



# PRODUKTRAPPORT

Daniel Skriver Hansen

H6PD100120



## Indholdsfortegnelse

<b>Titelblad</b>	<b>1</b>
<b>Forord</b>	<b>2</b>
<b>Læsevejledning</b>	<b>2</b>
<b>Problemformulering</b>	<b>3</b>
<b>Kravspecifikation</b>	<b>4</b>
Database	4
REST-API	4
User-Interface	5
<b>Testspecifikation</b>	<b>6</b>
Database	6
REST-API	6
User-Interface	7
<b>Brugervejledning</b>	<b>8</b>
Installation	8
Firebase oprettelse	9
App installation	9
Bruger anvendelse	10
<b>Teknisk produktdokumentation</b>	<b>11</b>
Database Design & funktionalitet	11
Software & kodeeksempler	12
Versionsstyring	13
<b>Sikkerhed</b>	<b>13</b>
<b>Source-kode</b>	<b>13</b>
<b>Konklusion</b>	<b>13</b>
<b>Bilag</b>	<b>14</b>

## Titelblad

# TECHCOLLEGE

Tech College Aalborg,  
Struervej 70,  
9220 Aalborg Ø

Sider: 20

Anslag: 14.934

**Elev:**

Daniel Skriver Hansen

**Firma:**

Bang & Olufsen A/S

**Projekt:**

Bestillings Organiserings  
System

**Uddannelse:**

Datatekniker m. Speciale i  
Programmering

**Projektperiode:**

26/10/2020 – 26/11/2020

**Afleveringsdato:**

18/11/2020

**Fremlæggesdato:**

27/11/2020

**Vejledere:**

Lars Thise Pedersen &  
Lærke Brandhøj Kristensen

## Forord

Denne produktrapport, som er en ud af to skrevne rapporter, udarbejdet i forbindelse med svendeprøveforløbet 2020, for datatekniker m. speciale i programmering.

Rapporten er skrevet og udarbejdet af Daniel Skriver Hansen.

## Læsevejledning

Denne produktrapport beskriver de tekniske aspekter af dette projekt. Den vil beskrive produktet i helhed, opsætning, workflow samt vise nogle kodeeksempler.

For at få mest muligt ud af denne rapport, anbefales det at læse procesrapport først, da den giver indblik i, hvilke tanker der er gjort før og under udvikling af projektet. Denne rapport vil efterfølgende uddybe den endelige opsætning.

## Problemformulering

Med udgangspunkt i den beskrevne case og problemformulering fra procesrapport, vil den tekniske vinkel på problemformuleringen være som følgende.

Vi skal formå at lave et system, som strukturerer måden, hvorpå at kunden kan bestille sin mad, tjeneren ikke glemmer / skriver kundens ordrer forkert og køkkenet overholder den tidshorisont, som de har sat.

Til dette skal der udvikles en app, til både kunder og køkkenpersonale, som snakker sammen via et REST-API.

REST-API'et skal have adgang til en database, hvori man kan gemme, den mængde af ordrer som er bestilt over en vis periode. Denne database skal være tilgængelig fra skyen.

## Kravspecifikation

Dette afsnit vil yderligere konkretisere de krav, som er til projektet. Jeg vil synliggøre hvilke krav som blev opfyldt, hvilke som mangler og hvad kravene reelt er.

Hvert krav er opbygget med følgende:

- **Krav Id:** Denne søjle viser det ID, som er blevet givet til kravet. Dette er gjort for bedre at kunne referere kravet igennem rapporten.
- **Krav:** Dette er en kort beskrivelse af kravet.

Derudover har jeg valgt at dele kravene op i tre kategorier, for at simplificere overblikket over kravene.

De tre kategorier er:

- **Database**
  - Kravene til, hvordan databasen skal være.
- **REST-API**
  - Kravene til, hvordan REST-API'et skal agere.
- **User-Interface**
  - Kravene til, hvordan User-interfacet skal formeres og bruges.

### Database

Krav Id	Krav
DB1	Databasen skal kunne modtage og afgive forudbestemte data.
DB2	Databasen skal kunne tilgås fra alle steder, hvor der er en internetforbindelse til stede.

