Quality control application for wood products

Case: Raute Oyj

LAB University of Applied Sciences
Bachelor of Engineering, Information and Communication Technology
2020
Aliaksei Prakapenka

Abstract

<table>
<tr><td rowspan="2">Author(s)
Prakapenka, Aliaksei</td><td>Publication type
Thesis, UAS</td><td>Completion year
2020</td></tr>
<tr><td>Number of pages
35</td><td></td></tr>
<tr><td colspan="3">Title of the thesis
Quality control application for wood products
Case: Raute Oyj</td></tr>
<tr><td colspan="3">Degree
Engineer (UAS)</td></tr>
<tr><td colspan="3">Name, title and organisation of the thesis supervisor
Matti Welin, Principal Lecturer, Information and Communication Technology</td></tr>
<tr><td colspan="3">Name, title and organisation of the client
Markus Halmetoja, Business Manager, Raute Oyj</td></tr>
<tr><td colspan="3">Abstract
The objective of the study was to research modern software frameworks and, based on the knowledge gained, develop the hybrid mobile application. The project was commissioned by Raute that serves wood products industry all over the world. With this application, it should be possible to control the quality of materials produced on Raute equipment.
The study of the Angular framework was based on its architecture. The main elements of the architecture, their tasks and features were considered. Also, the role of the Apache Cordova framework in the development process was studied, as well as the functions and the capabilities that it has.
The user interface description with the explanation of the methods used in the development was chosen as a way to demonstrate the software product.
The successfully developed mobile application became a result of the thesis. Now, Raute surveys clients and, according to their proposals, modifications will be made, and the application will be further developed.</td></tr>
<tr><td colspan="3">Keywords
Hybrid mobile application, Angular, Apache Cordova, user interface</td></tr>
</table>

Tiivistelmä

<table>
<tr><td rowspan="2">Tekijä(t)
Prakapenka, Aliaksei</td><td>Julkaisun laji
Opinnäytetyö, AMK</td><td>Valmistumisaika
2020</td></tr>
<tr><td>Sivumäärä
35</td><td></td></tr>
<tr><td colspan="3">Työn nimi
Puutuotteiden laadunvalvontasovellus
Case: Raute Oyj</td></tr>
<tr><td colspan="3">Tutkinto
Insinööri (AMK)</td></tr>
<tr><td colspan="3">Ohjaavan opettajan nimi, titteli ja organisaatio
Matti Welin, yliopettaja, Tieto- ja viestintätekniikka</td></tr>
<tr><td colspan="3">Toimeksiantajan nimi, titteli ja organisaatio
Markus Halmetoja, liiketoiminnan johtaja, Raute Oyj</td></tr>
<tr><td colspan="3">Tiivistelmä
Opinnäytetyön tavoitteena oli kehittää hybridi-mobiilisovellus, jonka avulla voidaan valvoa tuotteiden laatua tuotantoprosesseissa. Toimeksiantajana toimi puutuotetoimialaa palveleva Raute Oyj.

Työn aikana tutkittiin nykyaikaisia ohjelmistokehyksiä. Perehdytettiin Angular-ohjelmistokehyksen arkkitehtuurimalleihin ja selvitettiin Apache Cordovan ominaisuuksia ja toimintoja. Myöskin kuvattiin sovelluskehitystä ja siihen liittyvän menetelmiä, joita käytetiin käyttöliittymän toimintojen toteuttaminen.

Opinnäytetyön tuloksena saatiin toimiva mobiilisovellus, joka vastasi toimeksiantajan vaatimuksia. Raute tutkii tällä hetkellä asiakkaiden mielipiteitä, joiden perusteella tehdään tarvittaessa projektiin muutoksia.</td></tr>
<tr><td colspan="3">Asiasanat
Hybridi-mobiilisovellus, Angular, Apache Cordova, käyttöliittymä</td></tr>
</table>

Contents

1 Introduction

With the development of mobile technology, mobile devices have become important in the business world. From a simple device for talking and texting, the mobile device has changed into a business assistant whose role today is hard to overestimate. Nowadays, the phone contains plenty of information: ideas and thoughts, a calendar of meetings, correspondence with colleagues and partners, and many other useful data. Portability and mobility are the main advantages of these devices, and the ever-growing number of mobile applications only increases their relevance. To attract and maintain customer interest, modern companies are constantly improving their services and one of their priorities is to develop user-friendly and useful mobile applications.

Raute is the type of company which seeks to provide solutions and services that meet the customer's requirements. It is a company with worldwide fame in the wood processing industry. Its customers are veneer, plywood, LVL (Laminated Veneer Lumber) and sawn timber manufacturers. (Raute b.) For its customers, Raute provides a full range of services from project design to the maintenance of the existing production lines. One of the technological services offered to the client is an information and reporting system MillSights. Raute has decided to make a mobile application for product quality control as a part of this system. At the moment, clients use different tables and documents to record quality measurements and evaluation results of the product. This complicates the quality control process and generated numerous reports. The mobile application should facilitate this measurement process and simplify routine operations.

Thus, the goal of the thesis was to develop a mobile application for product quality control at different stages of the production process. During the design phase, Raute provided with a list of application functionality requirements. After reviewing these requirements, it was decided to create a hybrid application that can work on the Android and iOS platforms.

2 Operational Environments

2.1 Wood production process

According to Raute (a), wood processing consists of many different stages, and each stage has its line. On the log line, logs are measured, debarked, and cut to blocks. Then those blocks are peeled and cut into veneer sheets. To get the correct moisture content of the veneer sheets, they are passed through the drying line. On the scarf-jointing line and the composing line, the veneer sheets are joined. If there is a defect in a veneer sheet, it is repaired on the patching line. After those processes, the veneers are combined into plywood on the lay-out line and pressed on the pressing line. Lastly, the plywood gets its final size after the trimming line.

2.2 Raute goal

Even though almost all the lines work automatically, they still need human control. At some stages, it is necessary to measure the thickness or the length of products, or to take the readings from the sensors. And, at some parts of the lines, it is necessary to visually assess the quality of a product. Based on the gathered data, reports are created. It helps the workers to have a complete picture of the ongoing processes at each stage of the production. Besides that, data monitoring allows determining if any line needs additional configuration or to identify and resolve a malfunction. Due to the need to monitor the production process in real-time, Raute decided to create a mobile application that allows a worker to make measurements and transfer data to a server using a mobile phone.

2.3 Raute requirements

2.3.1 Data flow

There are three main nodes in the project through which data must flow: a mobile phone, a backend server, and a MillSights server (Figure 1). Using a phone, the user should be able to make measurements and enter the required data into the mobile application. The application must form a final object from this data and send it to the backend server. Paired with the backend, there should be a proxy server that accepts traffic only from the specific customer's public IP address, and transfer the data to the backend. The backend server must process the data and determine who owns this data. According to this information, the server understands to which MillSights server it must send the data. Thus, the backend server must act as a data explorer, but at the same time, it must validate the data and store it in its database. Raute creates a personal virtual network for each client. This network contains a server on which the Raute's software runs, and a proxy server that controls the data transfer.

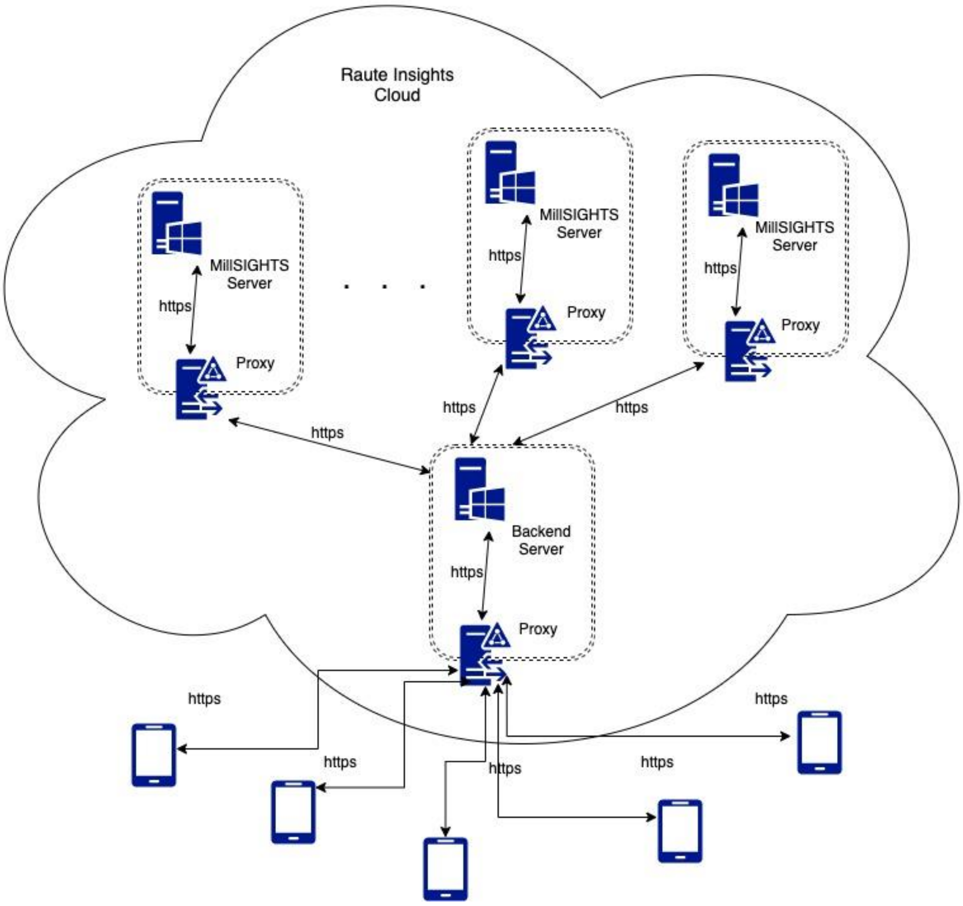

Figure 1. Data flow

2.3.2 Frontend requirements

To put a mobile application into operation, it must fulfil the following requirements:

