

WEB601 MS2 project report
Oleg Sivers
S2 2019

Table of contents

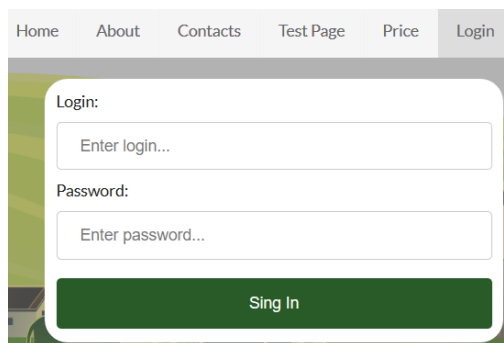
Table of contents	2
Git repositories	3
Checking implementation of APIs.....	3
Structure and navigation compared to MS1.....	4
Server	5
General information	5
Libraries and frameworks	5
Database structure.....	6
APIs.....	6
Client	7
General information	7
Libraries and frameworks	7
Client-Server interaction.....	8
How to test APIs.....	8
Responsive design.....	8
How to login.....	8
Possible issues.....	8
Conclusion.....	8
References	9

Git repositories







- Project: <https://github.com/sio2k1/WEB601PRJ>
- React application: <https://github.com/sio2k1/WEB601PRJ/tree/master/my-app>
- Express server: <https://github.com/sio2k1/WEB601PRJ/tree/master/server>
- Journal: https://github.com/sio2k1/WEB601_week_1/blob/master/journal.md

Checking implementation of APIs

1. Download project and get packages for server and client with “npm i”
2. Navigate to login page and enter credentials (login:Ali password:321)



3. Go to pricelist editor page in admin panel, using navigation bar or /admin/pl-editor
4. Using web interface perform update\insert\delete operations within the table:

Price list		Search			
Actions	Caption	Price	Units		
 	Paper3	4.7	Kg.		
 	Salt267uuur	111117	Kg77		
 	tr33rr	0	hh		

5. Navigate to /price to check changes

Home	About	Contacts	Test Page	Price	Admin	Logoff
Paper3			4.7	Kg.		
Salt267uuur			111117	Kg77		
tr33rr			0	hh		

6. Check database changes at \server\server_components\db\sdb.db with SQLite browser tool like <https://sqlitebrowser.org/dl/>

Structure and navigation compared to MS1

Site consists of several pages, which will be displayed for every visitor, this is the same as in MS1:

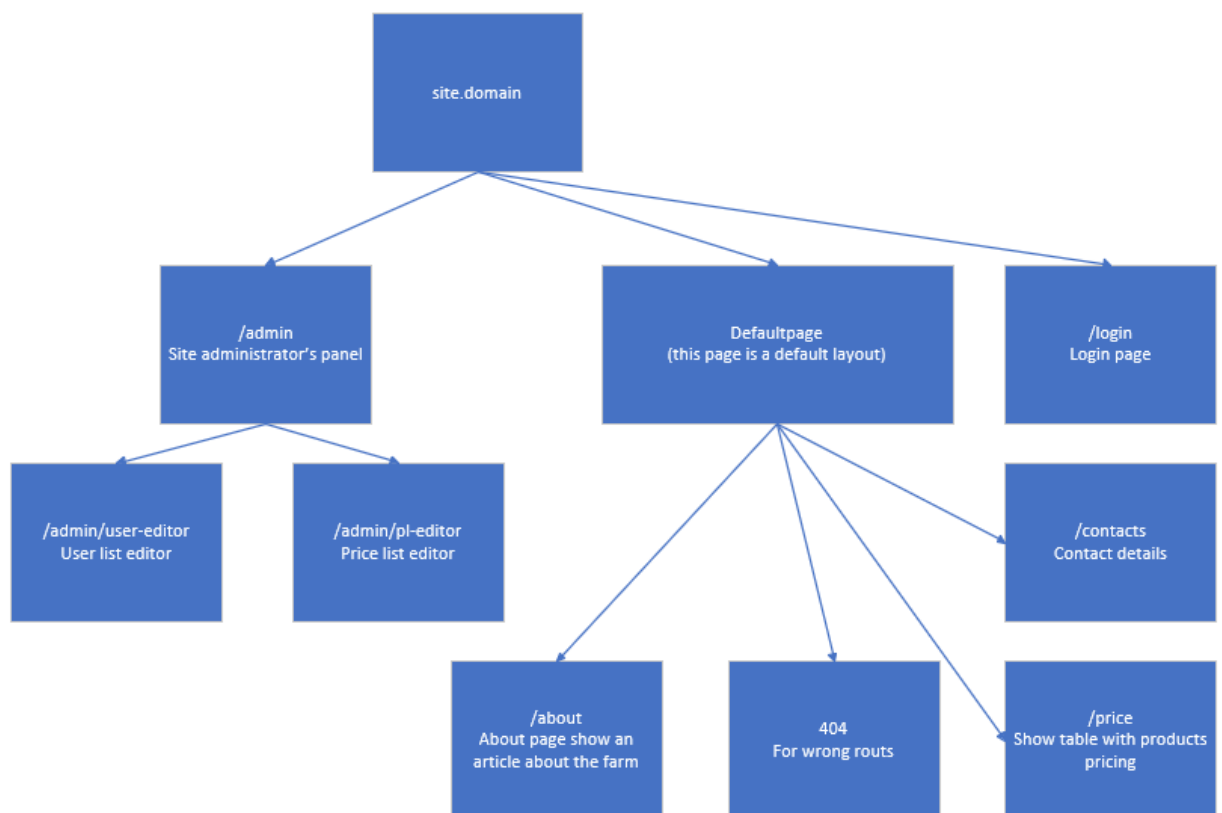
- Price list – displays price of goods
- About – article about farm
- Contacts – simple contact information
- Home page – landing page with logo
- 404 page – if route is not matching known route

