Coupon Matching as Progressive Web App (PWA)

Term paper for the course TM70303 Grundlagen Web Engineering.

Link to GitHub repository (https://github.com/Mystorius/coupon_matching):

- https://github.com/Mystorius/coupon_matching

Link to PWA hosted on google firebase (https://university-coupon-matching.firebaseapp.com/):

- https://university-coupon-matching.firebaseapp.com/

Link to drag-and-drop concept (http://jsfiddle.net/60wv8xqp/73/)

- http://jsfiddle.net/60wv8xqp/73/

Concept and Idea

The Idea was to build an coupon matching web app for the Mannheim CAS coupons given to students.

Each student gets three coupons for each day visiting lectures. One for coffee, one for a beverage and one for pastry of his choice.
Coupons can only be redeemed by a Coffee-Shop nearby Agata café and roastery (https://www.agata-kaffee.de/).
Each coupon is only valid for one day. Critically, coupons are not interchangeable by the Coffee-Shop.
You are not allowed to exchange a coffee coupon for a beverage one.
But students have different preferences. One dislikes coffee, the other prefers a pastry over a beverage.

Hence the Idea to build an web app where students can upload their existing coupons and submit their preferences.
In the backend, an algorithm will match offer to desired preference and automatically exchange the coupons.

Implementation

The project is divided into three pain parts:

- the Progressive Web App
- the front-end
- the back-end

The PWA parts includes all details why I chose to use and pwa and how I implemented it. Following with a description about the chosen front-end and lastly a short description about my back-end.

Progressive Web App (PWA)

Progressive Web Apps are user experiences that have the reach of the web, and are:

- **Reliable** - Load instantly and never show the 404, even in uncertain network conditions.
- **Fast** - Respond quickly to user interactions with silky smooth animations and no janky scrolling.
- **Engaging** - Feel like a natural app on the device, with an immersive user experience.

Reliable

When launched from the user's home screen, service workers enable a Progressive Web App to load instantly, regardless of the network state. This is particular useful with "not so reliable" DHBW W-Lan. One can add their coupon and submit their preferences regardless of their network state and once connection to the internet (back end) is available, the service worker will exchange the data without any user-interaction necessary. Also, redeeming of already exchanged coupons, will be available offline.

A service worker, written in JavaScript, is like a client-side proxy and enables to serve the desired cache regarding the current network state. For example with pre-caching key resources like the app-shell, default placeholder values and images / icons, you can eliminate the dependence on the network and ensuring an instant and reliable, app-like feeling.

Fast

According to Google and DoubleClick AdExchange statistics (https://developer.akamai.com/blog/2016/09/14/mobile-load-time-user-abandonment),
"3% of mobile users abandon sites that take longer than 3 seconds to load". And once loaded, users expect them to be fast, no janky scrolling or witch slow-to-respond interfaces.

Engaging

Progressive Web Apps are installable (with an "add to home screen pop up) and live on the user's home screen (looking just like an app), without the need for an app store. Which reduces the barrier of installing greatly, which is favorable because for more than 50% of users the avery app download is close to zero according to [comScore (https://techcrunch.com/2017/08/25/majority-of-u-s-consumers-still-download-zero-apps-per-month-says-comscore/?guccounter=1)](https://techcrunch.com/2017/08/25/majority-of-u-s-consumers-still-download-zero-apps-per-month-says-comscore/?guccounter=1)
So an PWA offers an immersive full screen experience with help from a web app manifest file and can even re-engage users with web push notifications. Like a notification about a new coupon match or their daily available matches.

Front-end

For the front-end I mostly used the [materializecss (https://materializecss.com/about.html)](https://materializecss.com/about.html) framework which follows google's material design language. Material Design is a design language that combines the classic principles of successful design along with innovation and technology([Google (https://design.google/)](https://design.google/)). For user interaction and displaying variable data (eg. coupon count etc.) I used the progressive framework [Vue.js (https://vuejs.org/v2/guide/)](https://vuejs.org/v2/guide/).

Your Coupons

This view displays the automatic matched coupons plus their own coupons, that are available to use. The view is divided into coupons that are viable to use *today* and *next Days*. This charakteristik is depended on the expire date of the coupons. The coupon count is stored in the database and gets updated with page refresh.

Limitations

A click on one of the three icons should open a generated QR-Code from the coupon ID (which can be scanned from the Coffee Shop) and remove the coupon from the database. This feature is not implemented yet.

Settings

This view displays the users offers and settings to choose wishes. The offers coupon coupon count depends on how may coupons a user created / uploaded. A new coupon can be created by clicking the floating *plus* button on the lower right. A user can then choose the coupon type and the expire date of the coupon and either upload it onto the database with *save* or cancel the operation with *discard*.

The wishes tab displays the users preferences for the matching algorithm. The default value is null so if the user firs uploads coupons the won't all get *matched away* before choosing a wish. The *submit wishes* button pushes the chosen options to the database.

Limitations

A click on one of the three icons should open a list with all created coupons for later editing or removal.
This feature is not implemented yet.

