Cupcake

Datamatiker 2. semester F24 - Hold B

Orn-Iliya Ananta Phak Petersen: cph-op91@cphbusiness.dk (Github: *Liya-ap*)
Ömer Ümit Øcalan: cph-oo203@cphbusiness.dk (Github: *qmer05*)
Rebecca Mary Buron Sørensen: cph-rs@cphbusiness.dk (Github: *rebeccaburon*)

Link til website intro:
https://cphbusiness.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=dc121877-5c35-4a44-90f2-b14e012581df

Indholdsfortegnelse

Indledning

I dette projekt har vi samarbejdet med virksomheden Olsker Cupcakes. Vi har udviklet et system, der kan modtage ordrer og hertil administrere dem for virksomhedens kunder. For at forstå projektets domæne vil vi begynde med en kort beskrivelse af virksomheden og redegøre for det teknologiske valg i opgaven. Herefter dykker vi ned i virksomhedens krav, som danner rammerne for de User stories, vi vælger at arbejde ud fra. For at imødekomme de opstillede krav og for at blive klogere på virksomheden, har vi lavet 4 diagrammer herunder Aktivitets-, Domæne-, ER- og Navigationsdiagram, som vi redegør for. Vi vil dernæst adressere nogle af de hjernevridende løsninger og særlige forhold som er blevet benyttet i programmet. Til sidst i rapporten vil vi skildre status på vores implementering og projektforløbet.

Baggrund

Projektet er baseret på en aftale med virksomheden Olsker Cupcakes, som er et økologisk iværksættereventyr fra Bornholm. De har ramt den helt rigtige opskrift og har specialiseret sig i at producere og sælge cupcakes til deres kunder. De har derfor brug for et effektivt system til at håndtere deres ordre-mængde. Hertil skal der laves en administrator platform, der giver en oversigt over de ordrer, som kunderne sender ind. Dette projekt vil så vidt muligt imødekomme disse krav og hjælpe Olsker Cupcakes med at fortsætte deres succesfulde forretning med en effektiv digital løsning.

Teknologivalg

Projektet er udviklet ved hjælp af en række teknologier, der skal omfavne de kriterier forretningen har givet. Udviklingen er foregået i IntelliJ IDEA version 2023.3.5.

Java version 11
Vi anvender Java version 11, til at udvikle vores backend-logik.

PostgreSQL database - (PgAdmin version 8.3??)
Vi anvender PostgreSQL til at opbevare og administrere data, som er relateret til ordrer, kunder og administratorer.

Javalin
Ved brugen af Javalin framework kan vi opbygge og håndtere de HTTP-anmodninger og svar, der er påkrævet.

Thymeleaf template engine
Vi bruger Thymeleaf template engine, til at generere HTML-indhold på serveren. Det giver os mulighed for at implementere data fra Java-koden direkte ind i HTML-skabelonerne.

JDBC

Vi bruger JDBC til at oprette forbindelse til og interagere med vores database fra Java-applikationen, så vi kan udføre database operationer.

Disse teknologier udgør grundlaget for projektet og danner rammerne for vores løsning. Vi vil i det næste afsnit reflektere over firmaets overvejelser med systemet, og de værdier fra systemet der kan føres til virksomheden.

Krav

Der blev etableret user-stories efter første kundemøde. User-stories har til hensigt at give en kort beskrivelse af, hvad kunden ønsker af funktionelle krav til produktet. Efter nøje og detaljeret samtale med kunden, har vi kunnet kortlægge specifikke user-story scenarier, som er godkendt af kunden.

Vi har dermed udviklet produktet på baggrund af det beskrevne, samt tolket på kundens krav. Dermed er produktet nødvendigvis ikke fyldestgørende for kunden, idet det endelige produkt skal testes og bedømmes af kunden, hvortil det kan kræve korrigering, tilføjelser og rettelser til produktet. Vi har formået indenfor tidsgrænsen at implementere **US-1** til **US-6**, som var de nødvendige krav, samt **US-7** og **US-9**, som skulle implementeres, hvis der var flere ressourcer til det.

