extended HTML, CSS, and JavaScript editor. (Jetbrains...2018)

It was chosen by the author because he has experience working in this IDE and it has many useful features.

1.1.2 XAMPP

XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use. (Apache Friends...2018)

XAMPP also provides *PhpMyadmin* which is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. (PhpMyAdmin...2018)

XAMPP package was chosen by the author because it has all necessary services for development of PHP application and does not require advanced knowledge about Apache web server and database server configuration.

1.1.3 PDO

The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. PDO provides a data-access abstraction layer, which means that, regardless of which database you are using, you use the same functions to issue queries and fetch data. (The PHP Group n.d.)

PDO is already build-in PHP 7 and does not require additional installation.

PDO extension was chosen by the author because it simplifies future change of database management system if needed.

1.2 Previous and similar solutions

1.2.1 Mobi-C

Mobi-c² is enterprise solution for mobile merchandising, bookkeeping, warehouse checking, and trade agent organizer. It supports integration with 1C³ bookkeeping software program on desktop PC or synchronization between mobile device and cloud server. The application allows administrator to see information about the company sales agent's location via GPS. Mobile application is available for Android⁴ devices on Google Play⁵ market. Server side of the system is required to install for administrator role (**Figure 1**).

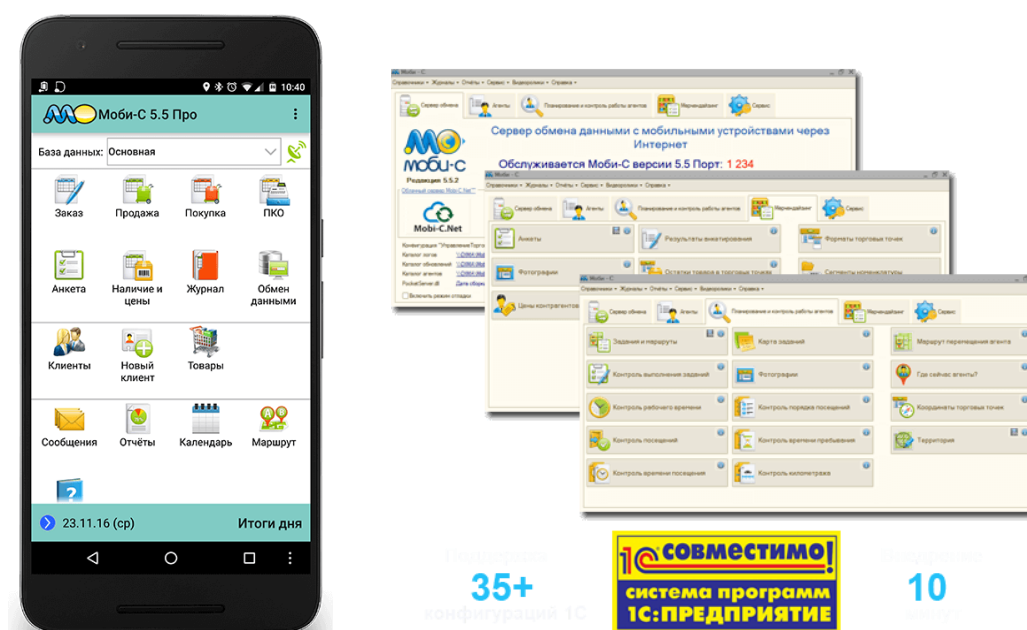


Figure 1. Mobi-c mobile interface (left) and Mobi-c desktop application interface (right).
(Source: author)

² <https://mobi-c.ru/>

³ <http://1c.ru/eng/title.htm>

⁴ <https://www.android.com/>

⁵ <https://play.google.com/>

1.2.2 Inventory, Purchase, Sales Order

Inventory, Purchase, Sales Order⁶ is enterprise solution for sales and stock management. It is designed for only one user role – retailer. The application allows user to keep detailed information of inventory on his mobile device and update stock movement in real time. Various reports can be generated to meet users reporting needs. The alerting system can notify user if product quantity in the inventory falls below set value (**Figure 2**).

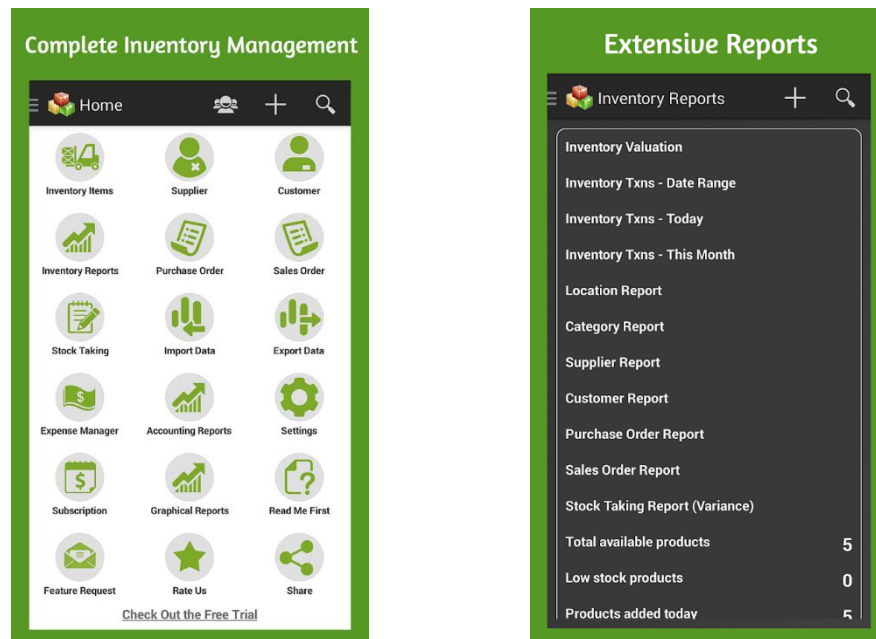


Figure 2. *Inventory, Purchase, Sales Order start menu (left) and report page (right).*
(Source: author)

