Delinnlevering 4 - cantclosevim

Joakim Tollefsen Johannesen
joakimtj@hiof.no

Niklas Berby
niklab@hiof.no

May 2, 2024

1 Plan for Brukertesting

Fra siste brukertest er det en del endringer som er gjort for å forhåpentligvis skape en mer behagelig og enklere brukeropplevelse for de som har null kjennskap til API'en.

Disse omhandler da det grunnleggende i API'et som for eksempel:

- Instansiering av hovedressurser.
  - Det 'fysiske' konseptene som `Harbor`, `Ship*`, et al.
  - Opprettelse av oppdrag i form av `Sailing`.
  - Opprettelse og bruk av Logging i simulering.
- Samspillet mellom objektene/klassene/typene.
- Kjøre enkle simuleringer uten å rote seg bort i den dypere funksjonaliteten til API'et.

Vi er da interessert om disse tiltakene har hjulpet, samt om de nye funksjonene som er implementert når en grad av vanskelighet som vi mener er akseptabelt. Den nye funksjonaliteten er som følger:

- Bygge strukturene for simulering av frakt.
  - `StorageArea`
  - `StorageColumn`
  - `Dock`
- Opprette kjøretoy som sørger for bevegelse av frakt innenfor havnen.
  - `AGV`
  - `Crane`
- Opprette fraktskipet og dets frakt.
  - `Container`
- Kjøre en enkel frakt simulering.

Brukertestingen vår er da hovedsakelig i to deler. Del 1 er mer fokusert på grunnleggende ting i API'en mens Del 2 er mer komplisert.

1.1 Del 1

I del én er testgruppen gitt oppgaver som skal i utgangspunktet hjelpe dem å bygge en grunnleggende forståelse av API'et.

Der det er relevant er de gitt en kort beskrivelse av konseptene. Som for eksempel om det er tvil i hva en havn ér.

Vi har planlagt følgende oppgaver der testgruppen har intellisens. Det blir også gitt hint om de står stille eller føler de trenger det for å forstå oppgaven. Hintene er gitt rekkefølge fra minste nyttig til mest nyttig.

1.1.1 Oppgave - Opprett en Havn

En havn er et vannområde der skip og båter kan bli fortøyd.

Opprett en havn og konfigurer den slik at skip kan fortøyes der.
Ta i bruk de klasser, metoder eller konstruktører dere føler er hensiktsmessige.

Her skal testgruppen opprette en havn med en eller flere kaiplasser.

Hint

1. Havn er 'Harbor'.
2. Kaiplass er 'Dock'.
3. En havn kan ha flere kaiplasser.
4. Benytt hjelpemetode for å lage vilkårlig antall kaiplasser.

1.1.2 Oppgave - Opprett et Skip

Et skip er et fartøy som i kommersielt bruk har enten passasjerer ombord eller gods den frakter.

Opprett et skip dere selv føler er best egnet til å senere utføre en enkel seiling. Det vil si frakt av gods ikke er nødvendig.

Her skal testgruppen opprette et skip. Skipet kan være både av type `CommercialShip` og `FreightShip` da simuleringsoppgaven ikke tar stilling til dette.

Hint

1. Skip er '`Ship`'.
2. '`Ship`' er en abstrakt klasse.
3. '`CommercialShip`' vil da være beste analog til et passasjer skip.
4. '`CommercialShip`' vil ikke kunne brukes til frakt av gods.

1.1.3 Oppgave - Opprett en eller flere Seilinger

En seiling er et 'oppdrag' der et skip reiser til en kaiplass. Seilingen skjer i et spesifisert tidsrom - altså fra en start-tid til en slutt-tid.

Opprett en eller flere seilinger med grunnlag i de tidligere oppgavene.

Her skal testgruppen opprette seiling(er) med objektene de opprettet i de forrige oppgavene.

Hint

1. Seiling er '`Sailing`'.
2. '`Sailing` krever et skip.
3. '`Sailing`' krever et reisemål.
4. '`Sailing`' krever tidsrommet for reisen.

1.1.4 Oppgave - Opprett en Logger

Det er flere typer loggere. Den mest enkle skriver kun relevante beskjeder under simuleringen til konsollen.

Opprett en logger for senere bruk.

