

Web Development

Service Desk Reporting System

Victor Anderssen

Bachelor's Thesis
Electrical Engineering
Vasa 2017



EXAMENSARBETE

Författare: Victor Anderssen
Utbildning och ort: Elektroteknik, Vasa
Inriktningsalternativ: Automationsteknik
Handledare: Kaj Wikman

Titel: Web Development - Service Desk Reporting System

Datum 10 april 2017 Sidantal 43

Abstrakt

Detta examensarbete gjordes på uppdrag av Wapice Ltd. Syftet med examensarbetet var att skapa en webbapplikation för övervakning av Service Desk-systemet som idag används hos Wapice Ltd. Det nya systemet kompletterar det befintliga och innehåller nya användargränssnitt för att övervaka statusen av Service Desk-systemet i realtid.

Examensarbetet resulterade i en webbapplikation som tillåter användare att skicka nya supportärenden och visa sina egna supportärenden. Administratörerna har tillgång till statistiska uppgifter om supportförfrågningarna, vilket hjälper till med beslutet om vilka förfrågningar som ska lösas först.

Språk: engelska Nyckelord: Web development, PHP, MySQL, JavaScript, Ajax, REST

BACHELOR'S THESIS

Author:	Victor Anderssén
Degree Programme:	Electrotechnics
Specialization:	Automation
Supervisor(s):	Kaj Wikman

Title: Web Development - Service Desk Reporting System

Date April 10, 2017 Number of pages 43

Abstract

This Bachelor's thesis was made on behalf of Wapice Ltd. The aim of the thesis was to create a web application for monitoring the service desk system currently deployed at Wapice Ltd. The new system complements the existing one, providing new user interfaces for monitoring the state of the service desk in real time.

The thesis resulted in a web application that allows users to submit new support tickets and view their own tickets. The administrators have access to statistical data about the support tickets, which helps with the decision on which tickets that should be solved first.

Language: english	Key words: Web development, PHP, MySQL, JavaScript, Ajax, REST
-------------------	--

Table of contents

Abbreviations

Preface

1	Introduction	1
1.1	Commissioner	1
1.2	Background.....	1
2	Task.....	2
2.1	Previous solution.....	2
2.2	Requirements.....	2
2.3	Aims and goals.....	2
3	Theory.....	3
3.1	HTML.....	3
3.2	CSS.....	4
3.3	SASS	5
3.4	JavaScript.....	6
3.5	Ajax	7
3.6	PHP	8
3.7	REST API.....	9
3.8	Web service	9
3.9	JSON.....	10
3.10	SQL.....	11
3.11	Authentication.....	12
3.11.1	LDAP	12
3.11.2	Shibboleth	12
3.11.3	SAML2.....	12
3.12	Sanitization.....	13
4	Development tools.....	14
4.1	Development environment.....	14
4.2	Visual Studio Code	14
4.3	Materialize framework.....	14
4.4	NPM	15
4.5	Gulp.....	16
4.6	Testing tools	17
4.6.1	Chrome Developer Tools	17
4.6.2	Postman.....	18
5	Solution	19
5.1	Planning	19

5.2	Application design	20
5.3	Navigation and language management.....	22
5.4	Data fetching	24
5.5	Service desk portal.....	25
5.6	Open tickets.....	27
5.7	Ticket escalation	28
5.8	Charts	31
5.8.1	Ticket overview charts	31
5.8.2	Ticket queues charts.....	32
5.8.3	Conversion of data for the ticket overview charts.....	33
5.8.4	Conversion of data for the ticket queues charts	37
5.8.5	Ticket queues selection	38
6	Result and discussion	40
6.1	Result	40
6.2	Problems encountered	40
6.3	Further development.....	40
6.4	Discussion	41
7	References.....	42

Abbreviations

SMS	Short Message Service.
RAM	Random Access Memory.
XML	Extensible Markup Language.
API	Application Programming Interface.
REST	Representational State Transfer.
SQL	Structured Query Language.
JSON	JavaScript Object Notation.
Array	Systematically arranged data in rows and columns.
SASS	Syntactically Awesome Style Sheets.
CSS	Cascading stylesheets.
DOM	Document Object Model.
LDAP	Lightweight Directory Access Protocol.
URL	Uniform Resource Locator.
Regex	A regular expression, search pattern.
IDE	Integrated Development Environment.
CRUD	Create Read Update Delete.
Duty Officer	Person responsible for tickets escalated by the end user.
NULL	A special marker and keyword indicating that something has no value.

Preface

This thesis is the final project of the degree programme in Automation engineering (Bachelor of engineering) at Novia University of Applied Sciences in Vaasa, Finland. The thesis work was carried out during the summer of 2016 in co-operation with the employer Wapice Ltd.

I first encountered Wapice in the beginning of 2016 when I applied for a summer job position at the head office in Vaasa. I have received a warm welcome at the office.

I would like to take the opportunity to thank everyone who have helped and supported me in the making of this thesis. Thank you Teemu Niemi and Olli Rajala at Wapice who has guided me in developing the right solution and who also lead the project. At Novia University of Applied Sciences I would like to thank my supervisor Kaj Wikman.

A special thank you to Esko Salonen who is the co-developer of the web application that this thesis is about.

Victor Anderssén

Vaasa, 21 February 2017

1 Introduction

This chapter consists of background information on the commissioner.

1.1 Commissioner

Wapice Ltd was established 1999 in Vaasa, Finland. Wapice is an independently and privately-owned company. Wapice is a software company providing solutions to domain leading companies globally. Wapice currently employs over 320 experts in seven locations around Finland. [1]

1.2 Background

Wapice has deployed a service desk system for managing support requests internally and for customer support requests. The IT-department which is responsible for managing these requests needed a tool that would help them monitor and prioritize incoming requests.

For Wapice, one of the most problematic parts about administrating a service desk system is to choose which tickets to solve first. It might not always be the most recently submitted tickets that are the most urgent. The tickets are prioritized according to severity but the deployed system had no good way of showing an overview of the newest tickets and their respective severities.

2 Task

The task was to develop a web application through which end users can file support tickets to the service desk. The IT-admins wanted statistical features for monitoring the state of the service desk and for prioritizing tickets. This would make it easier to determine which tickets to solve first.

2.1 Previous solution

The previous solution was a service desk system. Even though the previous solution had many wanted features, the previous system lacked a way of displaying the new tickets in a graphical way. Instead it displayed the new tickets in a list.

2.2 Requirements

The end user should be able to create new tickets and view their own tickets. They should also be given the option to escalate their own tickets to a service manager for further review in urgent cases.

The administrators should be able to see the overall status of the service desk system. This includes charts for all tickets created and closed the past month and all the newly created and successfully closed tickets in the system cumulatively since system deployment. A graphical overview of the tickets in the severity groups should be displayed to the administrators with the option to filter out certain tickets based on the queue that they belong to.

Administrators should be able to select a person that becomes the active duty officer. The duty officer will get an SMS upon a ticket escalation, which the end user can do in urgent situations when the ticket is not processed in a reasonable time.

2.3 Aims and goals

The aims of this thesis are to make the service desk easier to use for the end user. Give the administrators more analytic tools for better overall system monitoring. The web application developed should not replace the previously deployed system, it should complement the existing system by providing new features and simplify the interaction with the system both for the end users and the administrators.

3 Theory

This chapter consists of theory about the technologies used to create the web application.

3.1 HTML

Hyper Text Markup Language (HTML) is the building blocks used for creating websites. It defines the elements in the page but not the styling. Elements are represented by starting and closing tags. Browsers do not display the tags, but use them to render the content inside of them per the standard and browser specifications. Examples of content that can be displayed with HTML tags are images, videos, links and forms. [2]

Code example 1. Basic structure of a HTML document containing an unordered shopping list.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Shopping list</title>
</head>
<body>
  <h1 id="shoppingListHeader">My shopping list:</h1>
  <ul class="shoppingListItems">
    <li>Coffee</li>
    <li>Tea</li>
    <li>Bread</li>
  </ul>
</body>
</html>
```

3.2 CSS

Cascading Style Sheets (CSS) is a scripting language that is used to define the graphical properties of HTML elements such as color, placement, size, font and margins. To apply a style to an element, CSS uses rule sets. Rule sets are blocks of code that contain selectors that point to a class, ID or type of element. Inside of the rule set are properties and values. The CSS can be defined in a HTML document, but it is more common to store it in a separate file and use an include statement in the HTML document in which the styles should be applied.

Code example 2. Basic structure of a CSS file containing rule sets for elements in code example X.

```
/* Selector of the body element */
body {
    background-color: #ccc;
}

/* Selector of the element with an ID of shoppingListHeader */
#shoppingListHeader {
    font-size: 14px;
    font-weight: 400;
    color: black;
}

/* Selector of the element with a class of shoppingListItems */
.shoppingListItems {
    font-size: 12px;
    font-weight: 300;
    color: rgba(0, 0, 0, .86);
}
```

3.3 SASS

Syntactically Awesome Style Sheets (SASS) is a scripting language that compiles down to CSS and controls the look of web elements. It includes more functionality than CSS, for example nesting of selectors and the possibility to use variables. Colors for example can be stored in variables for later use. If the colors needed to be changed in the future, they can be replaced in one location in the code. It is possible to divide the SASS code into smaller files for easier management, these files would then get included in a main file that in turn gets compiled to one CSS file. The code becomes reusable with SASS. [3]

Code example 3. Basic example of SASS with nesting and a variable for the color attribute.

