U.P.B. Sci. Bull., Series C, Vol. C, Iss. 4, 2018 ISSN 2286-3540

A WEB BASED APPLICATION DESIGN FOR PRODUCT DATA ENGINEERING AND MANAGEMENT

Marian GHEORGHE[1], Aurel TARARA[2]

The informational environment involves a high volume of data transfer between web-based applications in the conditions of user's continuous mobility. This demands to update the applications by implementing new web technologies. The present paper contributes to the design of a web-based application for product data engineering and management, by integrating proper elements from modern web technologies, in order to avoid deployment problems to web servers, to allow flexibility and communication with other applications and hardware. The application exposes a REST API, JWT secured to exchange data with other applications. By means of endpoint, the backend API can be accessed in order to request JWT tokens used to authorize the requests for access to the application resources. The SPA approach reduces the resources usage and navigation/ data load time by means of components reusability and partial page load.

Keywords: PDM, PDEM, API, REST, SPA, JWT, Framework, Application.

1. Introduction

The continuous growth of processed data volume has determined the evolution of the concepts, architectures and web technologies, with a significant impact in industrial environments.

Product Data Management, PDM, as a computer-based system for development information infrastructure and cooperation on product design and manufacture, is part of Product Lifecycle Management, PLM. The Product Data Engineering and Management *Application*, PDEM.A01, is a web-based development application of PDM that improves data reusability and integrates constructive solutions configuration and technological validation tools, to support the analysis and configuration of certain customized product parts [1].

Modern web development technologies have been answered to different requests. A web API represents, at a higher level, a mechanism for code reuse,

[1] Professor, PhD, Doctoral School on Engineering and Management of Technological Systems, TCM Department, University POLITEHNICA of Bucharest, Romania, E-MAIL: marian.gheorghe@upb.ro

[2] PhD Candidate, Doctoral School on Engineering and Management of Technological Systems, University POLITEHNICA of Bucharest, Romania, e-mail; aurel.tarara@gmail.com

without needing to modify, understand or even see the implementation, and instead interacting only with the programming interface [2].

Representational State Transfer, REST, as an architectural style, improves the performance, the scalability of component interaction, the simplicity of interface, the modifiability of components, the portability, the reliability, the visibility of distributed systems [3].

In terms of REST APIs, recent versions of PHP frameworks can expose endpoints that may be consumed by multiple applications, developed for different types of devices e.g. desktop computers, laptops, mobile devices or complex computer systems used in industrial environments.

The frameworks facilitate the web programming and make it better organized. Frameworks increase programming productivity with the help of framework built-in functions. Widely used frameworks have a major security advantage since its users become long-term testers. The most popular frameworks are free and usually come along with a support team, documentation etc. [4]. Frameworks like Laravel package a collection of third-party components together with configuration files, service providers, prescribed directory structures and application bootstraps [5].

Laravel is based on the model-view-controller, MVC, paradigm and provides a scaffolding with places to put code in. Laravel starts out with a complete directory tree to organize the code, and also includes placeholder files to use as a starting point [6].

Modern JavaScript frameworks like Vue.js support the development of *single-page application*, SPA, that changes data asynchronously with the backend API. Vue.js allows to simply bind the data models to the representation layer. It also allows to easily reuse components throughout the application. A special architecture, Vuex, for centralized states allows a global application store to be created, the place where the global application state can be stored and managed [7].

In a SPA, the entire application runs as a single web page. In this approach, the presentation layer for the entire application has been factored out of the server and is managed from within the browser. It shares the objective to bring the power of a desktop app to the thin, cross-platform environment of a web browser. SPA applications render like a desktop application, but runs in a browser, they offer decoupled presentation layers, faster, lightweight transaction payloads and less user wait time. The SPAs code is easier to be maintained [8].

The requests made by the single page application to the backend API in order to exchange data imposes a high level of data security because these requests can be intercepted and/ or forged. Also, session stored authentication data in localStorage and sessionStorage is not persistent and is loosed if browser is refreshed. JSON Web Tokens, JWT, can be used as a method to secure requests.

JWT, as a standard for safely passing claims in space constrained environments, is a very compact, printable representation of a series of claims, along with a signature to verify its authenticity. The most important aspect of this is the standardization effort in the form of a simple, optionally validated and/or encrypted, container format [9].

2. Objective and research method

Recent web technologies allow developers to build better applications in terms of performance, maintainability and user experience.

Taking advantage of PHP and JavaScript frameworks, the time needed for applications development is reduced. Frameworks integrate code packages and plugins which add functionality to the application without consuming time to write code.

By developing backend REST APIs, the data can be exposed to multiple applications without rewriting the core code for each application or each type of device used by the end user.

SPA user interfaces allow the navigation thru different sections of the application more efficient and presents the data to the user faster since only sections of the web page are updated and since only requests for not loaded or affected data are made to the backend API.