1.2.3 Stock manager

Stock manager⁷ is advanced inventory and sales management mobile application within a company. It supports two user roles – administrator and user. The application allows synchronizing data among multiple users. In addition, it has internal messaging function. However, the application has short variety of report generation and provides statistical data only about orders and sold products. Unfortunately, the application lacks input

⁶ <https://play.google.com/store/apps/details?id=in.billionhands.instantinventory>

⁷ <https://play.google.com/store/apps/details?id=amalgame.emanager>

data validity check as, for example, it allows user to save character value in telephone number field. The application interface is displayed on **Figure 3**.

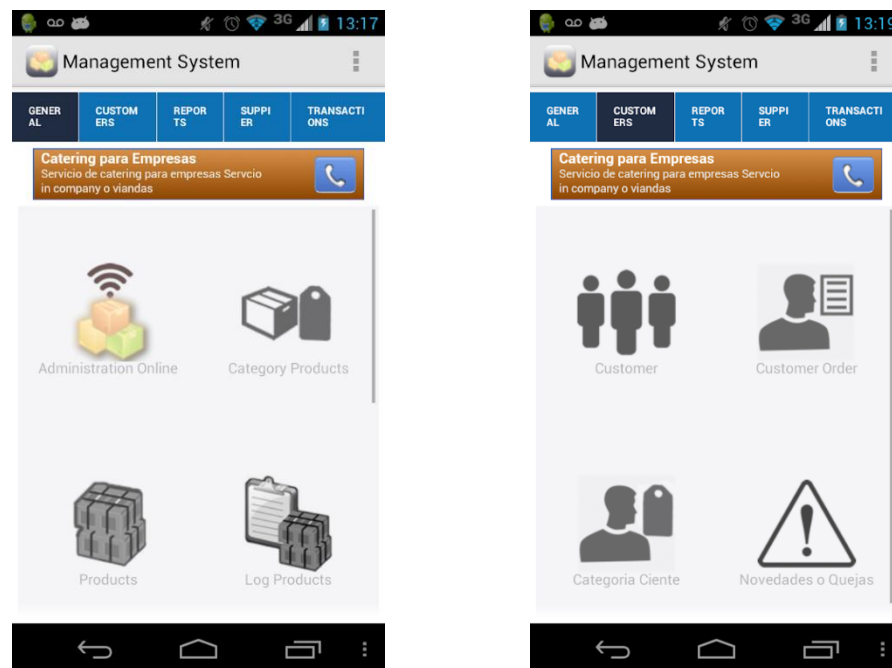


Figure 3. Stock manager main menu (left) and customer management menu (right).
(Source: author)

1.2.5 Conclusion

All main features of the abovementioned competing applications are presented in the **Table 1**.

Table 1. *Features of the competing applications compared to the author's application.*
(Source: author)

Application title	Multiple roles support	Manage stock	Generate statistics	Notifications	Order status	Generate reports invoices	Platform independent
This project	+	+	+	+	+		+
Mobi-c	+	+	+			+	
Inventory, Purchase, Sales Order		+	+			+	
Stock Manager	+	+				+	

As one can see from Table 1 even very elaborate competing applications still miss some important features, so there is still space for improvements, and the author intends to implement those missing features in his solution.

2 DESIGN AND ARCHITECTURE OF API

In this section, the author describes overall architecture of the application and its components, database structure and data flow diagrams. In addition, application key features are explained.

2.1 Functional requirements

The API development process includes following functional requirements:

- User can access from desktop web browser, mobile web browser or mobile client application;
- User can log in with unique credentials
- User can change password
- User can get unique identifier by entering own device id
- Admin can register new user
- Admin can reset user password
- Unique token can be created for registered user
- Registered user can read or modify own profile data with access token
- Admin can read, modify and delete profile data of other users
- Any user request is rejected if token has expired
- Each user has their own action permission set, depending on which user group they belong to.

2.2 Non-functional requirements

The API development process includes following non-functional requirements:

- API must work with JSON data serialization format
- API must return error messages in English language if any
- User must have stable internet connection to use API

2.3 Context level 0 diagram

Context diagram demonstrates how data is being processed between actors of the information system (**Figure 4**). Arrows demonstrate process format and action.

Processes are performed in following sequence:

- 1) Client inserts necessary data into client application form

- 2) Client application converts entered data into JSON format and sends AJAX request to API
- 3) API prepares database statement and sends query to database server
- 4) Database management system executes query and/or returns result of query to API, if select statement were performed
- 5) API returns data to client application in JSON format
- 6) Client application displays returned data.

