Syddansk Universitet, Odense
IMADA

Dato 27.11.2022

DM571
Software Engineering
Projekt

Projektet er udarbejdet af
Kasper Veje Jakobsen - kasja20@student.sdu.dk
Emil Czepluch - emcze20@student.sdu.dk
Nikolaj Dall - nidal20@student.sdu.dk
Tan Nhat Le - tale20@student.sdu.dk

Contents

1 Del 1

1.1 Antagelser

Vi antager at Inutilia Emptio (IE) er en handelsvirksomhed, der opererer på B2B-markedet. IE sælger køkkenudstyr til butikker såsom Imerco, Kop & Kande og lignende. IE's kunder har en indkøbsafdeling, der i stedet for at håndtere indkøb over telefonen, fremadrettet skal bruge vores softwareløsning til at bestille over nettet. Vi antager at leverandør A og B ikke sælger de samme varer.

1.2 Stakeholders

- **Hr. Manager**
  Beskrivelse: Hr. Manager er den primære kontaktperson hos IE, som vi skal levere et softwaresystem til. Han er den vigtigste stakeholder i dette projekt.
  Salience attributer: Power: Høj , Legitimacy: Høj, Urgency: Høj
  Klassificering: Core

- **Nuværende kunder**
  Beskrivelse: Tilbagevendende kunder, der handler hos IE. Disse kunder bliver brugere af systemet, og er derfor vigtige at håndtere.
  Salience attributer: Power: Mellem, Legitimacy: Mellem, Urgency: Lav
  Klassificering: Dormant / Discretionary / Dominent

- **Kundeservice-medarbejder**
  Beskrivelse: Disse medarbejdere skal hjælpe kunderne med eventuelle problemer under kundernes indkøb gennem vores system.
  Salience attributer: Power: Lav , Legitimacy: Mellem, Urgency: Mellem
  Klassificering: Discretionary / Demanding / Dependant

- **Leverandører**
  Beskrivelse: Leverandører af potter og pander til IE. Systemet skal bygges "rundt om" IE's leverandørers API'er. Det vigtigste for os hos leverandørerne er adgang til API'en og eventuelt hjælp til at bruge den.
  Salience attributer: Power: Lav, Legitimacy: Høj, Urgency: Lav
  Klassificering: Discretionary

- **Eksisterende softwareløsninger**
  Beskrivelse:
  Salience attributer: Power: , Legitimacy: , Urgency:
  Klassificering:

- **Konkurrenter**
  Beskrivelse:
  Salience attributer: Power: , Legitimacy: , Urgency:
  Klassificering:

1.3 Users

- **End Users**
  - **Kunder:** Et system for firmaer til at købe potter og pander til deres lager på en nem måde. Det nye online system vil være en vej til at holde på nuværende kunder ved at give dem en større fleksibilitet i forhold til indkøb af inventar. Det vil også være med til at trække nye kunder ind som gerne vil have en online løsning i forhold til at skulle ringe ind for at bestille inventar ved deres nuværende leverandør.
- **Super Users**, *Rangeret efter privilegier i systemet*
  - **System administrators:** Står for vedligeholdelse af systemet.
  - **Manager:** Til brug af diverse medarbejdere der har brug for et indblik i hvad der ligger internt i firmaet af inventar og kan tilføje nyt inventar.
  - **Kundeservice:** Flere privilegier end kunderne, men mindre end system administratorerne. De har nok redskaber til at kunne give kunderne en god service, uden at have fuld adgang til systemet, så de ikke udgør en sikkerhedstrussel.
- **System Users**
  - **Lager:** Bruges internt for at sikre at ordrerne bliver pakket korrekt og til de rigtige kunder.
  - **Betalingssystem:** Vi er afhængige af et betalingssystem som kan stå for at processere de betalinger vores kunder laver.
  - **Leverandør API:** Automatisk bestilling af nye varer til lageret, når inventaret kommer under et bestemt antal.
  - **Tid:** Bruges til at automatisere dele af vores system. Dette er smart så vi bruger mindre tid på ting der let kan automatiseres såsom dagsrapporter, effektivitetsammenligninger og salgsrapporter.

1.4 Backlog

Vi har valgt at strukturere vores Product Backlog ved at hver Item har en overskrift, en User Story, en estimation ved brug af T-Shirt størrelser og en prioritering fra Lav-Høj. Product Backloggen findes i Appendix 3.1.

Diskussion: Vi har valgt at give hvert item en overskrift. Overskriften består af et overemne og en titel. Dette gøres for at skabe overblik og klarhed omkring rammen af opgaven. Dertil kobles en beskrivelse for item'et, som er skrevet ved brug af user stories, dette for at fremhæve værdien af item'et for brugeren. Vi har estimeret vores items ved brug af T-Shirts størrelser. Vi har brugt magic estimation i vores team for at give det enkelte entreis i vores backlog den passende estimation. Vi har i samarbejde prioriteret itemsne for at opnå en prioritering alle i teamet er tilfredse med.