The objective of present research is to develop a web based application for product data engineering and management, by implementing modern web technologies, in order to avoid deployment problems to web servers with integrated new technologies, and to allow the communication with other applications and hardware.

The present research has been approached with regard to proper reference elements: selection of the PHP and JavaScript frameworks, definition of the backend and frontend APIs, etc.

3. Application design

Reference elements

Laravel 5.6 and Vue.js 2.5.7 have been chosen as the frameworks used for objective application backend and frontend APIs.

It has been established that the considered application will expose a package of end points in order to exchange data with other applications and a SPA approach is considered for the frontend API.

The backend API follows a model-view-controller, MVC, architecture made available by Laravel framework with the following main components: application data models, controllers that manage the data operations - *create*, *read*, *update*, *delete*, etc. - and exposed endpoints. Database connection and operations are simplified by Laravel and the ORM, Object-relational mapping, tool named Eloquent.

The application design base has been chosen as a suit of key functionalities that provides the user the ability to manage the product data.

The core functions of the application and their implementation approach are presented in Table 1.

Table 1

Application core functions and their implementation

Function	Implementation
Users authentication	JWT tokens are implemented to allow the secure communication between the frontend and the backend APIs
Users role/ permissions system	Gates and policies are implemented to restrict the application functionality by considering the user role
Product data definition	The frontend API allow project data to be defined by means of components and user interfaces. The backend API will allow the data CRUD, create, read, update, delete functionality
Advanced search by product attributes	Advanced search functionality is possible by means of models' relationships
Product data version control	Timestamps are registered in the database when data update action is called. Product data version will be automatically incremented
Product data lifecycle control	Product data is marked in the database as *for information, in study, released, or built*. Validation from users with higher rank is necessarily to change the data mark.
Product data sharing	Collaborative design is provided by means of product data transmittal based on email functions

The implementation of the application functionality considers the application scalability in terms of communication with other application or hardware. The backend API responses will be returned in JSON format.

Objective application backend API

In relation to Laravel base structure, the considered backend API is structured as presented in Table 2.

Table 2

Backend API base structure

Name	Description
Models	
User	Describes and interacts with database users table
Content	Describes and interacts with database content table
…	…
Caract_as	Describes and interacts with database assembly characteristics table
Caract_aux	Describes and interacts with database auxiliary parts characteristics table
Controllers	
AuthController	Contains methods for authentication
ContentController	Contains methods to create, update, delete, etc. content data
…	…
CaractasController	Contains methods to create, update, etc. assembly characteristics data
CaractauxController	Contains methods to create, update, delete, etc. auxiliary parts characteristics data
API routes	
api	Contains REST API routes

Laravel MVC approach and Eloquent allows a more efficient interaction and database information retrieval. The models describe the database tables structures and separate accessible or protected data by using fillable and hidden arrays of data. The user model defined for the considered application is as follows:

```
namespace App;
use Tymon\JWTAuth\Contracts\JWTSubject;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable implements JWTSubject {
        use Notifiable;
        protected $fillable = [ 'username', name, 'surname', rank, 'position', email, 'avatar', ];
        …
        }
```

To be noted that the User model includes support methods for the JWT authentication made possible by installing the Tymon JWT-auth package. Authentication methods are defined in the AuthController as follows:

```
namespace App\Http\Controllers;
use Illuminate\Support\Facades\Auth;
use Illuminate\Http\Request;

class AuthController extends Controller {
        public function __construct() {
                $this->middleware('auth:api', ['except' => ['login']]);
        }
        public function login() {
                $credentials = request(['username', 'password']);
                …
                return $this->respondWithToken($token);
        }
        public function me() {
                return response()->json(auth('api' )->user());
        }
        public function logout() {
                auth('api')->logout();
                return response()->json(['message' => Logged out']);
        }
        public function refresh() {
                return $this->respondWithToken(auth()->refresh());
        }
        protected function respondWithToken($token){
                return response()->json([
                'access_token' => $token,
                'user' => $this->guard()->user(),
                …
        }
        public function guard() {
                return \Auth::Guard('api'); }
        }
```

The login method receives a set of credentials and verifies that the credentials are registered in the database. If success a JWT access token is generated, this allows users to send request from the frontend API that will authorize data exchange.

The resource controllers allow data operations in relationship with REST verbs as GET, PUT, PATCH and DELETE. The responses generated are in JSON format.