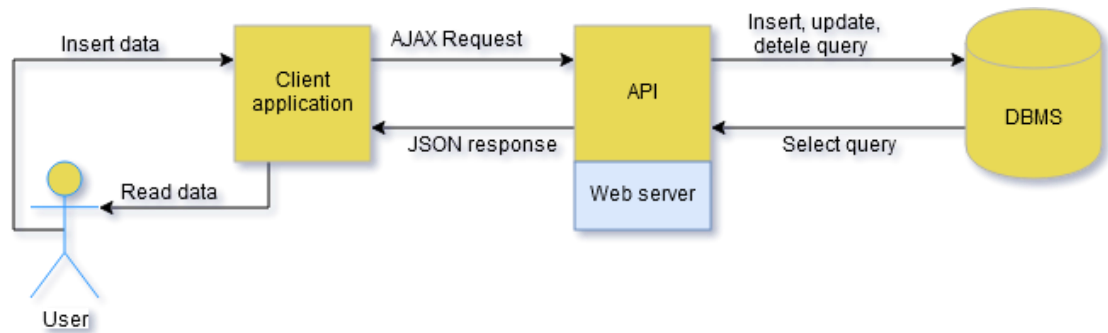


Figure 4. Context level 0 diagram. (Source: author)

2.5 Database development

Author has chosen MySQL database to store all necessary data because it is commonly used with PHP and the author already has experience working with MySQL databases.

Currently the database contains several entities and tables, which are designed for future API features, which are not yet implemented in the application as of writing this thesis. Most of the database tables are normalized until third normalization form, while tables for collecting statistical data are normalized only until second normalization, because more complicated queries will be performed with those tables (**Figure 5**).

For example, to implement the login feature, one needs to use `api_user`, `api_user_device` and `api_user_online` tables. API checks the data received from `api_user` and `api_user_device` tables and then sends a query to insert or update data row to the `api_user_online` table.

Database key entities are:

1. `api_user`
2. `api_order`
3. `api_company`
4. `api_notification`

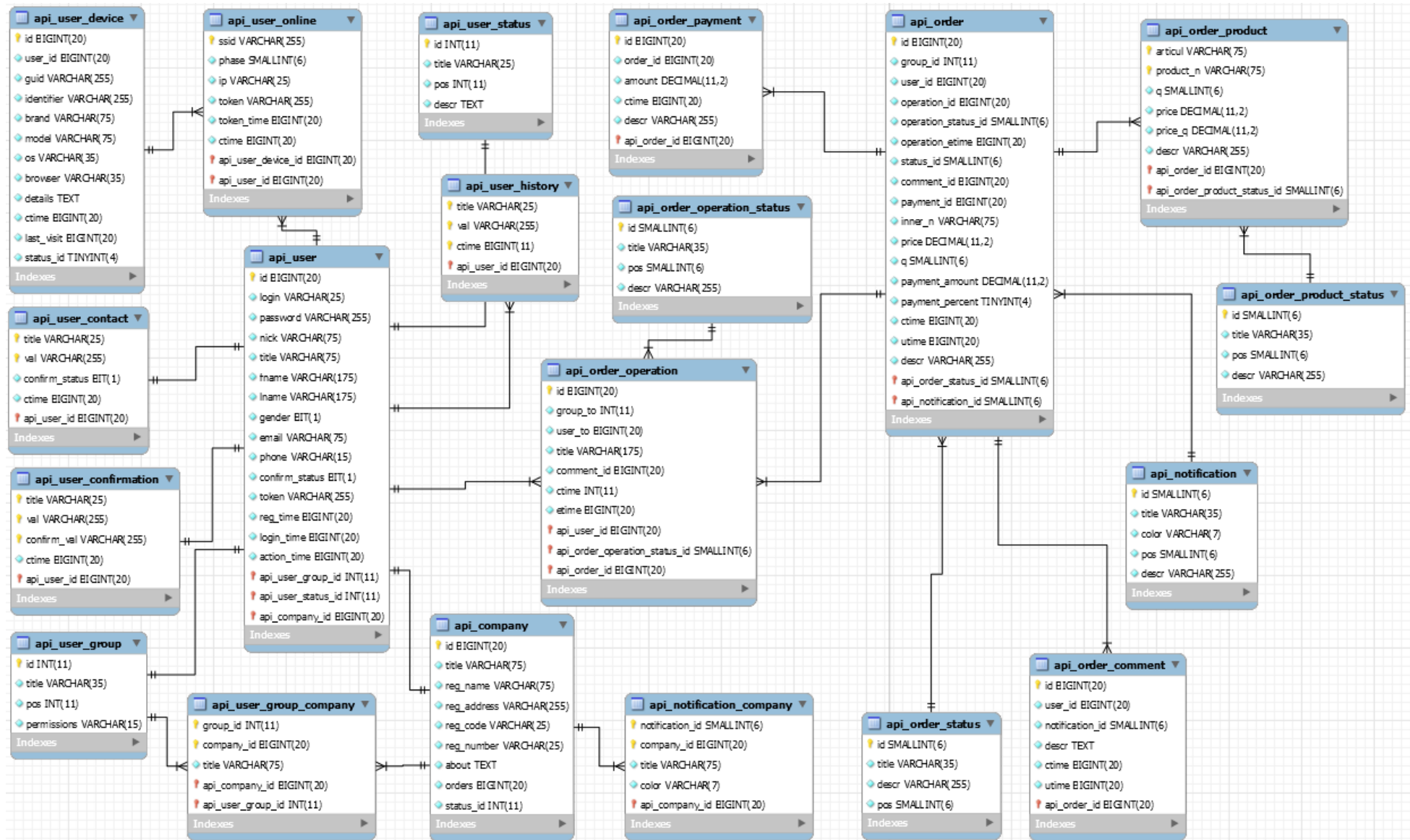


Figure 5. Database entity-relationship diagram. (Source: author)