```
$nav-text-color: rgba(0, 0, 0, .58)

.navbar
  font-size: 1rem
  color: $nav-text-color
  font-weight: 400
  .nav-item
    &:hover
      color: lighten($nav-text-color, 25%)
```

Code example 4. Compiled CSS of the code in example 3.

```
.navbar {
  font-size: 1rem;
  color: rgba(0, 0, 0, 0.58);
  font-weight: 400;
}
.navbar .nav-item:hover {
  color: rgba(64, 64, 64, 0.58);
}
```

3.4 JavaScript

JavaScript is an object-oriented scripting language. It can be run inside a host environment (typically a web browser). JavaScript can control the behavior of web pages and provide interactivity. It is more commonly used as Client-side JavaScript which can be used to control a browser and its Document Object Model (DOM). For example, responding to user events such as mouse clicks, form input, and page navigation. [4]

In this thesis jQuery was used. It is a JavaScript library which simplifies DOM manipulation and event handling. For example, selecting elements can be done with far less code than in JavaScript.

Code example 5. Toggling the visibility of an element with jQuery and JavaScript.

```
/* Toggle element visibility with jQuery */
$("#toggleProductInfoBtn").click(function() {
    $("#productInfo").toggle();
});

/* Toggle element visibility with JavaScript */
var button = document.getElementById('toggleProductInfoBtn');
button.onclick = function() {
    var productInfo = document.getElementById('productInfo');
    if (productInfo.style.display !== 'none') {
        productInfo.style.display = 'none';
    }
    else {
        productInfo.style.display = 'block';
    }
};
```

Server-side JavaScript has become more common nowadays. It extends JavaScript itself by supplying objects relevant to running JavaScript on a server. One example of such a server-side JavaScript runtime is Node.js, it is built on top of Google Chrome's V8 JavaScript engine. Node.js uses a non-blocking I/O system, no task is blocking the execution of another. Contrary to traditional web server technologies, where every connection creates a new thread or process which uses the servers RAM and finally will use up all the available memory, Node.js only uses a single thread to support up to tens of thousands concurrent connections. [5]

Node.js combined with other technologies can provide a full stack web development environment, one stack is MEAN (MongoDB, Express.js, Angular.js, Node.js).

3.5 Ajax

Asynchronous JavaScript and XML (AJAX) enables a website to request partially new data to parts of a page. Without having to reload the whole page. Since AJAX is asynchronous, the request will run in the background and the user's browser will continue with the processing of other events. This technique was used in this thesis project to fetch data for the charts and data tables displayed in the application. [6]

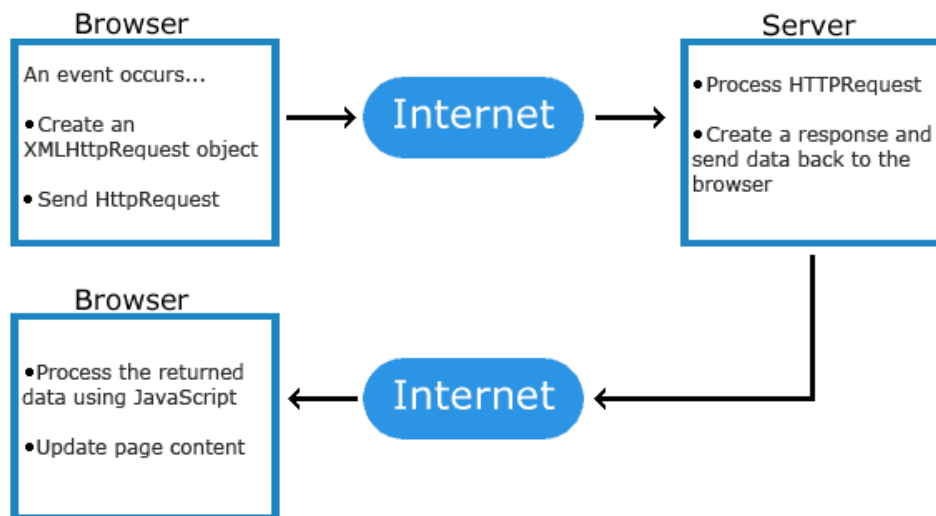


Figure 1. Example of how an AJAX request is processed. [7]

Code example 6. Example of an AJAX request for ticket statistics data using jQuery.

```

function loadStatistics() {
  $.ajax({
    data: {action: 'getStatistics'},
    url: 'ticketOverviewTools.php',
    method: 'GET',
    dataType: "html",
    success: function(data) {
      $("statistics").html(data);
    },
    error: function(jqXHR, textStatus, errorThrown) {
      console.log('getStatistics: Could not fetch any data from ticketOverviewTools.php');
      console.log('jqXHR: ', jqXHR);
      console.log('textStatus: ', textStatus);
      console.log('errorThrown: ', errorThrown);
    }
  });
}

```

3.6 PHP

PHP short for PHP: Hypertext Preprocessor is a server-side scripting language. It is interpreted on the server and the result is sent back to the user's browser as standard HTML. The user does not see the server side code. PHP can interact with many popular databases such as MySQL. PHP enables a web application to make use of user sessions to store user specific information. One of the more popular use cases for PHP is to use it within the LAMP (Linux, Apache, MariaDB / MySQL, PHP) stack, which is the case with this web application. [8] [9]

Code example 7. PHP document containing an unordered shopping list rendered using an array.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Shopping list</title>
</head>
<body>
  <h1 id="shoppingListHeader">My shopping list:</h1>
  <ul class="shoppingListItems">
    <?php
      $items = ['Coffee', 'Tea', 'Bread'];

      foreach ($items as $item) {
        echo '<li>' . $item . '</li>';
      }
    ?>
  </ul>
</body>
</html>
```

The code example above contains both HTML and PHP code. The PHP code is run on the server and standard HTML is sent back to the browser.

3.7 REST API

Representational State Transfer (REST) is an API (Application Programming Interface) that allows two systems to talk to each other. REST API's are stateless, meaning that each request made to the API contains all information necessary to fulfill the request. No session or state can be stored on the server side. [10]

REST API's are commonly accessed by an URI (Uniform Resource Identifier). REST can use HTTP methods to achieve CRUD (Create Read Update Delete) operations.

Table 1. REST actions and their CRUD equivalents.

HTTP method	CRUD
POST	Create
GET	Read
PUT	Update / Replace
PATCH	Update / Modify
DELETE	Delete

Using these methods, it is possible to expose an applications data through a REST API. This allows any other application to interact with the data that is provided by the API.

3.8 Web service

A web service is a communication method used to enable two systems to talk with each other. In this web application, a web service is used to expose certain ticket details through its REST API. This in turn gives the web application the capability to use information stored in the service desk database and use that information to draw charts and display data in any preferred way.

3.9 JSON

JavaScript Object Notation (JSON) is a lightweight data-interchange format. Compared to for example XML, JSON does not use opening and closing tags which makes it more readable. XML must be parsed with an XML parser. JSON can be parsed with a standard JavaScript method.

JSON is language independent. It can be fetched from a server and parsed. It consists of two types of structures. A collection of name and value pairs which can be compared to an associative array in other programming languages. The second structure type is an ordered list of values, which in other programming languages could be called array. [11]

Code example 8. Example of JSON data containing details of two tickets.

```
{
  "ticketDetails":[
    {
      "ticketID":"100",
      "ticketCreatedDatetime":"2016-10-27 12:51:21"
    },
    {
      "ticketID":"101",
      "ticketCreatedDatetime":"2016-10-27 13:02:23"
    }
  ]
}
```

Code example 9. Example of XML data containing details of two tickets.

```
<?xml version="1.0" encoding="UTF-8"?>
<tickets>
  <ticketDetails>
    <element>
      <ticketCreatedDatetime>2016-10-27 12:51:21</ticketCreatedDatetime>
      <ticketID>100</ticketID>
    </element>
    <element>
      <ticketCreatedDatetime>2016-10-27 13:02:23</ticketCreatedDatetime>
      <ticketID>101</ticketID>
    </element>
  </ticketDetails>
</tickets>
```

3.10 SQL

When information in an application needs to be stored in a way that it persists, a database is a standard way of achieving this goal. A database is a collection of data. Databases consist of tables which in turn consist of rows and columns. Each row is a representation of a record.

SQL, short for Structured Query Language is used for accessing and manipulating database systems such as MS SQL Server, MySQL and MariaDB. SQL uses queries to perform actions on a specified database. The queries can be complex and specific data can be fetched or changed.

To be able to uniquely identify records stored in a database table, a primary key is used. The primary key must be unique, it cannot be empty or NULL. Only one primary key column can exist in a database table. One common way of implementing a primary key is to use an automatically incremented number. Whenever a new record is inserted, the database creates a new automatically incremented number as the primary key for the record inserted.

Code example 10. SQL select statement example.

```
SELECT FROM Students WHERE Name = 'Victor' AND Age = 24;
```

Table 2. SQL result, one row is returned with the student named Victor.

ID (Primary Key)	Name	Age
1001	Victor	24

3.11 Authentication

This chapter consists of the basic theory behind the authentication methods used.