Compared to MS1 I added several pages which will be displayed after login:

- Admin – redirect to price list editor
- Price list editor – allow to modify pricelist contents in DB, using gui.
- User list editor – dummy component at this point

I also added layout for admin panel pages, so now my site has 3 types of layouts:

- Pages, that inherit layout from default page (/price, /contacts, /about, 404)
- Page with unique layout (/home, /login)
- Pages with layout for admin panel (/admin/pl-editor, /admin/user-editor)



Server

General information

My server application is made with JavaScript, running at server with Node.js. For building server, I used provided examples, libraries\frameworks documentation and stackoverflow for problem-solving: (Kahwaji, 2019), (Routing, .n.d.), (Knex.js - A SQL Query Builder for Javascript, .n.d.)

In the application I'm using several libraries and frameworks to cover common server-side tasks:

- Express.js framework for handling routs, accepting requests and output data to clients
- Knex.js to query database for adding, modifying, deleting and reading data
- Sqlite3 npm package to connect Snex.js with SQLite database
- body-parser npm package, to convert body part of client request to json

Libraries and frameworks

Express.js

Express.js is a framework for Node.js which provides variety of tools to web and mobile applications development. At my server project I'm using it to handle client-server data exchange over tcp\ip and for handling routs for REST API.

Server code, related to express, is designed and organized to be capable to add extra API entry points in future. For each API entry point we need to create folder with files with routes declaration and routes handling functions, example is -> server_components\pricelist folder. After all entry points and handlers designed, we can combine all routs in server_components\app.js file. Configuration file is located in server_components\config.js at this point we can set a port to listen to and SQLite DB path. To start application, we use entry.js file in server root folder as an entry point.

Knex.js and database provider

Knex is a JavaScript abstraction build above the T-SQL, mainly knex is generating sql statements for developer, based on java script code provided. Knex is supporting many relational databases, including MSSQL, Oracle, Postgres, SQLite and more. Knex.js also provides DB creation and migration functionality.

Server code related to knex in my code, mainly located in routes handling functions, example is \server_components\pricelist\handlers.js. Knex configuration file describes a database connection information like path, server address and user creds. Knexfile located at server_components\knexfile.js

SQLite database

It's a file based relational database, I've chosen it because my project is tiny and not requires some serious database management systems like MySQL of MSSQL, because of the small amount of data needed to be stored, still with knex we can easily change DBMS when it's needed.