2.6 API kernel development

API kernel consists of several base classes, which are going to be used by the whole system and will have minimal changes after implemented. The classes are listed below:

- 1) Db.php
- 2) Container.php
- 3) Customer.php, LogWriter.php, System.php
- 4) Request.php
- 5) Token.php
- 6) Translation.php
- 7) Validator.php

2.6.1 Database connector class

Database connector class contains all necessary information about database address, name, and user password. In addition, it contains PDO methods to prepare, execute statements and fetch received data.

2.6.2 Container class

Container class allows creating and saving other classes inside of its instances for future use. If a class instance already exists, then it is re-used without creating a new instance of the class.

2.6.3 Helper classes

Helper classes like Customer and LogWriter are used to interact with database by using more specific data queries. System.php contains a method *getIP*, which returns the IP address of the user, who sends the data request to the API.

2.6.5 Request class

Request class contains several methods to compare data received from the client application with database entries to check whether the client has entered correct credentials and whether it is authorized to receive any information. In addition, the Request class checks the user permissions for the current request. If there are any errors, they are added to the response array and sent in JSON format to the client application. If data, received from client application is correct, *checkAuth* method returns true and then data is passed to API controller. The whole work of request class is illustrated on **Figure 6**. Method of user permission check is illustrated on **Figure 7**. Detailed description of *checkAuth* method work is illustrated on **Figure 8**.

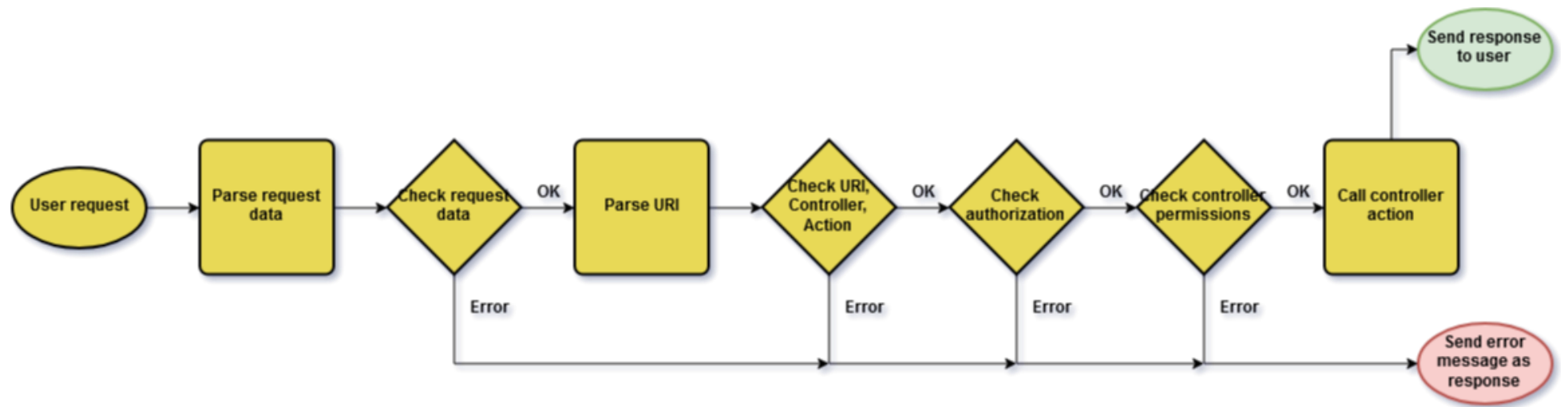


Figure 6. The whole work of Request class. (Source: author)