2 Del 2

2.1 Flows i systemet

Vi har i en fed teambuilding-aktivitet lavet to helt kanon UML Sequence Diagrams. For den første sekvens (Appendix 3.2 - 'Put item in a basket'), antager vi at slutbrugeren tilgår systemet gennem vores webbrowser-modul. I den anden sekvens (Appendix 3.3 - 'Check stock levels')) antager vi at en Super User tilgår systemet direkte gennem vores system-klient. Ydermere antages det at begge suppliers (A og B) har den samme API som kan kaldes for at tjekke lagerbeholdningen på varenene.

Derudover er der også lavet et UML Klasse-diagram, der i bagklogskabens lys måske skulle være lavet på forkant - så der var lidt mere mening med galskaben. Den kan findes i Appendix 3.4

Vi har brugt en klasse til at repræsentere en ordre, men i virkeligheden repræsenterer den klasse egentlig bare en dictionary. Det giver ikke rigtig mening på nuværende tidspunkt - men giver mulighed for videre implementation af nye features.

2.2 API Specifikation

API-specifikationen er vedlagt i Appendix 3.5. Vi bruger APIkeys til at verificere adgangen til API'en. Disse sendes med i request-headeren - hvilket også er implementeret i næste del af projektet.

Der er lavet et endpoint (GET /inventory) til at få en liste over alle produkter med tilhørende lagerværdier. Svaret gives som et array af json-objekter, hvor hvert json-objekt repræsenterer et produkt med dertilhørerende informationer.

Der er lavet et endpoint (POST /basket/create) til at oprette en kurv. Der skal sendes et brugerid (uid) med i request-body'en for at sikre samhørighed mellem bruger og kurv i systemet. Herefter returnes en basketId, som efterfølgende bruges til at opdatere kurven

Der er lavet to endpoints (POST + DELETE /basket/basketId/itemId) til at henholdsvis tilføje og fjerne produkter fra kurven. Når et produkt tilføjes skal der i request-body'en sendes et antal med. I begge tilfælde returneres hele kurven som et array af json-objekter, hvor hvert json-objekt repræsenterer et produkt i kurven med dertilhørerende informationer.

Vi har valgt at et produkt repræsenteres gennem et json-objekt, for at sikre nem interaktion mellem forskellige systemer og programsprog. Vi har valgt at en kurv og en inventory repræsenteres som et array af objekter, for at man nemt kan loope igennem alle produkter. De forskellige paths til endpointsne i API'en er bestemt ud fra et princip om, at det skal være nemt at sammenkæde path'en og den efterfølgende handling i systemet.

API'en er på level 2 af Richardson Maturity scale, da vi først og fremmest bruger de passende HTTP verber til at adskille typen af requests med POST, DELETE og GET. Derudover har vi adskilt de forskellige requests med passende paths.

2.3 Implementation og test

Vi har implementeret et system i Python, med de påkrævede funktionaliteter. Undervejs i processen, har vi valgt at holde os til at gemme i lokal hukommelse og systemet er derfor IKKE persistent ved genstart af serveren. For fremadrettet brug rådes det på det kraftigste at man bygger systemet op omkring en database.

Der er implementeret fem klasser (Basket, Inventory, Item, Order, User) og de er pakket ind i en Client-klasse, som udover at sørge for at gemme objekterne i localstorage også har flere metoder hvor de fem individuelle klasser er implementeret. Klasserne findes i "Python/classes/*.py" og clienten tilgåes i "Python/client.py".

Selve clienten eksponeres gennem en REST API, der er bygget op ved brug af python's flask-modul.

Vi har beregnet cyclomatic complexity af alle klasserne ved brug af programmet "radon" og der findes score på dem i Appendix 3.6

Unit testene er lavet ved at gøre brug af pythons unit testing framework - unittest. De klasser der er blevet testet er *Item, User* og *Basket*. Måden unittest er blevet brugt på er ved at lave forskellige unit tests, som hver tester en del af klasserne. Unit testene er bygget op på nogenlunde samme måde, der er et set up hvor nogle test objekter bliver lavet og så går hver unit test ind og bruger unittest metoder som assertEqual, assertIn, assertTrue, etc. til at tjekke at det respons der er kommet tilbage ved at bruge de enkelte klassers metoder også stemmer overens med det der man havde regnet med at få returneret.
Efter alle disse tests er lavet kan man bruge Coverage.py til at se hvor meget kode man dækker. Coverage.py kan både lave en tekstuel repræsentering, eller man kan få genereret en html der viser det på en visuel måde (Se README for commands på dette). Meget af vores error handling ligger i vores client, hvilket gør det lidt sværere at checke dårligt input ved kun at teste på disse 3 klasser, da det ikke giver en meningsfuld fejl tilbage. Den tesktuelle coverage rapport af testene kan findes i Appendix 3.7.
Grunden til at Item klassen ligger meget lavt i code coverage er fordi at under Item klassen ligger der kode til at skulle bruge supplier A og B's API'er, men da der ikke er nogen API at bruge, ligger der placeholder kommentarer der hvor implementation skal ligge. Dette resulterer i at vi ikke kan teste den del af koden og derfor ender med et dårligere code coverage på Item klassen.
Hvis man skulle gå mere ned i testing, kunne man have lavet end-to-end testing hvor man bedre kan emulere hvordan brugerene vil bruge hjemmesiden og tjekker flowesne igennem hjemmesiden. Man kunne også have gjort integrations testing hvor man går højere op en unit testene og kombinere de enkelte moduler og tester dem som en enhed.

