FlowMarkt Final Report

1. Introduction

FlowMarkt lowers transaction costs of recycling: time spent presenting and finding items, negotiating conditions as well as delivering goods. Users are geolocated, and near proximity buyers are preferred. FlowMarkt integrates to social media and digital communication tools to bring together remote users. FlowMarkt tracks “strength of trust”, which is calculated from past experiences, proximity and connection strenth between parties. Trust between parties is interpreted as “social capital” which automatically gives priority when FlowMarkt recommends further sale or exchange possibilities.

2. Design and Implementation

2.1 Design of services

OpenApi based dsl's were used during design phase to document api's. Code generation for rest service and client was planned to be based on Open Api spec, Swagger, Angular swagger generator and loopback swagger generator.

Application backend was planned to be implemented using Loopback, which is specialization of Express framework. LoopBack generator is able to generate rest endpoints from Open Api DSL's, but also models and crud operations for all supported db's, developer needs just to tie components together. For persistence was considered MongoDB or Cloudant.

2.1 Implementation of services

First implementatation of backend was json server. All fine here. Loopback 2.X and 3.X weren't tested, but instead IBM's commercial api connect product, which is extension of Loopback. IBM Api connect documentation was confusing: User interfaces and terminology seems to have changed pretty often, and hello world sample application with single message didn't work at all.

IBM Api connect worked partially. Generating models and using them locally with curl was ok (-k or –insecure added to each request to bypass ssl checks), but using insecure (self signed) ssl's from Angular app was showstopper. This isn't in any way Angular or IBM Api Connect specific problem.

During prototyping Api Connects Notes sample model was extended, and open api based model crafted during desing phase wasn't taken in use. This was partially up to time constraints, since there didn't seem to have enough time to start from clean slate. It might be that I later see how it would have worked to write completely new api here.

Changed Note type has 5 attributes (time and author added), of which one (title) is required.

```
properties:
  time:
    type: string
  author:
    type: string
  title:
    type: string
```

```
      content:
        type: string
      id:
        type: number
        format: double
    required:
      - title
    additionalProperties: false
```

Prototyping backend services process was:
- create api connect project
- start api connect edit web app
- extend existing model
- start api connect and test api's locally (partial success, curl ok)
- deploy api to bluemix, test api (failed due to ssl, no possibility to use http)
- convert bluemix api deployment from dea architecture to diego by hand
- test again (and it worked, as http was accepted)

Testing was done with curl from command line before services were integrated to client

```
curl --request POST \
  --url https://127.0.0.1:4002/api/Notes \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --header 'x-ibm-client-id: default' \
  --header 'x-ibm-client-secret: SECRET' \
  --data
'{"time":"0:59","author":"lopfiva","title":"fozofm","content":"kifeddeh","id"
:11.08522878}' -k

curl --request GET \
  --url 'https://127.0.0.1:4002/api/Notes' \
  --header 'accept: application/json' \
  --header 'content-type: application/json' \
  --header 'x-ibm-client-id: default' \
  --header 'x-ibm-client-secret: SECRET' -k
```

Curl worked. Calls from angular not.

During development IBM api connect was used locally with micro gateway, which worked ok. When deployed to IBM BlueMix DataPower gateway I tried to both configure SSL in use (only supporter for commercial license) and to use plain HTTP – both trials failed.

Here's small excerpt from log

```
Management URL:
https://new-console.ng.bluemix.net/apps/e5c7e9eb-aacd-4e75-aa19-20e4e75caab7
API target urls:
apiconnect-e5c7e9eb-aacd-4e75-aa19-20e4e75caab7.nikki-consulting-dev.apic.mybluemix.net
API invoke tls-profile: client:Loopback-client
Updated flowmarkt-server.yaml with tls-profile and target-url.
```

```
Found 1 files to publish.
Error during products publish: API level schemes must be set to use only "https"
```

After initial failure to implement services with Api Connect BlueMix DEA architecture was migrated to CloudFoundry Diego architecture and after deployment it suddely seemed all to work. No https, but plain http. No errors. Running. Miracle! (Happy was *I)*

Url for getting all items is:

http://apiconnect-e5c7e9eb-aacd-4e75-aa19-20e4e75caab7.nikki-consulting-dev.apic.mybluemix.net/api/Notes/

Note: Http. Existing url structure from Sample app.

2.3 Design of user interfaces

At design phase it was decided that app would later target mobile as progressive web app (PWA). Authentication was planned to happen using extenal identity providers and payment, if needed, using external payment providers.

2.4 Implementation of user interfaces

Application frontend was implemented using Angular (2.4). Angular-cli (1.0.beta..) was used to generate seed application, needed components, typescript classes, etc.

Cli generates routing module if new application is created with “--routing” option. Routing generation was forgotten, but this mistake was repaired later by hand.