To connect DB, I'm using sqlite3 npm package, which is recommended by knex documentation. Database located at \server_components\db\sdb.db

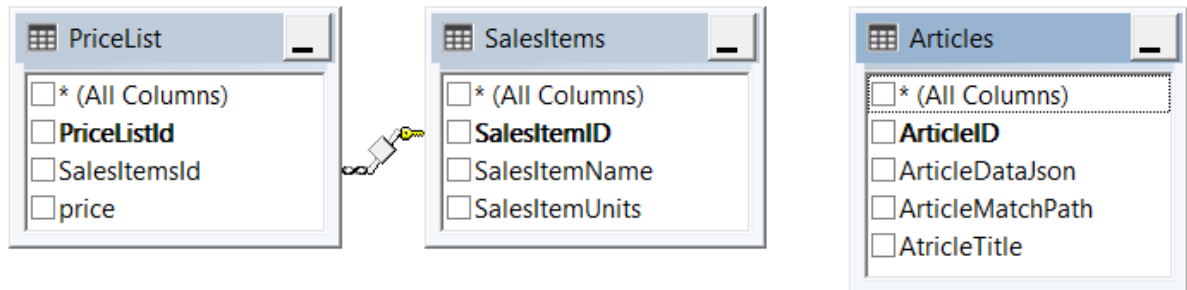
Links to the libraries:

- <https://github.com/expressjs/body-parser>
- <http://knexjs.org>

- <https://expressjs.com>
- <https://github.com/mapbox/node-sqlite3>

Database structure

I'm using SQLite file database with this tables:



At this point DB consists from three tables, one contains articles, another two is linked and contains price list entries with prices, captions and units. I connect to database using sqlite3 npm package, database is located in /server_components/db/sdb.db. Connection file is located in /server_components/knexfile.js, configuration file with DB path located in server_components\config.js. If future I'm planning to create table for users for authentication reasons to keep credentials and access rights.

APIs

There are two API paths in server application:

- Articles <http://localhost:3001/api/articles>
- Price list <http://localhost:3001/api/pricelist>

Articles only support requests to get all records and get specified record by /:id. In client side we have several articles at this point contacts and about.

Pricelist supports get, get by id, post, patch and delete requests:

- Get request will return all records from PriceList table with SalesItems table joined
- Get by id will return one record with PriceListId equal to /:id provided, if record not found or id is not an integer it will return an error
- Post will check request body to have a json with mandatory fields which are Price, SalesItemName, SalesItemUnits and if everything is ok it will insert data to PriceList and SalesItems tables and return inserted record, styling it like for "get by id" request, I forced to return inserted value to populate my table with pricelist with an actual id after inserting a record
- Patch will modify information for requested record (/:id) any of this is supported in json in request body: Price, SalesItemName, SalesItemUnits
- Delete will delete specified record in /:id

Client

General information

My client application is made with JavaScript, running at client with support of Node.js, as local webserver.

In the application I'm using several libraries and frameworks to cover common client-side tasks:

- React – allows to use component-based approach to JS based user interfaces, gives some extra functionality, like improved rendering according to data changes and JSX
- Redux – global data storage for all components inside application with subscriptions to data.
- react-router – npm package for React, to cover in-app routs.
- Axios – npm package to handle queries to external APIs
- material-ui & material-table npm packages I used to add editable table in my admin panel.

For building my project I used official documentation of react, material table, axios, react-router and provided examples: (React Getting Started, n.d.), (Material table documentation, n.d.), (Axios,2019), (ReactTraining, 2019), (Kahwaji, 2019)

Libraries and frameworks

React

Library that allows to develop user-interfaces with JavaScript originally released by Facebook, now it is an open-source project.

At my application I use react to build UI and split it into small components. All components in my application are located in /components folder, mainly first milestone was related to developing UI, so at this point I've made some improvements to component structure, but there are no significant changes compare to MS1, except API calls and small tune-ups

react-router

Npm package for handling application routs, used to attach component to route and display it on route access.

Routes defined in components/routing/routing.js. There are some static routes, like admin, login, pricelist and so on. Also, there are some dynamic routes, which we are populating from database, like contacts and about. At MS2 main changes are displaying of dynamic pages, defined in database and user-defined route containers and components like AdminLayoutRoute (components\admin\admin_layout.js). This user-defined route components combines route with particular layout of page, as I have several different layouts for pages in my app.