2.4 Arkitektur

De tre diagrammer fra C4-modellen findes i Appendix 3.8.

Layers: Softwaren er lavet i to lag. Et lag der er REST-API'en og et lag der er python-clienten. Som det er skildret i vores C4 model vil det sidste lag være et

præsentationslager, der i sidste del af denne opgave er lavet som en prototype.

Distribution Patterns: Vi bruger først og fremmest "Distributed Presentation", da tanken (hvis vi skulle lave det i virkeligheden) er at API'en og python-clienten køre på en intern server og at shopping-siden/app'en skulle distribueres ud på internettet.

Agile Principles: Vi mener at softwaren er bygget op på en måde, så det både er nemt at implementere nye metoder og funktioner samt revidere nuværende kode. Derudover har alle i teamet bidraget godt til processen, så selv hvis koden ikke er "the Agile way" er den blevet skabt "the Agile way".

Level 4: Det vil umiddelbart ikke skabe værdi for os, at lave Level 4 i C4-modellen, da vi i forvejen har lavet fine UML-diagrammer og sekvenser. Hvis ikke vi havde lavet UML klasse-diagrammer kunne det godt have skabt en god fælles forståelse for koden at lave Level 4.

2.5 User Interface

Vi har valgt at bygge en virkende prototype, som vi har hooket op med vores REST API, der er skrevet i python. Prototypen, der består af en simpel HTML/CSS/JS hjemmeside, tillader én bruger (hardcoded i js) med én apiKey (hardcoded i js), at gøre følgende:

1. Se alle produkter i inventory
2. Lægge x antal af et produkt i kurven
3. Fjerne et produkt fra kurven
4. Oprette en ordre med produkterne fra kurven

Undervejs snakker hjemmesiden sammen med python-systemet.

Vi har valgt at hjemmesiden består af to sider. En produkt-side hvor alle produkter kan ses samt en kurv-side hvor man kan se hvad man har lagt i kurven. På produktsiden fremgår alle produkterne i nogle kasser, hvor man kan vælge at tilføje et eller flere produkter i sin kurv. På kurvsiden fremgår en simpel liste med produkterne i kurven samt en samlet pris.

REST API'en håndterer http-forespørgslerne fra hjemmesiden og gemmer løbende information i localstorage. Når man starter serveren, initialiseres en simpel client med 6 produkter og 3 brugere. Igennem REST API'en kan man derudover

1. Få en liste over alle brugere (GET /users)
2. Oprette en bruger (POST /user)
3. Få en brugers indkøbskurv (GET /user/uid/basket)
4. Opdatere en brugers indkøbskurv (POST /user/uid/basket)
5. Slette et item fra en brugers indkøbskurv (DELETE /user/uid/basket)
6. Få en liste over alle ordre fra en bruger (GET /user/uid/order)

7. Oprette en ordre på en bruger (POST /user/uid/order)
8. Få alle ordre i system (GET /orders)
9. Få alle produkter i inventory (GET /inventory)

For at køre REST API'en gennem python skal man blot køre "Python/main.py"-filen. Vores dependencies er listet i "Python/requirements.txt". Vi anbefaler at køre serveren med python version >= 3.9. Serveren er sat op til at køre på "https//localhost:5000". Porten kan ændres nederst i "Python/main.py"-filen og øverst i "GUI/main.js"-filen, hvis det skaber problemer.

Hjemmesiden kan tilgås ved at åbne "GUI/products.html" i en browser. Vi anbefaler IKKE at der bruges Internet Explorer, Microsoft Edge eller lignende crappy browsers til at åbne siden.