Angular-Cli worked as expected, and even if result was working build pipeline, lot of generated template files, and not that much more, helped scaffolding to keep up consistent naming and structure of project.

Angular-Cli and Angular itself were stable, but not perfect. Some errors which came from missing module level imports were daunting to debug, since webpack had already collected everything to bundles and problems were only to be seen runtime. Confusion here was up to two implementations of Forms Angular has: I was using first parts from ReactiveForms without importing needed modules.

TypeScript was used instead of JavaScript. TypeScript is fantastic for java and C#/C++ developers: static typing to dynamically typed language. TS was used and it worked fine. Code is easy to write and compact. Generics, interfaces, .. nice..

Example TS dto (data transfer object) with 5 attributes:

```
export class FmItem {
  // added constructor: no need to define members in any other method
  constructor (id: number, title: string, content: string, author: string, time: string) {}
}
```

Angular http was used as out of box library, as it's reactive nature is pretty powerful. Angular-http uses internally Observables from RxJs, which makes asyncronous json calls elegant. It's possible to give observable to html template, and in html markup define that results are resolved asynchronously from reactive stream.

Example of reading all items from remote source in fm-item.service.ts

```
@Injectable()
export class FmItemService {

  (..)

  getAllItems() : Observable<FmItem[]>{
    return this.http
        .get(`${this.baseUrl}/${this.baseTag}`, {headers: this.headers})
        .map(response => response.json())
        .catch(this.handleError);
  }
```

How nice: getAllItems () returns Observable, which is using generics defined to have as return value list of FmItem's. Http itself is getting value as json response, mapping response to json structure, and after this Observable is returned if exceptions aren't catched.

Example of getting all items using service in fm-item-list.component.ts

```
export class FmItemListComponent implements OnInit {

  private items: Observable<FmItem[]>; // define result

  // inject service
  constructor(private itemService: FmItemService) {}

  ngOnInit() {
    this.items = this.itemService.getAllItems(); // get all items
  }

}
```

Important is that Observable is not subscribed here, but given to html to iterate over.

Example of showing all items from remote source in fm-item-list.component.html

```
<!-- items with async subscribes to returned observable-->
<div class="row" *ngFor="let item of items | async">
  <h3>{{item.title}}</h3>

  <!--truncate is used to shorten text of item.content if text is over given length -->
  <div>{{ item.content | truncate:250 }}</div>

  <!--link with id from current item can route user to details -->
  <a class="link" [routerLink]="['/items',item.id]">More ...</a>
  </div>
</div>
```

Note how async is used to define that subsribe starts here asyncronous operation to read data, and each item is used to render one row. In example is also shown how links are created.

Angular Forms can be written as template based or reactive. Template based is simpler and used here. Reactive forms give more flexibility and might fit better in when there's some other rendering than html used also. Template based was used as it was easier and worked.

It was thought that app would later target mobile as progressive web app (PWA). PWA approach wasn't tested as part of this project, but I believe implementation can be grown to direction of PWA. Here problem might be simple template based forms I was using just to get fast forward, but they are easily converted to Reactive declarative forms with form model.

Responsive layout was composed using Bootstrap. Angular material could have been used also, but there wasn't simply time to try it.

User management was planned to be implemented using external identity and access management (iam) solution like Keycloack (oss) or Auth0 (commercial). Security rules like passing authorization token to backend services could be then modeled using Open api spec (api key or oauth2 flow). KeyCloack and Auth0 weren't tested.

IBM Api connect supports user management out of box, but it wasn't used. It was tested how one can configure endpoints per environment, and at that point also sending application security credentials as part of http request (http header fields) was tested. Usage of headers enable later usage of JWT.

It was decided that if payment support is needed it's done using external payment gateways and integration solutions like Braintree or Speedly. Payment methods were not tested.

Angular Cli can be used to automatically publish app to github pages, which might be just ok during initial development to share results and get feedback. Easy as this: “ng github-pages:deploy”. Backend logic and database need to be hosted separately.

Unfortunately this didn't quite work and would have also needed changes to app as github pages has problems with spa's.

```
$ ng github-pages:deploy
Hash: ffe721bf92458c282331
Time: 103541ms
chunk {0} main.bundle.js, main.bundle.map (main) 3.47 MB {3} [initial] [rendered]
chunk {1} styles.bundle.css, styles.bundle.map, styles.bundle.map (styles) 120 kB {3} [initial] [rendered]
chunk {2} scripts.bundle.js, scripts.bundle.map (scripts) 617 kB {3} [initial] [rendered]
chunk {3} inline.bundle.js, inline.bundle.map (inline) 0 bytes [entry] [rendered]
stdout maxBuffer exceeded
Error: stdout maxBuffer exceeded
 at Socket.<anonymous> (child_process.js:259:14)
 at emitOne (events.js:96:13)
 at Socket.emit (events.js:188:7)
 at readableAddChunk (_stream_readable.js:176:18)
 at Socket.Readable.push (_stream_readable.js:134:10)
 at Pipe.onread (net.js:548:20)
```

Next trying Firebase. Simple, straightforward, but only after defining public directory as dist, removing first deployed project (which started to work ok locally after changing firebase.json, but never worked as hosted) and creating new project, which unfortunately wasn't automatically found by command line tools, and seeing that firebase cli's command line help missed way to define project id.

