# FlowMarkt Application Design and Software Structure 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

FlowMarkt uses dsl's to describe api's and models. Dsl's are to be used to document api's, automate tests (when possible) and generate artifacts. Code generation for rest service and client is based on Open Api spec, Swagger, Angular swagger generator and loopback swagger generator.

Application backend will be implemented using Loopback, which is specialization of Express framework. LoopBack generator is able to generate rest endpoints, but also models and crud operations for all supported db's, developer needs just to tie components together. For persistence we use mongoDB or if it's needed as service Cloudant could be good fit as well.

Application frontend will be implemented using Angular (>= 2.X). We use Angular-cli to generate seed application, needed components, typescript classes, etc. Cli generates routing module if new application is created with “--routing” option. Cli isn't handy only because it helps scaffolding app, but it also follows clear guidelines on naming and dividing components, includes live reloading support, builds with webpack, testrunners, etc.

Important notice is, that none of the choices done early on shouldn't compromise architectural guideline of targeting mobile as progressive web app (PWA). Cli will later on support creating PWA using “--mobile” option like “ng new pwa-app –mobile”. When project evolwes it's possible to add lighthouse to build pipeline to check that implementation develops to right direction.

It's all http & json, but it's not clear if rest api should be called using generated client library (swagger generators might not generate optimal code) or directly with Angular-http. If Angular-http is used we can use Angular-jwt extension from auth0 to ease usage of javascript web token (jwt). It should be noted that Angular-http uses internally Observables from RxJs, which makes asyncronous json calls elegant.

User management is implemented using external identity and access management (iam) solution like Keycloack (oss) or Auth0 (commercial). Security rules like passing authorization token to backend services are modeled using Open api spec (api key or oauth2 flow). External security system which

uses current standards is superior to any home-made “almost secure” system, but also allows easy integration and scales when complicated authorizations schemas might be needed in future.

If payment support is needed it's done using external payment gateways and integration solutions like Braintree or Speedly. No own hacks here. No direct integrations to single payment methods. Payments won't be published before they are initially needed, since there's possibility to fraud.

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.

For IDE there's free Visual Studio Code, but Webstorm should be also concidered. TypeScript is fantastic idea for java developers: static typing to dynamically typed language. With TS one needs to see what EcmaScript features currently used TS version is including. Angular might not use most recent TS version.

Fallback plan: If all goes wrong whole backed is implemented using PostgRest. Sql is then DSL for datamodel and rest api's, which should work pretty well.

### 2.1 The REST API Specification

Api is described using Open Api dsl. Definitions under aren't complete: groups, tags, resources (image, video, pdf) aren't yet included. Definitions are here to show direction of development.

Every get request which returns list allows usage of page and pageSize query parameters for pagination support.

User Management

| Method | Uri | Params | Returns | Description |
|---|---|---|---|---|
| put | signup | name, pw, pw | User, id created | Create user |
| post | login | Name, pw | user | Login user |
| post | logout | user | | Logout user |

Users needs to be created to system (signup), authenticated as part of creation of session (login) and authentication invalidated as part of tearing down session (logout).

Profile is kept separate from user even if every user has profile, which means that profile and user have 1=1 relationship. This means that to update users information one needs to use profile endpoints.

Items

| Method | Uri | Params | Returns | Description |
|---|---|---|---|---|
| get | items | search criteria | array of items | Get list of items |

| | | | | |
|---|---|---|---|---|
| post | items | new item | Item, id created | Create item |
| get | Items/{id} | id | item | Get item |
| put | Items/{id} | Id, item | item | Update item |
| delete | Items/{id} | - | - | Delete item |

It's all about items. Items can be searched, created, details of items can be seen, updated and items can be deleted. Items always have exactly one owner. Items owner has rights to update or delete item.

Additional resources like images, videos or pdf's are added and retrieved using separate calls which return resources. As resource are identified using url's they can also reside in extenal system, but access needs to be controlled using access rights. Access rights are same as to items.