### REST-API

Krav Id	Krav
RA1	Rest API'et skal kunne håndtere at modtage en ordre og sende informationerne de respektive steder hen.
RA2	REST API'et skal kunne håndtere at snakke om skrive til Databasen.
RA3	REST API'et skal kunne køres i en Docker Container fra en raspberry Pi

## User-Interface

Krav Id	Krav
<b>UI1</b>	UI'et skal være intuitivt og være let at bruge.
<b>UI2</b>	UI'et skal kunne bruges til at aflægge en ordre
<b>UI3</b>	UI'et skal kunne give information omkring, hvad der er forskellige ting på menuen.
<b>UI4</b>	Brugeren af UI'et skal kunne se, hvad da, indtil videre, har valgt fra menuen, via en scrollable ude i siden med de valgte menuer.
<b>UI5</b>	Brugeren skal kunne bekræfte at det er de valgte menuer, som brugeren har valgt.

## Testspecifikation

Under denne sektion vil der blive beskrevet test specifikationer ift. projektet. Test specifikationerne vil hovedsageligt bestå af Unit-Test, men det vil også kunne indføres i CI værktøjer som f.eks. CircleCi eller Jenkins.

Disse krav er opbygget på følgende måde:

- **Test Krav Id:** Denne søjle viser det ID, som er blevet givet til kravet. Dette er gjort for bedre at kunne referere kravet igennem rapporten.
- **Test Krav:** Dette er en kort beskrivelse af kravet.
- **Kravet:** Kravet fra kravspecifikationen som testes.
- **%:** Afkrydset, hvis dette ikke er formået at blive løst.
- **X:** Afkrydset, hvis kravet er løst.

### Database

Test Krav Id	Test Krav	Kravet	%	X
TDB1	Databasen skal kunne modtage et bestemt sæt data, og sende samme data tilbage igen.	DB1		X
TDB2	Databasen skal kunne ændre på data, og sende det ændrede data tilbage.	DB1	X	
TDB3	Databasen skal kunne kontaktes via REST-API'et	DB2		X

### REST-API

Test Krav Id	Test Krav	Kravet	%	X
TRA1	Rest-API'et skal kunne tilgås på "/" routen.			X
TRA2	Rest-API'et skal kunne modtage data via "/putOrder", med data via en POST.	RA1		X
TRA3	Rest-API'et skal kunne sende data baseret på, om der er en ordrer eller ej.	RA2		X
TRA4	Rest-API'et skal kunne håndtere at modtage data i JSON-format.			X
TRA5	Rest-API'et skal kunne snakke med databasen.			X



## User-Interface

Test Krav Id	Test Krav	Kravet	%	X
<b>TUI1</b>	Via UI'et skal man kunne tilføje menuer til ens ordrer.	UI2		X
<b>TUI2</b>	Via UI'et skal man kunne vælge mellem om det skal være køkken eller kunde, i FTS.			X
<b>TUI3</b>	Via UI'et skal man kunne sende en ordrer op til REST-API'et.	UI4, UI5		X
<b>TUI4</b>	Man skal kunne tilføje og slette menuer dynamisk.			X

## Brugervejledning

Denne brugervejledning vil give indblik i, hvordan produktet installeres og anvendes. Installationen kan forgå på flere måde, dog er anvendelsen meget lineær.

### Installation

Installationen af dette produkt kan forgå på flere måde. Jeg vil beskrive dem løbende som vi kommer igennem dette segment.

Før du går i gang med installationen, vil der være nogle programmer som skal installeres på dit system. Følgende programmer er nødvendige:

- **Python3** - <https://www.python.org/downloads/>
- **Git** - <https://git-scm.com/downloads>
- **Flutter** - [https://flutter.dev/docs/get-started/install?gclid=Cj0KCQiAhs79BRD0ARIsAC6XpaVyP3JLZw1N\\_LQLeJa5\\_1NtFupnfCtkl00NzyA4HhN\\_R4DB5riEr4QaAgO4EALw\\_wcB&gclsrc=aw.ds](https://flutter.dev/docs/get-started/install?gclid=Cj0KCQiAhs79BRD0ARIsAC6XpaVyP3JLZw1N_LQLeJa5_1NtFupnfCtkl00NzyA4HhN_R4DB5riEr4QaAgO4EALw_wcB&gclsrc=aw.ds)
- **Tekst editor, af eget valg**

For at komme i gang med installationen skal koden hentes fra github, denne gøres med denne kommando.