- Authorization. A user must pass the authorization by entering his username and a password. The backend server receives the data, checks it and, if the data is correct, generates a token and sends it back to the frontend. The token contains information connected to the user (last name, first name, company name, IP address of the MillSights server and some other information) and must be present in every further post request. As mentioned above, one of the server roles is data flow distribution. Precisely because of the token received along with the main data, the backend server can determine where to send this data. Another fact that the application must consider is the possibility of dynamically changing the IP address. For this reason, the backend server sends the list of allowed IP addresses along with the token. The application must check the client's public IP address to compare it with this list of valid IP addresses.

- Bluetooth. Since most of the measurements concern the dimensions of the product, the application must be able to work with Bluetooth devices measuring the thickness, length, width, and diameter.
- Camera. The application must be able to work with the phone camera. In some cases, for more complete information about the results of the production process, photo confirmation is required.
- Settings. The application must have a settings page. Some part of the settings should be common for any production process, and it is adjusted right on this page. Some part of the settings should appear on the screen dynamically, depending on the selected process. Also, the user should be able to select only the items that are currently needed from the proposed list of settings. The ability to change the order of measurements should be implemented as well. In case the user selects the measurements, which can be done by Bluetooth devices, a layout should appear on the screen where the user can select the measurement points.
- History. The application must save the data that is sent to the server. According to the logic of the application, the user can check the last ten measurements. Another important reason for storing the data is the ability to resend data. For some reason, there can be no connection with the server during the first attempt to send the data.
- Storage. Some application properties have data that should not be destroyed when the application is closed. For this reason, the application must store data such as a list of IP addresses, a token, settings, and measurements in the phone memory.
- Verification. It is important to check the entered data for validity, prohibit entering duplicate values in the measurement settings, and not allowing a user to leave the data filling form if the filling fields are left empty. Besides that, the application must inform a user if such a situation occurs.

3 Mobile application

Most people divide mobile apps according to their purpose. According to ThinkMobiles, there are 33 categories of mobile applications in Google Play and 24 categories in Apple App Store. Besides this, software developers divide mobile applications into three groups according to the specifics of their work: native application, mobile applications for websites, and hybrid.

3.1 Native application

Native applications are written in a native programming language for a specific platform. For Android, this language is Java or Kotlin, while for iOS, it is Objective-C or Swift. Since native apps are designed specifically for a specific platform, they can use all device capabilities - GPS sensor, camera, compass, contact list, etc. Native applications operate at high speed and have a perfect performance because they are optimized for a specific operating system. Users can install them through an application store. A disadvantage of native applications includes the high cost of application development since separate development, support and maintenance for each operating system is required. (ThinkMobiles.)

3.2 Mobile application for websites

These applications use a web browser on the user's device. In fact, they are websites that are adapted and optimized for phone usage and have advanced interactive features. Web applications are cross-platform, so they can be executed independently of any device platform. Programmers develop Web applications using the traditional web development languages like CSS, HTML and JavaScript. As a result, development and maintenance processes are faster than in native applications. A disadvantage is that applications do not have access to all the capabilities of a user's device. Also, such applications do not work without Internet access. (ThinkMobiles.)

3.3 Hybrid application

A hybrid application is a combination of web and native apps. Cross-platform is taken from web applications. The ability to use the functionality of a mobile device is taken from native applications. Technically, this is a web application wrapped in a native shell using frameworks like Apache Cordova, React Native, Xamarin, etc. Hybrid application development is faster and easier than native application development. A client gets, at the same time, an application that has several installation packages, each one for a specific operating system. However, this type of application has several limitations. It is not always possible to implement the user interface like in the native application. Also, heavyweight data processing is slow. (ThinkMobiles.)

4 Angular framework

According to the Adach, theoretically, it is enough to use HTML, JS and CSS to create a software product of any complexity. However, as more functionality is added to a project, it is more difficult to understand and maintain the code. To facilitate the programming process and make it controllable, developers use frameworks. One of the frameworks that are adaptable for websites building is Angular.

4.1 Basics of Angular

> *Angular, supported by Google, is an open-source software engineering platform used for building user interfaces (front-end) (AltexSoft 2020a).*

Angular has several features that make it comfortable and clear to work with it. One of the advantages of using Angular is its Component-based architecture (Figure 2). It means that each element of the user interface can be considered as a separate encapsulated component with its own internal functionality. (AltexSoft 2020a.)

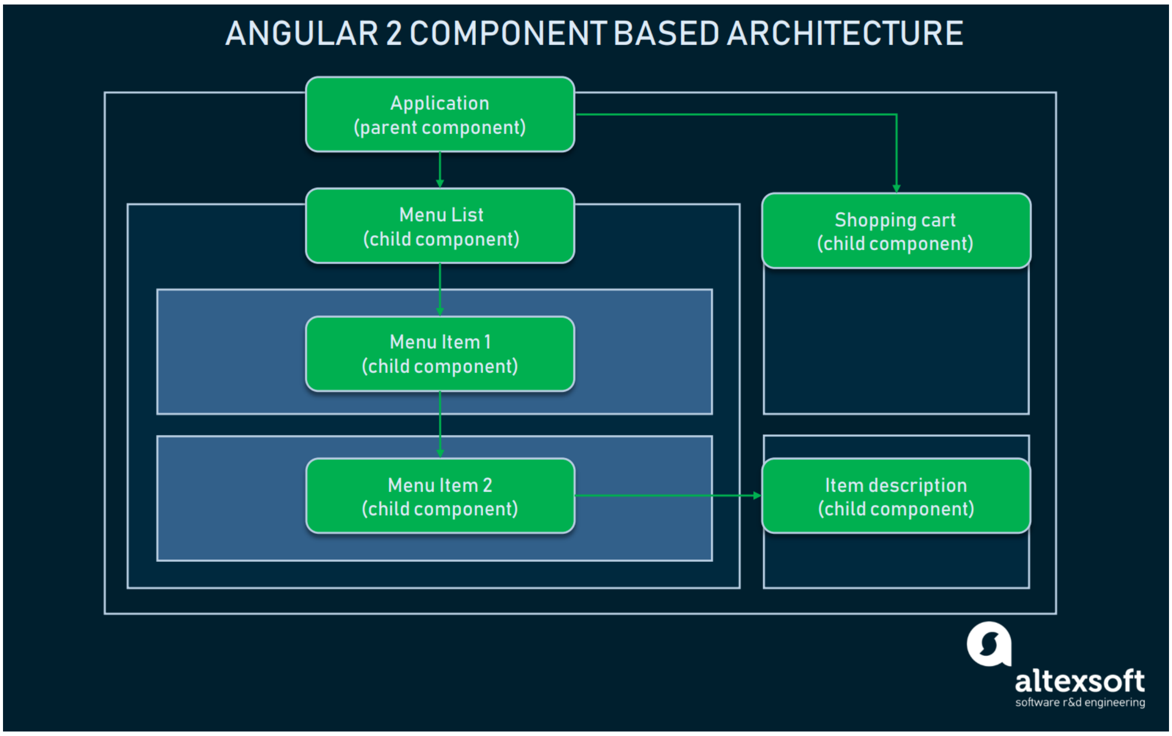

Figure 2. Component-based architecture (AltexSoft 2020a)

These components are easy to reuse, combine or simply remove from a project if it is no longer needed. Also, components independence increases the readability of the code, as well as makes it easy to test a web application. Another advantage is Angular Materials that is the library of user interface components. The components are adapted to work with

Angular and are easily integrated into the project. Also, Angular uses the library RxJS, which works with asynchronous data streams and facilitates such data type processing. One more advantage of using the Angular framework is the programming language TypeScript (TS) that is used to develop projects. (AltexSoft 2020a.)

TS, being a superset of the programming language JavaScript (JS), inherits the main advantages of JS and adds own features. In comparison with JS, TS offers strong compile-time data typing and supports classes, interfaces, and some other concepts from object-oriented programming. All these facts make the code more predictable. Furthermore, TS allows identifying common bugs at the compilation stage. (AltexSoft 2020b.)