3.11.1 LDAP

LDAP, short for Lightweight Directory Access Protocol is used for reading and writing to Active Directory. LDAP directory server stores data hierarchically. It is commonly used to store information about an organization and its assets and users. [12]

3.11.2 Shibboleth

Single sign-on (SSO) is a method that allows users to authenticate once in a service or web page and gain access to other services with a single authentication. Shibboleth is a web-based SSO. It provides SSO to services outside of the user's origin while still protecting their connection. [13]

3.11.3 SAML2

SAML, short for Security Assertion Markup Language is a standard method used for logging users in to an external destination.

One major advantage of this system is that the user does not have to type in their credentials when accessing a resource outside of the origin, due to the SSO (Single sign-on) login standard. [14]

3.12 Sanitization

When dealing with applications in general, one should never trust user input or for example session based variables. Sanitization is the removal of malicious code from the used input. User input can be forms that the user has filled in and submitted. User input can also be parameters in the URL. Always sanitize user input, escape characters, and use prepared statements to prevent attacks against the system for example malicious code injections.

For sanitization, a separate class was used. An instance of this class exists in the main controller *index.php* which can be applied to any input or session variable to ensure that the input does not contain any malicious code or characters.

Code example 11. Example of a sanitation class in PHP.

```
<?php
Class Sanitize
{
    public static function username($string)
    {
        $string = preg_replace( "/^[^a-zA-Z0-9 ]/", "", $string);
        return $string;
    }

    public static function email($string)
    {
        $string = preg_replace( "/^[^a-zA-Z0-9@._-]/", "", $string);
        return $string;
    }
}
?>
```

The sanitation class example above contains two out of several methods used for sanitizing data. These two methods can sanitize usernames and email addresses by replacing the characters that do not match the regex, with an empty character.

4 Development tools

This chapter consists of the development tools and environment used in the development process.

4.1 Development environment

The development environment used was a LAMP stack. This stack was chosen due to it being familiar to the development team and the amount of documentation available. Development took place in Windows and code ready to be tested was pushed to the development environment, which was a separate server dedicated to the testing of this application.

4.2 Visual Studio Code

For code editing (VSC) Visual Studio Code was chosen. It is a free text editor which is cross-platform and has a built-in package manager for adding more functionality to the editor. The editor is lightweight and very customizable due to the amount of community made packages that can be installed to extend the functionality of the editor. VSC is a more lightweight alternative to the full featured IDE Visual studio.

4.3 Materialize framework

Materialize was chosen as the front-end framework for the project. One great advantage of using a framework is not having to invent the wheel all over again. The framework provides styled, standard HTML elements out of the box. This means that there are predefined styles for elements like buttons, form elements and navigation elements. One thing to keep in mind when using any kind of framework that provides styling, is to only use the styles as a foundation. In this project, the styles were customized to better reflect the company's style guide line. If the styles were to be left unmodified it results in the application looking as many other sites built with the same framework.

Materialize contains a responsive grid system like many other front-end frameworks, for example like in Bootstrap and Foundation. This makes it relatively easy to build a responsive web application that will adapt its layout to the screen on which it is displayed on.

Material Design is a design language developed in 2014 by Google. The design is bold, graphic and intentional. An emphasis on user actions makes core functionality immediately apparent and provides waypoints for the user. [15]

Modern web design nowadays is leaning towards material design and this framework implements that well. One other popular framework choice would have been the twitter bootstrap framework, due to it being the industry de facto standard for web front-end development. [16]

4.4 NPM

NPM, short for Node package manager is a way of bringing in packages of code into a project. These packages contain code for solving very specific problems. NPM is part of the Node.js system itself.

The main benefit of this system is that it is possible to use packages that provide special functionality to your project. The packages can easily be managed, updated and deleted through the NPM. Chances are that common functions that the project requires, have already been written by someone else, thus there is no reason to reinvent the wheel. Instead, include the package with NPM and use the provided functions. The package names and version numbers are stored in a package.json file in the project directory. [17]

Code example 12. Example of a package.json file containing the gulp and gulp-sass packages.

```
{
  "name": "app_name",
  "version": "0.0.1",
  "description": "Application description",
  "author": "Authorname",
  "license": "License type",
  "devDependencies": {
    "gulp": "^3.9.1",
    "gulp-sass": "^2.3.2"
  }
}
```

4.5 Gulp

Gulp is a JavaScript development tool that can help automate the most common tasks in a web development project. These tasks can include but are not limited to, compiling SASS to CSS. Minifying files, watching the filesystem and executing tasks on file changes. Minifying files, referred to as minification is the process of removing all unnecessary characters from the source code to be minified, but maintaining the source codes functionality. Characters that can safely be removed in minification are comments, white space and line breaks. The result of minification is a smaller file size. This comes in play when a user visits a web site and the browser downloads the requested files. If these files are smaller, the page will load faster.

In this project, Gulp was used to compile the SASS down to CSS every time a SASS file was changed on save. This way the projects different page and component styles could be divided up into their own separate files for easier management. [18]

Once Node.js is installed, Gulp can be installed globally from the terminal with the following command:

```
npm install --global gulp-cli
```

To include Gulp into the projects package.json file. The following command is used:

```
npm install --save-dev gulp
```

Code example 13. Example of a gulp file that contains two build tasks, sass and sass:watch.

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('sass', function () {
  return gulp.src('./sass/main.scss')
    .pipe(sass({outputStyle: 'compressed'})).on('error', sass.logError))
    .pipe(gulp.dest('./css'));
});

gulp.task('sass:watch', function () {
  console.log("Watching sass/scss files now!");
  gulp.watch('./sass/**/*.scss', ['sass']);
  gulp.watch('./sass/**/*.sass', ['sass']);
});
```

To execute the `sass:watch` task defined in the `gulpfile`, the following command is used in the terminal:

```
gulp sass:watch
```

In this case `gulp` would have started the `sass:watch` task that would compile all the SASS automatically when it detects a file change in the project. The execution of the task would continue until it is explicitly stopped by the user.

4.6 Testing tools

This chapter consists of information about the tools used to test the web application.

4.6.1 Chrome Developer Tools

Chrome is a web browser developed by Google. It initially launched back in September of 2008. Since then, Chrome has taken the lead in the web browser market share. In January of 2017, Chrome had a web browser market share of 58,4%. [19]

Chrome developer tools is the built-in toolset in the chrome browser. It is especially useful when testing and debugging web applications. It can be used to change CSS directly in the browser, one can copy the parameters from the development tools and implement the changes in the code. This makes debugging style issues much faster than doing it the old way of updating the CSS code in the project and viewing the changes in the browser afterwards.

An even more powerful feature is the network tab. It displays all the assets that are requested, fetched, and loaded in to the user's browser. It is possible to record all the page network activity. This becomes important when for example reviewing time consuming queries. The developer tools show how long an individual request is taking. It therefore becomes easier to compare different methods of acquiring data, one can test different design patterns and algorithms and benchmark them against each other.

4.6.2 Postman

Postman is an application used for making requests to web services using their application programming interface (API's). Postman works by specifying a URL and a method. Additional data can be provided if one is for example making a POST request. Headers can also be added to the requests. This ease of use makes testing API's very fast and easy. Requests can be saved for later use and testing.

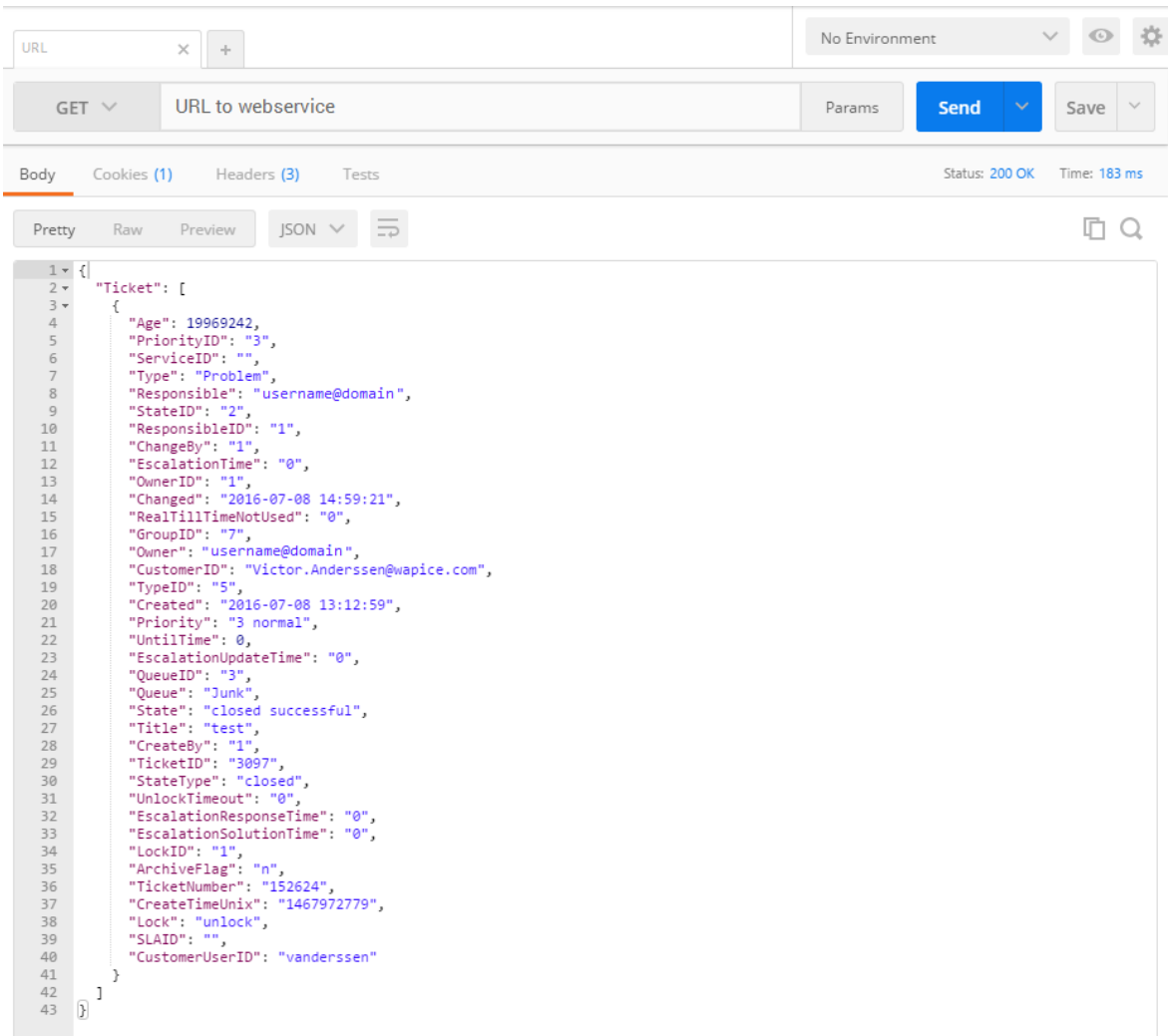


Figure 2. Example GET request in Postman. The result is JSON data.

5 Solution

This chapter consists of information about the planning and practical implementations of the application.

5.1 Planning

The planning process consisted of several meetings with the team involved in the application, where the application design was planned based on the requirements set by the project manager. Application design mockups were created in the planning process using basic HTML layouts to get a feel for how the application would look. These layouts could be changed fast and they were iterated upon several times.

The planning process did not stop when the implementation of the application started. Instead, the planning process continued along the whole project. This way the implementations could be reviewed by the development team and possibly altered.

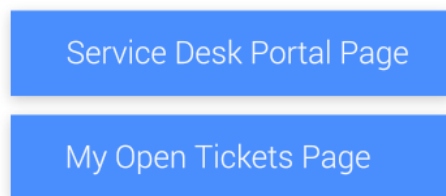
Every week the implemented functions were reviewed. New tasks were assigned and the planning of new tasks began.

5.2 Application design

Page access is divided into two categories. Administrators have access to all pages. Including the pages which contain analytical information about the service desk system and ticket details of all the tickets in the system. The normal users only have access to the main page where they can create new tickets and to the *my open tickets page*, where they can view all the tickets that they have created.

Page Access

Users & Administrators



Administrators Only

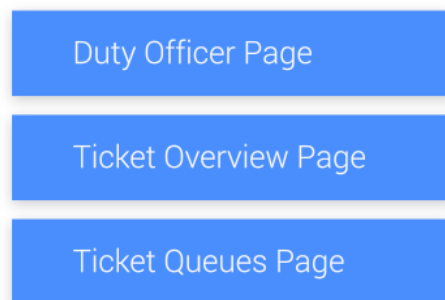


Figure 3. Page access diagram.

The *index.php* file serves as the applications main entry point, it is the applications main controller. Here the main session is established. All the required objects are instantiated. Validation is done to check whether the user is an admin or a normal user, this will restrict access to the administration pages in the application for the normal users. The information about the users is fetched from an LDAP server.

Code example 14. *index.php* serving as the applications main controller.