Funktionelle krav

US-1: Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, så jeg senere kan køre forbi butikken i Olsker og hente min ordre.

US-2: Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en ordre.

US-3: Som administrator kan jeg indsætte beløb på en kundes konto direkte i Postgres, så en kunde kan betale for sine ordrer.

US-4: Som kunde kan jeg se mine valgte ordrelinjer i en indkøbskurv, så jeg kan se den samlede pris.

US-5: Som kunde eller administrator kan jeg logge på systemet med email og kodeord. Når jeg er logget på, skal jeg kunne se min email på hver side (evt. i topmenuen, som vist på mockup'en).

US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.

US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, så jeg kan følge op på ordrer og holde styr på mine kunder.

US-8: Som kunde kan jeg fjerne en ordrelinje fra min indkøbskurv, så jeg kan justere min ordre.

US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde ugyldige ordrer. F.eks. hvis kunden aldrig har betalt.

Ikke-funktionelle krav

Olsker Cupcakes har udtrykt over for os, at de er et dybdeøkologisk cupcake producent fra Bornholm og er på iværksættereventyr. De har hyret professionelle fagfolk fra København, som har analyseret bageriets behov og kommet frem til de ikke-funktionelle krav, der er til produktet. Der er udformet et halvfærdigt mockup til forsiden. Firmaet har forhåbninger om at systemet vil øge deres omsætning og være med til at eskalere firmaet.

Vi har på baggrund af behovet udarbejdet vores eget mockup med inspiration fra fagfolkenes mockup. Dette har vi besluttet at gøre, idet vi mener, at vores egne designere er bedre klædt til

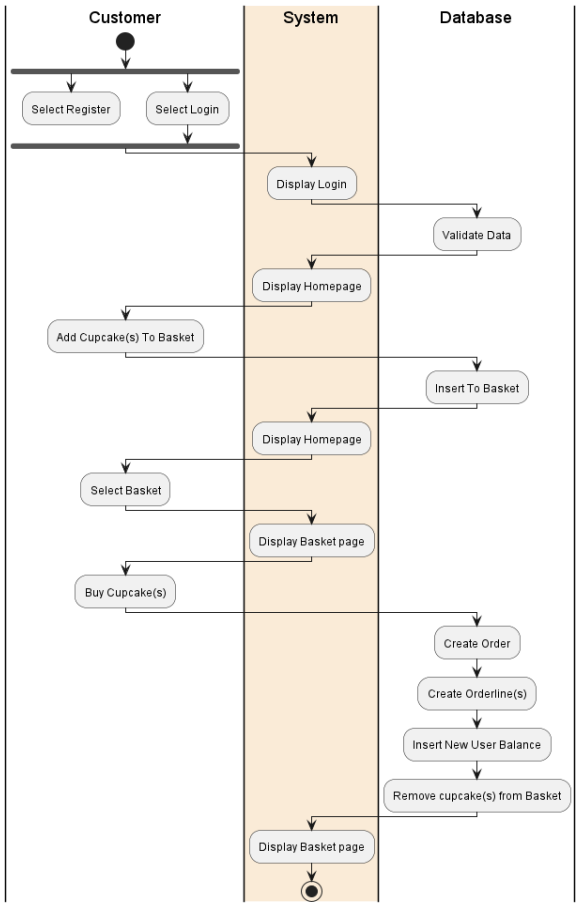
opgaven og grundet, at vi kan skabe en mere sammenhængende og gennemgående design på tværs af siderne, man skal kunne navigere igennem. Dette er naturligvis godkendt af kunden, som fik præsenteret vores Figma, hvilket faldt godt til. Se Figma Mockup i bilag.

Vores vision for systemet er at udvikle en intuitiv platform, hvorpå brugere kan bestille cupcakes efter ønskede behov. Siden skal være brugervenlig og nem at navigere igennem uden alt for mange distraherende elementer. Det værdi systemet skal tilføre Olsker Cupcakes er bedre kundetilfredshed via en flydende UX på baggrund af det valgte UI. Dette vil medføre tilbagevendende kunder samt positive brugeranmeldelser og promovering over for andre, hvis systemet efterlader brugerne et godt indtryk.