4.2 Angular architecture

4.2.1 Module

Angular uses a module as a mechanism to combine different application pieces. An application can be assembled from the modules that have certain functionality and interact with each other. Every Angular application has at least one root module called AppModule. Simple root NgModule definition is presented in Figure 3. (Angular a.)

src/app/app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Figure 3. App.module.ts (Angular a)

According to the Angular (a), the libraries required for the module are imported at the beginning of the file using the import directive. Every Angular library name starts with the @angular prefix, and each module must be defined by the @NgModule decorator. This decorator is the function that takes an object with properties which describe the module:

- Imports. This is an array for external modules, whose libraries are needed in a project. It can be an Angular built-in module or a module created by a developer.
- Providers. The services added to this array will be available both in this module and in any other module of an application.

- Declarations. This is a list of components, pipes and directives that are used in this particular module.
- Exports. A name of a module must be added to this list in case elements of this module are used in another module.
- Bootstrap. This array contains the root component that is called when an application is loaded.

4.2.2 Component

A component is an essential part of an application that defines and controls a view on the device screen. A component logic is defined inside a class where applied methods change the properties of the component and affect the view. As shown in Figure 4, the component class is described according to the principles of object-oriented programming and has the class name, attribute declarations, constructor, and methods. (Angular b.)

```
1  import { Component, OnInit, OnDestroy } from "@angular/core";
2  import { MatDialog } from "@angular/material/dialog";
3
4  @Component({
5    selector: "app-settings",
6    templateUrl: "./settings.component.html",
7    styleUrls: ["./settings.component.css"],
8  })
9  export class SetComponent implements OnInit, OnDestroy {
10   processIndex: number;
11   subprocessIndex: number;
12   toggleValue: string;
13   constructor(public dialog: MatDialog) {
14   }
15   ngOnInit() {
16   }
17   toggleSelected(index: number, checked: boolean) {···
18   }
19 }
```

Figure 4. Class of Angular component

According to Angular (c), Angular has its methods that are added into the class definition, like the method ngOnInit() (line 15 Figure 4). Such Angular methods are called lifecycle hook methods. The point is that Angular controls the vital activity of a component from the moment of its creation to the moment of destruction and the ngOnInit is one of the methods that is called during the component lifecycle. This feature allows adding functionality to the component at the various stages of its life. After the component has been constructed, its lifecycle hook methods are called in the following order:

- ngOnChanges() fires after the constructor and whenever a change happens in an input property of a component.
- ngOnInit() is called once after Angular has initialized data-bound properties of a component. Since the constructor does not have to perform complex initialization, this method does it. Also, the ngOnInit method is used to set up component initial configuration.
- ngDoCheck() is called each time a component property changes are checked, immediately after the ngOnChanges and ngOnInit methods. This allows checking the property value for additional conditions that Angular cannot check.
- ngAfterContentInit() is called once after external content has been added to a component or if the child tag <ng-content> has been replaced with HTML code from a parent component.
- ngAfterContentChecked() is called every time after content is checked by Angular.
- ngAfterViewInit() is called once after a component and its child views are initialized.
- ngAfterViewChecked() is called every time after a component and its child views are checked by Angular.
- ngOnDestroy() is called right before a component is destroyed and used to dispose of unnecessary event handlers.

4.2.3 Template and data binding

Interaction between components and views happens through templates. One component is mapped with only one template. The template contains Angular directives and binding markup in addition to HTML markup. The combination of these markups defines the Angular framework of how to display the view on the device screen. (Angular d.)

A dynamically changing component presentation is possible due to data binding (Figure 5) (Srashti 2018).

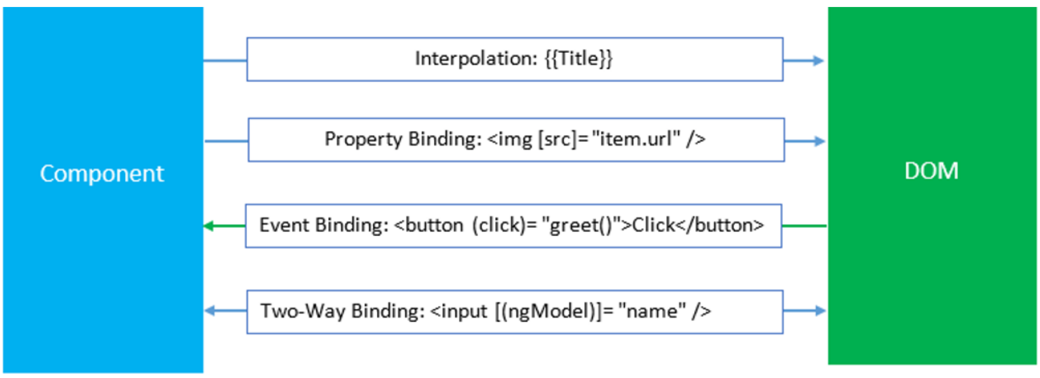

Figure 5. Data binding (Srashti 2018)

The documentation of Angular (e) describes when to use interpolation data binding. If it is necessary to display a component property in HTML code, developers apply this binding type. Interpolated text is enclosed in double curly braces and inserted between HTML tags as the text (line 1 Figure 6) or as the template expression (line 2 Figure 6). The point is that Angular primarily evaluates content between braces and then converts it to a string. Also, interpolation is used to pass a value from a component to template as a value of a DOM (Document Object Model) element property (line 3 Figure 6).

However, Angular (f) claims that the preferable way to pass the element property to DOM is property binding (line 5 Figure 6). Angular also draws attention to the fact that interpolation and data binding works only with DOM properties but not with HTML attributes. If it is necessary to use an attribute, the prefix attr with a dot is inserted before the attribute name (line 6 Figure 6).

Data binding that works in the opposite direction (from a template to a component) is called event binding (line 8 Figure 6). In other words, event binding allows associating an event that happened to an element with a method described in a component class. (Angular g.)

```
1  <p>{{title}}</p>
2  <span>{{1+1}}</span>
3  <img src="{{imageUrl}}">
4
5  <th [colSpan]="2"> Header </th>
6  <th [attr.colspan]="2"> Header </th>
7
8  <button (click)="increase()">Click</button>
```

Figure 6. Data binding.

The combination of property binding and event binding is two-way binding. Two-way binding allows dynamically changing a value on one side of the binding with the changes on the other one. For example, in Figure 7, the property ‘size’ is represented in square brackets. It means that its value depends on the variable ‘fontSizePx’. At the same time, this property is in round brackets. And it means that the variable ‘fontSizePx’ depends on the property ‘size’. If somewhere else the property ‘size’ changes its value, the variable ‘fontSizePx’ automatically takes the same value. (Angular h.)

src/app/app.component.html (two-way-1)

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
<div [style.font-size.px]="fontSizePx">Resizable Text</div>
```

Figure 7. Data binding – two-way binding1 (Angular h)

Also, two-way binding is used when working with input elements in forms. For instance, on Figure 8, the ngModel directive defines that once the value of the component property 'name' is changed in the first input field, it will be updated in the second input field and vice versa. In other words, a property is a transmitted and a received parameter at the same time. (Angular i.)

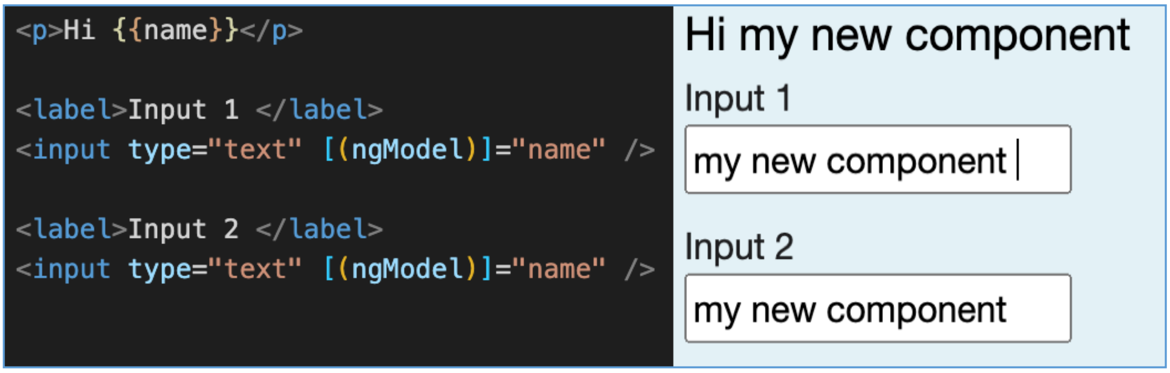

Figure 8. Data binding – two-way binding2

4.2.4 Directives

Directives define a set of instructions that apply to render HTML code. Angular divides them into attribute and structure directives. (Angular i.)

Attribute directives change the appearance or a DOM element behaviour. For example, a DOM element can get or lose one or more style classes using the NgClass directive. The object containing style classes can be assigned to an element directly in the template using conditional expression (Figure 9). Another attribute directive NgStyle is used in the same way as the NgClass directive. The only difference is that an element gets an object where the keys are the names of the CSS properties and their values are the possible values of the corresponding CSS properties. (Angular i.)

```
<div [ngClass]=
      "isChecked ? 'fewClasses' : ''">
  Few classes applied
</div>
```

```
/* in the component.ts
  fewClasses =  {
    newClass1: true,
    newClass2: true };
in the style.css
  .newClass1{ color: red };
  .newClass2 { margin: 20px }*/
```

Figure 9. NgClass directive

According to Angular (i), structure directives change DOM structure by adding or removing HTML elements. The NgIf directive allows adding, changing, or deleting an element under certain conditions (lines 1-2 Figure 10). All elements of the component array are processed one by one with NgFor directive (lines 4-6 Figure 10). The NgSwitch directive allows to embed switch-statement into a template and display a block depending on switch condition (lines 8-12 Figure 10) (Angular i).