Messages

| Method | Uri | Params | Returns | Description |
|---|---|---|---|---|
| get | messages | search criteria | array of messages | Get list of messages |
| post | messages | new message | message | Create message |
| get | messages | Empty or id | message | Get message |
| put | messages/{id} | message | message | Update message |
| delete | messages/{id} | - | - | Delete message |

Message is always written by single user (owner) and relates to item or other user (receiver). When related to item message can be read by owner of item. No other person than owner, receiver or items owner can ever see message. Only owner of message can delete or update messages.

As there's always only two parties in discussion messages don't need to be threaded and can be simply sorted by timestamp of original posting moment. Messages can be changed later, but as created and  updated timestamps are kept separate sorting doesn't change.

Profile

| Method | Uri | Params | Returns | Description |
|---|---|---|---|---|
| get | Profile/{id} | Id | profile | Get profile |
| post | profile/{id} | Id, profile | profile | Create profile |
| put | profile/{id} | Id, profile | profile | Update profile |

For simplicity profile is kept as one massive json structure containing all you ever think you could configure. This might not be optimal, but it gives lot of flexibility, as api's and backend doesn't need to be changed when new profile information or usage settings are introduced.

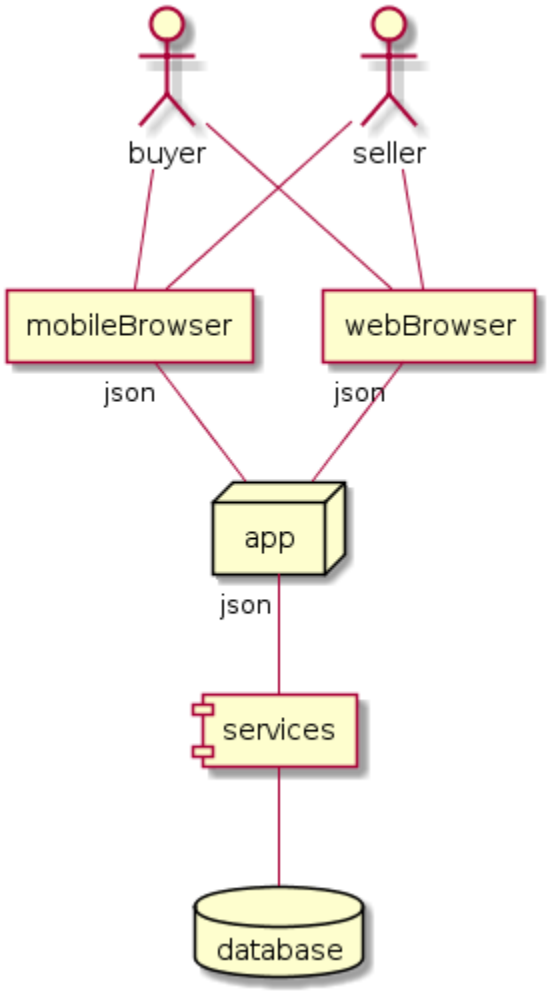
Security constraints are not yet defined, but most propably JWT tokens are sent in header “Authorization” field and checked in backend.

## 2.2 Front-end Architecture Design

Aim of application development is to develop progressive application which adapts itself to mobile or web user (capabilities of browser / device / usage situation).

Application is thin mv* app and user service endpoints defined using open api (swagger.io) dsl.

Communication between application and service endpoints is done using json.