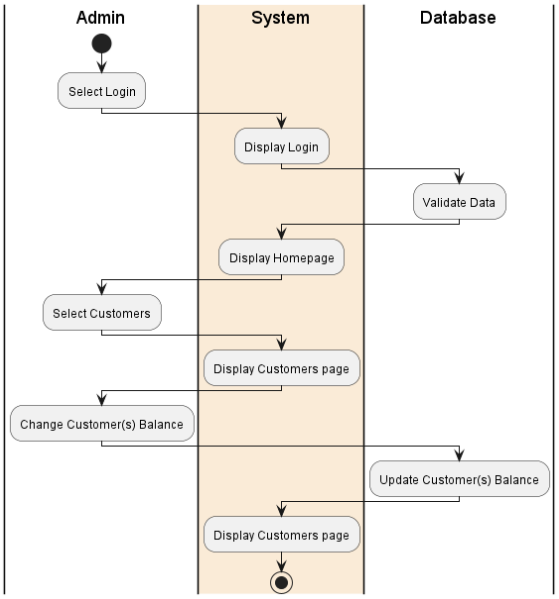
Vores UI er ikke blevet testet af brugere, idet der ikke var ressourcer nok. Det vil til enhver tid være at foretrække et brugerpanel bestående af individer med forskellige baggrunde herunder f.eks. forskellige demografiske forhold, idet vi umiddelbart definerer at Olsker Cupcakes har en bred målgruppe. For at øge tilfredsheden og brugeroplevelsen for systemet, bør brugerinddragelse derfor være en prioritet i det videre forløb. UI er dermed kun udviklet på baggrund af teamets egen kreativitet og viden fra Laws of UX (https://lawsofux.com/).

Aktivitetsdiagram

Et aktivitetsdiagram laves som regel i starten af et projektforløbet for at beskrive det overordnet workflow i programmet samt tjene som inspiration til domænemodellen, men dette har vi dog overset. Vi har dermed valgt at lave nogle aktivitetsdiagrammer efter færdiggørelsen af vores projekt for at vise det overordnede flow af vores program ud fra to udvalgte begivenheder, da vi mener at disse begivenheder indeholder de vigtigste funktionaliteter i vores program.



Figur 1 Aktivitetsdiagram over et enkelt køb af cupcakes lavet af en almindelig kunde.



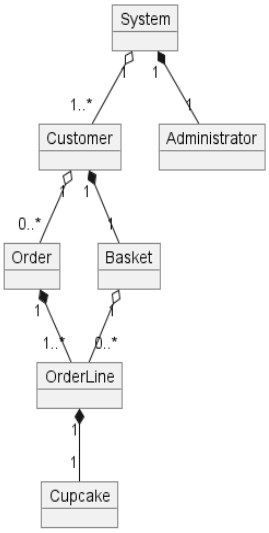
Figur 2 Aktivitetsdiagram over indsættelsen af et beløb til en kundes konto

Figur 1 viser det overordnet flow for når en almindelig kunde ønsker at bestille en eller flere cupcakes i en ordre. I dette aktivitetsdiagram er der antaget at kunden allerede har registreret sig og kan logge ind samt at de har nok penge på deres konto.

Figur 2 viser det overordnet flow for når administratoren ønsker at indsætte eller fratrække et beløb fra en kundes konto. I dette aktivitetsdiagram er der antaget at administratoren ikke har fratrukket for meget således at kunden går i minus eller har indsat for meget således at beløbet er større end den valgte datatype i vores database.