```
1   <p *ngIf="isHappy">Hello</p>
2   <p *ngIf="!isHappy">Bye-bye</p>
3
4   <ul>
5     <li *ngFor="let item of list">{{item}}</li>
6   </ul>
7
8   <div [ngSwitch]="count">
9     <ng-template ngSwitchCase="10">{{count * 10}}</ng-template>
10    <ng-template ngSwitchCase="100">{{count * 100}}</ng-template>
11    <ng-template ngSwitchDefault>{{count}}</ng-template>
12  </div>
```

Figure 10. Structure directive

In addition to built-in directives, it is possible to create custom directives. In general, the creation of directives looks like in Figure 11 but with decorator @Directive which gets an object containing a CSS attribute selector. (Angular j.)

```
@Directive({
  selector: '[name]'
})
export class className{
..
}
```

Figure 11. Custom directive

4.2.5 Services and dependency injection

Services in Angular represent a wide range of classes that perform some specific tasks, such as retrieving data from a server, storing data, etc (Figure 12) (Angular k).

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class StorageService {
  savedData: number[];

  constructor() {
    this.savedData = [];
  }

  saveData(element: number){
    this.savedData.push(element);
  }
}
```

Figure 12. Storage service

Unlike components and directives, services do not work with views and do not directly affect them. Services solve the problem of code repeating when it is needed to perform the same task in different components and classes. This is achieved by injecting the service into the component that wants to use it. This procedure is called dependency injection and happens when Angular creates an instance of the component class and check if any dependency is required in the constructor (Figure 13). (Angular k.)

```
export class SettingsComponent implements OnInit {

  constructor(private service: StorageService) { }
```

Figure 13. Dependency injection

4.2.6 Metadata

Most elements of the Angular architecture that were described above are classes. To distinguish between these classes, difference decorators like @Component, @NgModule, @Injectable or @Directive are used. They get an object with metadata that explains Angular how to process the class. (Angular l.)

4.2.7 The general picture of the Angular architecture

After having collected the information about the main elements explained above, the Angular architecture can be presented like in Figure 14 and its processes can be divided into some main points:

- Angular consists of at least one module.

- One of the important units of a module is a component that is associated with a template due to metadata.
- Components use services to help process data.
- Directives and data bindings effect on a template, and thus can change the view of its component.

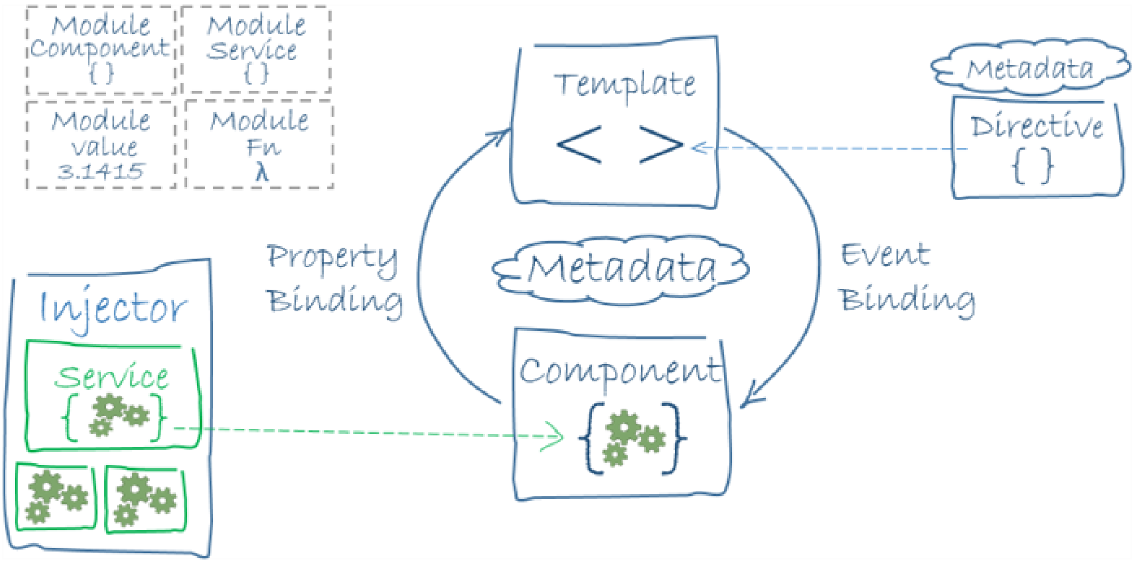

Figure 14. Angular architecture (Angular I)

5 Apache Cordova

Apache Cordova allows building mobile applications using CSS3, HTML5, and JavaScript, instead of using a platform-specific programming language. It does this by converting CSS, HTML, and JS into code that any platform treats as web content. This extends the capabilities of HTML and JavaScript to work with a variety of mobile devices. A Cordova application is called a hybrid application. It means that it is neither truly native applications - because it has not been developed using native developing tools, nor web - because it is not just a web application, but an application that also can use the mobile device functionality and work without an Internet connection. (Griffith.)

JavaTpoint describes a Cordova application as an application that consists of only one page, which occupies all the space on the device screen. This page contains a WebView, which is regarded as an embedded browser that manages web content written on HTML with CSS and JS (Figure 15). Any mobile operating system has its WebView. This fact allows Cordova to create the WebView for a specific operating system, without changing web content. To access the functionality of a mobile device, Cordova uses plugins. They can be official plugins developed by Cordova community or custom plugins, but all of them are written on native code and offer a JavaScript API. The application using JavaScript code connects to native components via this API. (JavaTpoint.)

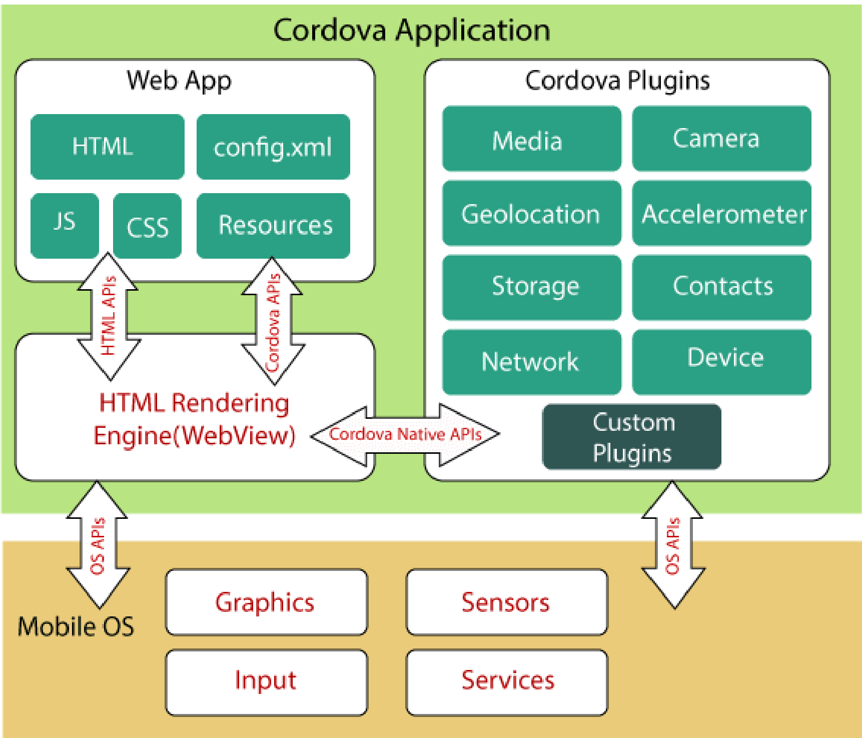

Figure 15. Cordova architecture (JavaTpoint)

6 Implementation

6.1 Installation and setting up the development environment

The created application is essentially composed of two different applications that were merged into one hybrid application. The web application was created using the Angular framework. The wrapper for this web application was created by Apache Cordova. Besides this, it was used a lot of other development tools. Apache Cordova requires the NodeJS software platform because a Cordova command-line interface (CLI) tool runs on it (Apache Cordova a). Since Angular uses the TypeScript language that browsers do not understand, NodeJS is required as well. It transpiles TypeScript code in plain JavaScript code that browsers can render. (Gudelli 2019.) Additionally, NodeJS has a node package manager, which is a command-line utility. It allows installing difference npm-packages from repositories. (OpenJS Foundation 2011.) Since, at the moment, only the version of the application for the Android platform has been created, just the extra developing tools for Android will be listed: the integrated development environment Android Studio, the Java software development kit and the build automation tool Gradle. This set was needed to deploy the Cordova application for the Android platform (Apache Cordova b).

To use the Cordova CLI, the Apache Cordova utility was installed (Figure 16). This CLI was used to create the project, add various platforms and plugins to the project, as well as build and run the application (Apache Cordova c).

