Masaryk University
Faculty of Informatics

A Mobile Application for the Administration of the Kentico System

Bachelor's Thesis

Linda Hansliková

Brno, Fall 2016

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Linda Hansliková

Advisor: Bruno Rossi, Ph.D

Acknowledgement

My thanks go to Brunno Rossi and Marek Fešar my advisors for allowing me to proceed with this topic and their advice and patience. And also to my friends Matej Pavla, Jozef Vilkolák and Roman Mačor for their support and their help with proofreading this text.

Abstract

In a time where time is more precious than money it is crucial for people to accomplish a task as quick as possible. When creating various web-sites, the Kentico Enterprise Marketing Solution (KEMS) is a helpful tool to safe time and therefore money. KEMS is a content management system (CMS) which allows clients to create and manage their web-sites using a single user interface (UI). This thesis is about adding an extension to the said system which allows administrators to administrate their site from their smartphones. The functionality implemented should reflect the basic needs of an administrator of the KEMS. The extension consists of two parts: the custom web application programming interface (API) and the mobile application (app). The custom web API was used to call the Kentico API (KAPI) and retrieve data and the mobile app was used as a gateway for the user and the custom web API (CAPI).

Keywords

Mobile, Mobile Application, Kentico, Javascript, JQuery, WebAPI, Apache Cordova

Contents

List of Figures

1 Introduction

KEMS is a content management system (CMS) which allows clients to form and manage their web-sites using a single user interface (UI). In this thesis we created a mobile app called KenticoApp which calls an API that we also developed by extending the API of KEMS. An API is a collection of functionality which a programmer is able to utilise in a third party app. The KenticoApp makes it possible for clients to manage their site from their smartphones. It consists of two parts: the CAPI backend, which stores and retrieves data from and to the database, and the mobile client app, which allows the user to communicate with the system. The functionality is divided into three main categories. The first category represents the system tasks such as restarting the server, cleaning unused memory or cache and reading the event-log or general system information. The second one operates with the users and their roles. It offers the editing of the user's first and last name and adding or removing their roles. The third and last category makes it possible to create or delete roles and edit them by adding or removing permissions. To be able to perform all of the above actions the user has to be authenticated and authorized first. The authentication credentials are checked against the KEMS database using KAPI. Only global administrators are authorized.

The backend was implemented in C# .NET and communicates with the KAPI. The mobile client app is a Cordova app written in JavaScript, HyperText Markup Language (HTML) and Cascading Style Sheets (CSS). The communication is ensured by asynchronous JavaScript and Extensible Markup Language (Ajax) in the format JavaScript Object Notation (JSON). For the purpose of version control and backup we decided to use a technology called Git. Our Git project was hosted on the web-based Git repository hosting service called GitHub. GitHub is an industry standard for hosting open-source software source code.

Chapter one introduces KEMS, web API and hybrid mobile applications. In the second chapter we describe the application architecture and the implementation of the extension of the KEMS in more detail. Finally, we valorise the achieved result and suggest other potential extensions or solutions.

2 Analysis

2.1 Section 1

Example citation[2] [1] *Example italic* Example reference to other section 2.1 Example of escapes $ %

Example paragraph

2.2 Section2

Section 2...

Eample enumerate:

1. item 1
2. item 2
3. item 3

Example desciption:

Term1 description term 1..

Term2 Tdescription term 2..

```
Example code sample
```

This text references image 2.1

2.3 Kentico CMS

2.4 Web Application Interface

2.5 Hybrid Mobile application

1. Example footnote

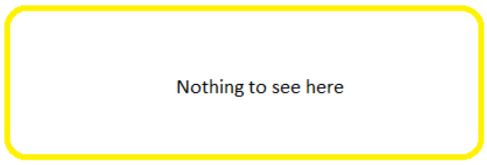

Figure 2.1: This is an empty example image

3 Implementation

3.1 Application Overview

This thesis consists of two parts. The first of which is the CAPI backend. It stores and retrieves data from and to the database via calls to the KAPI. It itself is called from the mobile client app, called KenticoApp, through which the user is able to communicate with the system and manage his site.

