STUDENTRAPPORT

NTNU

KANDIDATNUMMER(E)/NAVN:
10085, Cathrine Kristiansen

DATO:	FAGKODE:	STUDIUM:	ANT SIDER/BILAG:
08.12. 2025	IDATT1003_1	Dataingeniør (BIDATA), Programmering 1	23 /

FAGLÆRER(E) :

- MUHAMMAD ALI NOROZI
- Arne Gerhard Styve
- Ivar Farup
- Surya Bahadur Kathayat

TITTEL :

Studiedagbok

SAMMENDRAG:

Denne rapporten beskriver utviklingen av et konsollbasert dagboksystem laget i Java som en del av emnet IDATT1003, Programmering 1, ved Norges teknisk-naturvitenskapelige universitet (NTNU). Oppgaven baserer seg på et objektorientert design, med fokus på kvalitet i kode og arkitektur. Systemet gjør det mulig for brukeren å registrere, vise, søke og slette dagbokinnlegg, samt administrere forfattere og vise statistikk over deres innlegg. Arbeidet bygger på sentrale prinsipper som kohesjon, kobling, enkeltansvar og separasjon av bekymringer, som danner grunnlaget for drøftingen av løsningens kvalitet.

Arbeidsprosessen har fulgt en iterativ og inkrementell utviklingsprosess, med støtte av verktøy som IntelliJ, Git, Checkstyle, SonarLint og PlantUML. Systemet er testet med både enhetstester og brukertesting, som videre har bidratt til forbedring av inputvalidering og design. Resultatet ble et fungerende system som oppfyller hovedkravene, men med tydelige forbedringsmuligheter.

Denne oppgaven er en besvarelse utført av student(er) ved NTNU.

Deklarasjon om KI-hjelpemidler

Har det i utarbeidingen av denne rapporten blitt anvendt KI-baserte hjelpemidler?

[ ] Nei

[x] Ja

Hvis *ja*: spesifiser type av verktøy og bruksområde under.

Tekst

[ ] **Stavekontroll**. Er deler av teksten kontrollert av:
Grammarly, Ginger, Grammarbot, LanguageTool, ProWritingAid, Sapling, Trinka.ai eller lignende verktøy?

[ja] **Tekstgenerering**. Er deler av teksten generert av:
ChatGPT, GrammarlyGO, Copy.AI, WordAi, WriteSonic, Jasper, Simplified, Rytr eller lignende verktøy?

[ ] **Skriveassistanse**. Er en eller flere av ideene eller fremgangsmåtene i oppgaven foreslått av:
ChatGPT, Google Bard, Bing chat, YouChat eller lignende verktøy?

Hvis ja til anvendelse av et tekstverktøy - spesifiser bruken her:

Det er fått hjelp av ChatGPT i enkelte deler av rapporten. Alle avsnitt og innhold er skrevet av meg, men jeg har fått hjelp til redigering av flyt og formelt språk i enkelte deler av oppgaven.

Kode og algoritmer

[**ja**] **Programmeringsassistanse**. Er deler av koden/algoritmene som i) fremtrer direkte i rapporten eller ii) har blitt anvendt for produksjon av resultater slik som figurer, tabeller eller tallverdier blitt generert av: *GitHub Copilot, CodeGPT, Google Codey/Studio Bot, Replit Ghostwriter, Amazon CodeWhisperer, GPT Engineer, ChatGPT, Google Bard* eller lignende verktøy?

Hvis *ja t*il anvendelse av et programmeringsverktøy - spesifiser bruken her:

I oppgaven har jeg brukt ChatGPT både som sparringspartner og til å generere noe av koden.