```
<?php
// Start session
session_name('SessionName');
session_start();

// Connections & database handling
$ldapConnection = new LdapConnectionHandler();
$serviceDeskRestHandler = new ServiceDeskRestHandler();
$databaseConnection = new DatabaseConnectionHandler();

if (empty($_SESSION['userName'])) {
    session_unset();
    session_destroy();
    session_name('SessionName');
    session_start();
}

$_SESSION['userName'] = '';
if (empty($_SESSION['userName']) || empty($_SESSION['userDisplayName'])) {
    $_SESSION['userName'] = getUsername();
    $_SESSION['userDisplayName'] =
        $ldapConnection->getDisplayname
        (Sanitize::username($_SESSION['userName']));
    $_SESSION['admin'] =
        $ldapConnection->isAdmin
        (Sanitize::username($_SESSION['userName']));
}

$userName = Sanitize::username($_SESSION['userName']);
$userDisplayName = Sanitize::username($_SESSION['userDisplayName']);
$admin = Sanitize::safe($_SESSION['admin']);
```

5.3 Navigation and language management

The application display language gets determined by the session variable named *lang* which is a URL parameter. If this session variable is not set by the user then the case might be that the user has set the preferred language in a previous session, thus saving the language setting in a cookie in the user's browser. If the language setting cannot be found in neither of the variables, then the application will default to English.

Page and language parameter example.

```
index.php?page=main&lang=en
```

In the code example above, the user would be served the main page in English. The strings for each language are stored in separate language configuration files containing multidimensional arrays.

Example language configuration, a string stored in a multidimensional array.

```
$lang['en']['launcher']['supportRequest']['header'] = 'Support request';
```

Code example 15. index.php determining the language of the application.

```
if (!empty($_GET['lang'])) {
    $langSelected = Sanitize::safe($_GET['lang']);
    setcookie('lang', $langSelected, time()+60*60*24*365*10, '/');
}
else if (!empty($_SESSION['lang'])) {
    $langSelected = Sanitize::safe($_SESSION['lang']);
}
else if (!empty($_COOKIE['lang'])) {
    $langSelected = Sanitize::safe($_COOKIE['lang']);
}
else {
    $langSelected = 'en';
}

switch ($langSelected) {
    default:
    case '':
    case 'en': $langSelected = 'en'; break;
    case 'se': $langSelected = 'se'; break;
    case 'fi': $langSelected = 'fi'; break;
}

$_SESSION['lang'] = $langSelected;
```

Page routing is achieved with a URL parameter as shown in the *Page and language parameter example*.

If this parameter is set, it is sanitized to prevent any malicious code injections. Then the user gets redirected to the requested page, stated in the switch case. Some pages have a script associated with them. This is to minimize the amount of JavaScript that each page must load in. However, if the parameter is not set the main page is shown. Both administrators and normal users have access to the main page.

Code example 16. index.php determining which page to serve to the user.

```
// Get page
if (empty($_GET['page']) == false) {
    $page = Sanitize::safe($_GET['page']);
}
else {
    $page = 'main';
}

$script = '';
$pageName = '';

// Get page .php file
switch ($page) {
    default: $pageFile = 'error.php'; break;

    case 'main': $pageFile = 'launcher.php';
    $pageName = $lang[$langSelected]['launcher']['navbarText']; break;

    case 'ticketOverview': $pageFile = 'ticketOverview.php';
    $pageName = $lang[$langSelected]['ticketOverview']['navbarText'];
    $script = 'ticketOverview.js'; break;
}

if ($script != '') {
    echo '<script src="js/pageJs/'.$script.'"></script>';
}
}
```

5.4 Data fetching

Data in the application is fetched through the service desk REST API. To handle these requests in the application, a handler class is used. The handler class has several methods for fetching different data from the API. To be able to display details about specific tickets, a handler class method is used that returns an array of all the ticket ID's that matches the search query give in the request.

Example of a search query for tickets having a closed state:

```
https://webserviceurl.com/api/&StateType=closed
```

Example response containing an JSON array of the ticket ID's whose state is closed:

```
[ '101', '102', '103', '104' ]
```

Fetching details of the tickets is done by using a handler class method which can fetch ticket details in batches. The reason for this is that the API cannot handle a request that consists of all the ticket ID's in the system all at once. The request containing the ticket ID's would be too large. The limitation here is due to the maximum character length that the web service can handle in the request URL.

The API can safely handle the fetching of details for 500 tickets per request. When requesting ticket details the application sends a request to the API containing a maximum of 500 ticket ID's. The API in return responds with the detailed information about the requested tickets. This recursive data fetch is a better way of fetching ticket details, than to request details for one ticket at a time.

5.5 Service desk portal

The main page of the application is called *service desk portal*. It serves as a tool to the end user for filing new tickets to the service desk. There are categories for each type of ticket that the user can choose from in an accordion menu. A general service ticket can be categorized into one out of four different severity queues.

The severity queues have two deadlines associated with them, first response time and resolution time. Deadline times are relative to the creation time of a ticket. These deadlines serve as goals for the service desk agents solving the tickets.

First response time marks the time when the ticket should have been replied to and the work solving the ticket would have begun. Resolution time is the total amount of time that it should take to solve a ticket in a severity queue. It is possible to file more specific tickets such as access right changes, purchases, user account management and permission changes. These tickets will have the same deadline times as severity queue three, which serves as the default severity queue that should be used in most cases.

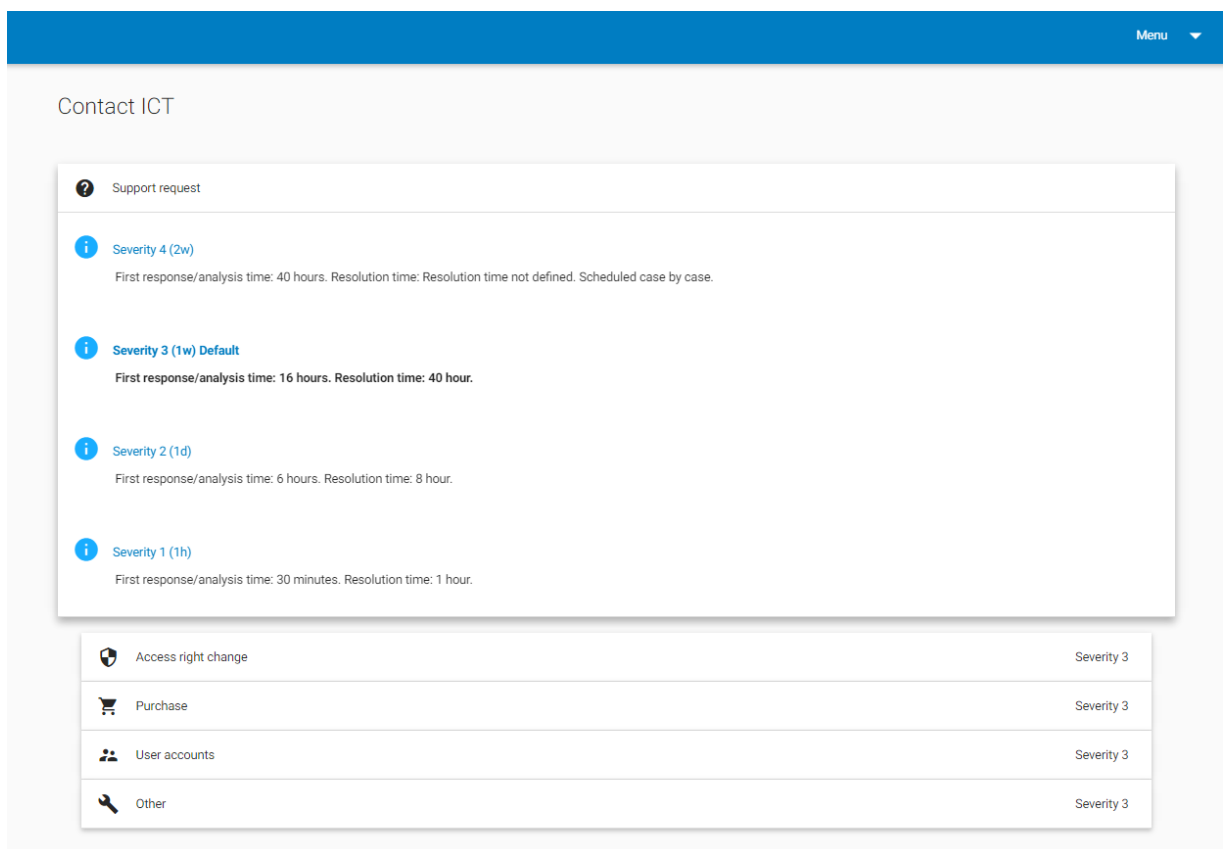


Figure 4. The main page of the application, used to file new tickets.

The deadline times for the severity queues are the following:

Table 3. Severity response and resolution times.

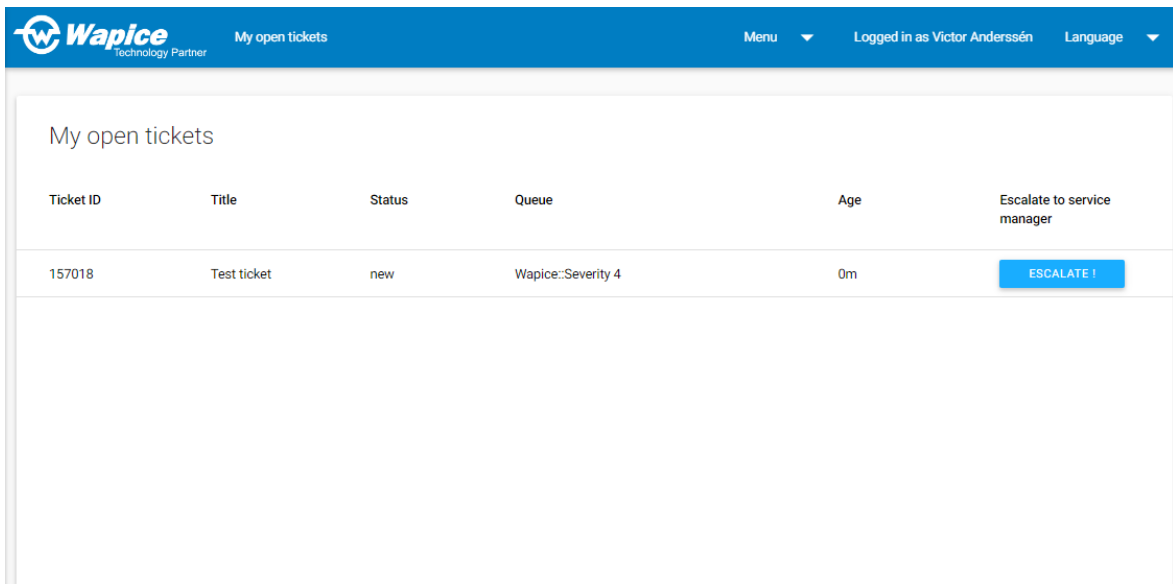
Severity 1	First response time: 30 minutes. Resolution time 1 hour.
Severity 2	First response time: 6 hours. Resolution time 8 hours.
Severity 3 (Default)	First response time: 16 hours. Resolution time 40 hours.
Severity 4	First response time: 40 hours. Resolution time is not defined. Scheduled case by case.

If the user has open tickets in the service desk, then the five newest open tickets are displayed at the bottom of the main page.

5.6 Open tickets

The *My open tickets* page shows all the open tickets belonging to the user. The tickets are displayed in a table format with cells displaying the tickets internal service desk ID, ticket title, status, queue, and age since creation. For each ticket, there is the possibility to escalate the ticket to a service manager if the resolution of the ticket is taking too long and the problem that the ticket should resolve is urgent and important.

The user can also access their closed tickets from the main menu. This redirects the user to the existing service desk customer front-end. This section was not implemented as a separate page in this web application, but it is considered as a future implementation.



Ticket ID	Title	Status	Queue	Age	Escalate to service manager
157018	Test ticket	new	Wapice::Severity 4	0m	ESCALATE !

Figure 5. My open tickets page, displaying one test ticket.

5.7 Ticket escalation

From the *My open tickets* page the user can escalate tickets to a service manager or duty officer. The escalate button opens a modal with a text area where the user should describe the problem and why it is critical that the ticket gets solved as fast as possible. This system is in place in case of tickets that are filed because of critical problems, not being resolved in a reasonable time. Finally, the user confirms the escalation by clicking on the escalate button in the modal. The modal text submission is handled with Ajax as shown in the example below.

Code example 17. Ticket escalate functions.

```

/* Extract data from the submitted modal */
function escalateTicket(ticketId){
    $("ticket").val = ticketId;
    $("escalate-text-message").val = $("escalate_placeholder").val;
    $('#escalate-modal').openModal();
}

/* Post data */
function escalateTicketConfirm() {
    var ticketId = $("ticket").val;
    var message = $("escalate-text-message").val;