The CAPI partially follows the representational state transfer (REST) architecture by using appropriate hyper text transfer protocols (Http). We use for example POST requests for creating or GET requests for reading resources from the backend. The usage of status codes, such as 200 for OK, 403 for forbidden or 503 for service unavailable, is also a RESTful convention. Our backend is stateless. This is achieved by using ATs instead of storing the user session across multiple Http requests. One of the reasons why we cannot call this application RESTful is it does not follow the fundamental concept of identifying all resources and relationships between them. For example our *System* "resource" contains the method *ClearCache()* and *ShowEventlog()*. These should be identified in separate resources *CacheClearer* and *Eventlog*.

The communication between the CAPI and KenticoApp is ensured by Ajax using the JSON format. It is an effective way to broadcast information via a simple string.

3.2 Extending Kentico

3.2.1 Custom Kentico Module

The CAPI was created using the .Net framework. It uses KAPI calls and is called by the KenticoApp. For executing an API call, the user has to be signed into the system and have the proper authorization. TODO:

3.2.2 Kentico 9.0 API

3.3 Web API Application

3.3.1 Microsoft Web API 1.0

TODO:API Controller, Filters, Recieving and sending response (JSON), REST: HttpError Codes, stateless, token, not restful The API call structure is demonstrated in the illustrated code below.

```
[Authorize]
[HttpPost]
[Route("kenticoapi/users/edit-user")]
public HttpResponseMessage EditUser([FromBody]JObject
    postData)
{
  string username, firstName, surname;
  try
  {
    username = postData["username"].ToObject<string>();
    firstName = postData["firstName"].ToObject<string>();
    surname = postData["surname"].ToObject<string>();
  }
  catch (Exception e)
  {
    return Request.CreateResponse(HttpStatusCode.
        ServiceUnavailable, new { errorMessage = e.Message
         });
  }
  try
  {
    UserInfo updateUser = UserInfoProvider.GetUserInfo(
        username);
    if (updateUser != null)
    {
      // Updates the user's properties
      updateUser.FirstName = firstName;
      updateUser.LastName = surname;

      // Saves the changes
      UserInfoProvider.SetUserInfo(updateUser);
    return Request.CreateResponse(HttpStatusCode.OK, new
        { user = updateUser });
    }
  } catch(Exception e)
31    {
32      return Request.CreateResponse(HttpStatusCode.
          ServiceUnavailable, new { errorMessage = e.Message
          });
33    }
34    return Request.CreateResponse(HttpStatusCode.
        ServiceUnavailable, new { errorMessage = "User␣is␣
        null" });
35  }
```

The annotation from the 1st is noted either in front of a particular method, or in front of a whole controller so that all it's methods are affected. It which was implemented as our custom AuthenticatorFilter and checks if the user is authenticated so the call can be executed. If successfully authorized, the user is stored into the request properties from where he can be retrieved with the following command:

```
UserInfo user = (UserInfo) Request.Properties["
   LoggedUserInfo"]
```

as it is done in the method GetCurrentUser(). Line 2 ensures that only POST requests are handled by the method. POST requests send data from the client to the server as opposed to GET requests which demand data from the server. In this example the system stores updated user information from the KenticoApp into the database. The 3rd line represents the route where the call can be excessed through the client app. The 4th line is the head of the method. It's return type enables the client to receive a StatusCode and a value, which is the content of the Http response message. The parameters are passed on from the client as one object in the JSON format. On the lines 6 to 12 the JSON object is parsed into separate parameters as strings. This is done in a try-catch block to handle possible exceptions and return the wished response message on the line 15. The CreateResponse() method is of the class Request and it's parameters are the status code 503 service unavailable and an object with the error message of the caught exception from line 13. The line 19 gets the user with the parsed username and stores it in the variable called updateUser of the type UserInfo. This type is defined in the KAPI documentation and has attributes such as username, user ID, user first and last name, etc. Line 20 checks if the updateUser is not null. Lines 23 and 24 change the updateUser's first and last name. On the line 27 the updateUser is inserted in the database. Line 28 returns the status code 200 OK and the updateUser object in JSON format. The lines 30 to 33 are similar to lines 13 to 16. If updateUser is null the response is status code 503, the same as on line 15, and the error message"User is null".