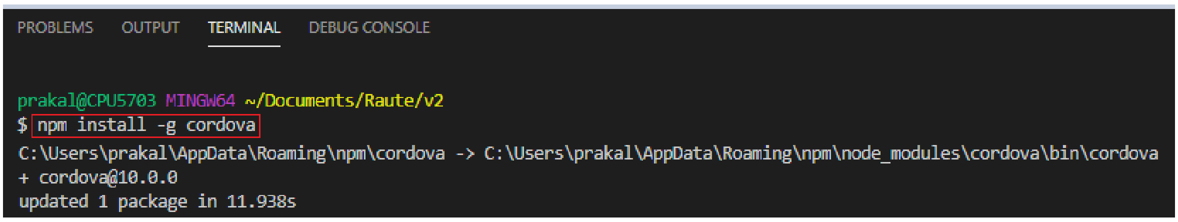

Figure 16. Cordova installation

Every command starts with the special word ‘cordova’. Figure 17 demonstrates how the project with the name hybridApp was created.

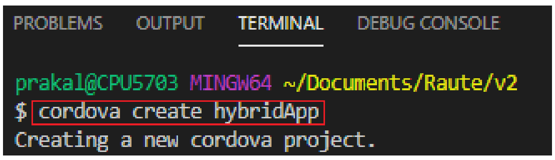

Figure 17. Cordova project creation

After the application was created, it became possible to add the support for the Android platform (Figure 18).

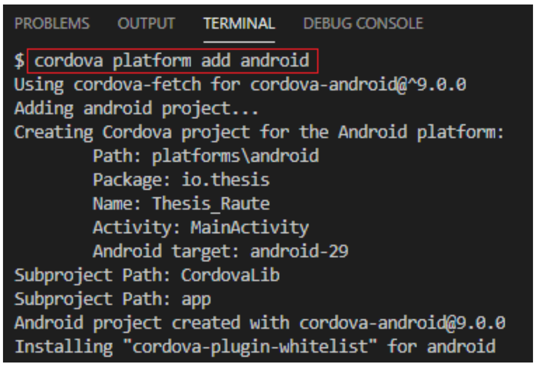

Figure 18. Android platform

Given the need to interact with the functions of mobile devices, the necessary plugins were added to the project. For example, the plugin for working with a camera was added like in Figure 19.

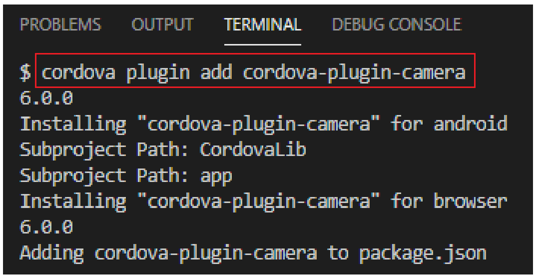

Figure 19. Cordova plugin camera

After adding all dependencies, the final directory structure looked like in Figure 20.

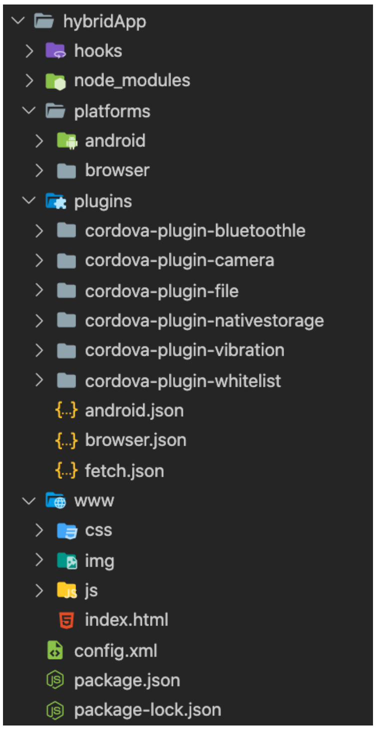

Figure 20. Directory structure

The Apache Cordova (d) documentation explains what each item means:

- The hooks folder stores special scripts that are executed before and after CLI commands.
- The node_module folder contains different libraries that are needed during the development.
- The platforms folder contains subfolders with source code for the target platforms.
- The plugins folder is used to store plugins that a developer adds to the project.
- The www folder is the folder where Cordova takes the files to render.
- The config.xml file is the configuration file that controls application behaviour
- The package.json file describes the project and contains information about the packages that were added to the project.

Angular, like Apache Cordova, managed the project through its command-line interface. To install this CLI, the node package manager was used (Figure 21).

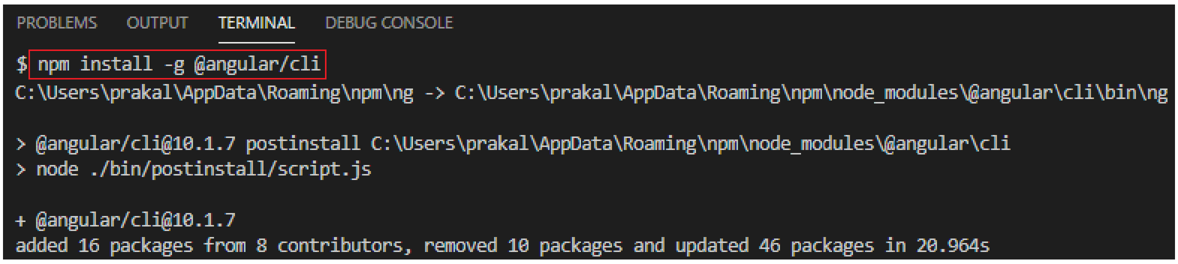

Figure 21. Angular CLI installation

Angular CLI used the special word ‘ng’ to manage the project. Figure 22 shows how the project mobileQC was created. (Angular m.)

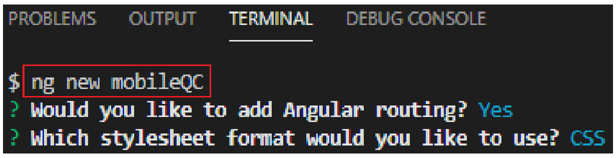

Figure 22. Angular project creation

For this project, 20 components and 8 services were created. The commands in Figure 23 show how these elements were added to the project.

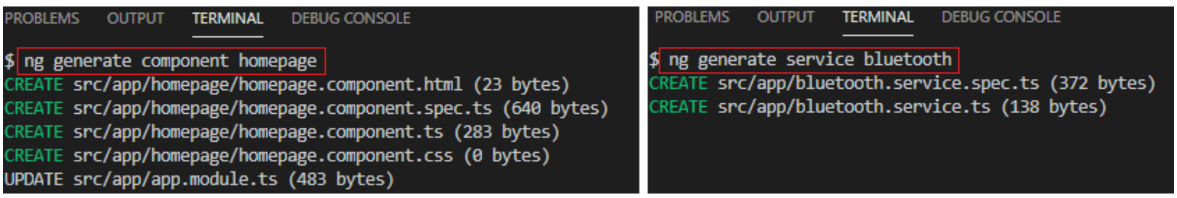

Figure 23. Component and service creation

To compile the project into the output directory the command 'ng build' was used. By default, the compiled files get into the output directory 'dist'. (Angular o.)

Once both projects have been created and configured, they were merged into one project. For this purpose, all content of the Cordova project, except the package.json file, was moved to the root of the Angular project. After that, information from the Cordova package.json file was merged to the Angular package.json file. For the reason that Cordova expects to find web content in the www folder, it was needed to change the Angular output directory value from 'dist' to 'www' in the angular.json file. (Nnanna 2017.)

One more file was modified according to Palme (2018). The index.html file has the head section where the tag <base> content was changed (line 6 Figure 23).

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="utf-8">
5      <title>RauteClient</title>
6      <base href="./">
7      <meta name="viewport" content="width=device-width, initial-scale=1">
8      <link rel="icon" type="image/x-icon" href="favicon.ico">
9      <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
10   </head>
11   <body>
12     <app-root></app-root>
13     <script type="text/javascript" src="cordova.js"></script>
14   </body>
15 </html>
```

Figure 23. index.html

This tag specifies a base URL and by default, its value is href = '/' which means the absolute path. But since Cordova does not handle such paths, this value was changed to the relative path by adding dot before the slash. Also, the script cordova.js was added to HTML code (line 13 Figure 23). This is the library that is used to interact with device hardware. The last file that was modified was the Angular main.ts file. This file bootstraps the application. The point is that Cordova, like Angular, has different lifecycle stages and the application should be bootstrapped only after all resources have been loaded. This lifecycle stage is called 'deviceready' and it was processed by adding the JavaScript event listener to the file (Figure 24).

```
const onDeviceReady = () => {
  platformBrowserDynamic().bootstrapModule(AppModule);
};

document.addEventListener('deviceready', onDeviceReady, false);
```

Figure 24. Event listener

6.2 User Interface and implementation

The first action performed by the application is to check if there is a token already on the mobile device storage. This token stores user data and gives the right to use the application. If there is no token on the phone or the token is expired, the client must go through the verification process by entering the personal username and the password on the login page (Figure 25).

Figure 25. Login page

To check authorization, the service Auth has been created in the application. This service itself uses three different services: the Storage service (line 26 Figure 26), the JwtHelperService service (line 32 Figure 26), and the Request service (line 34 Figure 26).