    $.ajax({
        data: { ticketId: ticketId, message: message },
        url: 'escalation_tools.php?action=escalate',
        method: 'POST',
        dataType: "html",
        success: function(data) {
            $('#escalate-modal').closeModal();
            location.reload();
        },
        error: function(jqXHR, textStatus, errorThrown) {
            console.log('escalateTicket: Could not post any data to
            escalation_tools.php?action=escalate');
            console.log('jqXHR: ', jqXHR);
            console.log('textStatus: ', textStatus);
            console.log('errorThrown: ', errorThrown);
        }
    });
}

```

The Ajax POST request gets sent to the script *escalation_tools.php* where it gets processed and the escalated ticket gets updated with a new article telling that the user has escalated the ticket. The ticket also gets moved to a separate queue for escalated tickets.

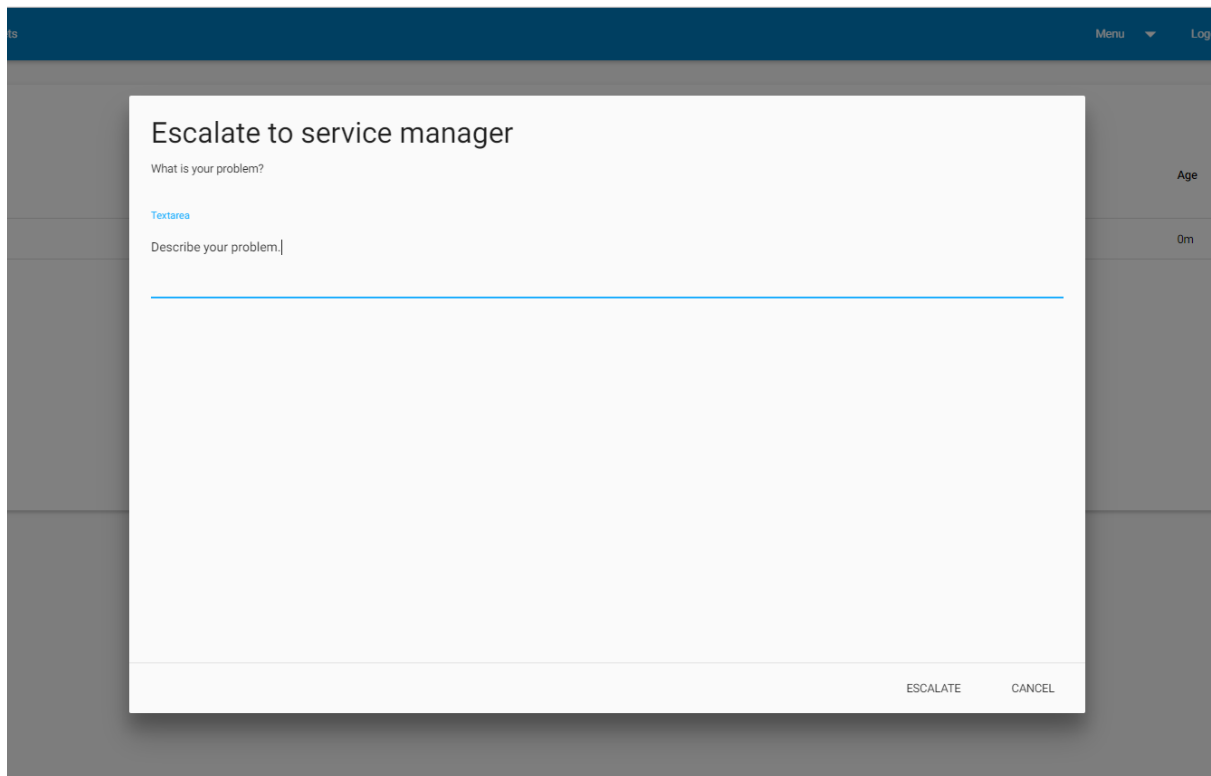


Figure 6. Escalate to service manager modal in the *My open tickets page*.

When the ticket gets escalated, a duty officer gets an SMS about the escalation. The SMS is sent by a script that was in place before this application was being developed. The duty officer can be set from the administrator configuration page in the web application. The page displays a list of persons belonging to a duty officers LDAP group. The assignment of a new duty officer is done by selecting a new person from the list and confirming the change by pressing set.

Figure 7. Duty officer assignment page.

Table 4. Database table of the current duty officer.

NewDutyOfficer	TimeUpdated	ID	UserWhoUpdated
PersonName2	2017-03-17 13:24:22	1	PersonName1

The current duty officer is stored in one row in a database. When a new duty officer is assigned by an administrator from the duty officer page the row gets updated.

Duty officer table columns:

- NewDutyOfficer – Who is assigned as the new duty officer.
- TimeUpdated – The time when the update took place.
- ID – Primary key, never changes.
- UserWhoUpdated – Who did the duty officer change?

5.8 Charts

The charting library NVD3.js was used to visualize data. It is based on the D3.js library. It can be installed and saved as a project dependency with NPM using the following terminal command:

```
npm install --save nvd3
```

5.8.1 Ticket overview charts

The ticket overview page consists out of two line charts. The first one displays new and closed tickets for the past month. The second chart displays new and closed tickets cumulatively since the deployment of the service desk system. Both chart have a horizontal timeline beneath. The timeline determines the size of the time axis that is used in the chart. The timeline can be changed by dragging its starting and ending points.

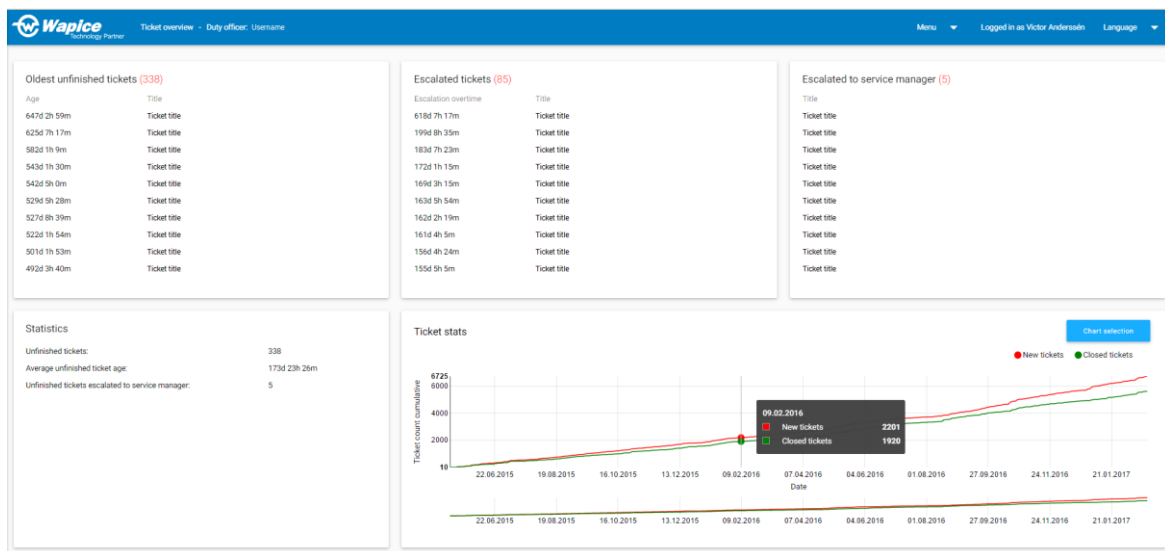


Figure 8. Ticket overview page.

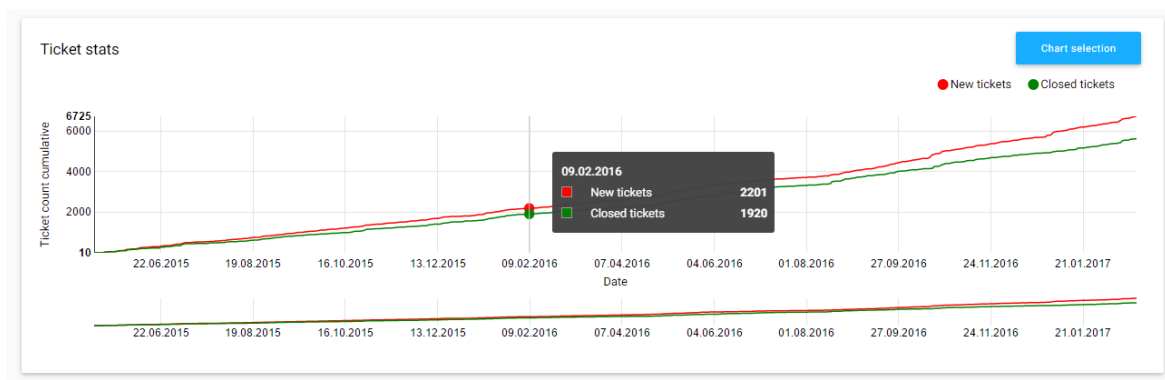


Figure 9. Ticket overview page, cumulative chart displaying new and closed tickets, since deployment of the service desk system.

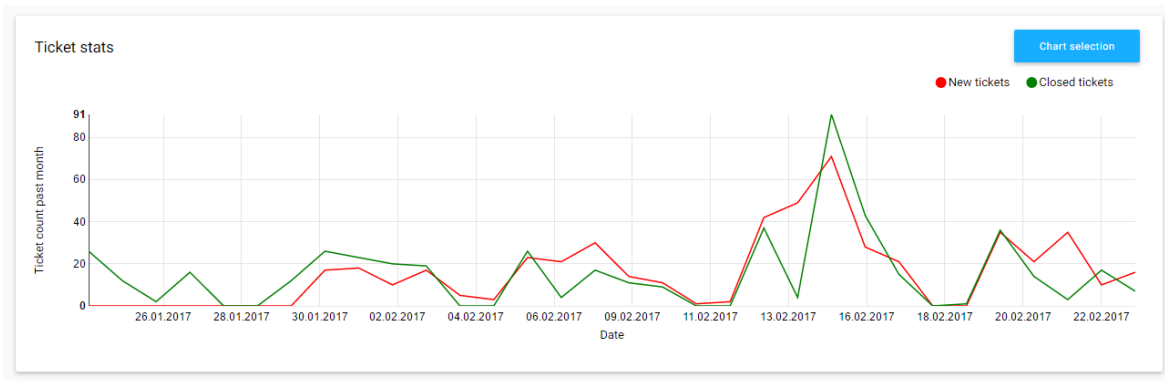


Figure 10. Ticket overview page, chart displaying new and closed tickets past month.

5.8.2 Ticket queues charts

The ticket queues page consists out of four scatter charts with the same design, but different data. These charts display all the tickets belonging to a severity queue, there are four severity queues in total. The tickets are displayed as circles. If the user hovers over a circle, additional ticket information is displayed. The information displayed in the hover state is the ticket ID, the ticket age, the queue that the ticket belongs to and the ticket title. One severity chart may contain tickets from multiple queues. The charts are updated in real time with Ajax calls. If a circle is clicked, it redirects the user to the internal service desk page containing all information about the ticket.

The circles may have one out of three possible colors. When a ticket or circle is created, it is initially displayed in a green color. This due to the ticket's age has not surpassed the first response time, which is displayed as a horizontal yellow line. When the tickets age is between the first response time line and resolution time line, the ticket is displayed in a yellow color. Finally, when the tickets age surpasses the response time it is displayed in a red color. However, if a ticket gets successfully closed at any time, the ticket or circle is automatically removed from the chart.

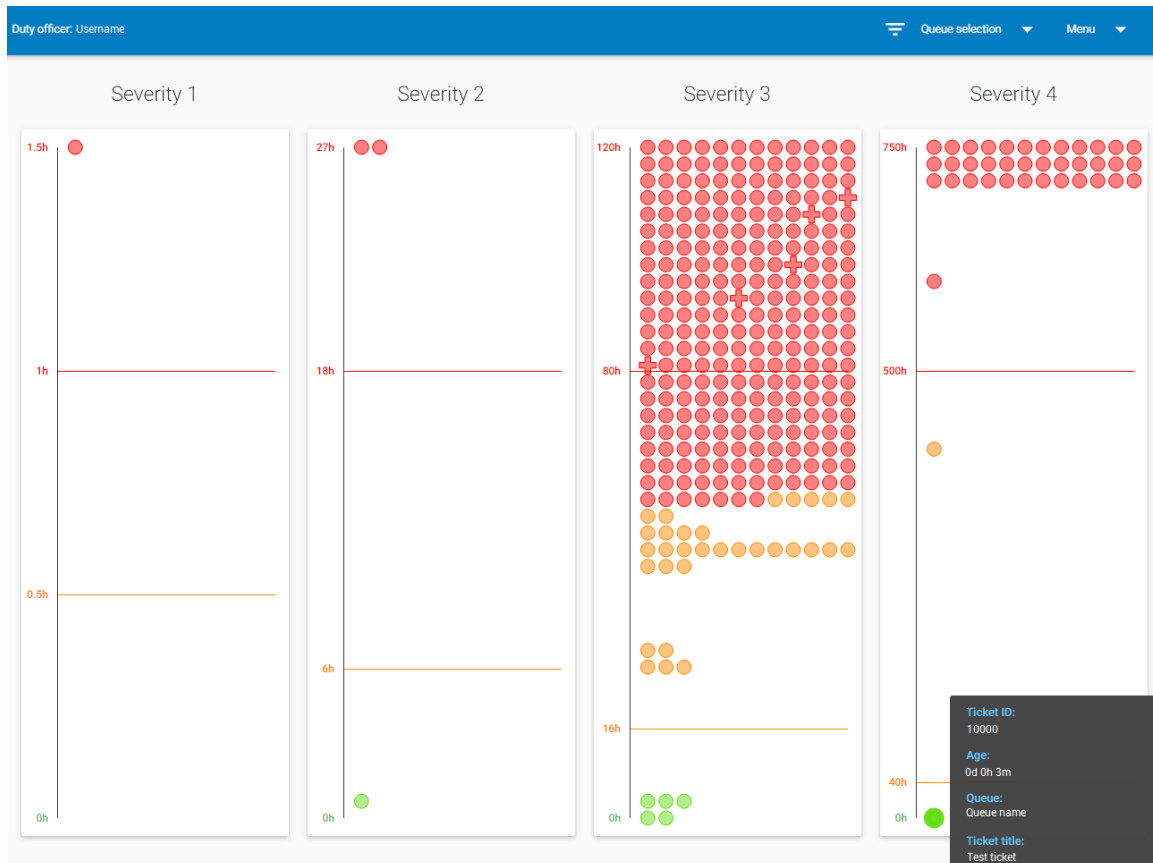


Figure 11. Ticket queues page, displaying tickets as color coded circles.

5.8.3 Conversion of data for the ticket overview charts

The *ticket count past month* chart uses date time as its x-axis. The y-axis displays the total amount of new and successfully closed tickets for a given date. The second chart *ticket count cumulative*, displays all new and successfully closed tickets cumulatively on a given date since the deployment of the system.

For the *ticket count past month* chart, only the data from the past month needs to be fetched from the API. This is done by passing in a parameter into the request that filters out all tickets that have a created at timestamp older than 50000 minutes. For the *ticket count cumulative* chart, the parameter is not used. This way all the tickets are fetched.

Since the charts display the newly created tickets per day and the successfully closed tickets as separate data series on their own lines, the requests are separated for each of the data series.

The difference between these two requests is the API parameter called *StateIDs* which determines which ticket state the returned tickets will have. For new tickets *StateIDs=1* is used and for successfully closed tickets *StateIDs=2* is used.

Code example 18. Fetching ticket details for successfully closed tickets from the service desk API.