The application ContentController includes index, store, show, update and destroy methods as follows:

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Content;
use App\Http\Resources\ContentResource;
class ContentController extends Controller {
        public function index() {
                return ContentResource::collection(Content::all());
        }
        public function store(Request $request) {
                $this->validate($request, [
                'name' => 'required|string|max:100|unique:contents',
                'idno' => 'required|numeric|max:255',
                'drawing' => 'required|string|max:255',
                'type' => 'required|string|max:255',
                'description'  => 'required|string|max:255', ]);
        }
        public function show(Content $content) {
                return new ContentResource($content);
        }
        public function update(Request $request,  $id){
                …
        }
        public function destroy(Role $role) {
                $role->delete();
                return response()->json(null, 204);
        }
}
```

The backend API exchanges data with the frontend API by means of guarded endpoints as follows:

```
Route::group(['prefix' => 'auth'], function ($router) {
        Route::post('login', 'AuthController@login');
        Route::post('logout', 'AuthController@logout');
        Route::post('refresh', 'AuthController@refresh');
        Route::post('me', 'AuthController@me');
});
Route::group(['middleware' => 'jwt.auth'],
        function($router) {Route::apiResource('/user', 'UserController');
                …
                Route::apiResource('/content', 'ContentController');
        });
```

The first group of end points allows request to be sent in order to get an access JWT token, to logout users, to refresh an expired token and to get the authenticated user. The second group of end points allows application data exchange.

Objective application frontend API

The frontend of the considered application is a SPA developed using Vue.js framework in conjunction with Vue Router and Vuex to allow SPA routing and global data storage. The frontend API base structure is developed as presented in Table 3.

Table 3

Application backend API base structure

Name	Description
General files	
app.js	Application main file
init.js	Authentication methods, JWT token request and storage
routes.js	Contains application routes
sore.js	Main Vuex store file
Components	
Users.vue	Contains the template, methods etc. for users interface
….	…
Content.vue	Contains the template, methods etc. for content interface
Modules	
init.js	App initialization store module file
…	…
content.js	Content store module file

The main page contents the HTML classic declarations, meta fields, style sheets and JavaScript files links. To be noted that SPA behavior is assured by the main component where the rest of the APP components are rendered. The page code is as follows:

```
<!doctype html>
<html lang="{{ app()->getLocale() }}">
<head> … </head>
<body>
        <div class="container" id="app">
                <main-app></main-app>
        </div>
        <script src="{{ asset('js/app.js') }}"></script>
```

```
</body>
</html>
```

The content of the application components respects the following model:

```
<template>
        //html layout
</template>
<script>
        export default { data() { //component data },},
        methods: { // component methods }, },
        computed: { //component computed properties}}
</script>
<style> // styles </style>
```

Performance of the developed application

During the development and use of the considered application, relevant observations have been made, as follows.

The application is highly maintainable, due to the logical and decentralized structure of frontend code separation from the backend and MVC pattern, allowing development of user interfaces without modifying the backend API code and structure.

The application exposes a REST API, JWT secured to exchange data with other applications which adds flexibility to the application. By means of endpoint, the backend API can be accessed in order to request JWT tokens. The tokens are used to authorize the requests for access to the application resources.

In industrial environment of E-House development and building [10], in the case of an operational application, PDEM.A02, less requests are sent to the backend. Vuex store assures data persistence and change between components and asynchronous requests allow retrieval of partial data from database. To retrieve parts data from the database, sends three less requests to the backend than the classic application, and no new requests if the page is refreshed.

The SPA approach reduces the resources usage and navigation/ data load time by means of components reusability and partial page load. Forms are reused thru the application. In case of adding or editing information only form elements are updated in the page. The rest of page and information does not change.

4. Conclusions

The objective application has been designed based on new concepts and web technologies.

The application is highly maintainable, and newer functionality can be added more efficiently by installing code packages supported by Laravel and Vue.js frameworks.

Generally, the implementation of modern web technologies improves the application, minimizes the resources usage, reduces the development time, and allows the system to exchange data more efficiently thru the REST API, secured with JWT tokens.

Acknowledgement

This work has been supported by the Sectorial Operational Program for Human Resource Development, financed from the European Social Fund and the Romanian Government under the contract number POSDRU/159/1.5/ S/132397.

R E F E R E N C E S

[1]. *A. Tarara, M. Gheorghe*, Development of a product data engineering and management web based application, U.P.B. Sci. Bull., Series D, Vol. 78, Issue 1, pp. 279 – 290, ISSN 1454-2358, 2016.
[2]. *J. Stylos*, Making APIs more usable with improved API design, documentation and tools, PhD Thesis, Carnegie Mellon Univeersity, 2009.
[3]. *F. Doglio*, Pro REST API development with Node.js, Apress, 2015.
[4]. *N. Prokofyeva, V. Boltunova*, Analysis and Practical Application of PHP Frameworks in Development of Web Information Systems, J. of Procedia Computer Science, 2017, Vol. 104 Iss. C, 51-56.
[5]. *M. Stauffer*, Laravel up & running, O`Reilly Media, 2016.
[6]. *M. Bean*, Laravel 5 Essentials, Packt Publishing, 2015.
[7]. *O. Filipova*, Learning Vue.js 2, Packt Publishing, 2016.
[8]. *S. Emmit*, SPA Design and Architecture, Manning Publications, 2016.
[9]. *S. Peirott*, The JWT Handbook, Auth0, 2017.
[10]. ***, Product Portfolio, IMSAT E-House Solution Division.