3 Appendix

3.1 Product Backlog

- **Bruger: Oprettelse**
  Beskrivelse: Som ikke-eksisterende bruger vil jeg oprette mig som bruger, så jeg kan få adgang til systemet.
  Estimation: S
  Prioritering: Høj

- **Bruger: Browsing**
  Beskrivelse: Som bruger vil jeg se hvilke produkter IE udbyder, så jeg kan beslutte mig for hvilke jeg vil købe.
  Estimation: L
  Prioritering: Mellem

- **Bruger: Shopping**
  Beskrivelse: Som bruger vil jeg tilføje og fjerne produkter til/fra min indkøbskurv, så jeg kan købe produkterne.
  Estimation: S
  Prioritering: Høj

- **Bruger: Betaling**
  Beskrivelse: Som bruger vil jeg godkende og betale for produkterne i min indkøbskurv, så jeg kan modtage dem.
  Estimation: M
  Prioritering: Høj

- **Bruger: System Admin**
  Beskrivelse: Som systemadministrator vil jeg have fuld adgang til systemet, så jeg kan ændre ting løbende.
  Estimation: XL
  Prioritering: Høj

- **Bruger: Kundeservice**
  Beskrivelse: Som kundeservicemedarbejder vil jeg have priviligeret adgang til systemet, så jeg kan hjælpe kunderne med deres ordrer.
  Estimation: L
  Prioritering: Mellem

- **Produkter: Håndtering**
  Beskrivelse: Som en manager vil jeg tilføje, fjerne og redigere produkterne i systemet, så jeg kan holde systemet opdateret.
  Estimation: L
  Prioritering: Mellem

- **Lager: Indkøbsautomatisering**
  Beskrivelse: Som lagersystem vil jeg automatisk indkøbe nye varer når lagerbeholdningen kommer under en vis grænse, så der altid er varer på lager.
  Estimation: XL
  Prioritering: Høj
- **Lager: Lister**
  Beskrivelse: Som en manager vil jeg hente en liste over produkter samt deres lagerbeholdning, så jeg kan holde mig opdateret.
  Estimation: S
  Prioritering: Mellem
- **Lager: Notifikation**
  Beskrivelse: Som lager vil jeg have en notifikation om gennemførte ordre, så mine medarbejdere kan pakke varene og sende ordren.
  Estimation: M
  Prioritering: Lav
- **Ordre: Lister**
  Beskrivelse: Som en manager vil jeg hente en liste over ordrer samt deres status, så jeg kan holde mig opdateret.
  Estimation: S
  Prioritering: Lav
- **Ordre: "Database"**
  Beskrivelse: Som ordresystem vil jeg gemme alle ordrer samt deres status i systemet, så det ikke bliver glemt.
  Estimation: S
  Prioritering: Mellem
- **Rabatkuponer**
  Beskrivelse: Som manager vil jeg kunne oprette rabatkuponer, så kunderne kan få rabat.
  Estimation: S
  Prioritering: Lav
- **Optid**
  Beskrivelse: Som bruger vil jeg have adgang til system 99% af dagen, så jeg kan købe ind når det passer mig.
  Estimation: XL
  Prioritering: Høj
- **Sikkerhed**
  Beskrivelse: Som manager vil jeg have sensitiv virkosmhedsdata og brugerinformation beskyttet, så vi ikke bliver banket af myndighederne.
  Estimation: XL
  Prioritering: Høj

