**EINDONTWERP**

|  |  |
| --- | --- |
| **Naam** | Timo Brouwer |
| **Studentnummer** | 23055987 |
| **Klas** | 1 |

**OPDRACHT**

Geef hieronder een korte beschrijving van je opdracht (een samenvatting van je productvisie) zodat je docent OPT3 ook begrijpt met welke opdracht jij bezig bent (neem dit onderdeel over uit je laatst ingeleverde opdracht en vul aan of wijzig de casus aan de hand van eigen bevindingen of op basis van feedback die je hebt gekregen van docenten en/of medestudenten):

“Freezer" is een app die 65-plussers helpt hun vriezerinhoud te beheren door het scannen van producten en het uitprinten van stickers met QR-codes, namen, en houdbaarheidsdata. Dit vereenvoudigt het bijhouden van voorraden en vermindert voedselverspilling.

**USER STORIES**

Bij de oplevering van het ontwerp hieronder ben ik uitgegaan van de volgende User Stories (gebruik de ‘+’ rechts van de tabel om een User Story toe te voegen):

|  |  |
| --- | --- |
| **ID** | **Omschrijving** |
| US1 | Als gebruiker wil ik producten kunnen toevoegen aan mijn vriezerinventaris zodat ik het niet meer op papier hoef te schrijven en het niet meer hoef te onthouden. |
| US2 | Als gebruiker wil ik een duidelijk overzicht met producten in de vriezer zodat ik minder voedselverspilling heb. |
| US3 | Als gebruiker wil ik producten kunnen verwijderen nadat ik ze heb gebruikt zodat ik geen dubbele producten koop. |
| US4 | Als gebruiker wil ik een rapportage van mijn voedselvoorraad zodat ik dit kan delen met bijvoorbeeld mijn mantelzorger. |
| US5 | Als mantelzorger wil ik een Admin menu en een normale gebruikersmenu zodat ik de voorraad kan beheren van mijn client zonder dat hij wijzigingen kan toepassen |

**EINDONTWERP**

In het eindontwerp heb je Patterns en SOLID-principes toegepast en je hebt Classes toegevoegd die je nodig hebt om je applicatie te laten werken (zoals voor het tonen van een GUI of Command Line voor bijvoorbeeld een menu). In het ontwerp heb je ook de types van je properties, parameters en methodes toegevoegd. Zie het volgende document voor voorbeelduitwerking en aanwijzingen:

‘[OPT3 16.2 - Ontwerp en realisatie met patterns - Toelichting.pdf](https://brightspace.hhs.nl/content/enforced/18624-H-SE-S2OPT3-20_2022_VT/documenten/OPT3%2016.2%20-%20Ontwerp%20en%20realisatie%20met%20patterns%20-%20Toelichting.pdf)’

Afbeelding met tekst, schermopname, zwart-wit, ontwerp

Automatisch gegenereerde beschrijving

\* Controleer of het plaatje goed leesbaar is.

**PATTERNS**

Beschrijf hieronder hoe je het Template Method Pattern en een tweede Pattern hebt toegepast in bovenstaand eindontwerp:

|  |  |
| --- | --- |
| **Pattern** | **Beschrijving** |
| Template Method Pattern | Na(a)m(en) van template method (s): run()  Ik heb een basisprogramma (**AppMenu**) gemaakt dat een lege **run()** functie heeft. Twee andere programma's (**UserAppMenu** en **AdminAppMenu**) vullen deze **run()** functie in met specifieke stappen voor gebruikers en beheerders. Dit helpt om de algemene opzet van het menu hetzelfde te houden, maar laat de details per type gebruiker verschillen. Het doel is om de code netjes en overzichtelijk te houden. |
| Abstract Factory Pattern | Hoe heb ik dit toegepast?   1. **Basisplan (UserFactory interface):**    * Dit is het algemene plan voor het maken van menu's. Het zegt alleen dat er een methode **createAppMenu()** moet zijn die een menu maakt, maar het zegt niet precies welk menu. 2. **Specifieke plannen (AdminFactory en GebruikerFactory):**    * Deze volgen het algemene plan en zeggen precies welk menu ze maken. **AdminFactory** maakt een beheerdersmenu (**AdminAppMenu**) en **GebruikerFactory** maakt een gebruikersmenu (**UserAppMenu**). 3. **De gebruiker kiest:**    * In de **User** klasse vraagt het programma de gebruiker om te kiezen tussen admin of gewone gebruiker.    * Afhankelijk van de keuze, gebruikt het programma de juiste fabriek (**AdminFactory** of **GebruikerFactory**) om het juiste menu te maken en te laten draaien.   **Waarom heb ik dit gedaan?**  Het doel is om de code netjes en flexibel te maken. In plaats van precies te moeten weten welk menu gemaakt moet worden, kan de **User** klasse gewoon het algemene plan (**UserFactory**) volgen. Dit maakt het makkelijk om nieuwe menu's toe te voegen zonder veel aanpassingen aan de bestaande code te doen. Hierdoor blijft de code overzichtelijk en makkelijk te onderhouden. |

**SOLID-PRINCIPES**

Beschrijf hieronder voor alle SOLID-principes hoe je die hebt toegepast in Bovenstaand eindontwerp:

|  |  |
| --- | --- |
| **SOLID-principe** | **Beschrijving (gebruik een spatie als je geen toelichting geeft)** |
| Single Responsibility | Elk van de klassen in het project heeft een specifieke verantwoordelijkheid. Bijvoorbeeld, de **Manager**-klasse is verantwoordelijk voor het beheer van producten, terwijl de **Email**-klasse verantwoordelijk is voor het verzenden van e-mails. Dit helpt bij het onder controle houden van een duidelijke en eenvoudige codebase. |
| Open-Closed | De klassen in het project zijn ontworpen om open te zijn voor uitbreiding maar gesloten voor wijzigingen. Bijvoorbeeld, de **AppMenu**-branch kan worden uitgebreid door nieuwe klassen toe te voegen (zoals **AdminAppMenu** en **UserAppMenu**) zonder de bestaande code te wijzigen. Dit maakt het gemakkelijker om nieuwe functionaliteit toe te voegen zonder de bestaande code te verbreken. |
| Liskov | De klassen in het project voldoen aan het substitutieprincipe, wat betekent dat objecten van een afgeleide klasse zich kunnen gedragen als objecten van de basisklasse. Bijvoorbeeld, **AdminAppMenu** en **UserAppMenu** zijn afgeleid van **AppMenu** en kunnen naadloos worden gebruikt waar een **AppMenu** wordt verwacht, zoals in de **User**-klasse. |
| Interface Segragation | De **UserFactory**-interface volgt het principe van interface segregatie door slechts één methode, **createAppMenu**, te definiëren. Dit zorgt ervoor dat klassen alleen afhankelijk zijn van de methoden die ze nodig hebben, waardoor onnodige afhankelijkheden worden vermeden. |
| Dependency-Inversion | Het project past het principe van dependency inversion toe door hogere modules niet afhankelijk te maken van lagere modules, maar beide afhankelijk te maken van abstracties. Bijvoorbeeld, zowel **AdminAppMenu** als **UserAppMenu** maken gebruik van de **Manager**-klasse via de **ProductManager**-interface, waardoor de implementatiedetails van **Manager** kunnen worden gewijzigd zonder de hogere modules te beïnvloeden. |