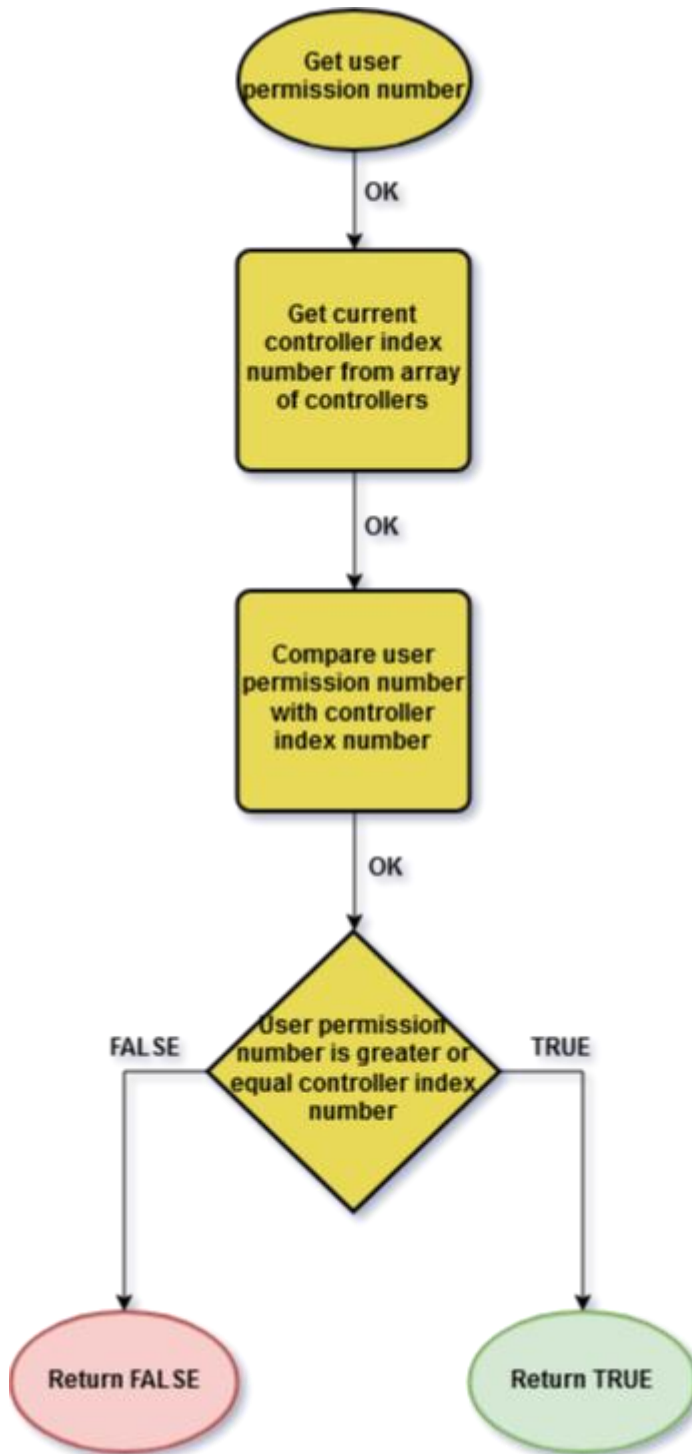


Figure 7. Data process sequence in `check permissions` method of `Request` class. (Source: author)