Her skal testgruppen opprette en logger som de skal bruke i simulerings klassen for å få utskrift av hendelser i simuleringen.

Hint

1. Loggere har LogLevel(s) som bestemmer hvilke meldinger som skal skrives ut.

2. Verbøse-meldinger, Advarsels-meldinger etc.

3. Simulering krever en logger.

1.1.5 Oppgave - Opprett og Kjør en Simulering

En simulering kjører de oppdragene (seilingene) som er gitt.

Opprett simulatoren og konfigurer tidligere objekter for å kunne kjøre en simulering.

Den største oppgaven i fase 1. Det viktigste punktet i denne oppgaven er at de får med seg at seilinger er lagret i havnen som skal utføre den. De må også lære at de selv må ta hensyn til om en kaiplass er opptatt og selv opprette seilinger der det ikke er noen konflikter mellom forskjellige skips reiser for at de skal utføres.

Hint

1. Simulatoren heter '`SimulateHarbor`'

2. Simulatoren krever en Logger for å kunne utføre simuleringen.

3. Simulatoren utfører de seilingene i de havnene som er gitt som argument.

4. Simulatoren utfører kun de seilingene som er innenfor det gitte tidsrommet til simuleringen.

1.2 Del 2

I del to skal testgruppen utføre en mer avansert seiling. Dette vil kreve en dypere forståelse av API'et da de må opprette ressurser for konsepter de kanskje ikke har noen tidligere kunnskap om. Mer spesifikt: frakt av gods.

I del to får de tilgang til dokumentasjon og et klassediagram.

1.2.1 Oppgave - Frakt: Havn Konfigurasjon

Tilføy eller gjør endringer på havnene dere lagde for å støtte bevegelse av gods i havnen.

Her skal testgruppen opprette StorageArea og StorageColumn(s). De skal også opprette AGV'er.

Hint

1. Havner har et område dedikert til lagring.
2. Lagringsområdet har også flere steder der gods er lagret.

1.2.2 Oppgave - Frakt: Kaiplass Konfigurasjon

Tilføy eller gjør endringer på kaiplassene til havnene for å støtte lossing av gods.

Her skal testgruppen legge til et vilkårlig antall kraner og et vilkårlig antall AGV'er.

Hint

1. Lossing av gods håndteres av kraner.
2. AGV'er flytter godset fra kaiplassen til lagringskolonnene.

1.2.3 Oppgave - Frakt: Fraktskip

Opprett et skip og dets last. Lag seilingene som skal utføres.

Her skal testgruppen opprette fraktskip med last. Last (kontainere) er enten instansiert separat eller laget generisk med hjelpemetoder. Dette er ment å teste hvor behjelpelige hjelpemetodene våre er og de valg brukere av API'en muligens tar med henhold til egne styrker og tidligere erfaringer.

Så skal de opprette seilinger som skal utføres.

Hint

1. Last kan også opprettes internt i fraktskip med hjelpemetoder.

1.2.4 Oppgave - Frakt: Simulering

Simuler seilngene dere lagde i forrige oppgave.

Dette er kun for å se om det fungerer som de har tenkt.

2 Gjennomføring og Evaluering av Brukertest

Brukertesten viste følgende svakheter i API designet:

- Det er ikke intuitivt at seilinger lagres i en havn.
- Det er derfor heller ikke intuitivt at en 'passer' in havnene som skal simuleres inn som argument til simulerings-metoden.
- For at et skip skal kunne seile må det være forankret til en kaiplass. Det er ingenting i API'et som gjør dette klart for bruker utenom en vag feilmelding.
  - Dette var noe vi trodde vi hadde endret/løst tidligere. Men her ser vi at hva enn vi gjorde tydeligvis ikke fikset det.
- Flere events burde legges til.
- Måten frakt av gods innenfor havner fungerer er for 'opaque' og gir ikke bruker evnen til å feks bestemme hvor frakt skal eventuelt gå.