Domæne model og ER diagram

Domænemodel



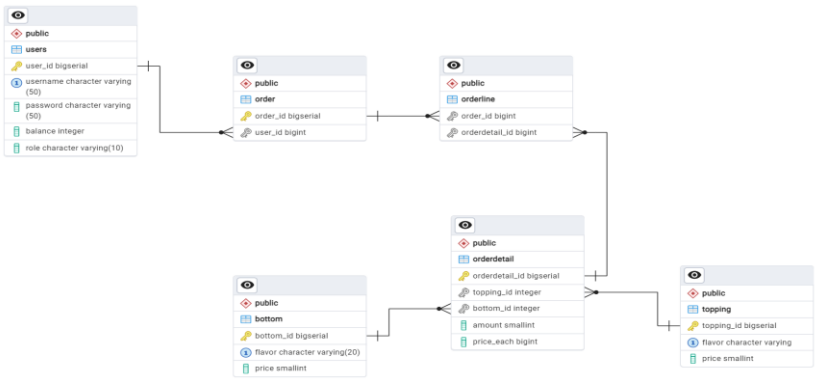
Figur 3
Domænemodel over Cupcake programmet

Figur 3 viser det domænemodel, vi udarbejdede helt i begyndelsen af projektforløbet under planlægningsprocessen. Vores overordnet program ser vi som en enhed *"System"*, hvorved en kunde og administrator har adgang til at benytte det. Der er kun en administrator til programmet, men der kan være mange kunder.

I vores program skal den enkelte kunde have en kurv hvor de har mulighed for at tilføje alle de cupcakes de ønsker. Kunden har derefter mulighed for at lave en ordre ved at købe de cupcakes der er i kurven. Når kunden har lavet en ordre, kan de vælge at lave flere ordrer derefter, hvis ønsket.

Vi kunne muligvis have undladt at have *"OrderLine"*-enheden med i domænemodellen, da den egentlig bare repræsenterer *"Cupcake"*-enheden.

ER diagram

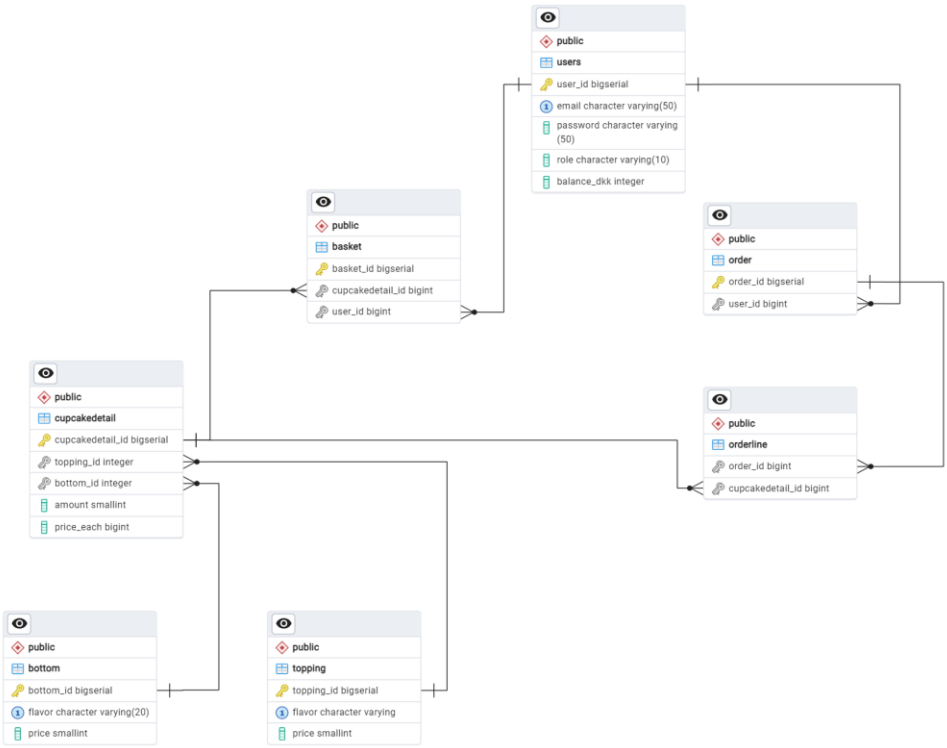


Figur 4
ER-diagram over vores database før oprettelsen af databasen.

Figur 4 viser vores oprindelige ER-diagram over vores database, som vi udarbejdede helt i begyndelsen af projektforløbet under planlægningsprocessen.