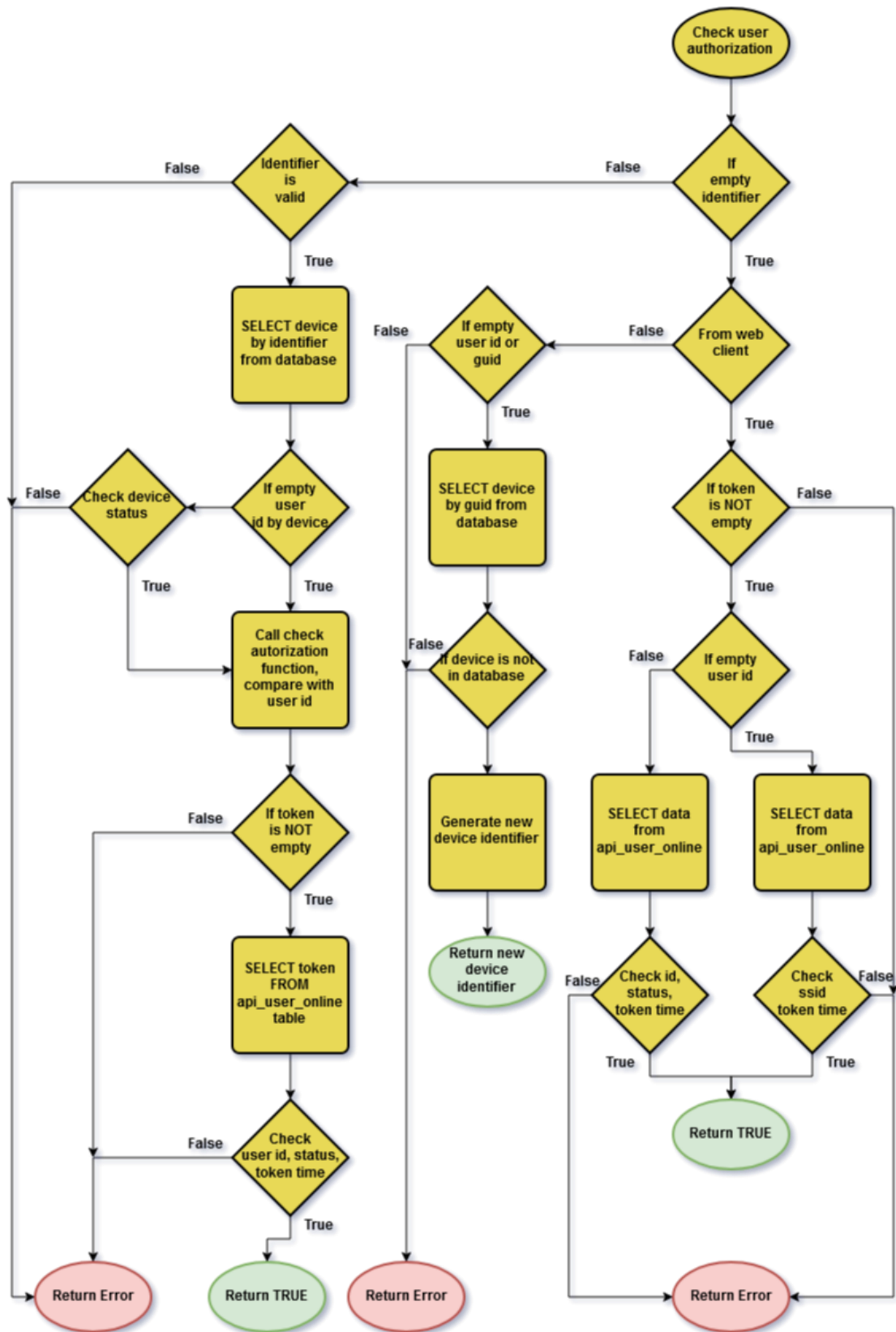


Figure 8. Detailed description of checkAuth method of Request class. (Source: author)

2.6.6 Token class

Token class contains methods *generate* and *generateIdentifier* to generate user token and unique identifier based on device id. In case of success, token and identifier are inserted in database and returned as the result of method.

2.6.7 Language translation class

Checks current API locale and creates alias between translation and error messages files. English language is set as default.

2.6.8 Validator class

Validator class contains several methods, which check if, for example, user email has correct format or received data type is integer and returns Boolean value.

2.7 API development

API consists of several classes and files and belongs to exact company, who is going to use API. Directories, classes and files are listed below:

1. API configuration: config.php, constants.php
2. Controller: Login, Profile
3. Translation: errors.php
4. ApiRoot
5. ApplicationController
6. Index.php

2.7.1 API configuration

config.php

This file has API default language setting, array of controllers and their actions which are used in API.

constants.php

This file has database connection parameters, token active time, path of base directory and few more common constant values.

2.7.2 API controllers

Controller is an entity, which defines set of action that will be performed by the API according to user request.

Login

Login controller class contains methods *main* and *token*. Former method checks if user logs in from web client or mobile application, updates database table *api_user_device* and *api_user_online* with new token and visit time. Latter method sets API response with newly generated token for user

Profile

Profile controller class contains method *main*, *users* and *password*. Method *main* allows to read and modify user own profile data if correct credentials were received from client application. Returns profile data as response. Method *users* is available for Top Manager or Admin role and it returns array of users and some of their credentials who are registered in the application. Method *password* allows any user to change own password.

Registration

Registration controller class contains method *main*. It allows to register a new application user with entered credentials. Credentials, received in request, are checked for validity and then added to *api_user* table. Available only for Administrator and Top Manager user role.

2.7.3 Translation

Translation directory contains *errors.php* file. The file has error messages and their ID numbers of default locale.

2.7.4 ApiRoot

ApiRoot class contains method *services*, which registers instances of translation, validator, request, token classes and puts them into container. There are also methods *getConfig* and *getConfigByKey*. Former returns contents of company API configuration file, latter creates hash table of configuration elements and returns list of key-value pairs.

2.7.5 AppController

AppController is parent class of controllers. It defines the data format returned to the user and gets processed elements from container to send as response.

2.7.6 Index.php

Index.php is starting point of API work. It creates *Api* object, loads file with constant values and class autoloader.

2.8 Client application front-end development

A simple web client application is created to imitate client application data requests to API (**Figure 9**). Client application consists of *jQuery* library file, JavaScript client with API address parameters, several HTML files with web forms to enter data, CSS file to set form and font size.

Web client application uses HTTP protocol to send data requests to web server, where API is located.

The screenshot shows a web browser window with the address bar displaying 'api.dreamsfm.eu/WpApp/web/login.html'. The page content is as follows:

Get identifier

Login for customer

Generate token

Figure 9. Client application web form for user login. (Source: author)

2.9 API features

2.9.1 User identification

In order to identify API user, the combination of following data values are used:

1. Session ID
2. User login
3. User password
4. User identifier (mobile application client only)
5. User device GUID (mobile application client only)
6. Ip address
7. Access token

2.9.2 Token based authentication

A token is a piece of data that has no meaning or use on its own, but combined with the correct tokenization system, becomes a vital player in securing your application. Token based authentication works by ensuring that each request to a server is accompanied by a signed token, which the server verifies for authenticity and only then responds to the request. (Auth0...n.d.)

Token user authentication implemented in this work for extra security reasons. Basic idea of token authentication is taken from common definition of access token, but implementation in this work is different:

- Token lifetime is defined in constants.php file as `TOKEN_GENERATION_TIME`
- Token expiry time is updated with every user request while token is valid
- Token is linked with user identifier and user id or just with user id if web client used

Token generation is based on current time specific format, which is passed to *uniqid* function as prefix. Result of *uniqid* function is string value, based on prefix and current time in microseconds (**Figure 10**).