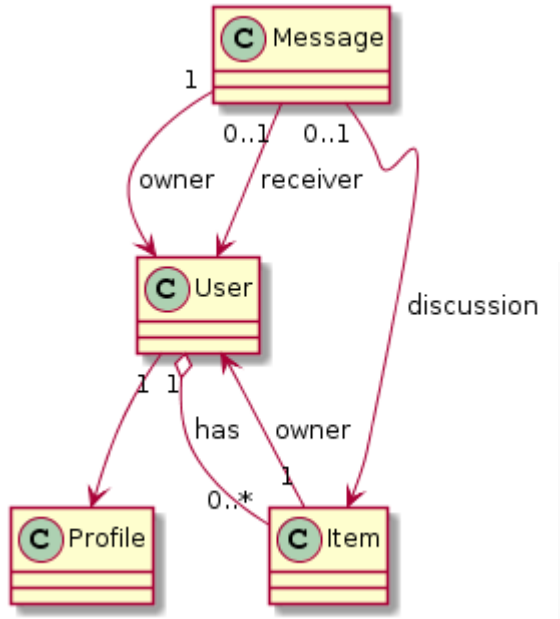

```
@startuml
actor buyer
actor seller
agent mobileBrowser
agent webBrowser
node app
component services
database database

buyer -- mobileBrowser
seller -- mobileBrowser
buyer --  webBrowser
seller --  webBrowser
mobileBrowser "json" -- app
webBrowser "json" -- app
app "json" -- services
services -- database
@enduml
```

App artifacts, services and database are all preferably hosted by 3rd parties and scale when there starts to be additional transactions. Pictures and other blobs can be hosted externally to system itself.

## 2.3 Database Schemas, Design and Structure

Database structure is pretty simple. User is central class, which is linked to exactly one profile, can have multiple items, can send and receive messages directly or thru discussions connected to items. User can be read as “seller” or “buyer”.

```
@startuml
User "1" --> Profile
User "1" o-- "0..*" Item : has
Item "1" --> User: owner
Message "1" --> User : owner
Message "0..1" --> User : receiver
Message "0..1" --> Item : discussion
@enduml
```




Referental integrity rules and references between objects can be controlled programmatically, which makes NoSql databases fitting here. I don't really see if there's reasons to rule out relational databases, except that it might be easier for partitioning and sharing data to multiple servers (maybe because of amount of data, but also to have data close to services If services can be distributed globally to be close to users location).

2.4 Communication

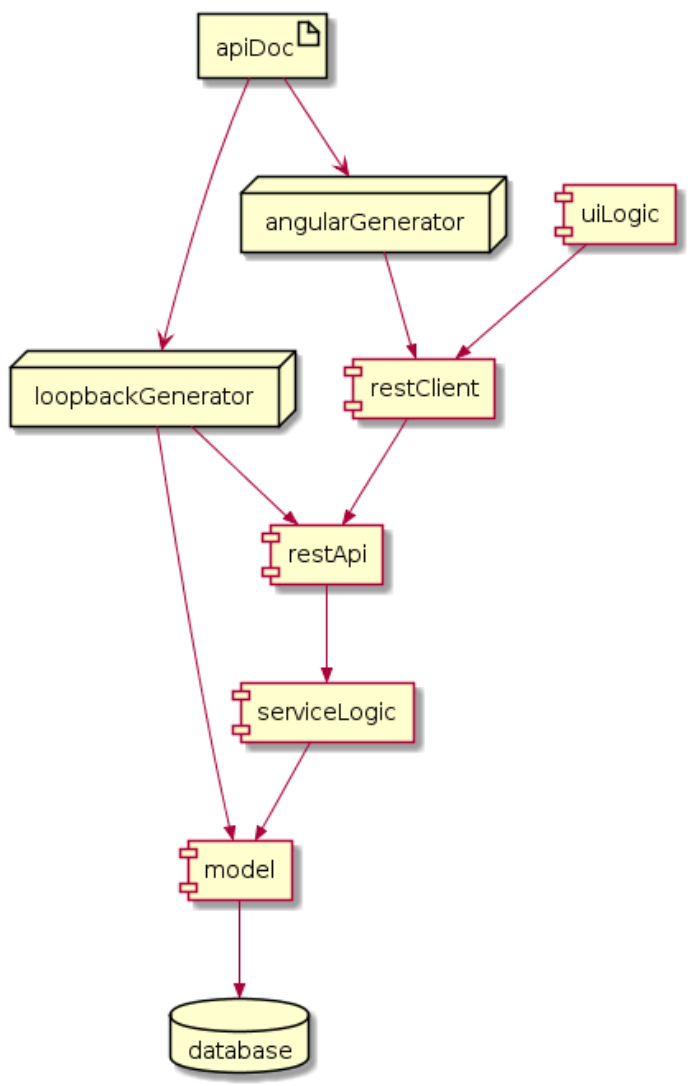
Messages are expressed in json. See appendix 1 for structure. Note that example is taken from pet store and modified to contain only proof of concept messages. Real message content is to be interactively added as implementation of ui is getting forward.