3.2 UML Sequence Diagram: Put item in basket

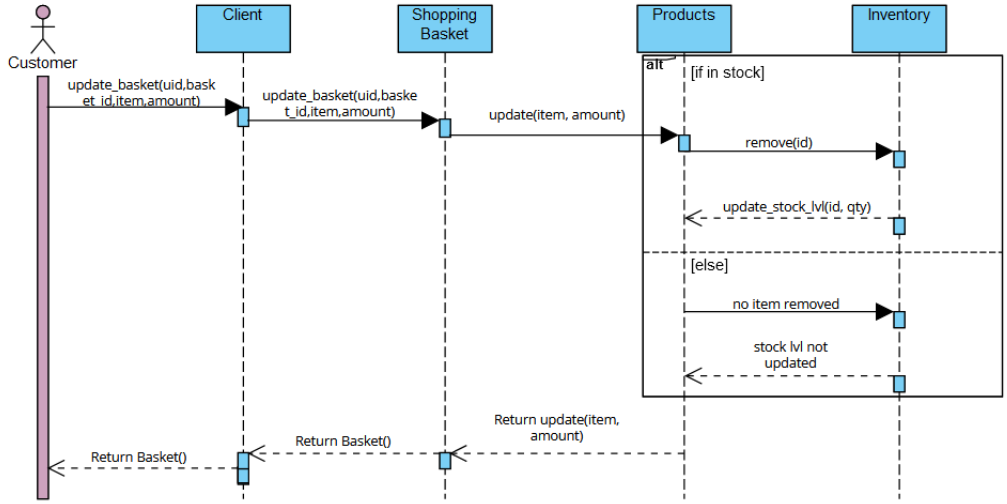

Figure 1: UML Sequence Diagram: Put item in basket

3.3 UML Sequence Diagram: Check Stock Levels

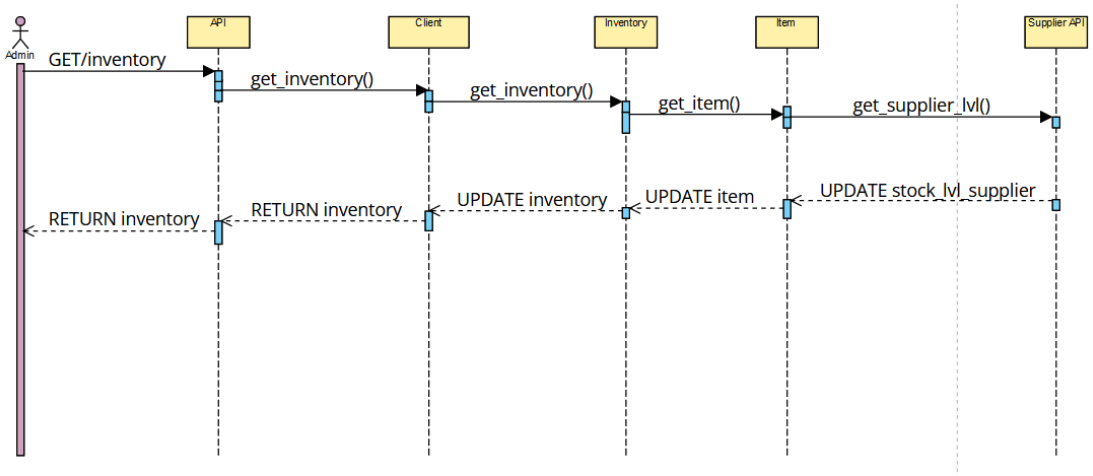

Figure 2: UML Sequence Diagram: Check Stock Levels

3.4 UML Class Diagram

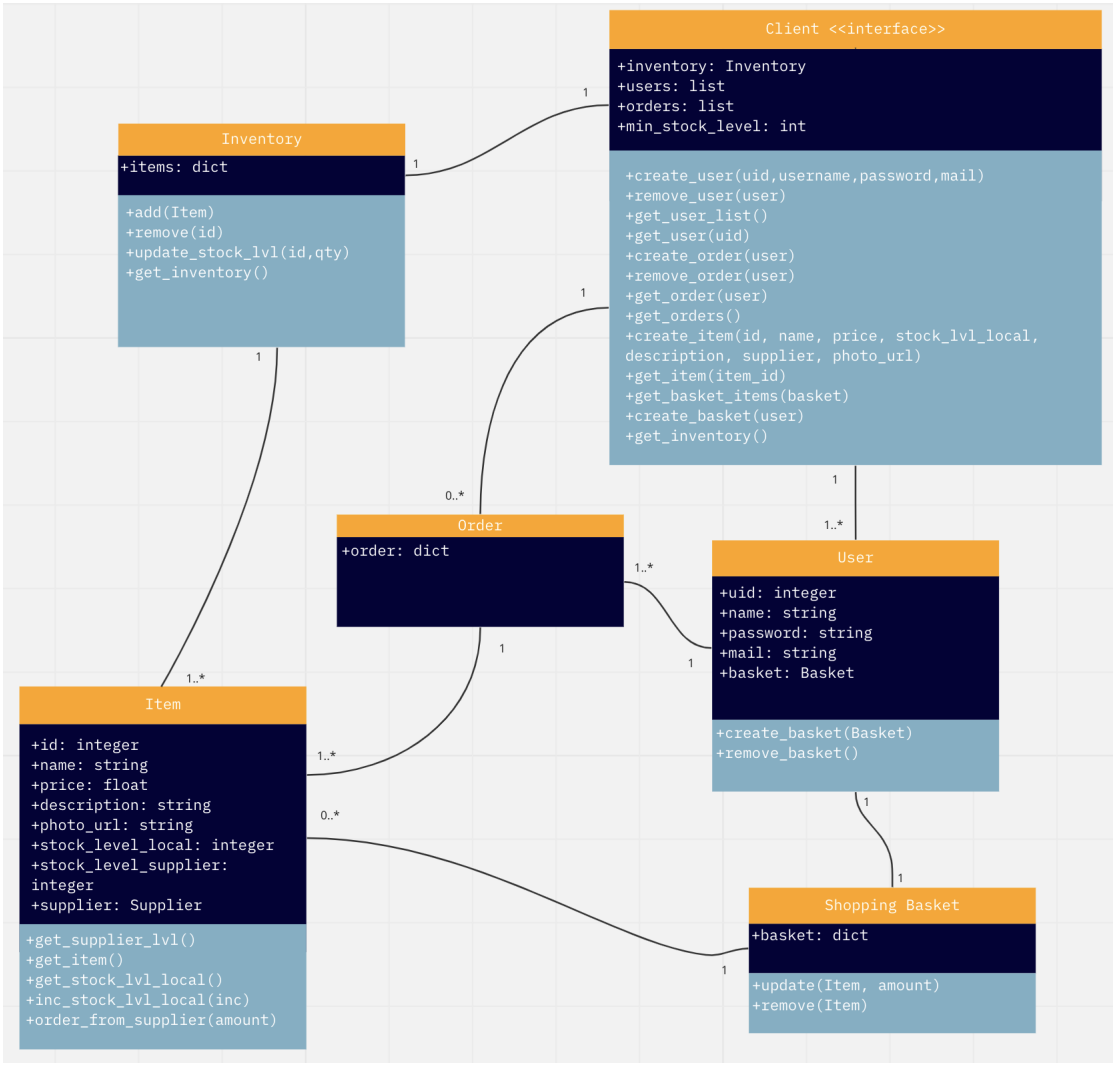

Figure 3: UML Class Diagram

3.5 OpenAPI Specifikation

Herunder følger vores OpenAPI-specfikation. For bedre readability, kan den også åbnes i afleveringsmappen under filnavnet: "OpenAPI.yaml"

```
openapi: 3.0.3
info:
  title: Shopping Basket
  description: This is an awesome app for a shopping basket
  contact:
    email: kasja20@student.sdu.dk
    name: Tan Nhat Lee, Nikolaj Dall, Emil Czepluch, Kasper Veje Jakobsen
  version: 1.0.0
tags:
  - name: Inventory
  - name: Shopping Basket

paths:
  /inventory:
    get:
      tags:
        - Inventory
      summary: Fetch inventory status
      description: Fetch a list of products with stock levels
      security:
        - apiKey: []
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/AllItems'

        '400':
          description: Bad request
        '401':
          description: Unauthorized - API Key not accepted
        '418':
          description: The server refuses to brew coffee because it is,
              permanently, a teapot

  /basket/create:
    post:
      tags:
        - Shopping Basket
      summary: Create Basket
      description: Create a new shopping basket
      requestBody:
        description: User ID for connection between user-object and
            basket-object
        required: false
        content:
          application/json:
```

```
            schema:
              $ref: '#/components/schemas/User'
    security:
      - apiKey: []
    responses:
      '200':
        description: Successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ShoppingBasket'
      '400':
        description: Bad request
      '401':
        description: Unauthorized - API Key not accepted
      '403':
        description: User not found
      '406':
        description: User not authorized
      '418':
        description: The server refuses to brew coffee because it is,
           permanently, a teapot
/basket/{basketId}/{itemId}:
  post:
    tags:
      - Shopping Basket
    summary: Add item to basket
    description: Adds an item (identified by the itemID) to the given
       basket (identified by the basket ID)
    parameters:
      - name: basketId
        in: path
        description: ID of shoppingbasket
        required: true
        schema:
          type: string
      - name: itemId
        in: path
        description: ID of item
        required: true
        schema:
          type: string
    requestBody:
      description: The quantity of the item, you want to add to your
         basket
      required: true
      content:
        application/json:
            schema:
              type: integer
              example: 5
    security:
      - apiKey: []
```

```
    responses:
      '200':
        description: Successful operation, returns the entire
            shoppingbasket
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AllItems'
      '400':
        description: Bad request
      '401':
        description: Unauthorized - API Key not accepted
      '403':
        description: Basket not found
      '418':
        description: The server refuses to brew coffee because it is,
            permanently, a teapot
      '498':
        description: Amount not in stock
  delete:
    tags:
      - Shopping Basket
    summary: Remove item from basket
    description: Removes an item from the basket, and returns the
        remaining basket
    parameters:
      - name: basketId
        in: path
        description: ID of shoppingbasket
        required: true
        schema:
          type: string
      - name: itemId
        in: path
        description: ID of item
        required: true
        schema:
          type: string
    security:
      - apiKey: []
    responses:
      '200':
        description: Successful operation, returns the entire
            shoppingbasket
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/AllItems'
      '400':
        description: Bad request
      '401':
        description: Unauthorized - API Key not accepted
      '403':
```

```
          description: Basket not found
        '418':
          description: The server refuses to brew coffee because it is,
              permanently, a teapot

components:
  securitySchemes:
    apiKey:
      type: apiKey
      name: Authorization
      in: header
  schemas:
    AllItems:
      type: array
      items:
        allOf:
          - $ref: '#/components/schemas/Item'
    Item:
      type: object
      properties:
        itemId:
          type: integer
          example: 1234
        stockLvlLocal:
          type: integer
          example: 5
        stockLvlSupplier:
          type: integer
          example: 5
        supplier:
          type: string
          example: 'Supplier A'
        photoUrl:
          type: string
          example: 'https://www.stockimage.com/pot'
        price:
          type: integer
          example: 17.95
        name:
          type: string
          example: 'Perfekt Pande'
        description:
          type: string
          example: 'Denne Pande best r af en l kker jern-legering, der
              g r maden super spr d'
        qty:
          type: integer
          example: 5
    ShoppingBasket:
      type: object
      properties:
        basketId:
          type: string
```

```
        example: LKJHI78oyhLO/8hiulhi67g
    User:
      type: object
      properties:
        uid:
          type: string
          example: JKhlIUYo879pjip98?IJP()/
```