**git clone <https://github.com/skriverthefirst/BOS.git>**

Med dette link henter du hele source koden til systemet, det vil komme i en struktur som ser sådanne ud:

- **doc** – Dokumentationen til dette projekt
- **src** – Alt source koden til projektet.
  - **Api** – Dette er restserveren, der bruges til kommunikationen mellem de forskellige devices som har appen installeret. Dette er også servicen som har adgang til databasen.
    - **Auth** – Her vil din key.json ligge, der bliver brugt som nøgle til din database.
  - **App** – Dette er appen som du skal bruge på dine devices.

For at få systemer op og køre, vil du inde i API folderen finde en .py fil, der hedder **api.py**, dette er filen som starter serveren og som alt kommunikation kører igennem. Når denne fil bliver startet, vil forbindelsen til databasen også blive oprettet.

Serveren startes derefter ved at skrive "flask run", mens man befinder sig inde i denne folder. Dette er forudsat at du hentet pakken Flask, som gøres ved at skrive "Python3 -m pip install flask"

Første gang denne server startes, vil der ikke være en database tilgængelig, dette oprettes inde på <https://firebase.google.com>. Der vil komme en forklarende guide på, hvordan oprettelsen af databasen skal ske, senere i rapporten.

## Firestore oprettelse

For at få en database til at fungere sammen med dette produkt, vil det kræve at der bliver oprettet en database inde hos Firestore. Dette er gratis, til et vist omfang.

Hvis man følger linket længere oppe, til firebase, så vil man komme ind på start siden, hvor man kan oprette sin konto, derefter trykker man på + ikonet og sætter sit projekt op, dette skal man bare følge step-for-step.

Når dette er færdig, vil man blive givet en authentication token, som man skal smide ind i "key.json" filen, for at kunne få forbindelsen til databasen.

Det vil også kræve at du får ind i "database.py" og ændrer projectId, til ID'et på din database i denne sætning:

```
firebase_admin.initialize_app(cred, {"projectId": "bestillingsorganiseringssystem"})
```

## App installation

Da projektet er udviklet via en simulator på computeren, er det også det jeg tager udgangspunkt i, i denne installations rapport.

For at få gang i appen kræver dette at du har Flutter installeret, og en simulator kørende på din PC. Dette kan enten være en Android eller IOS simulator (udviklingen er sket på en simuleret iPad 8 gen), da flutter compile filer til begge platforme.

Når du står inde i mappen "App" vil du skulle gå en folder længere ind (bestillingsOrganisering), da det er her Flutters "Pubspec.yaml" fil ligger. Dette er en af de filer som har med i sin bygge proces for at kunne få de nødvendige pakker med ind i APK (Android) og IPA (IOS) filerne.

For så at starte appen på det simulerede device, skriver man, fra roden af "bestillingsOrganiserings" mappen, "flutter run" – derefter vil flutter bygge projektet og uploade det til simulatoren.

## Bruger anvendelse

I denne sektion vil der blive beskrevet, hvordan anvendelsen af dette produkt vil være. Dette vil være meget lineært og kun være beskrevet for delen som kunden skal bruge, da køkkenet ikke har et funktionelt design.

Appen er programmeret med funktionalitet fremfor design, i tankerne.

Se Bilag 1.

Den første skærm man kommer ind på, når man åbner appen første gang, vil være FTS (First time setup) siden.

Denne side giver dig 2 knapper at vælge i mellem, Consumer og Kitchen. For at sætte den op til at kunder kan vælge menuer, trykkes der på "Consumer".

Se Bilag 2.

Når denne er trykket på, vil appen bevæge sig videre til "TableNumberPage", hvor det er nødvendigt at skrive sit bordnummer ind, sådan at tjenerne ved, hvilket bord der har bestilt hvad.

Se bilag 3.