```
24  isAuthorized(): Promise<boolean> {
25    return new Promise((resolve, reject) => {
26      this.storage.getDataFromStorage('token').then(
27        (res: string) => {
28          this.token = res;
29          this.userInfoService.userInfo.next(
30            jwtHelper.decodeToken(this.token).user.lastname
31          );
32          const isExpired = jwtHelper.isTokenExpired(this.token);
33          if (!isExpired) {
34            this.request.setHeaderToken(this.token);
35            resolve(true);
36          } else { resolve(false); }
37        },
38        // storage is empty
39        (rej) => resolve(false)
40      );
41    });
42  }
```

Figure 26. Authorization checking

In the Storage service, interaction with the phone storage is implemented via the Cordova plugin NativeStorage (GitHub). This plugin keeps, retrieves, or erases data by special reference to value and returns an answer in the callback function (Figure 27).

```
getDataFromStorage(storageId: string) {
  return new Promise((resolve, reject) => {
    NativeStorage.getItem(
      storageId,
      (data) => resolve(data),
      (err) => reject(err)
    );
  });
}
```

Figure 27. Storage service

If the token is retrieved, the JSON Web Token helper service checks it. In the case, if this token is not expired (line 32 Figure 26) it is passed to the Request service (line 16 Figure 28). In any other cases, the application waits if the client fills the fields on the login page and then it sends the username and the password to the Request service (line12 Figure 28).

To transfer the data for verification to the server, the Request service (Figure 28) uses the Angular httpClient service that performs HTTP requests. As was mentioned in section 2.3 of this document, the data that is sent to the server must be sent along with the token that can prove authorization and gives needed information to the server. According to Schmitz (2019), the token can be passed in an Authorization header (line 18, line 23 Figure 28).

```
11  constructor(private http: HttpClient) {}
12  login(url, name, pass): Promise<any> {
13    return this.http.post(url, {username: name, pass}).toPromise();
14  }
15
16  setHeaderToken(token: string) {
17    this.header = new HttpHeaders({
18      Authorization: `Bearer ${token}`,
19    });
20  }
21
22  postRequest(url: string, body): Promise<any> {
23    return this.http.post(url, body, { headers: this.header }).toPromise();
24  }
```

Figure 28. Request service

After successful authorization, the client is taken to the home page (Figure 29).

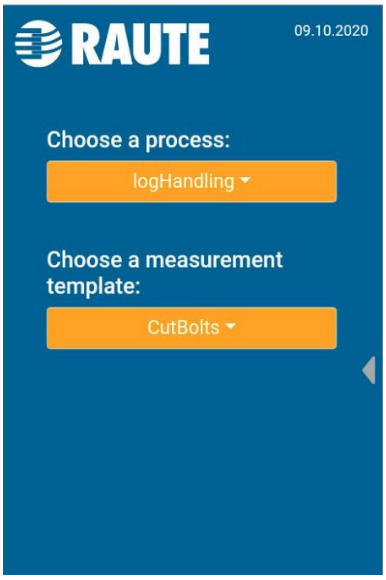

Figure 29. Home page

On this page, the client selects a technological process from the first drop-down list. Based on the selection, the program dynamically creates a list of templates for the second drop-down list. This process uses the Settings service, which is responsible for processing these lists. The interaction between the component and the service was done by using the RxJS library.

The RxJS library is used for transferring data asynchronously, and to handle event-based behaviour of a component. The library works as follows (Figure 30). The service creates the Subject object that is both observable and observer. On the service side, it uses the next() method to transfer data to everyone who has subscribed to receive this data from it. Components use the same object to subscribe to receive data. Thus, every time the data is transmitted, the components receive it.

```
//  service A
const mySubject = new Subject();

mySubject.next('data');
...
mySubject.next('new data');

// some component
...
consctuctor(servA: AService)
servA.mySubject.subscribe( data => console.log('I got ' + data));

//  result is:
//  'I got data'
//  'I got new data'
```

Figure 30. RxJS

After selecting a template, the user goes to the Settings page. This page is conventionally divided into three parts.

The first part shows the basic settings, which are independent of the choice of the template (Figure 31).

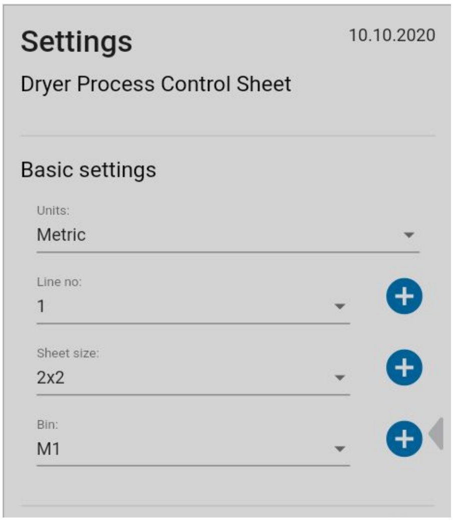

Figure 31. Basic settings

The implementation features include adding new values (Figure 32) and deleting existing ones.

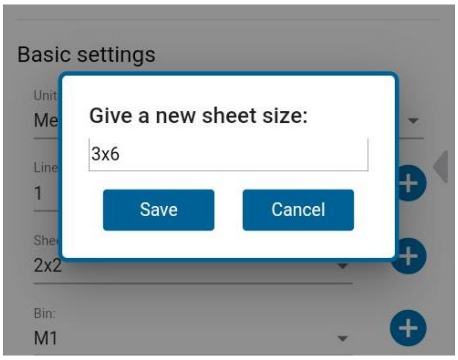

Figure 32. Adding a new value

The Angular service MatDialog was used to add new values. This service provides the ability to use the Dialog component from the Angular Material Component library. First, an object of the MatDialog class was created. This object opens a modal window by the name of the Dialog component (line 327 Figure 33), passes input data to it (line 329 Figure 33) and returns an instance of MatDialogRef. Further, this instance subscribes to receive data from the modal window after it is closed (line 331 Figure 33).

```
addSheetSize() {
  const inputDialogRef = this.dialog.open(InputDialogComponent, {
    panelClass: 'custom-dialog-container',
    data: { hint: 'Give a new sheet size:' },
  });
  inputDialogRef.afterClosed().subscribe((input) => {
    if (input && input.trim().length && !this.sheetSizes.includes(input)) {
      this.sheetSizes.push(input);
      this.settingsForm.controls.sheetSizeSelect.setValue(
        this.sheetSizes.indexOf(input)
      );
    }
```

Figure 33. Sheet size adding

The contextmenu event was used to implement an item removal from the dropdown list. In the browser version, this event is fired when an element is right-clicked. The same event fires on a mobile device, if one makes a long press on the item on the device screen.

The second part of the Settings page contains fields that depend on the user-selected template (Figure 34). These fields are implemented using the Angular MatCheckboxModule module. It allows selecting the required fields for the measurements.

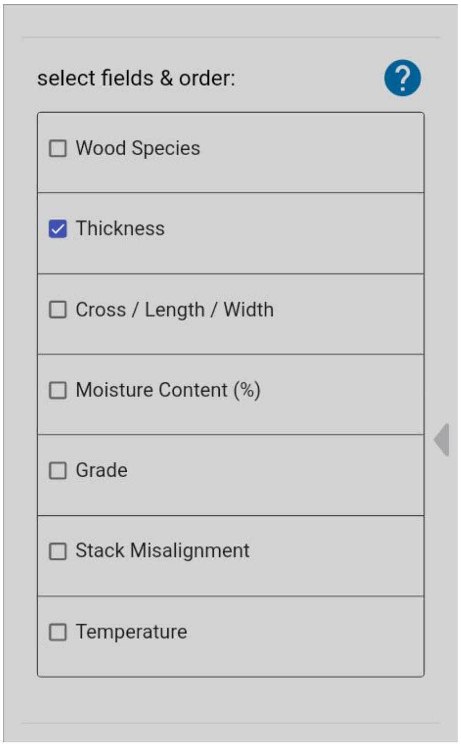

Figure 34. Checkbox fields

Besides that, based on customer requirements, the ability to reorder the fields was implemented (Figure 35).

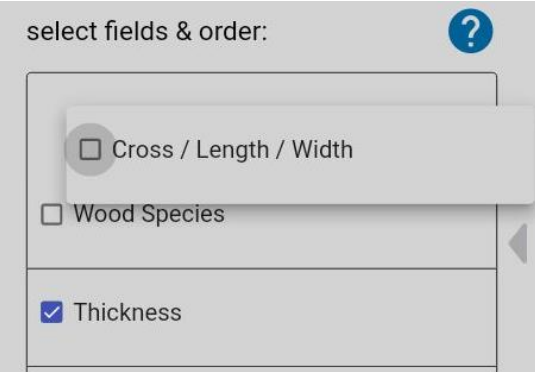

Figure 35. New order on measurements

For these purposes, Angular has the DragDropModule module in which the cdkDropList and the cdkDrag directives are implemented. The general algorithm is shown in Figure 36. The program listens for the drop event and, when the item is moved, starts the moveItemInArray method for execution. This method changes the order of the elements in the array.