1. Det jeg har fått hjelp til er forslag til konvertering av brukerinndata for to datoer til LocalDateTime i Java ved bruk av DateTimeFormatter. Eksempel på koden så slik ut:

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyyMMdd");

LocalDateTime fraDato = LocalDate.parse(fraInput, formatter).atStartOfDay();
LocalDateTime tilDato = LocalDate.parse(tilInput, formatter).atTime(23, 59, 59);
```

Forklaring med min egen forståelse:

DateTimeFormatter.ofPattern("yyyyMMdd"):	Her opprettes et formatteringsobjekt som kan tolke datoer skrevet som åtte

	sammenhengende tall, f.eks. 20250615. Dette representerer år, måned og dag.
LocalDate.parse(fraInput, formatter):	Dette konverterer brukerens tekstinput (en streng) til et LocalDate-objekt.
.atStartOfDay():	*Gjør om datoen til et LocalDateTime-objekt, klokken 00:00:00, slik at hele dagen regnes med.*
tilDato = ... atTime(23, 59, 59):	*For sluttdatoen settes tidspunktet til slutten av dagen for å inkludere alle innlegg som ble skrevet den dagen.*

Ved å bruke LocalDateTime for både start- og sluttdato kan jeg sammenligne dato og tid direkte med tidsstemplene i dagbokinnleggene. Dette gjør det mulig å filtrere hvilke innlegg som ligger innenfor tidsintervallet som er valgt. Selv om ChatGPT foreslo koden, har jeg tilpasset den til mitt eget prosjekt. Jeg har også forsikret meg om at koden fungerer sammen med resten av systemet, spesielt i søkefunksjonen finnRegistrertDagbokinnleggEtterDato(), som nå returnerer en liste av innlegg mellom to datoer.

2. Jeg fikk også hjelp til å strukturere en oversikt over forfattere og antall dagbokinnlegg i en 2D-liste med kolonner. Forslaget ble gjennomgått, tilpasset og kvalitetssikret av meg. Kodeutdraget jeg brukte i koden var:

```
public record AuthorPostCount(Author author, int postCount) {}

public List<AuthorPostCount> createAuthorPostList(DiaryEntryRegister register) {
    List<AuthorPostCount> list = new ArrayList<>();

    for (Author author : register.getAllAuthors()) {
        int count = numberOfPostPerAuthor(register, author);
        list.add(new AuthorPostCount(author, count));
    }