Efter at man har skrevet sit bordnummer ind, kommer vi nu videre til at man kan vælge, hvad man gerne vil have at spise. Dette forgår på "MenuPage" siden. Her vil man, naturligvis, kunne se, hvad man har bestilt, ude i siden. Når man har valgt de menuer som man gerne vil have, kan man trykke afslut ordrer, som vil føre til næste side.

Se bilag 4.

Denne side skal forsikre kunden om, at man har valgt de rigtige menuer, hvis de valgte menuer er inde på siden, kan man vælge at trykke på fluebenet, hvis menuerne er forkerte, kan man vælge at trykke på krydset, så vil man komme tilbage og kunne vælge andre.

## Teknisk produktdokumentation

Denne del af rapporten vil gå mere i dybden af, hvordan programmet er lavet. Den vil beskrive database designet, give mere viden omkring, hvordan den database, som er valgt, virker. Den vil også vise nogle kodeeksempler, som vil hjælpe læseren med at forstå koden bedre.

### Database Design & funktionalitet

Under denne sektion, vil læser få information omkring database designet, både det aktuelle databasedesign og hvordan dette bruges i forhold til appen.

Se bilag 5.

Designet af denne database er meget simpelt, da firestore er en NoSQL, dokument database. Firestore fungerer således at man har en collection, inde i den collection kan man lægge dokumenter, som indeholder fields af data. Inde i disse fields ligger der Key / Value pairs, som indeholder den indsatte data.

Hvis man kigger lidt på koden, så vil man få fat i det specifikke dokument ved at refererer til det på sin database klient.

*Databasebiblioteket er skrevet i Python*

```
Def __init__(self):  
    self.db = firestore.client()  
    self.food_ref = self.db.document('orders/foods')
```

Dette vil give os en reference til det specifikke dokument, hvorpå vi kan ændre de data som dokumentet indeholder, tilføje nye data eller slette data.

## Software & kodeeksempler

Denne sektion vil give læseren, via små forklaringer omkring et stykke kode, mere information omkring koden.

### Dynamisk generering af UI-elementer

Inde i filen MenuPage.dart i src/app/bestillingsOrganisering/pages/consumer (sti taget relativ til github rod), vil man finde dette stykke kode:

```
chosenMenuItems.map<Widget>((menu)=>Container(  
    child: Column(  
        children: <Widget>[  
            Text(menu.title)  
        ],  
    ),  
),  
).toList(),
```

Dette giver en dynamisk måde, hvorpå man kan generere UI elementer ud fra en liste. Den tager en liste og mapper dem ind til widgets, i dette tilfælde er widget'en en Container. Ved brugen af dette giver det mulighed for at tage data fra det enkelte element i listen, som ses ved at "menu.title" bliver brugt i eksemplet.

### Responsivt UI-design

Flere steder i appkoden vil man kunne finde denne form for linje, i en form eller anden:

```
width: (MediaQuery.of(context).size.width) / 3,
```

I forhold til "Width" vil dette gøre at vi kan køre denne app på brede, såvel som smalle, devices og stadig have det samme layout at brugerfladen. Der er en lignende funktionalitet til "Height".

### Python Paths

Læseren vil, i python filerne, finde denne linje kode:

```
dirname = os.path.dirname(__file__)
```

Dette gøres for at være sikker på at stien til de andre python filer, som denne fil bruger, altid er den samme, uanset miljø, så længe de ligger relativ til denne fil.

## Versionsstyring

Til projektet er der benyttet Github, hvilket er en cloud baseret Git service.

Det har primært været brugt under udviklingen af dette projekt, sådan at min kode har været sikret, hvis der skulle ske fejl på mit hardware.

## Sikkerhed

I forhold til sikkerhed i dette projekt, er databasen det eneste som har en sikret måde at kommunikere på, da det skal kommunikere oppe i skyen. Dette kan begrundes med, at hele produktet kan køre på et internt netværk og at dataene der sendes, ikke indeholder sensitiv information. Sikkerheden for databasen er i form af en Authentication token som bruges til at lave forbindelsen til databasen i skyen, denne token er individuel, afhængig af database.

## Source-kode

Alt source-kode til dette projekt kan findes inde på følgende Github:

<https://github.com/skriverthefirst/BOS>

## Konklusion

Overordnet set vil jeg ikke bedømme projektet til at være så fuldent som jeg gerne ville ønske det. Der er en lang række af ekstra design features jeg gerne ville have implementeret og videre udviklet, f.eks ville jeg gerne have haft at jeg havde fået lavet min rest-server i docker.