```
// html  file
<div cdkDropList (cdkDropListDropped)="drop($event)">
  <div *ngFor="let item of items" cdkDrag>{{item}}</div>
</div>

// ts file
drop(event: CdkDragDrop<string[]>) {
    moveItemInArray(this.items, event.previousIndex, event.currentIndex);
  }
```

Figure 36. Drag and Drop

The last part of the Settings page appears when the user selects Thickness, Cross / Length / Width, or both fields from the list of the fields (Figure 37).

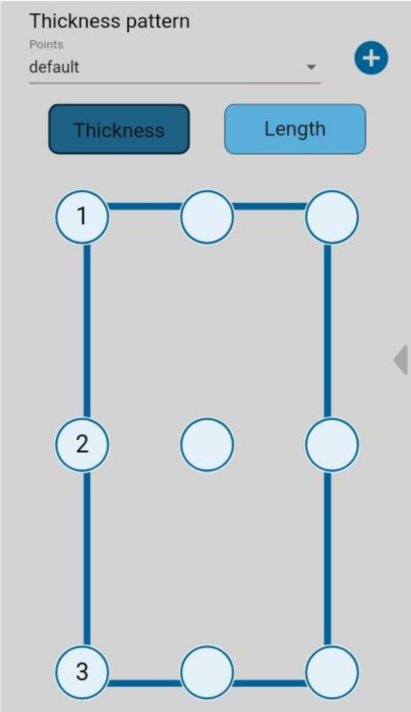

Figure 37. Measurement points

The image is a plywood sheet layout with measurement points. The user chooses which points and in what order to measure. Clicking on the last selected point removes it from the list of the measured points. This behaviour has been achieved with objects properties management. The objects were created based on two object interfaces (Figure 38).

```
1  export interface IPoint {
2    orderOfCreation: number;
3    orderOfMeasurement: number;
4    value: number;
5    isBLE: number;
6  }
7
8  export interface IPattern {
9    name: string;
10   lastIndex: number;
11   pattern: IPoint[];
12 }
```

Figure 38. Interfaces

The object, based on the IPattern interface, stores information about all points on the layout as the array of points (line 11 Figure 38). Each point is an object with a creation sequence number assigned by the application (line 2 Figure 38) and a measurement sequence number assigned by the user (line 3 Figure 38). Using these properties, as well as the lastIndex property (line 10 Figure 38), the program controls the behaviour of the layout.

After confirming the selection, the user proceeds to the measurement page (Figure 39).

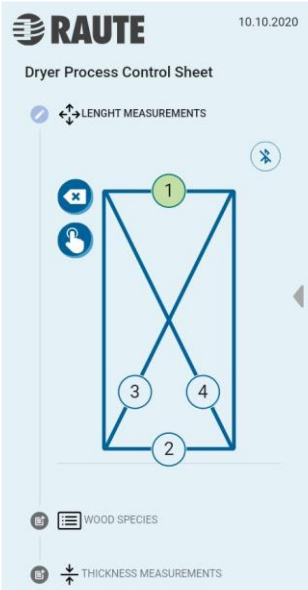

Figure 39. Measurements

The component that is responsible for this page requests data from the Settings service. The component displays the received data using the Angular Stepper module. This module allows displaying one measurement information per step. The transition to the next step is possible only when the data entry at the current step is completed.

Some measurements can be carried out by the user using Bluetooth devices (Figure 40).

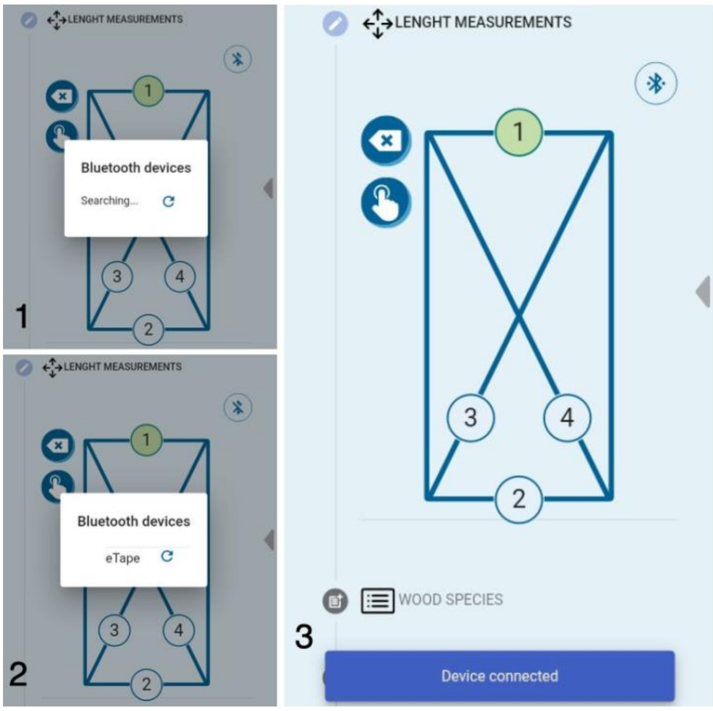

Figure 40. Bluetooth connection

The Bluetooth service was created to interact with Bluetooth devices. This service implemented control over the mobile device Bluetooth module using the Cordova plugin cordova-plugin-bluetoothle. In this plugin, functions are callback-based (Figure 41). For the correct operation of the program, 17 different functions from this library were used. The main steps in the development of the Bluetooth service were initialization, scanning, connection, subscription, disconnection, closure.

```
bluetoothle.method_name(successResult => {/*success Action*/}, error =>{/*error action*/});
```

Figure 41. Bluetooth method

Many fields can only be filled manually on the measurement page (Figure 42).

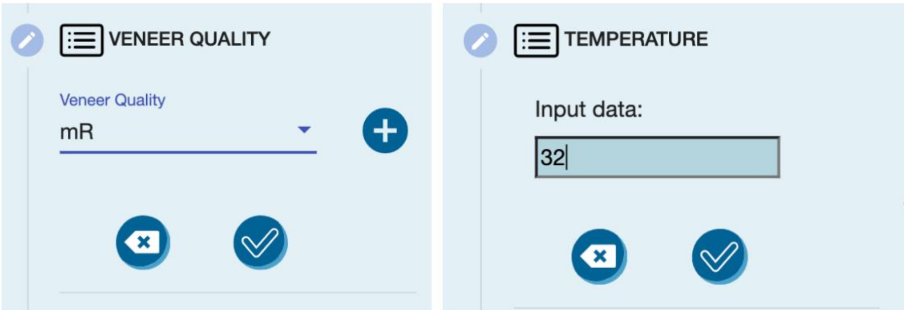

Figure 42. Manual input

Creation of such fields and processing of the entered data is implemented using the Angular Reactive Forms module. The form containing fields is based on the FormGroup class, and input fields are based on the FormControl class. This approach allows managing field values, pass fields as FormControl objects, iterate them, as well as create validation rules.

The final step on the page Measurements allows adding a comment, taking a picture, and sending all measurements to the backend (Figure 43). To work with a camera, Cordova has a Camera plugin. The picture is received into the callback function and it has the base64 encoded string format. The picture is sent to the server in the same format.

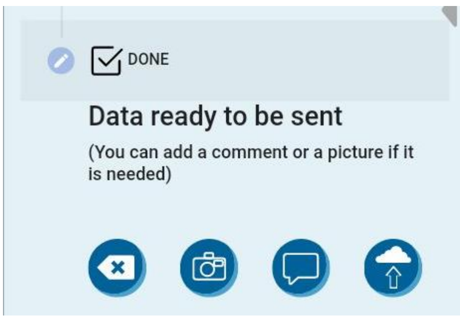

Figure 43. Final step

In case of successful data transfer, the program displays a message about it. If the data transfer fails, the program informs that the user can try to send the data again on the Upload measurements page.

Also, an exclamation mark appears on each page to remind the user of the unsent data. Angular ngIf directive is used to display the exclamation mark on the screen (line 9 Figure 44).

```
9   <div *ngIf="unsentRows!=0" class="unsentAlert">
10    <mat-icon class="unsentSign" (click)="showUnsentWarning()">warning</mat-icon>
11  </div>
```

Figure 44. Exclamation mark

The Result service is responsible for processing and storing the sent data. Also, any component can use this service to check the amount of unsent data. (line 2 Figure 45.)