    return list;
}
```

Forklaring med min egen forståelse:

Java records:	Er en kortfattet måte og representere en datastruktur som er mindre rddig og mer utsatt for feil, i forhold til for eksempel List.<Object>.

AuthorPostCount:	Fungerer som en egen hjelpeklasse som gjør at hver rad med beholdninger består av (Author, antall innlegg).
Metodens funksjoner:	Henter alle forfattere fra registeret, teller antall innlegg per forfatter og legger resultatet in i listen.
Hvorfor dette alternativet:	Gir lettere lesbarhet, sterkere typesikkerhet og mindre sjanse for logiske feil.
Kvalitetssikre KI-generert kode:	Siden KI ikke kjenner hele koden min er det lett for at den genererer feil slik at funksjonaliteten ikke blir slik jeg ønsker.

Bilder og figurer

Bildegenerering. Er ett eller flere av bildene/figurene i rapporten blitt generert av: *Midjourney, Jasper, WriteSonic, Stability AI, Dall-E* eller lignende verktøy?

Hvis ja til anvendelse av et bildeverktøy - spesifiser bruken her:

Andre KI-verktøy

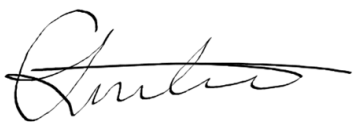
ja

Andre KI-verktøy. har andre typer av verktøy blitt anvendt? Hvis ja spesifiser bruken her:

Google transelate - Jeg har brukt Google Translate til å få hjelp med oversettelse av enkelte setninger mellom norsk og engelsk. All faglig tekst er skrevet, kontrollert og tilpasset av meg selv.

x

Jeg er kjent med NTNUs regelverk: *Det er ikke tillatt å generere besvarelse ved hjelp av kunstig intelligens og levere den helt eller delvis som egen besvarelse.* Jeg har derfor redegjort for all anvendelse av kunstig intelligens enten i) direkte i rapporten eller ii) i dette skjemaet



Underskrift/Dato/Sted

INNHOLD

Innholdsfortegnelse

1 Introduksjon 1
1.1 Bakgrunn 1
1.2 Kravspesifikasjon 1
1.2.1 Overordnet beskrivelse 1
1.2.2 Funksjonelle krav 1
1.2.3 Ikke-funksjonelle krav 2
1.2.4 Use-case diagram 3
1.3 Avgrensninger 3
1.4 Begreper/Ordliste 4
2 Teori 5
3 Metode 5
3.1 Utviklingsprosess 6
3.2 Verktøy 7
3.3 Bruk av KI verktøy 8
4 Resultat 8
4.1 Teknisk Design 8
4.2 Implementasjon 8
4.2.1 Oversikt over alle klasser som er brukt i programmet 9
4.2.2 En oversikt over alle biblioteker og API som er brukt i programmet 9
4.2.3 Sortering av dagbokinnlegg: 10
4.2.4 UML-diagram 10
4.3 Testing 13
4.3.1 Testklasser 13
4.3.2 Brukertesting 14
4.4 Utrulling til sluttbruker (deployment) 14
5 Drøfting 14
5.1 Drøfting av løsning/design 14
5.2 Drøfting av prosess 15
5.3 Drøfting av bruken av KI-verktøy 15
5.4 Konklusjon – erfaring 16

Figurliste

Figur 1 Use Case diagram.....3
Figur 2 Klassediagram 11
Figur 3 Aktivitetsdiagram 12
Figur 4 Sekvensdiagram 12

Tabelliste

Tabell 1 Funksjonelle krav 1
Tabell 2 Ikke-funksjonelle krav 2
Tabell 3 Begreper og ordliste 4
Tabell 4 Verktøy 7
Tabell 5 Oversikt over klasser 9
Tabell 6 Bibliotek/API 9
Tabell 7 Testklasser 13

1 INTRODUKSJON

1.1 Bakgrunn

Prosjektet er en del av fagets læringsmål om objektorientert programmering, utvikling av Java-programmer samt UML-modellering. Oppgaven ble publisert i tre deler og har som mål å lage et konsollbasert program med funksjonalitet som gir brukeren mulighet til å opprette, søke og behandle en digital dagbok.

1.2 Kravspesifikasjon

1.2.1 Overordnet beskrivelse

Systemet er et konsollbasert dagboksystem hvor en vanlig bruker kan legge til, hente frem og administrere sine dagbokinnlegg.

1.2.2 Funksjonelle krav

Nummer:	**Krav:**
1.	Systemet skal kunne registrere nytt dagbokinnlegg, med id, tittel, innhold, tidspunkt og forfatter.
2.	Systemet skal kunne vise alle registrerte dagbokinnlegg.
3.	Systemet skal kunne finne dagbokinnlegg mellom to valgte datoer.