Dette skyldes formentlig min fejlvurdering af, hvor lang tid der skulle bruges på at skrive rapporter.

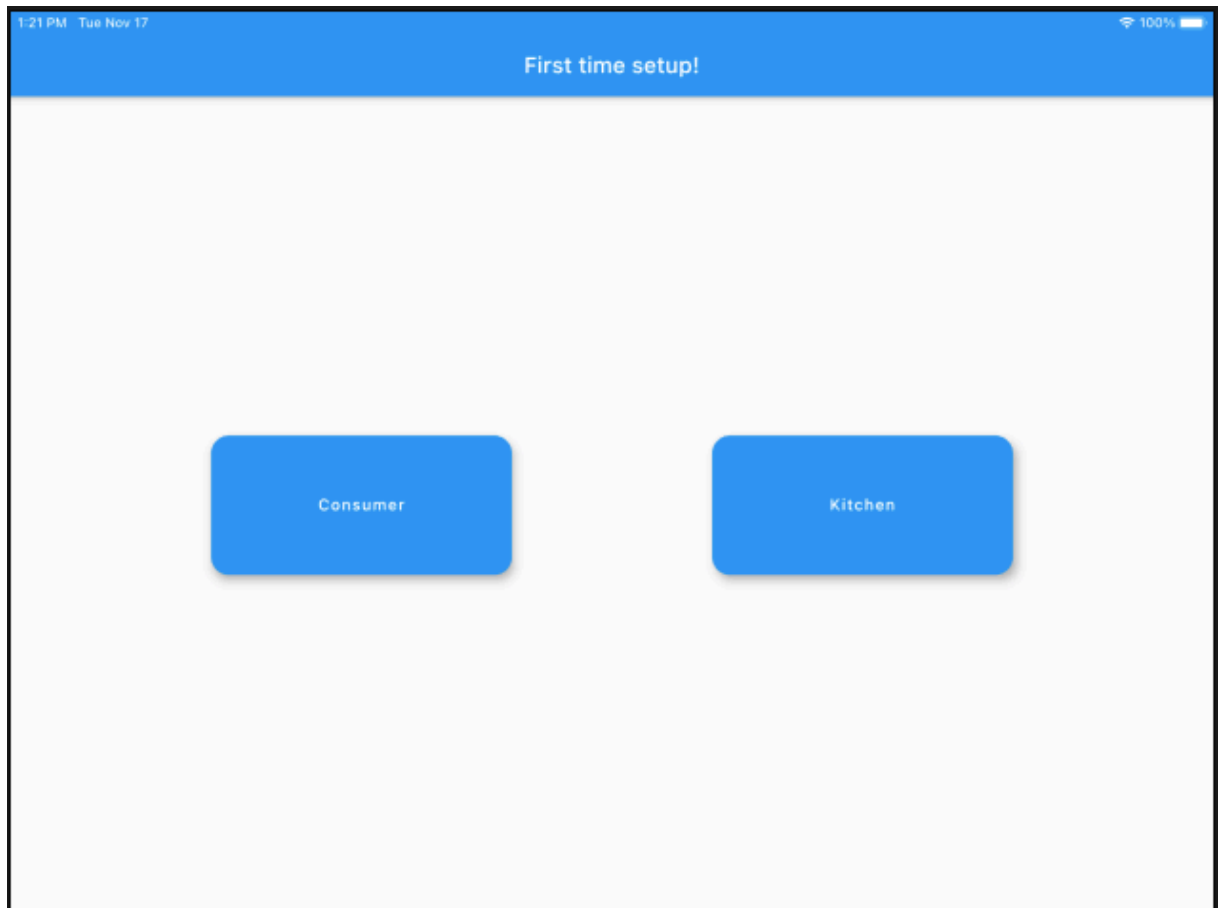
Det realiserede forløb afviger en del fra det estimerede. Dertil sagt, er det svært at vurdere, hvor lang tid udvikling af et produkt kan tage, når man bruger teknologier som man ikke er erfaren indenfor.