3 Evaluering av tildelt gruppes API

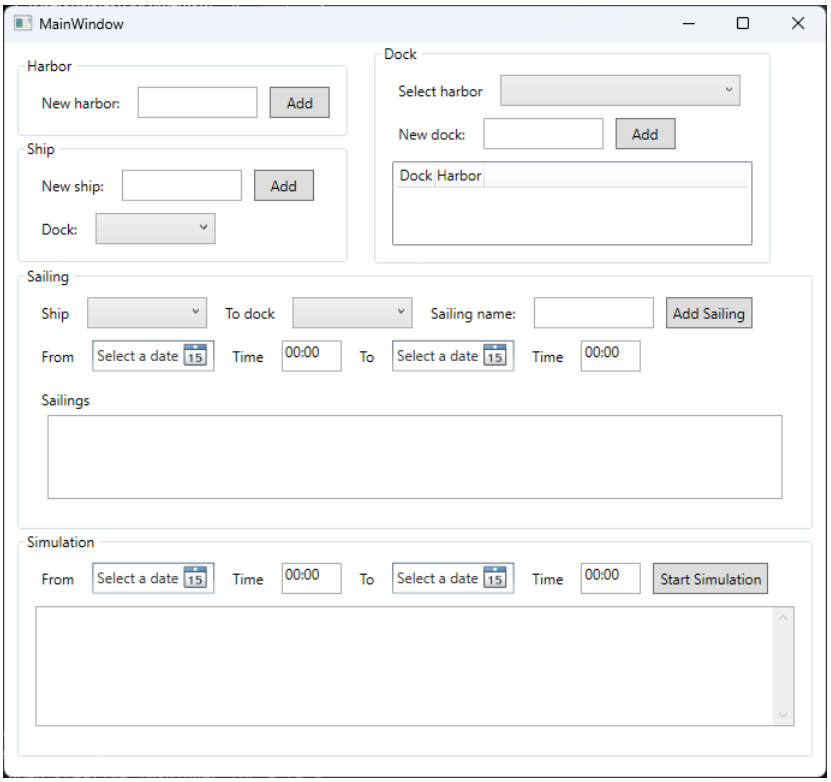
JECH sitt rammeverk er en service-basert løsning for varehus-administrering.

I forhold til vårt eget API er deres mye mer event-basert. De fleste hendelser i API'et 'raiser' en event, med dette kan vi selv velge hva som skal gjøres med disse hendelsene. I vårt tilfelle skrev vi ut til konsollen informasjon om hva som skjedde.

Med at rammeverket er service-basert var det litt vrient å forstå sammenhengen. Det er en del servicer som må instansieres og som ikke er helt entydig med første øyekast. Dette kan være konsekvens av manglende erfaring med slike service-baserte systemer men vi mener også at det kunne vært noe mer beskrivende XML-kommentarer. For eksempel, 'Klasse for håndtering av Palle-objekter' er en enkel forklaring på hva det gjør men det kunne vært noe mer informasjon om det faktiske bruksområdet.

Å 'subscribe' og 'raise' events med kode er som nevnt litt tvetydig i starten når det er så mange forskjellige events. Men etter første metode for å håndtere disse eventsene var det intuivt hvordan resten av API'en fungerte. Det er et sammenhengende rammeverk uten 'overraskelser'.

4 Implementasjon av grafisk grensesnitt



Som krav til denne delinnleveringen skulle vi implementere et grafisk grensesnitt for å kunne sette opp havner, skip, kaiplasser og seilinger. Det skulle også være mulighet til å kjøre simuleringer på seilingene, samt å kunne se en logg over hva som skjedde under simuleringen. Vi benyttet WPF for denne delen.

Etter å ha lagt til de nødvendige objektene, kan en simulering kjøres ved å trykke på "Start Simulation"-knappen. Output fra simuleringen vises i en `RichTextBox` nederst i grensesnittet. [1]

4.1 Bruk

Grensesnittet gjør det mulig å opprette havner, skip, kaiplasser og seilinger. Visse felt er påkrevd for å opprette objektene, og det er lagt til validering for å hindre at brukeren oppretter objekter uten nødvendig informasjon.

[1] Visse begrensninger medfølger.

Kaiplasser og seilinger krever at det er skip og havner tilgjengelig, så det er lagt til stopper for å hindre at brukeren oppretter disse objektene uten at det er skip eller havner tilgjengelig. Dette gjøres kun i koden til grensesnittet, og ikke i APIen.