4.	Systemet skal kunne slette et dagbokinnlegg basert på id.
5.	Systemet skal kunne søke etter dagbokinnlegg basert på ord i enten tittel eller innhold.
6.	Systemet skal kunne vise antall dagbokinnlegg per forfatter.
7.	Systemet skal kunne avslutte programmet via menyen.

Tabell 1. Funksjonelle krav.

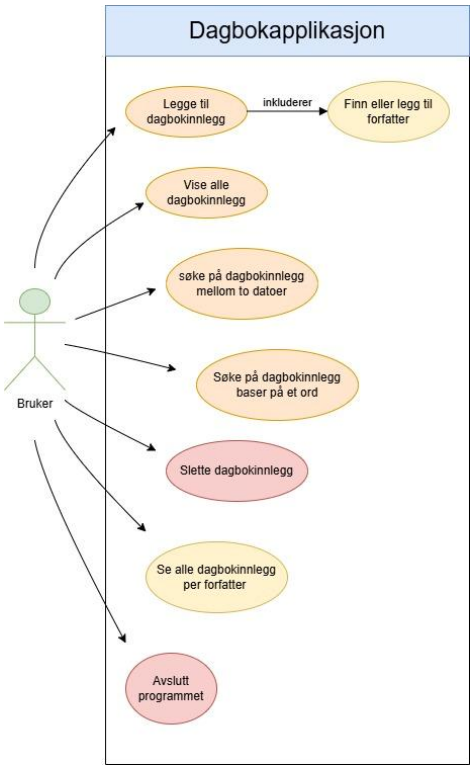
1.2.3 Ikke-funksjonelle krav

	Krav:
1. Brukervennlighet:	• Et system som viser enkel menystruktur som er lett å forstå. • Gi forståelige feilmeldinger med forklaringer. • Systemet skal ha enkle inputforespørsler med tekst og tall.
2. Ytelse:	• Systemet skal kunne håndtere dagbokinnlegg uten at programmet blir tregere. • Systemet skal kunne gjennomføre søk på dato og nøkkelord innen 1 sekund.
3. Pålitelighet og robusthet:	• Systemet skal kunne håndtere ugyldig input uten å krasje. • Systemet skal sjekke at forfatter, tittel og innhold ikke er tomme før et innlegg opprettes. • Systemet skal gi beskjed hvis brukeren prøver å slette et innlegg som ikke finnes.
4. Vedlikehold:	• Koden skal ha klare ansvarsområder fordelt mellom klasser. • Systemet skal la bruker legge til nye funksjoner.
5. Loggføring:	• Systemet skal logge alle hendelser til konsollen. • Systemet skal gi ut loggførte meldinger med nyttig informasjon for feilsøking.

Tabell 2. Ikke-funksjonelle krav.

1.2.4 Use-case diagram

Use-case diagrammet viser hvilke funksjoner brukeren kan utføre i systemet.



Figur 1. Use Case diagram

1.3 Avgrensninger

Dette prosjektet har noen avgrensninger relatert til brukerfunksjonalitet og og teknologi. Siden prosjektet er laget som et grunnleggende java-prosjekt, lagres ikke filer i en database og forsvinner når programmet avsluttes. Funksjonaliteten er derfor begrenset med tanke på registrering, visning, søk og sletting av dagbokinnlegg. Prosjektet er i seg selv laget som et konsollbasert program uten grafisk brukergrensesnitt, og eksisterende brukerinnlegg kan ikke redigeres. Siden programmet er laget som en frittstående applikasjon, har det ingen autentisering eller sikkerhetsmekanismer, og kun en type bruker har tilgang til funksjonene. Samtidig er ikke programmet laget for å håndtere store mengder data eller omfattende feilmeldinger.

1.4 Begreper/Ordliste

Begrep (Norsk)	**Begrep (Engelsk)**	**Betyding/beskrivelse**
Dagbokinnlegg	Diary Entry	En digital inngang som lar brukeren registrere nye dagbokinnlegg.
Forfatter	Author	En digital inngang som lar brukeren registrere en ny forfatter.
Dagbok	Diary	Samlingen av alle dagbokinnlegg som hører til systemet.
Dagbokregister	Diary Entry Register	En liste av alle oppførte dagbokinnlegg i systemet.
Forfatterregister		En liste over alle forfattere i systemet.
Dato – og tidsstempel	Timestamp	Informerer om når ett innlegg enten ble opprettet eller endret.
Menyvalg	Menu Option	Et valg brukeren kan velge i en en meny liste som vises i konsollen.
Posttelling per forfatter	Author Post Count	En oversikt som viser hvor mange innlegg forfatteren har i registeret.
UML		Unified Modeling Language, et standardisert og visuelt modelleringsspråk
Brukergrensensitt	UI – User Interface	Kontaktpunktet der mennesker samhandler med digitale systemer.
Kohesjon	Coheison	et prinsipp som forteller om graden av samarbeid i en programkode.
Kobling	Copling	Refererer til graden av avhengighet mellom ulike klasser eller moduler i et program.
Forstå, vedlikehold, teste, gjenbruke og utvikle	SOLID	Et akronym for fem grunnleggende designprinsipper innen objektorientert programmering (OOP) i Java og andre språk. Er ment å hjelpe utviklere med å lage programvare som er lettere å forstå, vedlikeholde, teste, gjenbruke og utvide over tid.