3.3.2 CAPI Token Management

For user authentication we decided to use access tokens (AT). ATs are leveraged to secure the communication between a user and the system. After signing in the user is given a random generated unique AT by the system which stores it in it's database. Before every API call, the system requires the user's AT and then checks it against the database. For the call to be executed the AT has to exist in the database with the corresponding user ID and must not be expired. If this is not the case the user is redirected to the welcome page, where he has to sign in. To represent and store the ATs in the database in our project we were inspired by the layered application design pattern, more specifically by it's data access layer (DAL). This pattern is used to ensure security and scalability of an application by partitioning it into three layers. The first and lowest layer is needed to operate the database called DAL, it represents entities. The next layer is the business logic layer which contains the logic of the system and the last one is the presentation layer utilised to display the application through a UI to users. For the purpose of this thesis we decided to represent the ATs as an entity using the Entity Framework. The entity contains the user identification (ID), a unique pseudo-random code and an expiration date and time (expiration) as can be seen in the following example code.

```
public class Token
   {
       [Required]
       public int UserID { get; set; }

       [Required][Key]
       public string Code { get; set; }

       [Required]
       public DateTime Expiration { get; set; }
   }
```

The ID is of the type int and is equal to the user's ID who "owns" the AT. The code is of the type string and is generated with the pseudo-random number generator Random. *The chosen numbers are not completely random because a mathematical algorithm is used to select them, but they are sufficiently random for practical purposes.*[1] Right after generating the code is tested against the database if no AT with the same one exists. If the code is already taken, another one is generated and tested. If not, the token entity is assigned the code, user ID and date and time 10 minutes from the assignment. The expiration is of the type DateTime. After every executed API call the AT's expiration is set to 10 minutes from calling. Before every API call the system searches it's database for expired ATs and deletes them.

3.4 Cordova Mobile Application

3.4.1 Apache Cordova

For the implementation of the mobile app we leveraged the Apache Cordova framework (ACF). The reason being it is less demanding to learn and supports seven platforms. As opposed to the Xamarin framework (XF) supporting only three. Even though XF should be faster than ACF, and therefore offer a smoother user experience, the difference between execution times of non performance sensitive apps on today's devices is negligible. We did not consider development in native languages, such as Android Java or iOS SWIFT, because of their steep learning curve and the ability to deploy only to one platform. The development was divided into two stages. For creating the UI we decided to use JQuery Mobile. It is an HTML5-based user interface framework which allows users to design aesthetically pleasing mobile elements by utilising the languages CSS and HTML. Document object model (DOM) elements are individual parts of a web page described by tags such as div, span, input or others. These tags assign styling and properties to elements. For the DOM manipulation we used the JQuery library which has a small learning curve and offers a fast way to add, modify, style and delete elements or change their behaviour. It also offers a set o handy helper functions which provide easy to use interface for frequently used operations in web development, e.g. Ajax(). TODO: Cordova vs. native vs. Xamarin, PhoneGap, Cordova wrapper