Vi valgte at opdele *bottom* og *topping* ud i deres egne tabeller og bruge dem som *foreign keys* i *orderdetail* tabellen for at følge 3NF.

Der er også skabt en mange-mange relation mellem *order* og *orderdetail* forbundet gennem *orderline* tabellen der benytter hvert tabellers *primary key* som *foreign keys*. Dette skyldes at forskellige ordrer kan indeholde mange cupcakes og de samme typer cupcakes kan være i mange forskellige ordrer.



Figur 5
ER-diagram over vores endelige database

Figur 5 viser vores endelige ER-diagram over vores database, hvilket er blevet auto-genereret ud fra databasen. Vi har ændret navn på vores tidligere *orderdetail* tabel til *cupcakedetail* for at gøre det mere specifikt, da det alligevel kun er cupcakes det repræsenterer. Under *users* tabellen havde vi også ændret *username* til *email* grundet at vi overså at kunderne kun skulle oprette en bruger med en e-mail og ikke brugernavn.

Den største ændring, vi dog har lavet til databasen, er tilføjelsen af *basket* tabellen. Vi valgte at tilføje denne tabel, da en kunde skulle have mulighed for at tilføje cupcakes til kurven uden at købe dem, og næste gang kunden loggede ind, så ville de tilføjede cupcakes stadig ligge klar i kurven. Kunden skulle dermed ikke starte forfra med deres bestilling.

Enhver ny kunde der oprettes i systemet får i *users* tabellen automatisk tildelt rolen *user* under *role* samt får tilføjet *500* til deres konto under *balance_dkk*.

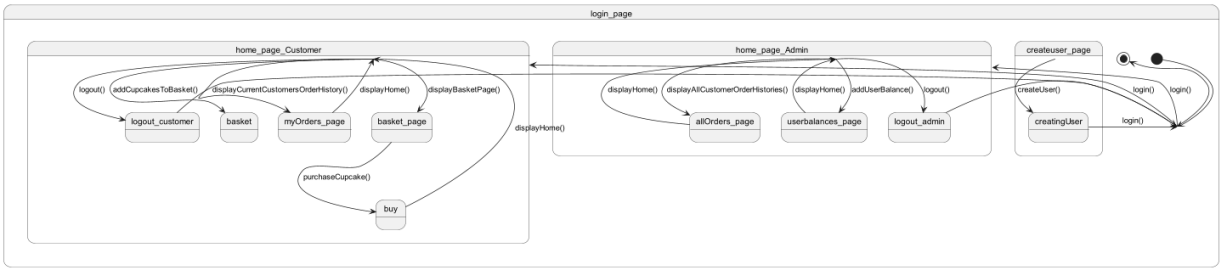
Navigationsdiagram

Fælles Navigations bar

Uafhængig af, om man er logget ind som kunde eller som administrator, har begge mulighed for at trykke på *Home* knappen, og vende tilbage til den forside, der gør sig gældende. Derudover kan begge bruger logge af, fra hvilken som helst side.

Adgangsbegrænsning

Baseret på bruger-rollerne er nogle af siderne begrænset. For eksempel er det kun administrator der kan tilføje en valutabalance til en kunden og det er kun en kunde som får mulighed for at vælge Cupcakes og hertil købe dem.



Beskrivelse af navigations mønstre

Applikationen består af fire sider: login side, opret bruger side, forside for administratorer og forside for kunderne. Brugeren starter på *login_page* og kan her enten vælge at logge ind med en eksisterende bruger, herunder admin, eller oprette sig som ny bruger. Hvis brugeren vælger at oprette sig som ny kunde, bliver man ført videre til createuser_page hvor man her opretter sig. Hvis oprettelsen er succesfuld, kommer man tilbage til *login_page*, og kan nu logge ind med sin nye bruger.

Når brugeren er logget ind som kunde, åbner *home_page_* Customer op, og brugeren kan nu vælge imellem at tilføje *Cupcakes* til deres kurv (*basket*), se egen ordrehistorik (*myOrders_page*), eller se *Cupcakes* i egen kurv (*basket_page*). Vælger kunden at få fremvist sin kurv, kan kunden købe de *Cupcakes* der er deri.