Dog er jeg godt tilfreds med, hvad jeg har lært af dette projekt.

Ud fra sektionen "Teknologi & begrundelse" fra procesrapporten, kan man se at flutter er et sprog i udvikling (da brugen af det er steget med 9% på et år), og dette projekt har givet mig mere blod på tanden til at ville lære og udvikle min evner inden for dette framework.

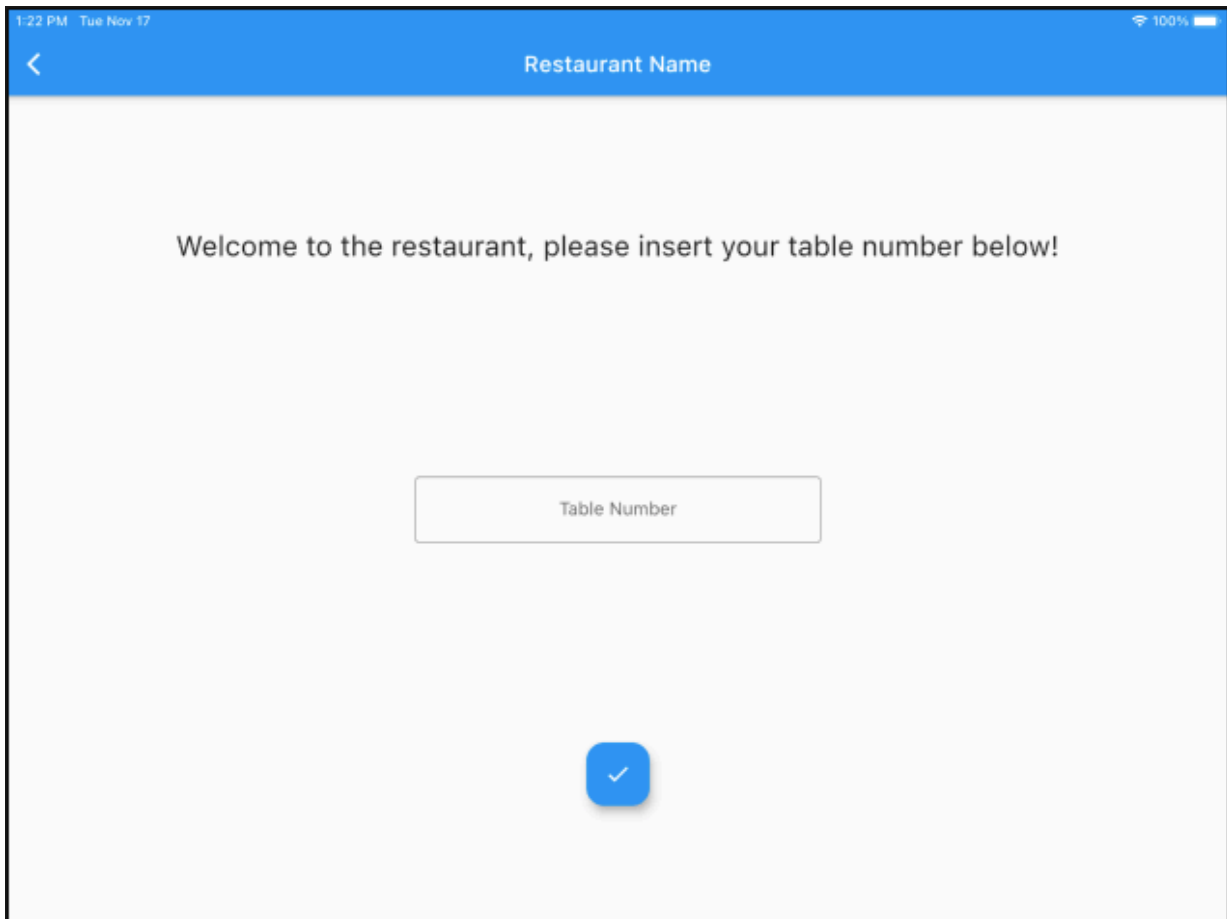