```
$searchString = 'StateIds=2&TicketCreateTimeNewerMinutes=50000';  
  
// Get all ticketID's for the successfully closed tickets the past month  
$ticketIDArray = array();  
$ticketIDArray = $serviceDeskRestHandler->searchTickets($searchString, False);  
  
// Get the details for all the tickets in $ticketIDArray  
$allTicketDetails = $serviceDeskRestHandler->getTicketDetailsMultirequest($ticketIDArray);
```

Once the ticket IDs and the ticket details have been fetched, an array containing a key named *x* and the ticket date time value in the format Y-m-d is created. This array can contain multiple occurrences of the same date time value.

Code example 19. Creating a temporary array containing the created date time of the tickets.

```
// Extract the dates
$ticketDetailsArray = array();
foreach ($allTicketDetails['Ticket'] as $ticket) {
    // Take away the H:m:s from the date times
    $time = strtotime($ticket->Created);
    $newformat = date('Y-m-d', $time);
    array_push($ticketDetailsArray, array("x" => $newformat));
}
```

A temporary array is created containing a key value pair. The key is *x* and the value is a date time in the format Y-m-d. The length of the array is 31, the maximal amount of days in a month. This array length determines the amount of days that the chart is finally going to display. The value *y* gets initiated to the integer 0 for all array entries. This value is a counter of how many of the tickets fetched that have the same date time.

Code example 20. Creating an array of date times ranging from today back 31 days.

```
// Create an array of datetimes.
$weekdaysDatetimeArray = array();
for ($i = 31; $i >= 0; $i--) {
    array_push($weekdaysDatetimeArray, array("x" =>
    date('Y-m-d', strtotime('-'.(int)$i.' day')), "y" => 0));
}
```

The final array is created by counting the occurrences of the same date time in *Code example 18* and incrementing the counter by one for each occurrence. Finally, the key *x* containing the date time gets formatted to a UNIX timestamp in milliseconds. The reason for this is that the charting library expects date time values in this time format.

The process differs for the *ticket count cumulative* chart and the array length is the total count of all days since system deployment. The *y* value gets cumulatively added for each day.

Code example 21. Counting how many tickets that have the same date time for the past month chart.

```

$arrayIndex = 0;
foreach ($weekdaysDatetimeArray as $row) {
    $counter = 0;
    for ($i = 0; $i < count($ticketDetailsArray); $i++) {
        if ($row['x'] == $ticketDetailsArray[$i]['x']) {
            $counter++;
        }
        else {
            // Do nothing -> try a new row in $ticketDetailsArray
        }
    }
    // Add the count to the correct row in $weekdaysDatetimeArray
    $counter = (int)$counter;
    $weekdaysDatetimeArray[$arrayIndex]['y'] = $counter;
    $arrayIndex++;
}

// Format time to UNIX timestamps
$count = count($weekdaysDatetimeArray);
for ($i = 0; $i < $count; $i++) {
    $datetime = $weekdaysDatetimeArray[$i]["x"];
    $unixtimestamp = (strtotime($datetime))*1000;
    $weekdaysDatetimeArray[$i]["x"] = $unixtimestamp;
}

$jsonEncoded = json_encode($weekdaysDatetimeArray);
echo $jsonEncoded;

```

Code example 22. JSON output from the calculations where the *x*-value is the date time in milliseconds since the UNIX epoch. The *y* value is the total count of tickets on that date.