4.2 Restriksjoner

Oppgaven var klar på at om rammeverket var manglende i visse områder, skulle vi jobbe rundt det ved å heller gjøre endringer i grensesnittet. Dette har vi delvis gjort i form av validering av input. Likevel måtte vi ty til noen kreative løsninger for å få alt til å fungere.

En simulering krever en liste over havner. Dette er fordi simuleringen kjører alle seilingene som er registrert for hver havn. Vi var usikre på hvordan vi skulle implementere dette, og la derfor til en "master harbor" som fikk tilføyd alle seilingene. Dette skjer bak kulissene, og oppleves ideelt sett ikke av brukeren. Vi er klare over denne restrisjonen, og ønsker å forbedre dette i fremtidige iterasjoner. Det hadde vært mulig å gjøre en fiks for dette allerede nå, men vi valgte å fokusere på grensenittet og la eventuelle fikser til APIen vente til senere.

Kraner, AGVer og lagringsområder i havnene er ikke implementert i det grafiske grensesnittet. Vi tolket at disse ikke var nødvendige for å kunne kjøre en simulering, og valgte derfor å la de frafalle i denne omgang.

Output fra simuleringen vises i en `RichTextBox`, men på grunn av måten APIen er implementert på, er det ikke mulig å få en kontinuerlig oppdatering av denne. I tillegg er det ikke mulig å stoppe en simulering, og outputen fra simuleringen har ingen timestamp slik som den vil ha i en konsollimplementasjon. Dette er fordi eventene som vi abonnerer på ikke gir like fullstendig informasjon som de som blir skrevet til konsollen i `ConsoleLogger` klassen. Dette hadde nok en gang vært mulig å fikse, men vi lot det være for denne iterasjonen.

5 Refleksjon over WPF dokumentasjon

URL til hovedsiden for dokumentasjon som ble benyttet mest: `https://learn.microsoft.com/en-us/dotnet/desktop/wpf/`

WPF, som et rammeverk utviklet av Microsoft, har en omfattende dokumentasjon. Den inkluderer alt fra en oversikt over alle klasser og metoder,

til eksempler på bruk. Den har også en fin forklaring på forskjeller mellom WPF og Windows Forms, som er et annet rammeverk for å lage grafiske grensesnitt i C#.

Dokumentasjonen er oversiktlig og lett å navigere. Den har en sidebar med et søkefelt som gjør det enkelt å finne det du er ute etter.

Som et grensesnitt som er tett integrert med C#, er det også en del dokumentasjon som er relevant for C# generelt. Dette er også inkludert i dokumentasjonen. Vi følte at Visual Studio har verktøy som gjør WPF lett å bruke, og IntelliSense var også veldig nyttig for å finne ut hva som var tilgjengelig.

6 Diskusjon om Regler fra Boken

Boken er en essensiell ressurs for utviklere som lager et API. De råd og anbefalninger den gir er ikke kun relevante for de som jobber med .NET men også de som jobber i andre språk og verktøy. Den gir utviklere viktig informasjon om hvordan man utvikler gjenbrukbare og vedlikholdbare rammeverk som er brukervennlige.

Noen nøkkeltemaer vi vil ta ut fra boken er:

1. `Konsistens` Å være konsistent i API-design er avgjørende. Dette gjør API-en lettere å lære og bruke riktig, siden brukerne kan dra nytte av kjent mønstre og praksiser.

2. `Ergonomi` Boken understreker viktigheten av å lage et API som er enkelt og intuitivt å bruke. Dette innebærer å redusere kompleksiteten og unngå unødvendig fleksibilitet som kan forvirre brukeren.

3. `Ytelse` Rådene inkluderer også overveielser om ytelse, der det anbefales å velge design som tilbyr optimal ytelse uten å komplisere API-et unødvendig.

4. `Versjonering og bakoverkompatabilitet` Strategier for hvordan man skal håndtere endringer i et rammeverk over tid uten å bryte eksisterende kode.

5. `Dokumentasjon og vedlikehold` God dokumentasjon er avgjørende for ethvert rammeverk, og boken gir retningslinjer for hvordan man

dokumenterer API-er effektivt. Det understrekes også viktigheten av å vedlikeholde og oppdatere rammeverket regelmessig.