Here we go.
- create firebase account at https://console.firebase.google.com/
- go to your project directory
- ng build -prod
- npm install -g firebase-tools
- firebase login
- firebase init (give dist as public directory, index.html should not be overridden)
- firebase serve
- firebase deploy (when needed with --project <id-here> option)

On the process I needed to change url's at environment.ts to point production url's.

Result can be seen at:

https://flowmarkt-client.firebaseapp.com/

Unfortunately SOP blocks all traffic as in CORS header there's address of local installation Angular CLI uses to serve content.

```
Cross-Origin Request Blocked: The Same Origin Policy disallows
reading the remote resource at
http://apiconnect-e5c7e9eb-aacd-4e75-aa19-20e4e75caab7.nikki-con
sulting-dev.apic.mybluemix.net/api/Notes. (Reason: CORS header
'Access-Control-Allow-Origin' does not match
'http://localhost:4200').
```

Please test with Chrome by allowing “unsafe scripts” until this problem is solved

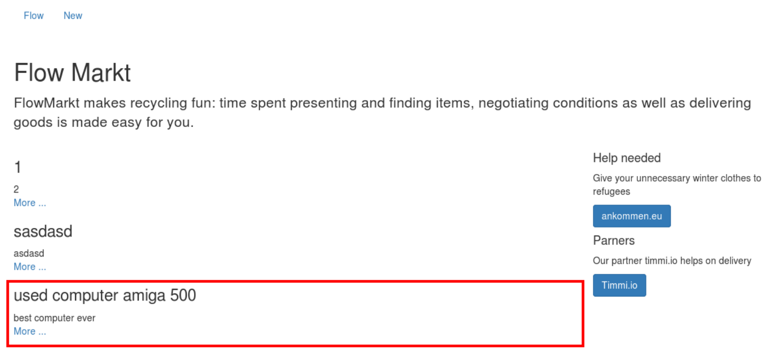
- https://support.google.com/chrome/answer/1342714

3. Screenshots

Application has suffered conciderable scope creeping, but contains currently

- view to list all items, view to look details of single item, view to add new item

3.1 list view (opening page)


Flow New

Flow Markt

FlowMarkt makes recycling fun: time spent presenting and finding items, negotiating conditions as well as delivering goods is made easy for you.

1

2
More ...

sasdasd

asdasd
More ...

used computer amiga 500

best computer ever
More ...

Help needed

Give your unnecessary winter clothes to refugees

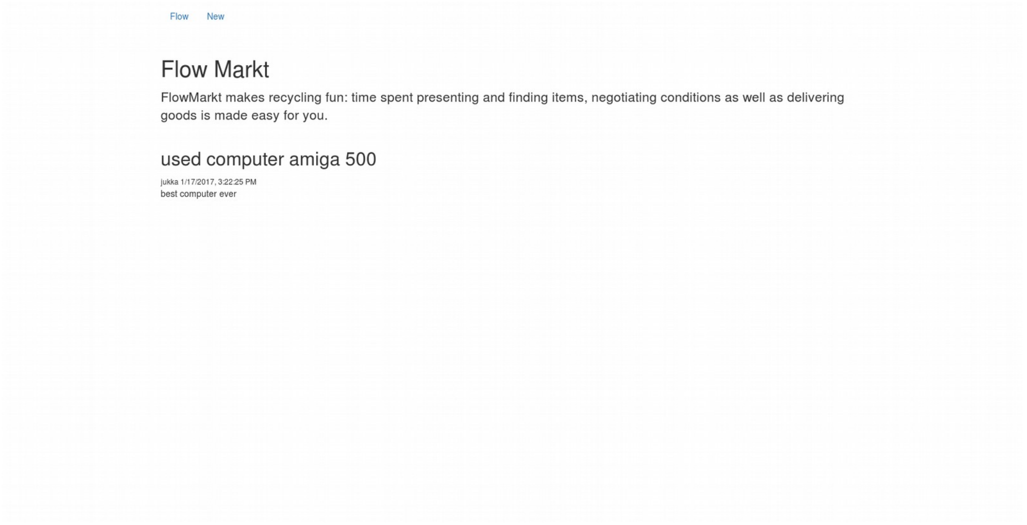
ankommen.eu

Parners

Our partner timmi.io helps on delivery

Timmi.io

3.2 details view (single selected item page)