If it feels strange that decisions are pushed forward please consider how much learning possibilites that kind of “decide as late as possible” strategy is including before decision. If decisions would be done early on lot of learning would need to be taken in thru change management or existing implemention would need to be refactored heavily.

## 3. Conclusions

Api docs are written using swagger yaml dsl, which is used to generate service skeleton, model and rest client code. Ui logic and service logic needs still to be added.

```
@startuml
artifact apiDoc
node loopbackGenerator
node angularGenerator
component restClient
component restApi
component model
database database
component uiLogic
component serviceLogic

apiDoc --> angularGenerator
apiDoc --> loopbackGenerator

angularGenerator -->> restClient
loopbackGenerator -->> restApi
loopbackGenerator -->> model

uiLogic -->> restClient
restClient -->> restApi
restApi -->> serviceLogic
serviceLogic -->> model
model -->> database
@enduml
```




Note: as swagger publishes api metadata thru http url it is also possible to consume it from angular without generation. What approach is taken is decided after PoC (Proof of Concept) is done.

## 4. References


Last responsible moment
- http://wirfs-brock.com/blog/2011/01/18/agile-architecture-myths-2-architecture-decisions-should-be-made-at-the-last-responsible-moment/

UML dsl used in documentation
- http://plantuml.com/

Rest api & Json style guides
- https://blog.philipphauer.de/restful-api-design-best-practices/
- https://google.github.io/styleguide/jsoncstyleguide.xml

Swagger
- https://blog.philipphauer.de/enriching-restful-services-swagger/
- http://editor.swagger.io/#/
- https://apihandyman.io/openapi-trek-into-fastness-nordic-apis-summit-2016/

Example apis
- https://github.com/OAI/OpenAPI-Specification/blob/master/examples/v2.0/yaml/petstore.yaml
- https://github.com/zalando/shop-api-documentation/wiki/Api-introduction

`loopback swagger` generator
- https://strongloop.com/strongblog/enterprise-api-swagger-2-0-loopback/
- https://loopback.io/doc/en/lb3/Swagger-generator.html

Angular 2: swagger generators
- https://github.com/signalfx/swagger-angular-client
- https://github.com/signalfx/swagger-client-generator
- https://github.com/NSwag/NSwag

Typescript
- https://www.typescriptlang.org/

Angular cli
- https://cli.angular.io/
- https://www.sitepoint.com/ultimate-angular-cli-reference/

Angular style guide
- https://angular.io/styleguide

PWA
- https://developers.google.com/web/progressive-web-apps/
- https://developers.google.com/web/progressive-web-apps/checklist
- https://developers.google.com/web/tools/lighthouse/
- https://github.com/angular/mobile-toolkit/

Calling rest services
- https://scotch.io/tutorials/angular-2-http-requests-with-observables
- https://auth0.com/blog/angular-2-series-part-3-using-http/

Persistence
- https://loopback.io/doc/en/lb2/Database-connectors.html
- https://www.mongodb.com/
- https://cloudant.com/