```

[ {
  "x": 1485208800000,
  "y": 2
}, {
  "x": 1485295200000,
  "y": 5
}, {
  "x": 1485381600000,
  "y": 0
} ]

```

5.8.4 Conversion of data for the ticket queues charts

Data for the four charts is comprised of x and y values. Where the y value is a relative value between the created at time and the resolution time * 1.5. The reason for the resolution time being one and half times greater is so that if there are many tickets which have surpassed the resolution time. The tickets would have some extra graphical space in the top of the chart where they could be displayed, otherwise the red resolution time line would be at the very top of the chart.

The x value is a generated value between one and twelve. The x value determines the circles position in the x -axis. There can be a maximum of twelve tickets in one chart that have a similar y -value that aligns them to the same line in the y -axis. These tickets would get distributed in the x -axis and assigned x -values from one to twelve. The positioning system of the circles is dynamic in the sense that it will rearrange all circles when an update of the chart occurs. If new circles are added or if circles are removed, then the position of all circles will get re-evaluated to display them all correctly.

5.8.5 Ticket queues selection

If the user would like to filter out certain queues from any of the charts in the *ticket queues* page, they can do this through the queue filter menu at the top of the page. This menu shows all the available queues that belong to the four severity queues. The users filter choice is stored in a cookie, so that the same filter will be applied the next time the user visits the page. To change the filter setting the user marks checkboxes for the desired queues and finally applies the changes with the *change queues* button.

Due to the large number of available queues to choose from in the filter menu, there are buttons in the queue selection menu for checking or unchecking all queues. From the start, there is a default filter configured. The user can revert to this filter by clicking on the defaults button.

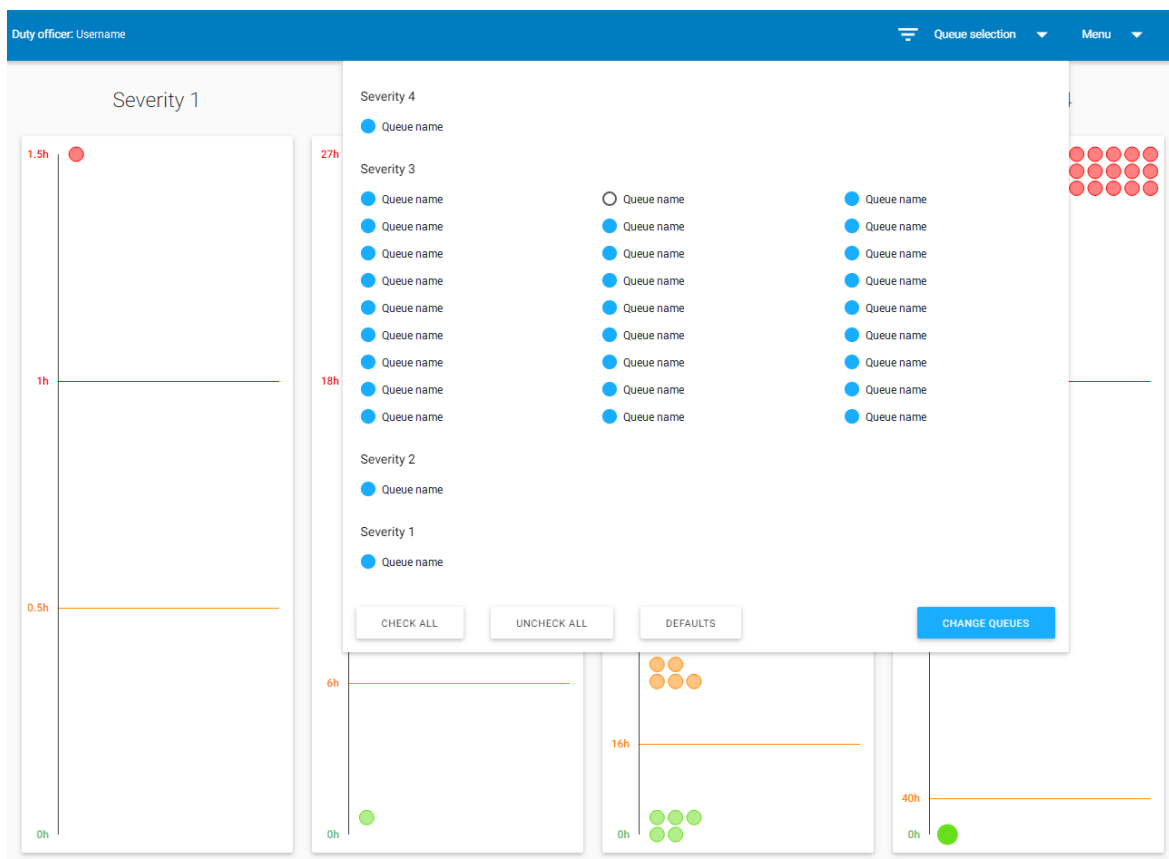


Figure 12. Ticket queues page filter menu.

If the cookie is not set and the queues currently displayed are the predefined queues from the main configuration. Then the script fetches the default queues from the main configuration. This scenario would take place if the user has not chosen any filter and the page gets refreshed.

If the queues currently displayed are the same as in the main configuration and the user's cookie is set. This scenario would take place when the user initially visits the page. Then the script fetches the *queueID's* from the user's cookie and parses them and the script proceeds with updating the charts with the queues specified by the user's cookie. The cookie can be set with no queues in it. This would make all the charts display no data.

Queues chosen in the filter by the user, gets passed to from the JavaScript handling the filter menu to the PHP script that fetches and parses new data based on the filter selection. the PHP script starts off by checking if the queues fetched are not empty and different from the default queues in the main configuration of the web application. If this is true, the script continues updating the charts with the queues specified by the user. This scenario would take place when the user would have chosen a new filter and confirmed it.

6 Result and discussion

This chapter consists of the results, problems encountered and discussion.

6.1 Result

The resulting application fulfilled the requirements. The application was taken into production when it was finished. The application has enabled the service desk agents to prioritize tickets more easily by having access to the graphical representations of the tickets, thus the service desk system has become more efficient since users do not have to wait as long for their tickets to get solved. Due to the new analytics tools such as the ticket charts and status view, it is now possible to spot problems faster than before.

6.2 Problems encountered

The largest problem encountered in the development process was the lack of documentation on how the deployed service desk system worked. It was quite difficult to fetch large amounts of data from the service desk web service without any significant delay. The solution to this problem was to divide the data to be fetched into smaller chunks. This way it was possible to recursively fetch the ticket ID's and then the details for each ticket.

6.3 Further development

A feature that could be implemented, is the displaying of the users closed tickets instead of redirecting the user to the service desk customer frontend.

General optimizations to the recursive queries for fetching ticket details should be done to improve the overall user experience and to minimize loading times. The limitations of API request optimization, may lay in the capabilities of the API itself.

The Gulp.js workflow could further be improved. A task for automatic minification of JavaScript files could be added. To save some prototyping time, live reload could be used in a Gulp task. Live reload refreshes the web browser whenever a file changes, so that the change in code is instantly shown in the browser. This enables faster iteration of design ideas.

6.4 Discussion

At the start of this project, PHP was chosen as the main language for the application backend. It was used without any framework. It could have been better considering future feature implementations to have used a small framework to structure the application in a more modular and easily upgradable way, for example by using the Slim framework. Although it has been easy to implement new functions in the application, due to the backend that has been custom written for this application.

The JavaScript charting library chosen was NVD3.js while it provides easy to use standard charts. The library itself is not the best choice when it comes to developing custom charts, such as the ticket queues chart in this thesis. A better choice for building custom charts would have been the library D3.js on which NVD3.js is based on. D3.js has its own disadvantages one being that its learning curve is quite steep compared to simpler alternatives.

7 References

- [1] Wapice Ltd, "Wapice Ltd.," [Online]. Available: <https://www.wapice.com/about-us/wapice>. [Accessed 27 October 2016].
- [2] W3C, "HTML5," 28 October 2014. [Online]. Available: <https://www.w3.org/TR/html5/>. [Accessed 26 October 2016].
- [3] N. Weizenbaum, C. Eppstein and H. Catlin, "Sass Basics," [Online]. Available: <http://sass-lang.com/guide>. [Accessed 25 October 2016].
- [4] Mozilla Developer Network and individual contributors, "JavaScript Guide - Introduction," 5 July 2016. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction>. [Accessed 2 November 2016].
- [5] N. Foundation, "About Node.js®," [Online]. Available: <https://nodejs.org/en/about/>. [Accessed 20 February 2017].
- [6] Tutorialspoint, "jQuery - Ajax," 2016. [Online]. Available: <https://www.tutorialspoint.com/jquery/jquery-ajax.htm>. [Accessed 25 10 2016].
- [7] W3Schools, "How AJAX Works," [Online]. Available: <https://www.w3schools.com/php/ajax.gif>. [Accessed 20 February 2017].
- [8] php.net, "What can PHP do?," [Online]. Available: <http://php.net/manual/en/intro-whatcando.php>. [Accessed 25 10 2016].
- [9] B. Ajzele, Modular Programming with PHP 7, Birmingham: Packt Publishing, 2016.
- [10] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000. [Online]. Available: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. [Accessed 9 March 2017].
- [11] W3CSchools, "JSON - Introduction," [Online]. Available: http://www.w3schools.com/js/js_json_intro.asp. [Accessed 25 10 2016].
- [12] UnboundID, "Basic LDAP Concepts," [Online]. Available: <https://www.ldap.com/basic-ldap-concepts>. [Accessed 31 January 2017].
- [13] Shibboleth Consortium, "How Shibboleth Works: Basic Concepts," [Online]. Available: <https://shibboleth.net/about/basic.html>. [Accessed 31 January 2017].
- [14] OneLogin Inc, "Dev Overview of SAML," [Online]. Available: <https://developers.onelogin.com/saml>. [Accessed 31 January 2017].

- [15] Google, "Material design - Introduction," December 2016. [Online]. Available: <https://material.io/guidelines/>. [Accessed 20 February 2017].
- [16] A. Wang, A. Chang, A. Mark and K. Louie, "About," 2016. [Online]. Available: <http://materializecss.com/about.html>. [Accessed 23 November 2016].
- [17] npmjs.com, "What is npm?," 4 November 2016. [Online]. Available: <https://docs.npmjs.com/getting-started/what-is-npm>. [Accessed 29 November 2016].
- [18] Gulp, "gulp API docs," 13 May 2016. [Online]. Available: <https://github.com/gulpjs/gulp/blob/master/docs/API.md>. [Accessed 29 November 2016].
- [19] Awio Web Services LLC, "Browser & Platform Market Share January 2017," 31 January 2017. [Online]. Available: <https://www.w3counter.com/globalstats.php?year=2017&month=1>. [Accessed 21 February 2017].
- [20] A. Aziz and S. Mitchell, "An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET," February 2007. [Online]. Available: <https://msdn.microsoft.com/en-us/library/bb299886.aspx>. [Accessed 20 February 2017].