```
40  ngOnInit() {
41    this.resultService.initObserver.subscribe(res => {
42      this.unsentRows = this.resultService.howManyUnSent();
```

Figure 45. Number of unsent data

The Upload measurements page contains the data of the measurements and shows the amount of unsent data (Figure 46). There is also an icon located opposite each record, which dynamically changes its content depending on the status of the record.

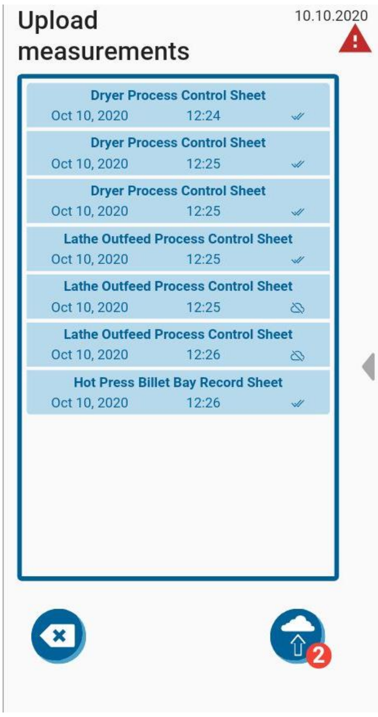

Figure 44. History page

The measurements list is implemented using the Angular MatExpantion module. This module allows displaying an array of data in the form of the expandable detailed view (Figure 47).

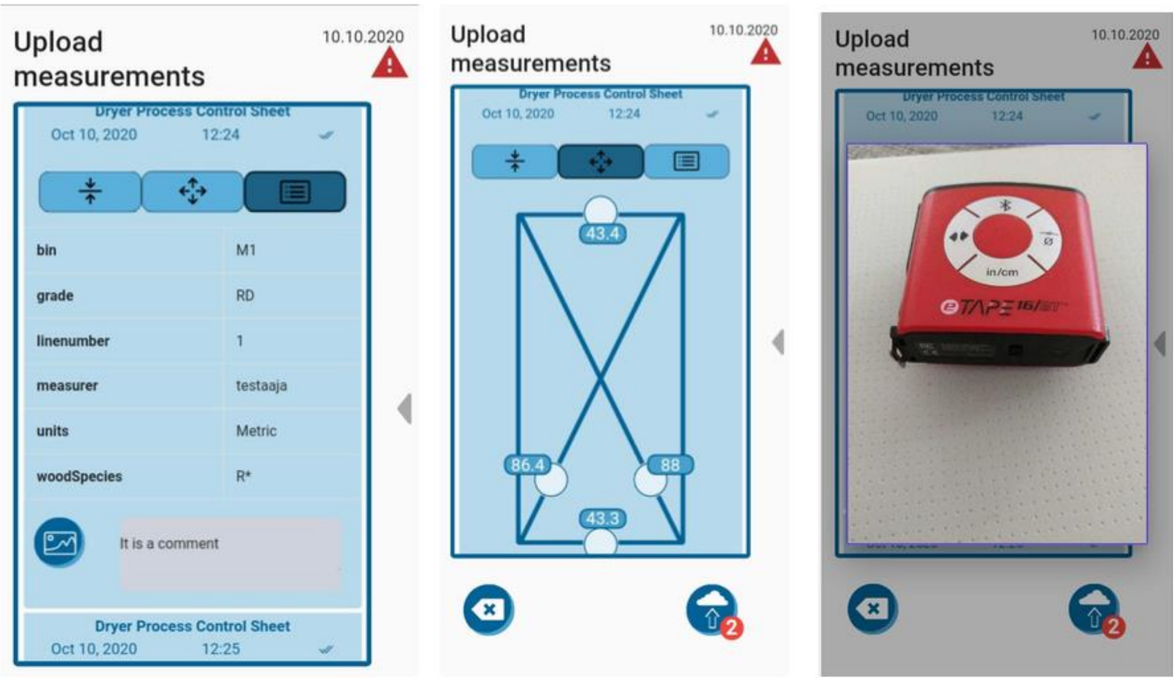

Figure 47. Detailed information

7 Conclusion

The application was commissioned by Raute, the manufacturer of production lines of wood products. The company constantly creates new services for its customers to facilitate the production process. At one of the workshops, Route asked to develop the product quality control application running on various mobile operating systems. The requirement was to create a full-stack system: the client-side (frontend) and the server software part (backend). Although the work was implemented in close cooperation with a backend application developer, the thesis focused on the process of creating the client application.

To accelerate the product creation process, it was decided to develop the hybrid mobile application. The hybrid application combines the development of several programs, different in purpose. Having familiarized with modern trends in the development of this type of application, the Angular framework and the Apache Cordova framework were chosen for work.

Angular has existed for about ten years and its functionality is constantly increasing. In this work, it was impossible to explore all the features of this framework. Therefore, it was decided to describe the core of any framework - its architecture. The thesis describes in detail the interaction of various Angular units, their tasks and features of use. The Apache Cordova framework was described in the same way.

There are not many sources describing the interaction of Angular and Apache Cordova. Therefore, the practical part describes in detail the installation and the configuration of both frameworks. The main functions implementation of the user interface was also described.

The main work on creating an Android application is completed and potential clients' surveys are currently being conducted. The received feedbacks are analysed, and improvements will be made soon. After the final version of the software product is approved, it is planned to install software tools to create a version for iOS.

References

Adach, D. What is Angular? Retrieved on 21 September 2020. Available at https://mdbootstrap.com/education/angular/crash-1-what-angular-is/

AltexSoft 2020a. The Good and the Bad of Angular Development. Retrieved on 21 September 2020. Available at https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/

AltexSoft 2020b. The Good and the Bad of TypeScript. Retrieved on 21 September 2020. Available at https://www.altexsoft.com/blog/typescript-pros-and-cons/

Angular a. Introduction to modules. Retrieved on 22 September 2020. Available at https://angular.io/guide/architecture-modules

Angular b. Introduction to components and templates. Retrieved on 22 September 2020. Available at https://angular.io/guide/architecture-components

Angular c. Hooking into the component lifecycle. Retrieved on 22 September 2020. Available at https://angular.io/guide/lifecycle-hooks

Angular d. Template syntax. Retrieved on 24 September 2020. Available at https://angular.io/guide/template-syntax

Angular e. Interpolation and template expressions. Retrieved on 24 September 2020. Available at https://angular.io/guide/interpolation

Angular f. Attribute, class, and style bindings. Retrieved on 25 September 2020. Available at https://angular.io/guide/attribute-binding

Angular g. Event binding (event). Retrieved on 25 September 2020. Available at https://angular.io/guide/event-binding

Angular h. Two-way binding [(...)]. Retrieved on 25 September 2020. Available at https://angular.io/guide/two-way-binding

Angular i. Built-in directives. Retrieved on 28 September 2020. Available at https://angular.io/guide/built-in-directives

Angular j. Structural directives. Retrieved on 28 September 2020. Available at https://angular.io/guide/structural-directives

Angular k. Introduction to services and dependency injection. Retrieved on 29 September 2020. Available at https://angular.io/guide/architecture-services

Angular l. Introduction to Angular concepts. Introduction to services and dependency injection. Retrieved on 29 September 2020. Available at https://angular.io/guide/architecture

Angular m. Setting up the local environment and workspace. Retrieved on 5 October 2020. Available at https://angular.io/guide/setup-local

Angular n. Ng generate. Retrieved on 5 October 2020. Available at https://angular.io/cli/generate

Angular o. Ng build. Retrieved on 5 October 2020. Available at https://angular.io/cli/build

Apache Cordova a. Get Started Fast. Retrieved on 1 October 2020. Available at https://cordova.apache.org/

Apache Cordova b. Android Platform Guide. Retrieved on 2 October 2020. Available at https://cordova.apache.org/docs/en/9.x/guide/platforms/android/index.html

Apache Cordova c. Create your first Cordova app. Retrieved on 2 October 2020. Available at https://cordova.apache.org/docs/en/latest/guide/cli/index.html

Apache Cordova d. Cordova Command-line-interface (CLI) Reference. Retrieved on 2 Oktober 2020. Available at https://cordova.apache.org/docs/en/9.x/reference/cordova-cli/index.html#directory-structure

GitHub. Cordova plugin NativeStorage. Retrieved on 7 October 2020. Available at https://github.com/TheCocoaProject/cordova-plugin-nativestorage

Griffith, C. What is Apache Cordova? Retrieved on 30 September 2020. Available at https://ionicframework.com/resources/articles/what-is-apache-cordova

Gudelli, A. 2019. Is Node.Js Required For Angular 2 Or Angular? Retrieved on 1 October 2020. Available at https://www.angularjswiki.com/angular/is-node-js-required-for-angular-2-or-angular/

JavaTpoint. Cordova Architecture. Retrieved on 30 September 2020. Available at https://www.javatpoint.com/cordova-architecture

Nnanna, J. 2017. Convert your Angular (v7 updated) project to mobile app using Cordova. Retrieved on 6 October 2020. Available at https://medium.com/@nacojohn/convert-your-angular-project-to-mobile-app-using-cordova-f0384a7711a6

OpenJS Foundation 2011. What is npm? Retrieved on 1 October 2020. Available at https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

Palme, E. 2018. How to wrap an Angular app with Apache Cordova. Retrieved on 6 October 2020. Available at https://medium.com/@EliaPalme/how-to-wrap-an-angular-app-with-apache-cordova-909024a25d79

Raute a. Retrieved on 15 September 2020. Available at https://academy.raute.com/

Raute b. About Raute. Retrieved on 9 October 2020. Available at https://www.raute.com/about-raute

Schmitz, H. 2019. Angular Authentication with JWT. Retrieved on 12 October 2020. Available at https://developer.okta.com/blog/2019/05/16/angular-authentication-jwt

Srashti, J. 2018. Demystified Angular Databinding. Retrieved on 24 September 2020. Available at https://www.dotnettricks.com/learn/angular/data-binding-methods

ThinkMobiles. What are the popular types and categories of apps. Retrieved on 17 September 2020. Available at https://thinkmobiles.com/blog/popular-types-of-apps/