Application security - keycloack
- http://www.keycloak.org/
- https://www.gitbook.com/book/keycloak/securing-client-applications-guide/details

- http://paulbakker.io/java/jwt-keycloak-angular2/

Application security - auth0
- https://auth0.com/how-it-works
- https://auth0.com/blog/angular-2-authentication/
- https://auth0.com/blog/introducing-angular2-jwt-a-library-for-angular2-authentication/
- https://auth0.com/lock

Payments
- http://ecommerce-platforms.com/ecommerce-selling-advice/choose-payment-gateway-ecommerce-store
- https://developers.braintreepayments.com/
- https://developers.braintreepayments.com/start/hello-client/javascript/v2
- https://docs.spreedly.com/

App hosting
- https://pages.github.com/

IDE
- https://code.visualstudio.com/
- https://www.jetbrains.com/webstorm/

PostgRest
- http://postgrest.com/
- http://postgrest.com/examples/start/

# Appendix 1: api dsl

Please copy paste to http://editor.swagger.io/#/ to see / edit.

```
swagger: '2.0'
info:
  version: '1.0.0'
  title: Swagger FlowMarkt (Proof of concept)
  description: A simple API that contains small subset of FlowMarkt functionality
  contact:
    name: FlowMarkt team
    email: jukka.nikki@jukkanikki.com
    url: http://www.jukkanikki.com
  license:
    name: MIT
    url: http://opensource.org/licenses/MIT
host: api.flowmarkt.io
basePath: /api
schemes:
```

```
  - http
consumes:
  - application/json
produces:
  - application/json
paths:
  /items:
    get:
      description: Returns all items from the system that the user has access to
      operationId: findItems
      produces:
        - application/json
        - application/xml
        - text/xml
        - text/html
      parameters:
        - name: tags
          in: query
          description: tags to filter by
          required: false
          type: array
          items:
            type: string
          collectionFormat: csv
        - name: limit
          in: query
          description: maximum number of results to return
          required: false
          type: integer
          format: int32
      responses:
        '200':
          description: item response
          schema:
            type: array
            items:
              $ref: '#/definitions/item'
        default:
          description: unexpected error
          schema:
            $ref: '#/definitions/errorModel'
    post:
      description: Creates a new item in the store.  Duplicates are allowed
      operationId: addItem
      produces:
        - application/json
      parameters:
        - name: item
          in: body
```

```
      description: Item to add
      required: true
      schema:
        $ref: '#/definitions/newItem'
  responses:
    '200':
      description: item response
      schema:
        $ref: '#/definitions/item'
    default:
      description: unexpected error
      schema:
        $ref: '#/definitions/errorModel'
/items/{id}:
  get:
    description: Returns an Item  based on a single ID
    operationId: findItemById
    produces:
      - application/json
      - application/xml
      - text/xml
      - text/html
    parameters:
      - name: id
        in: path
        description: ID of item to fetch
        required: true
        type: integer
        format: int64
    responses:
      '200':
        description: item response
        schema:
          $ref: '#/definitions/item'
      default:
        description: unexpected error
        schema:
          $ref: '#/definitions/errorModel'
  delete:
    description: deletes a single item based on the ID supplied
    operationId: deleteItem
    parameters:
      - name: id
        in: path
        description: ID of item to delete
        required: true
        type: integer
        format: int64
    responses:
```

```
        '204':
          description: item deleted
        default:
          description: unexpected error
          schema:
            $ref: '#/definitions/errorModel'
definitions:
  item:
    type: object
    required:
      - id
      - name
    properties:
      id:
        type: integer
        format: int64
      name:
        type: string
      tag:
        type: string
  newItem:
    type: object
    required:
      - name
    properties:
      id:
        type: integer
        format: int64
      name:
        type: string
      tag:
        type: string
  errorModel:
    type: object
    required:
      - code
      - message
    properties:
      code:
        type: integer
        format: int32
      message:
        type: string
```