Vælger brugeren at logge sig ind som administrator, kan brugeren vælge at få vist en fuld ordrehistorik på alle kunder (*allOrder_page*) eller tilføje en balance til kunden (*userbalance_page*).

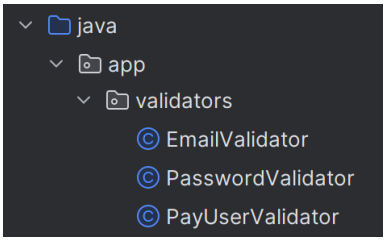
Hvis brugeren vælger at logge ud (*logout_admin* eller *logout_customer*) kommer brugeren tilbage til *login_page*.

Dette navigation diagram giver en indsigt i applikationens flow og skal hertil hjælpe med at skabe et overblik over, hvilke funktioner der gør sig gældende for bestemte brugere. Vi vil i næste afsnit kigge nærmere på de særlige forhold, der er benyttet i programmet.

Særlige forhold

Dette afsnit bruges til at beskrive særlige forhold der benyttes i programmet.

Validering af brugerinput herunder validering af password



En package "*validators*" er oprettet under app for at samle alle brugerinput validering. *EmailValidator* klassen har til hensigt at validere brugerens email, *PaswordValidator* brugerens password og *PayUserValidator* skal validere at admin kan hæve/indsætte penge ind på en brugers konto.

```
public class EmailValidator {  2 usages  Ømer

    private static final String EMAIL_REGEX =  1 usage
            "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@" +
                    "[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";

    private static final Pattern pattern = Pattern.compile(EMAIL_REGEX);

    public static boolean isValidEmail(String email) {  1 usage  Ømer
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }
}
```

I klassen *EmailValidator* findes en konstant String REGEX, som har til formål at indeholde et mønster, som bruges til tekstbehandling og mønstergenkendelse. Her definerer *EMAIL_REGEX* strukturen af en gyldig e-mailadresse.

Klassen Pattern har en compile metode der tager imod en REGEX og gemmer et returneret Pattern objekt i pattern, som skal bruges i *isValidEmail* metoden.

IsValidEmail tager imod en *String email* som parameter, og inde i metoden anvendes en *matcher object*, som tjekker hele input String og hvis hele mønsteret matcher returneres *true* eller *false*. For at se nedbrydning af EMAIL_REGEX opbygning - se bilag.

```
private static void createUser(Context ctx, ConnectionPool connectionPool) {
    String email = ctx.formParam( key: "email");
    String password1 = ctx.formParam( key: "password1");
    String password2 = ctx.formParam( key: "password2");

    if (EmailValidator.isValidEmail(email)) {
        if (PasswordValidator.isValidPassword(password1)) {
            if (password1.equals(password2)) {
                try {
                    UserMapper.createuser(email, password1, connectionPool);
```

isValidEmail metoden anvendes i *UserController* klassens *createUser* metode. Først gemmes de indtastede brugerdata fra html siden i String variabler gennem Context klassens *formParam* metode.

Derefter tjekkes om den indtastede email opfylder vores opstillede krav til en e-mail og returnerer true eller false. Hvis der returneres true fortsætter valideringsprocessen og tjekker om de indtastede adgangskoder er fyldestgørende, hvortil der gennem *UserMapper* klassen gemmes en ny bruger i databasen.

Session attributter

Den eneste information vi har valgt at gemme i session er et *user*-objekt når en kunde eller admin logger ind i systemet. Vi benytter denne session attribut til bl.a. at vise brugerens e-mail i bunden af vores side mens de er logget på. Derudover bliver den brugt til at tage fat i den rigtige *user_id* for kunden der er logget ind for at gemme kundens ordre eller kurv rigtigt i vores database ud fra id'et.

Status på implementation

Manglende implementering af user stories

Vi satte et mål for at implementere og færdiggøre **US-1** til **US-6**, hvilket det lykkedes os. Vi implementerede dog ved et tilfælde også **US-9** uden at tænke nærmere over det, da det virkede som en naturlig del af **US-6**. Dermed manglede vi at implementere **US-7** og **US-8**, hvilket udelukkende skyldes manglende tid.