## Bilag

### Bilag 1.



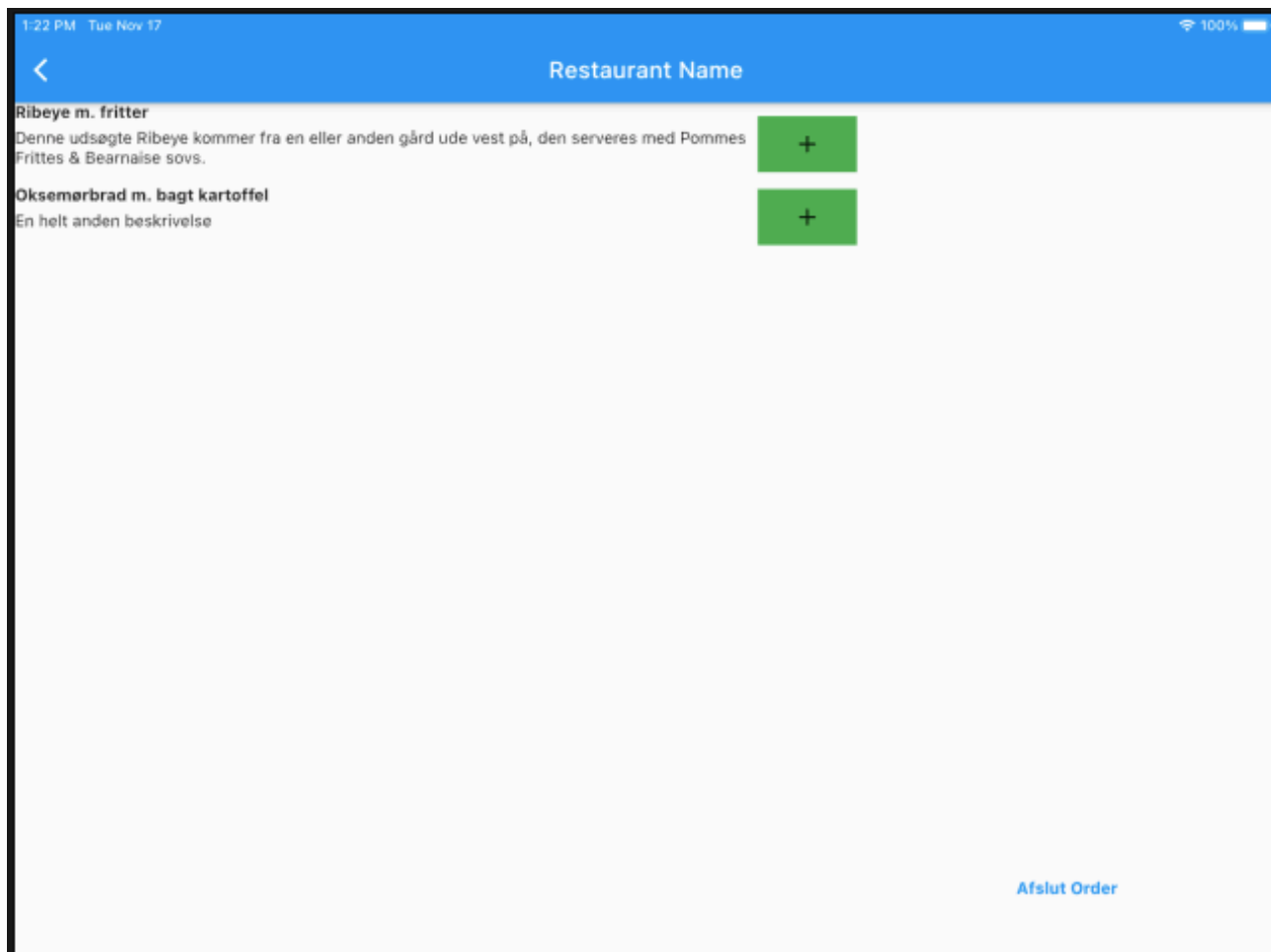


## Bilag 2.

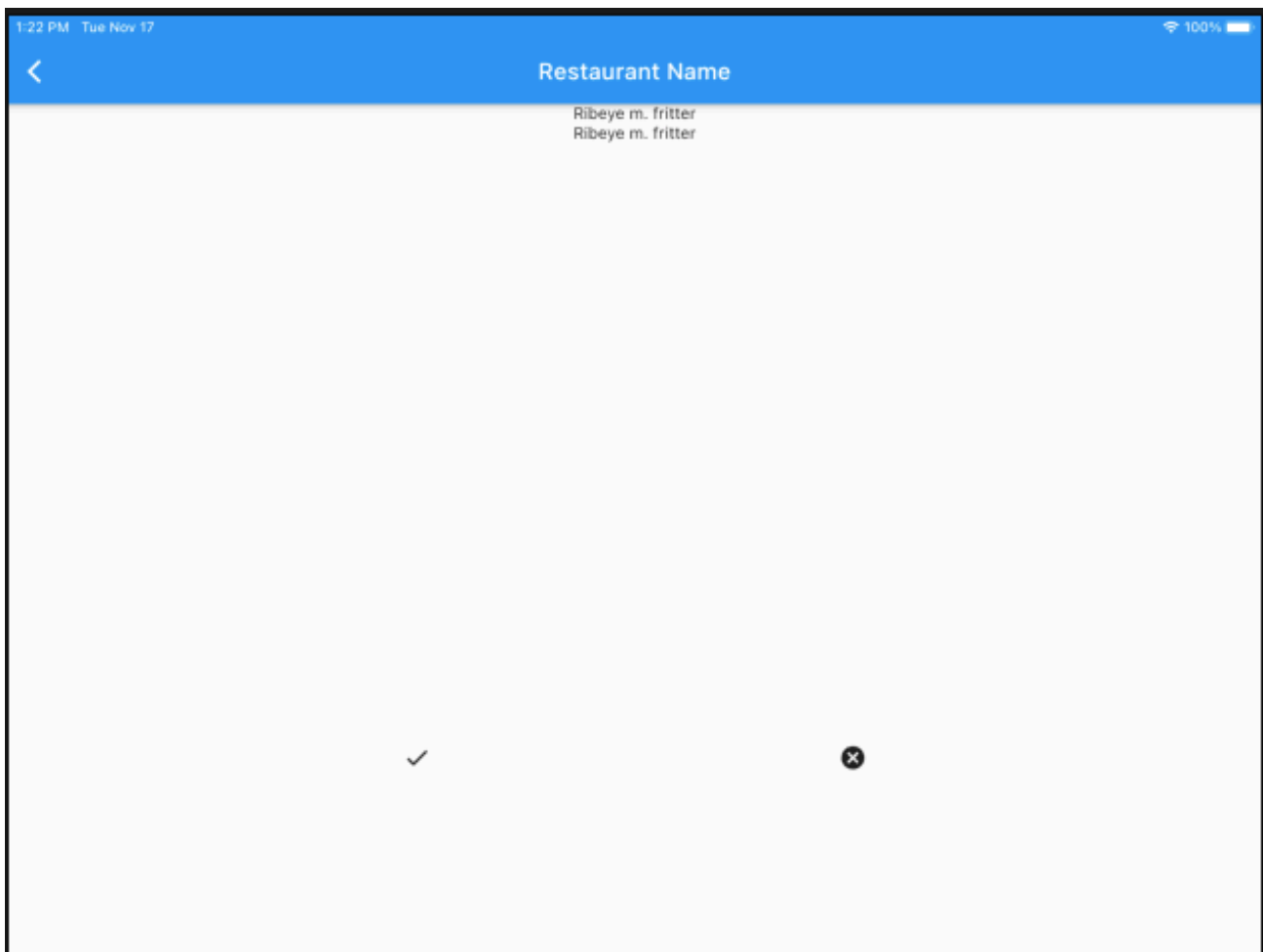


The screenshot shows a mobile application interface for a restaurant. At the top, there is a blue header bar with a back arrow on the left, the text "Restaurant Name" in the center, and a status bar on the right showing "1:22 PM Tue Nov 17" and "100%" battery. Below the header, the main content area is light gray and contains the text "Welcome to the restaurant, please insert your table number below!". In the center of this area is a white rectangular input field with the placeholder text "Table Number". Below the input field is a blue circular button with a white checkmark inside, indicating a confirmation or next step.

## Bilag 3.



## Bilag 4.



## Bilag 5.