3.6 Cyclomatic Complexity

```
Python\client.py
    M 32:4 Client.get_user - A (3)
    M 47:4 Client.get_order - A (3)
    C 8:0 Client - A (2)
    M 25:4 Client.get_user_list - A (2)
    M 56:4 Client.get_orders - A (2)
    M 70:4 Client.get_basket_items - A (2)
    M 11:4 Client.__init__ - A (1)
    M 17:4 Client.create_user - A (1)
    M 21:4 Client.remove_user - A (1)
    M 36:4 Client.create_order - A (1)
    M 43:4 Client.remove_order - A (1)
    M 62:4 Client.create_item - A (1)
    M 66:4 Client.get_item - A (1)
    M 81:4 Client.create_basket - A (1)
    M 85:4 Client.get_inventory - A (1)
Python\main.py
    C 56:0 Basket - B (6)
    M 85:4 Basket.post - B (6)
    C 152:0 Order - B (6)
    C 30:0 User - A (5)
    M 59:4 Basket.get - A (5)
    M 124:4 Basket.delete - A (5)
    M 155:4 Order.get - A (5)
    M 175:4 Order.post - A (5)
    M 33:4 User.post - A (4)
    C 193:0 Orders - A (4)
    C 14:0 UserList - A (3)
    M 196:4 Orders.get - A (3)
    C 211:0 Inventory - A (3)
    M 17:4 UserList.get - A (2)
    M 214:4 Inventory.get - A (2)
Python\classes\basket.py
    M 10:4 Basket.update - A (4)
    C 4:0 Basket - A (3)
    M 25:4 Basket.remove - A (2)
    M 7:4 Basket.__init__ - A (1)
Python\classes\inventory.py
    C 4:0 Inventory - A (2)
    M 22:4 Inventory.get_inventory - A (2)
    M 7:4 Inventory.__init__ - A (1)
    M 10:4 Inventory.add - A (1)
    M 14:4 Inventory.remove - A (1)
    M 18:4 Inventory.update_stock_lvl - A (1)
Python\classes\item.py
    M 55:4 Item.get_supplier_lvl - A (5)
    M 83:4 Item.order_from_supplier - A (5)
    C 41:0 Item - A (3)
    F 1:0 get_stock_lvl_supplier_A - A (2)
    F 10:0 get_stock_lvl_supplier_B - A (2)
    F 19:0 order_from_supplier_A - A (2)
```

```
    F 30:0 order_from_supplier_B - A (2)
    M 44:4 Item.__init__ - A (1)
    M 62:4 Item.get_item - A (1)
    M 76:4 Item.get_stock_lvl_local - A (1)
    M 79:4 Item.inc_stock_lvl_local - A (1)
Python\classes\order.py
    C 6:0 Order - A (3)
    M 7:4 Order.__init__ - A (2)
Python\classes\user.py
    C 4:0 User - A (2)
    M 7:4 User.__init__ - A (1)
    M 13:4 User.create_basket - A (1)
    M 17:4 User.remove_basket - A (1)
```

3.7 Unit Test Coverage Report

```
Name                  Stmts   Miss  Cover
---------------------------------------
classes\basket.py        15      0   100%
classes\item.py          46     16    65%
classes\user.py          11      0   100%
test\__init__.py          0      0   100%
test\test.py             93      1    99%
---------------------------------------
TOTAL                   165     17    90%
```