Token generation method is called once authenticated user logs in or when sends data requests, which refresh user token expiry time with every request. Requires user id as incoming parameter. As result of method it token value is updated in database and returned to client side application. Error is returned otherwise.

```

public function generate( $customer_id ) {
    $token = '';
    $err = '';
    $now = time();

    if( empty( $customer_id ) || !$this->validator->isPositiveInteger( $customer_id ) ) {
        $err = $this->translation->getErrorById(1007);
    } else {
        $token = uniqid(date( format: 'YmdHis-' ));
    }

    if( $err == '' && $token != '' ) {
        $sql = 'UPDATE api_user_online SET token = \'' . $token . '\', token_time = ' . $now . ' WHERE user_id = ' . $customer_id;
        $this->db->execSql($sql);

        return $token;
    }

    return Container::getInstance()->get('Request')->setResponse( ['error' => $err] );
}

```

Figure 10. PHP Code of token generation function. (Source: Author)

2.9.3 User role permissions

The central notion of Role-Based Access Control (RBAC) is that users do not have discretionary access to enterprise objects. Instead, access permissions are administratively associated with roles, and users are administratively made members of appropriate roles. This idea greatly simplifies management of authorization while providing an opportunity for great flexibility in specifying and enforcing enterprise-specific protection policies. Users can be made members of roles as determined by their responsibilities and qualifications and can be easily reassigned from one role to another without modifying the underlying access structure. (Ferraiolo; Cugini; Kuhn 2015)

Role-based access control allows set, what API using permissions has specific employee (**Figure 11**).

The database table `api_user_group` stores information about each role and its controller permission (**Figure 12**).

The idea to implement role-based access control in this work has following reasons:

- To allow admin and manager set permissions to other department roles
- To allow roles see and modify only data they are permitted
- To avoid sudden or mistaken deletion of data by non-managing roles

Controller\ User role	admin	top manager	manager	shop assistant
login	3	3	3	3
profile	3	3	2	1
registration	3	3	0	0
meaning	0 = restricted	1 = read	2 = read + write	

Figure 11. API user roles and their permissions against controller actions. (Source: author)

id	title	pos	permissions
1	admin	10	333
2	top manager	20	333
3	manager	30	320
4	warehouse worker	40	310
5	adjuster	50	310
6	shop assistant	60	310

Figure 12. Database table *api_user_group*. (Source: Author)

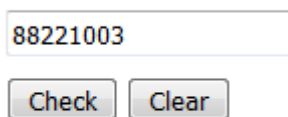
3 RESULT OF DEVELOPMENT

The author has decided to demonstrate user requests and API response with help of HTML forms and AJAX requests. This imitates future client application requests and displays what user will see after it.

3.1 Get identifier by sending device GUID

Unregistered user has to enter device GUID. Time of action is returned to client application as result, moreover a new entry is added into api_user_device database table and will be used in registration (**Figure 13**).

Get identifier



88221003

Check Clear

Response

time = 08.05.2018 16:54:41

Figure 13. Get identifier by user id. (Source: author)

3.2 Login for registered user

In order to log-in, registered user has to enter identifier, login name and password. Action time, user new token and user id is returned as result, while new entry is added into api_user_online database table (**Figure 14**).

Login for customer



q5d21NKZS/PiPUm54aky

ac

.....

Login Clear

Response

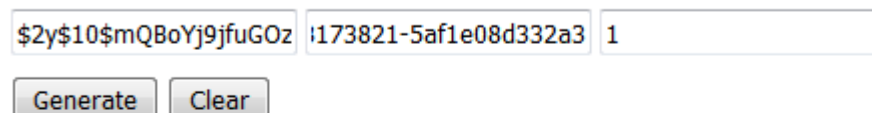
```
time = 08.05.2018 17:38:21
data
  action = main
  params
    token = 20180508173821-5af1e08d332a3
    id = 1
```

Figure 14. Log in for registered user. (Source: author)

3.3 Token refresh for online user

In order to update token lifetime, registered user has to send identifier, current token value and user id. Token is returned to client application as result. Token refresh time and token expiry time is updated in api_user_online database table. (**Figure 15**)

Generate token



\$2y\$10\$mQB0yJ9jfuGOz 173821-5af1e08d332a3 1

Generate Clear

Response

```
time = 08.05.2018 17:57:38
data
  action = token
  params
    token = 20180508175738-5af1e512614f2
```

Figure 15. Refresh token for online user. (Source: author)

3.4 Error message if token has expired

Whenever users have not been active for some time, they receive action time, error number and message (**Figure 16**).

Generate token

<input type="text" value="q5d21NKZS/PIPUm54aky"/>	<input type="text" value="173821-5af1e08d332a3"/>	<input type="text" value="1"/>
<input type="button" value="Generate"/>	<input type="button" value="Clear"/>	

Response

```
time = 08.05.2018 17:48:15
error
  0
  Key = 1004
  Message = The token expired.
```

Figure 16. Error message if token has expired. (Source: author)

CONCLUSION

Trading and service sectors in Estonia play important role in establishing country's strong and stable economics. In addition, Estonia has great infrastructure for digital solutions, good internet connection, e-country services, digital signature, e-recipe from the family doctor etc. This means, that more digital solutions can be implemented in our everyday life.

Comparing to similar solutions, application, suggested by the author offers to involve more roles to cooperate within the company, not just administrator and sales agent, but also warehouse assistant and delivery agent. In addition, the author's solution is able to interact with client mobile, web or desktop application.

The aim of thesis was to develop a web application for adjustment of small trading company working processes.

According to main tasks, the following functionality elements were developed:

- Database with necessary tables
- Login, profile and registration controllers
- User authentication and authorization
- Error messages
- Web client forms

The application and database, which are implemented in this thesis are prepared for adding more functionality and features in the future development.