Back-end / Hosting

The back-end is provided by google firebase and includes a cloud storage called *firestore* and the function to perform the matching. The back-end architecture would ideally be the following:

- Each user has its own unique ID
- Created coupons and wishes are attached to each unique ID
- Each created coupons the thrown into a large pool (table) called *offers* and wishes are thrown into a *demand* pool
- A matching algorithm following the [stable marriage problem (https://arxiv.org/ftp/arxiv/papers/1608/1608.07575.pdf)](https://arxiv.org/ftp/arxiv/papers/1608/1608.07575.pdf) and the [hospitals / residents problem (https://pdfs.semanticscholar.org/77d9/f84082674888ca90ca662847983381b23338.pdf?_ga=2.154836731.2131887508.1535974286-2007767658.15359742860)](https://pdfs.semanticscholar.org/77d9/f84082674888ca90ca662847983381b23338.pdf?_ga=2.154836731.2131887508.1535974286-2007767658.15359742860) will match both pools

Things to do

- User authentication mechanism
- Improvement on the service worker caching:
  - serve custom 404 page
  - serve offline banner
  - serve last saved state while 404
  - ...
- Matching algorithm
- removing the all listed *limitations*
- so much more...

Screenshots

Link to PWA screenshots:

- Mobile
    - Overview (https://snag.gy/wi16Pt.jpg) - https://snag.gy/wi16Pt.jpg
    - Wish selection (https://snag.gy/6zyfJe.jpg) - https://snag.gy/6zyfJe.jpg
    - Creating coupon (https://snag.gy/0VYRS7.jpg) - https://snag.gy/0VYRS7.jpg
    - Calendar view (https://snag.gy/nUDzM2.jpg) - https://snag.gy/nUDzM2.jpg
- Desktop
    - Overview (https://snag.gy/vl74G2.jpg) - https://snag.gy/vl74G2.jpg
    - Wish selection (https://snag.gy/vl74G2.jpg) - https://snag.gy/vl74G2.jpg
    - Creating coupon (https://snag.gy/P45Tra.jpg) - https://snag.gy/P45Tra.jpg
    - Calendar view (https://snag.gy/iNxte0.jpg) - https://snag.gy/iNxte0.jpg

What's included

```
│   database.rules.json
│   Documentation.md
│   firebase.json
│   MIT-LICENSE.md
│   README.md
│
└───public
    │   index.html
    │   manifest.json
    │   service-worker.js
    │
    ├───fonts
    │   └───roboto
    │           Roboto-Bold.woff
    │           Roboto-Bold.woff2
    │           Roboto-Light.woff
    │           Roboto-Light.woff2
    │           Roboto-Medium.woff
    │           Roboto-Medium.woff2
    │           Roboto-Regular.woff
    │           Roboto-Regular.woff2
    │           Roboto-Thin.woff
    │           Roboto-Thin.woff2
    │
    ├───images
    │   │   ic_refresh_white_24px.svg
    │   │
    │   └───icons
    │           Icon-144.png
    │           Icon-192.png
    │           Icon-36.png
    │           Icon-48.png
    │           Icon-512.png
    │           Icon-72.png
    │           Icon-96.png
    │
    ├───scripts
    │       app.js
    │       firebase-app.js
    │       firebase-firestore.js
    │       jquery-3.3.1.js
    │       jquery-3.3.1.min.js
    │       materialize.js
    │       materialize.min.js
    │       vue-firestore.js
    │       vue.js
    │       vue.min.js
    │
    └───styles
            materialize.css
            materialize.min.css
            style.css
```

Documentation

You can find the code documentation at Documentation.md (Documentation.md)

Drag and Drop

Drag&Drop erlaubt es dem Nutzer auf ein Element zu klicken, dann die Maustaste gedrückt zu halten, um das Element an einen anderen Ort zu ziehen und die Maustaste dort los zu lassen, um es dort abzulegen.
Eine durchsichtige Darstellung des Elements, das gezogen wird, folgt dem Mauszeiger während der Durchführung.

Damit ein Element per Drag&Drop bewegt werden kann, muss die Eigenschaft 'draggable' auf true gesetzt werden.

Während des Drag&Drop Vorgangs werden unterschiedliche Events ausgelöst. Für ein Basisbeispiel sind drei events notwendig:

- ondragstart - Wird beim Start des Zieh-Vorgangs ausgelöst und definiert die Daten welche gezogen werden. (dataTransfer.setData)
- ondragover - Wird ausgelöst wenn ein ziehbares Element über ein gültiges Feld gezogen wird in dem es abgelegt werden kann. Über dataTransfer.dropEffect wird das visuelle Feedback, über die zugelassene Aktion, für den Nutzer definiert.
- ondrop - Wird ausgelöst wenn ein ziehbares Element in ein gültiges Feld abgelegt wird. Über dataTransfer.getData() werden die abgelegten Daten abgerufen und können verarbeitet werden.

Mittels ev.preventDefault() in den events ondragover und ondrop wird das Standardverhalten des Browsers unterdrückt. So dass z.B. Links beim Ablegen nicht geöffnet werden.

Incorporated

- JQuery (https://jquery.com/) A library for supporting quick and easy javascipt
- materialize.js (https://materializecss.com/) materialize.js and materialize.css for styling
- vue.js (https://vuejs.org/) The Progressive JavaScript Framework
- vue-firestore.js (https://github.com/gdg-tangier/vue-firestore) Vue.js binding for firebase cloud firestore.
- firebase (https://firebase.google.com/) Firebase is Google's mobile platform that helps you quickly develop high-quality apps and grow your business.

Hosting

- Hosted on google firebase

Key Resources

- Your First Progressive Web App - Google Developers (https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/)
- Progressive Web App (https://ryanwhocodes.github.io/template-progressive-web-app/)

License

Author

- Benjamin Pohl