3.8 C4 Model

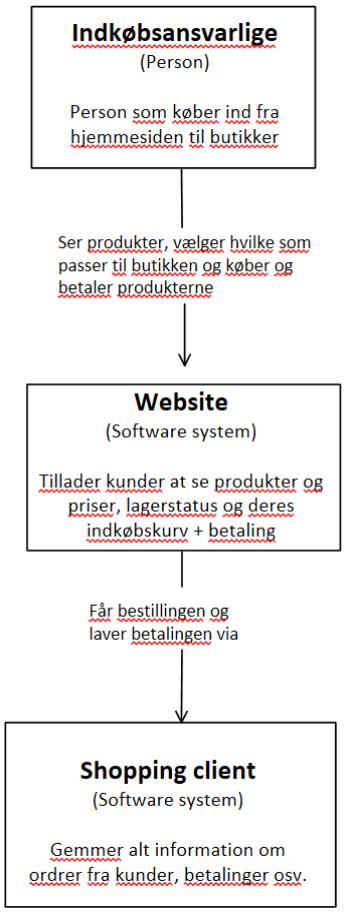

Figure 4: C4 Model: Level 1 - Context

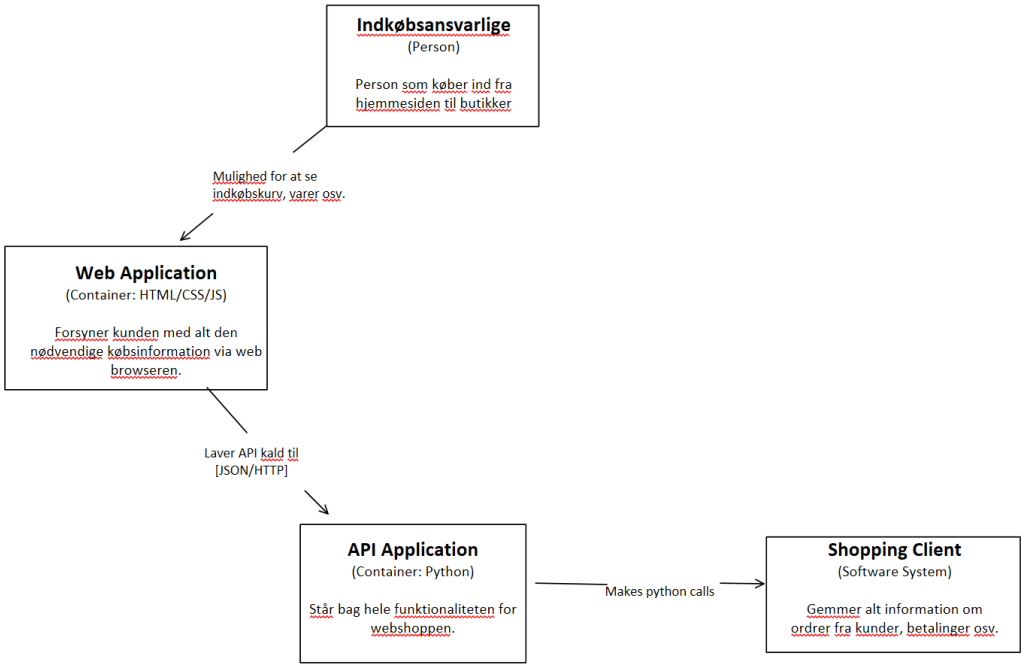

Figure 5: C4 Model: Level 2 - Container

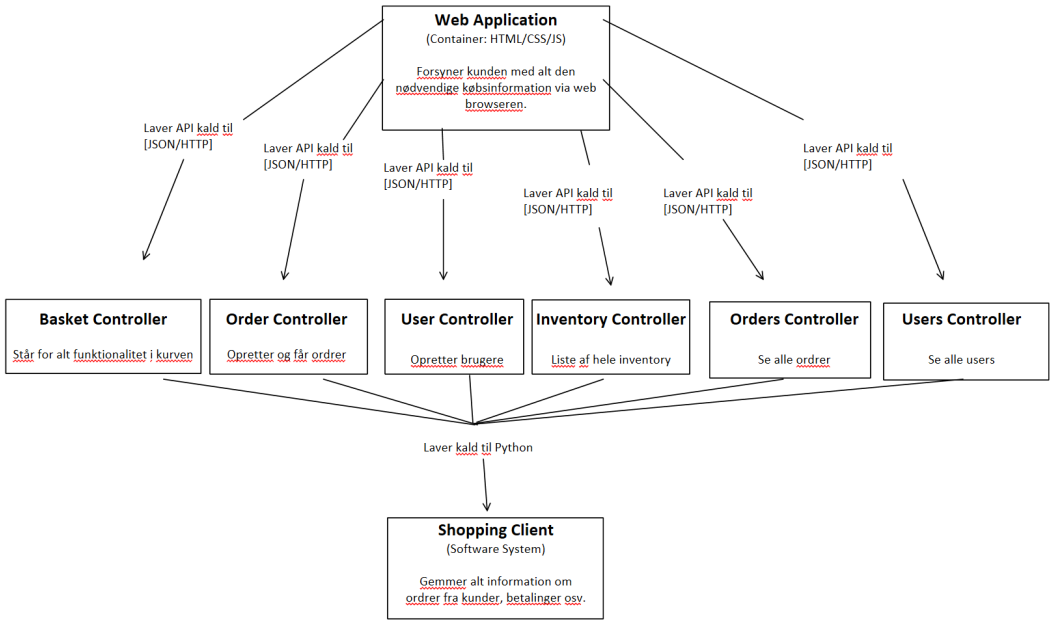

Figure 6: C4 Model: Level 3 - Component