There are several tasks and features for future development such as:

- Order management
- Order status confirmation
- Statistics collection
- Push notifications

RESÜMEE

Töö teemaks oli tööprotsesside korralduse parandamine väikses kaubandusettevõttes. Töö eesmärgiks oli andmebaasi loomine ja veebirakenduse arendamine, mis võimaldaks erinevate juurdepääsuõigustega kasutajate juurdepääsu infosüsteemile. Rakendus pidi sisaldama registreerimise ja sisselogimise ning kasutaja profiili loomise võimalusi. Samuti pidi rakenduses olema alus uute võimaluste lisamiseks tulevikus.

Töö koosneb kolmest peatükist. Esimene peatükk räägib tehnoloogiate ja arendustööriistade valikust, ning samuti kirjeldab sarnaseid lahendusi, nende eeliseid ja piiranguid. Kolme kõige sarnasema lahenduse puhul tehti põhjalikum analüüs ja võrdlus.

Teine osa kirjeldab rakenduse arhitektuuri, rakenduse võtmekomponente ja funktsionaalseid põhivõimalusi ning andmebaasi. Kolmas osa sisaldab näiteid rakenduse funktsionaalsete võimaluste kasutamise kohta.

Rakenduse arendamist ajendas probleem, millele osutasid erinevate kaubandusettevõtete töötajate vahel läbi viidud küsitluse tulemused Ameerika Ühendriikides. Üle poole vastajatest polnud rahul ettevõttesiseste tööprotsesside korralduse kvaliteediga töötajate või ettevõtte harude vahel. Küsitluse tulemusena tehti kindlaks mitu aspekti, mis takistavad töötajate edukat koostööd. Käesoleva töö autor pakub tarkvaralise lahenduse, mis võimaldab ettevõttel leevendada või ära hoida suure osa probleemidest, millele juhib nimetatud küsitlus.

Vastavalt püstitatud ülesannetele, autor lisas rakendusse järgmisi funktsionaalseid elemente:

- Andmebaas rakenduse jaoks;
- Registreerimise, sisselogimise ja kasutaja profiili moodulid;
- Kasutajate autentifitseerimine ja autoriseerimine;
- Vigade tekstide ja koodide genereerimine;
- Veebiliides rakenduse kliendipoolse osa töö imiteerimiseks.

Tulevikus plaanib autor lisada rakendusse järgmised võimalused: telliste haldamine, kuna see on üks võtmekomponentidest kaubandusettevõtte jaoks; tellimuse staatuse kinnituse, et kõik tellimuse töötlemisega seotud osapooled oleksid kursis, mis seisus tellimus on; statistika kogumise selleks, et juhtkonnal oleks kergem orienteeruda ettevõtte töös ja kergem teha ettevõtte juhtimisega seotud otsuseid; *push*-teavitused tellimuse

osapoolte informeerimiseks sellest, et on vaja sooritada mingi toiming tellimuse eduka täitmise jaoks.

REFERENCES

Wikipedia 2018. REST. Available at https://en.wikipedia.org/wiki/Representational_state_transfer, accessed May 5, 2018

W3Schools 2018. Asynchronous JavaScript (AJAX). Available at https://www.w3schools.com/xml/ajax_intro.asp, accessed May 5, 2018

ECMA International 2017. JavaScript Object Notation (JSON). Available at <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>, accessed May 5, 2018

The PHP Group n.d. PHP Data Object (PDO). Available at <http://php.net/manual/en/intro.pdo.php>, accessed May 5, 2018

Kemp; Appelquist; Malhotra; Raman 2010. Web Applications Architecture. Available at <https://www.w3.org/2001/tag/2010/05/WebApps.html>, accessed April 23, 2018.

Adjustment n.d. Cambridge Dictionary Cambridge University Press. Available at <https://dictionary.cambridge.org/dictionary/english/adjustment>, accessed April 23, 2018

Wikipedia 2017. Access token. Available at https://en.wikipedia.org/wiki/Access_token, accessed May 5, 2018

Ferraiolo, David F; Cugini, Janet A; Kuhn, D. Richard 1995. *Role-Based Access Control (RBAC): Features and Motivations*. National Institute of Standards and Technology U.S. Department of Commerce. Gaithersburg MD 20899. Available at https://www.researchgate.net/profile/D_Kuhn2/publication/238743515_Role-based_access_control_features_and_motivations/links/562e7d3808ae04c2aeb5d98b.pdf, accessed May 8, 2018

Auth0 n.d. Token Based Authentication Made Easy. Available at <https://auth0.com/learn/token-based-authentication-made-easy>, accessed May 8, 2018

Katcher, Bruce L. 2018. *How to improve interdepartmental communication*. Discovery Surveys, inc. Sharon and Wellfleet, MA. Available at <http://www.discoverysurveys.com/articles/itw-017.html>, accessed May 8, 2018

APPENDICES

The source code

Archive with application contains following items:

- Database – dreamsfmeu_api.sql
- Application files
- Installation instruction – api_details.txt

The application can be run on local web server with support of PHP version 7.1 or higher and MySQL client version 5.0.12 or higher.

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Maksim Vassiljev,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Web application API for adjustment of small trading company working processes, supervised by Andre Säask and Aleksandr Sokhin,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Narva, 23.05.2018