Manglende CRUD metoder

Vi mangler umiddelbart ingen CRUD metoder ift. de user stories vi nåede at implementere, men da vi ikke fik tid til **US-8** er kunden tvunget til at købe de cupcakes der er lagt i kurven. Hvis kunden ikke har nok penge på kontoen, har vi valgt at fjerne alle cupcakes fra kurven så kunden har mulighed for at forsøge igen, for at arbejde uden om det problem der vil opstå med at kunden vil sidde fast med en masse cupcakes i kurven indtil administratoren indsætter et beløb på kundens konto.

Manglende responsiv design

Vores hjemmeside fungerer ikke tilfredsstillende på mindre skærme som bl.a. mobiltelefoner. Designet er ikke super responsivt men vi har forsøgt at gøre brug af *media query* i vores css, dog grundet manglende tid og prioritering af funktionalitet frem for design, så blev det ikke færdiggjort.

Manglende unit tests

Vi har i denne omgang ikke prioriteret at lave unit tests til vores program og derfor mangler vi dem også. Dermed er det også svært at konkludere med fuld overbevisning at alt fungerer efter hensigten. Vi kan kun bedømme om vores program fungerer ud fra det visuelle, når vi kører det.

Proces

Projektforløbet var planlagt til at følge projektmodellen Kanban, hvor vi lavede et Kanban-board for at visualisere arbejdsprocessen og sikre en løbende og effektiv opgavehåndtering. Målet var at skabe overblik og styring af arbejdsopgaverne, samt at tilpasse os eventuelle ændringer i opgaverne. I praksis fulgte vi denne plan ved at oprette et Kanban-board i GitHub, hvor hver især kunne se opgaverne og deres status (Se bilag). Vi afholdte både fysiske-og onlinemøder for at diskutere fremskridt, udfordringer og justere opgavefordeling efter behov. Det gik godt med at holde styr på opgaverne og sikre en løbende opdatering af Kanban-boardet. Dog havde vi en smule ujævn fordeling af opgaver, da erfaringsniveauet spænder bredt i gruppen. Den ujævne fordeling af opgaver resulterede i en asymmetrisk fremdrift, hvor nogle gruppemedlemmer færdiggjorde deres opgaver hurtigere end andre og påtog sig herefter nogle nye. Dette skabte en ubalance i arbejdsbyrden. For at imødekomme denne problematik forsøgte vi at uddelegere opgaver, med en passende og udfordrende arbejdsbyrde. Dette etablerede fremadrettet en mere ensartet og harmonisk arbejdsproces.

Under projektforløbet har vi fået en bedre forståelse og hertil færdighed indenfor Kanban board. Ydermere er vi blevet bedre til organisering og styring af vores arbejdsopgaver og vi vil i fremtiden holde fokus på en kontinuerlig evaluering af arbejdsprocessen og fordeling af opgaver.

Bilag

Nedbrydning af EMAIL_REGEX udtryk:

- `^` angiver starten af linjen.
- `[_A-Za-z0-9-\\+]+` matcher én eller flere tegn, som kan være understregningstegn, bogstaver (både små og store), tal, bindestreger eller plus-tegn.
- `(\\.[_A-Za-z0-9-]+)*` tillader nul eller flere forekomster af et punktum efterfulgt af én eller flere af de samme tegn som den første del (før "@"-symbolet).
- `@` matcher det bogstavelige "@"-tegn.
- `[A-Za-z0-9-]+` matcher én eller flere tegn, som kan være bogstaver (både små og store), tal eller bindestreger.
- `(\\.[A-Za-z0-9]+)*` tillader nul eller flere forekomster af et punktum efterfulgt af én eller flere bogstaver eller tal.
- `(\\.[A-Za-z]{2,})` matcher et punktum efterfulgt af mindst to bogstaver (f.eks. ".com", ".org").
- `$` angiver slutningen af linjen.

Figma Mockup:

Kanban board for Cupcake project :