Flow New

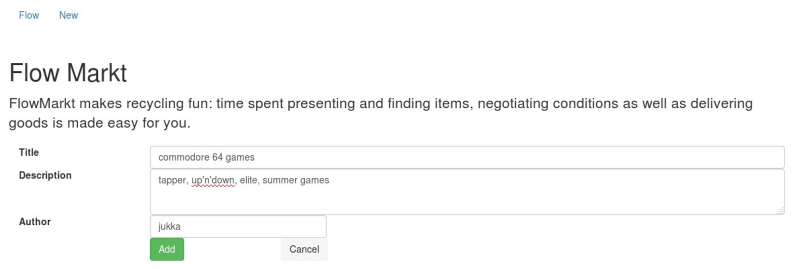
Flow Markt

FlowMarkt makes recycling fun: time spent presenting and finding items, negotiating conditions as well as delivering goods is made easy for you.

used computer amiga 500

jukka 1/17/2017, 3:22:25 PM
best computer ever

3.3 add item view (form to add new items)


Flow New

Flow Markt

FlowMarkt makes recycling fun: time spent presenting and finding items, negotiating conditions as well as delivering goods is made easy for you.

Title	commodore 64 games
Description	tapper, up'n'down, elite, summer games
Author	jukka

Add Cancel

4. Conclusions

Trying to hit moving target is hard. Running to Angular 2 train in middle of learning AngularJS 1 did take some energy, but it was worth it. Angular 2 and Angular-Cli gave clear structure and TypeScript was pretty familiar language for OO-programmer like me.

I feel Angular is good framework for structured enterprise apps, but it might not be perfect choice for everyone. Just take React, or better Cycle.Js, if you're eating bits and pissing bytes, or Elm is you feel like being pure is THE goal. For others Angular does the work.

I'm not decided if Javascript is server side technology of choice for me. It's pretty astonishing what one can generate from DSL's and then extend to full blown app, but for server side space there's also lot of other good options. If Javascript is your first or second programming language just keep with it (tame it with TypeScript, please) and do the whole end-to-end tube with it, but if there's already more programming languages in your CV please wait to see use case before selecting backend technology for project in hand.

Firebase was hosting option that came to me as option when everything else seemed to fail. I'm curious to try persistence and authentication features of Firebase, possible with AngularFire module. If present persistence models are easy to use this might be way to go for streightforward projects.

Now after this course, some extra courses from sitepoint (bootstrap, sass, es6, ..) and reading Secrets of Javascript Ninja (2nd ed, manning), Funtional Programming in Javascript (manning) and Reactive Programming with RxJs (Pragmatic programmer) I feel I'm somewhere near to understand what I do. I'm reading currently Angular 2 Development with Typescript and hope it helps me to tie all together to cohesive base knowledge.

For everyone wanting to learn ES6 and have good base knowledge of JS I can recommend Secrets of Javascript Ninja (2nd ed, manning). Other books I've read are also ok, but nothing beats understanding basics well when it comes to debugging & fine tuning "80% ready" code – which just has this one small problem that it crashes sometimes ..

Biggest challenge to me was amount of time I managed to invest. Having family, 2 kids, didn't make it any simpler. It seems that in real world everything takes quite lot of time, and entropy is waiting always around the corner.

Sorry to say, but it's not all ready here: if I have time I'll update readme files to github for more info. Currently github contains random notes, which aren't in sync with code. You might still want to read them, but please see if code has changed already to contain more complete code example.

Github for client side code (see comments in code, this is best I can now do)
https://github.com/nikkijuk/fullstack-web-developer-transformation/tree/master/fs-capstone/client

Github for server side code (completely uncommented, mostly generated)
https://github.com/nikkijuk/fullstack-web-developer-transformation/tree/master/fs-capstone/server

5. References

- Open Api initiative
    - https://www.openapis.org/
- Api Connect
    - https://developer.ibm.com/apiconnect/getting-started/
- CloudFoundry Diego
    - https://www.ibm.com/blogs/bluemix/2016/11/bluemix-cloud-foundry-upgrading-dea-diego-architecture/
- Angular CLI
    - https://github.com/angular/angular-cli
- Angular production and development configuration
    - http://tattoocoder.com/angular-cli-using-the-environment-option/
- Angular material
    - https://material.angular.io/
- Deploy using angular cli
    - http://developer.telerik.com/featured/quick-angular-2-hosting-angular-cli-github-pages/
- Firebase
    - https://console.firebase.google.com
    - https://firebase.google.com/docs/cli/
- Firebase storage
    - https://firebase.google.com/docs/storage/
- Firebase authentication
    - https://firebase.google.com/docs/auth/
- Deploy to Firebase tutorial (partially outdated)
    - https://medium.com/codingthesmartway-com-blog/hosting-angular-2-applications-on-firebase-f194688c978d#.bs5u4s4tg