Axios

Npm package for fetching data and connect our application with remote APIs.
At my project all API calls related data are located in /api_list folder.
There are two axios init files: api_articles_axios and api_pricelist_axios
These files are containing connection information about particular API
There is also api_operations file, which cover base APIs calls for get, post, patch and delete.

To use api_operations in components\containers we need to call corresponding function(i.e. FDelete for delete, FUpdate for patch, etc.) For mentioned functions we need to provide params: api -> which is axios config info and in some situations json for request body, json should contain the "id" field for patch and delete methods. Sample code I'm using to get data from API:

```
import api from '../api_list/api_articles_axios'
import * as operations from '../api_list/api_operations'
this.setState({...this.state, isFetching: true});
const inData = await operations.FGet(api)
this.setState({data: inData, isFetching: false});
```

material-ui and material-table

I'm using the material table npm package to cover table base editing for my pricelist. Material table support column definition, paging and events for delete\edit and create rows.

Component which using material table is located at components\admin\price_editor\pl_editor.js

Client-Server interaction

Based on http protocol over tcp\ip by sending web requests form client to server and getting response on requests. Server support REST api while handling routes. At client we form requests with axios and handle them with express.js at server.

How to test APIs

For testing reasons we can use Postman software at our APIs endpoints, we need to fill postman body with corresponding json to post or patch, example json:

```
{ "data": {  
  "PriceListId": "9",  
  "SalesItemName": "Salt267uuur",  
  "Price": "111117",  
  "SalesItemUnits": "Kg77",  
  "id": "9"  
}  
}
```

Responsive design

I've done it with pure css and media queries, we can use them to override css classes, depending on screen resolution. Typical example is components\admin\admin_layout.scss

For admin layout I override grid from using columns to using rows, so in small displays admin layout will look different. Navigation bar css for responsive design was picked from w3cschools examples (Responsive Web Design - Media Queries, n.d.).

How to login

To login to website we need to use any login from \my-app\src\jsondata\users.json Example record:

```
"login": "Ali", "pwd": "321"
```

Possible issues

SQLite DB connection

I checked connection only within windows env, if knex is unable to connect to DB please check:

\server\server_components\config.js for DB path

Express listing host

Host is specified with "0.0.0.0", so express will start listening on every available IP in system could be changed in \server\server_components\app.js

Conclusion

At this point my project consist of client and server, client can display several pages with styles, supports responsive design for all pages, can do request to server via http web requests, my server has

two API endpoints one is for price list and it supports patch\delete\get\post and another for articles which supports only get requests.

Site is almost fully operational, besides login which is not finished for now. For my project I used these libraries\frameworks:

- React
- Redux
- react-router
- Axios
- material-ui
- material-table
- Express.js
- Knex.js
- Sqlite3
- body-parser

References

Kahwaji, A. (2019, September 25). alikahwaji/API-Knex. Retrieved October 6, 2019, from <https://github.com/alikahwaji/API-Knex>.

Routing. (n.d.). Retrieved October 6, 2019, from <http://expressjs.com/en/guide/routing.html>.

Knex.js - A SQL Query Builder for Javascript. (n.d.). Retrieved October 6, 2019, from <http://knexjs.org/>.

React Getting Started. (n.d.). Retrieved October 1, 2019, from <https://reactjs.org/docs/getting-started.html>.

Material table documentation. (n.d.). Retrieved October 11, 2019, from <https://material-table.com/>.

Axios. (2019, October 16). axios git. Retrieved October 16, 2019, from <https://github.com/axios/axios> .

Kahwaji, A. (2019, October 7). alikahwaji/class-example. Retrieved October 10, 2019, from <https://github.com/alikahwaji/class-example> .

ReactTraining. (2019, October 14). ReactTraining/react-router. Retrieved October 15, 2019, from <https://github.com/ReactTraining/react-router> .

Responsive Web Design - Media Queries. (n.d.). Retrieved October 17, 2019, from https://www.w3schools.com/css/css_rwd_mediaqueries.asp .