3.4.2 JQuery Mobile

3.4.3 Ajax

TODO: ajax communication with web API

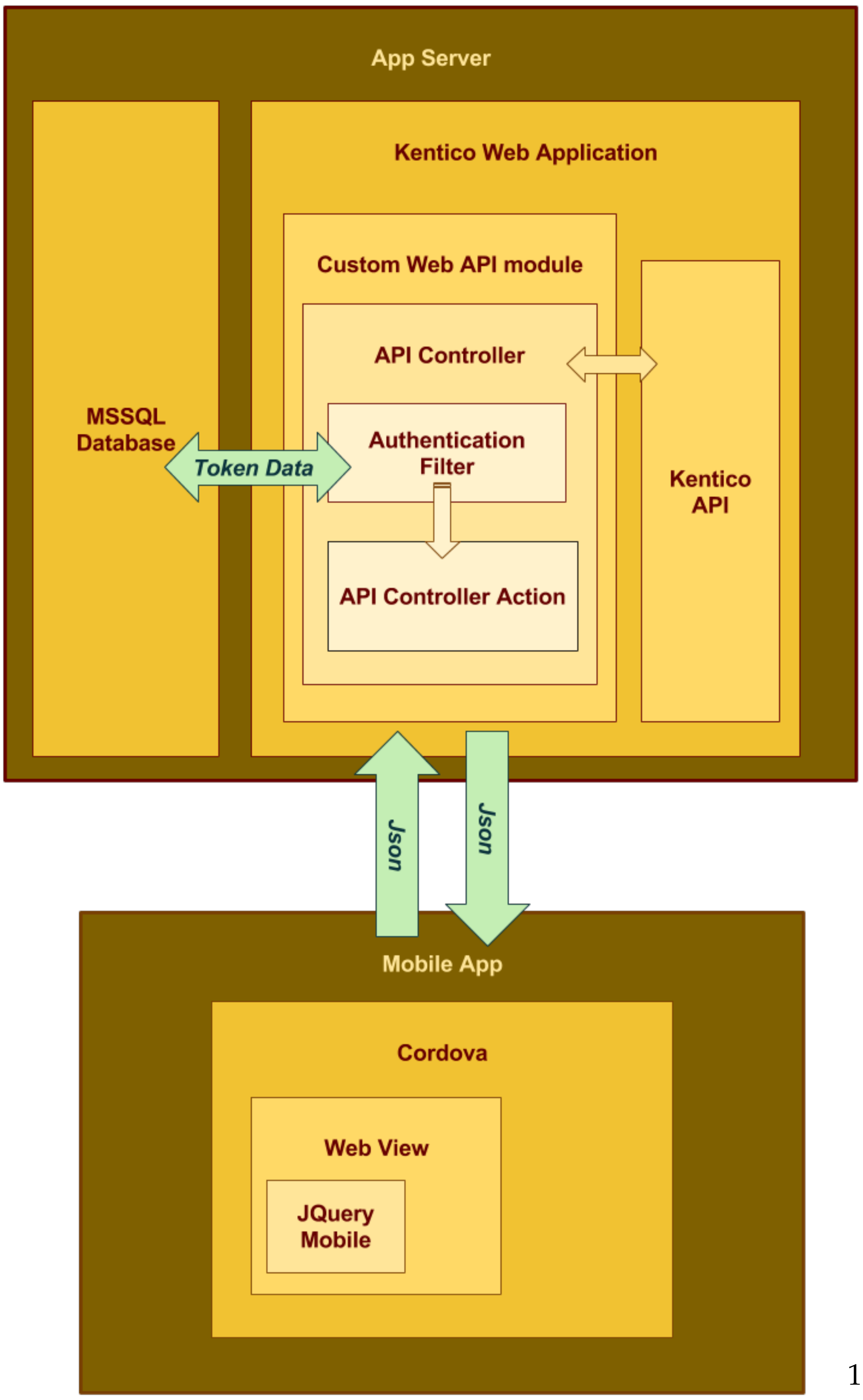

Figure 3.1: Architecture overview

4 Conclusion

4.1 Evaluation

TODO: Functionality

4.2 Future Work

TODO: Ability to choose between available sites on Kentico server, Access control, Security Token, Forgotten Password, polished UI

Bibliography

[1] Msdn documentation - random. `https://msdn.microsoft.com/en-us/library/system.random%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396`. [cit. 2016-12-09].

[2] A Great Man. About great men. *Greatness*, 15(9):100–101, 2345.