Enkeltansvarsprinsippet	RSP - Single Responsibility Principle.	Et grunnleggende prinsipp innen programvaredesign som sier at en klasse, modul eller funksjon kun skal ha én grunn til å endre seg.
Separasjon av bekymring	SoC - Separation of conserns	Adskillelse av ansvarsområder som er et grunnleggende prinsipp for å organisere kode.

Tabell 3. Begreper og ordliste

2 TEORI

I utviklingen av programvaren er flere grunnleggende prinsipper innen objektorientert design brukt, særlig kohesjon, kobling, enkeltansvarsprinsippet, separasjon av bekymringer og UML-modellering.

Kohesjon og kobling brukes for å vurdere kvaliteten i et objektorientert system. Lav kobling betyr at klasser er lite avhengige av hverandre, mens høy kohesjon innebærer at en klasse har et tydelig og avgrenset formål. Sammen bidrar disse prinsippene til et mer robust og vedlikeholdbart design [1].

Enkeltansvarsprinsippet (Single Responsibility Principle) sier at en klasse skal ha ansvar for kun én funksjon eller ett område. Dette gjør koden enklere å forstå, teste og videreutvikle, og reduserer risiko for feil ved endringer [2].

Separasjon av bekymringer (Separation of Concerns) handler om å dele systemet inn i tydelige deler som håndterer hver sin type oppgave. Dette gir bedre struktur, enklere feilsøking og mulighet for å utvikle ulike deler av programmet parallelt [3].

UML (Unified Modeling Language) brukes som et visualiseringsverktøy for å planlegge struktur og arbeidsflyt i programmet. Diagrammer som klassediagram, aktivitetsdiagram og sekvensdiagram gjør det mulig å kommunisere design på en måte som er uavhengig av programmeringsspråket, og gir et tydeligere bilde av systemets arkitektur [4].

3 METODE

Metoden beskriver både arbeidsform og hvilke verktøy som ble brukt.

For å få en oversikt over prosessen tegnet jeg opp klassediagram og aktivitetsdiagram. På denne måten fikk jeg mulighet til å tenke igjennom struktur, klasser, relasjoner og ansvar før jeg begynte å kode. Jeg arbeidet iterativt og inkrementelt for å kontinuerlig forbedre programmet, hvor både programkode og rapport ble utarbeidet parallelt.

Siden kravene ble publisert delvis, måtte jeg jobbe fleksibelt. Jeg startet med å utarbeide grunnfunksjonaliteten i del 1 for deretter å jobbe parallelt med del 2 og del 3, senere i prosessen. Enhetstester ble utført hyppig gjennom hele prosessen.

Jeg benyttet IntelliJ som arbeidsverktøy både for klasser, tester og UML-diagram. Jeg brukte også diagarms.net for å gjøre UML-diagram mer interessante å lese. Verktøy som Checkstyle, SonarLint og Logger bidro til kontinuerlig kvalitetssikring og feilhåndtering. Commits ble utført og sjekket inn i Github med jevne mellomrom. ChatGPT ble brukt som sparringspartner.

3.1 Utviklingsprosess

Prosjektet var utarbeidet med en prosessmodell der arbeidet ble publisert i tre deler fordelt på fem uker, med krav til programkode og rapport. Jeg planla å arbeide iterativt og inkrementelt, hvor hver del bygget videre på det neste.

Del en var utarbeidelse av grunnstruktur med domeneklasser og påfølgende tester. Del to gikk ut på å implementere registerklasser og mer funksjonalitet. Ved tredje del implementerte jeg brukergrensesnitt og eventuelt egne utvidelser. Tester ble også utført etter andre og tredje del. Modellen gjorde det mulig å motta tilbakemelding og justere programmet etter hver del.

Jeg planla å bruke tilbakemeldingene aktivt og samtidig jobbe parallelt med rapporten for å utarbeide en god løsning etter kravspesifikasjonene. Dette ga meg også mulighet til god refleksjon underveis. Hyppige innsjekkinger i Git ga meg god kontroll på utviklingen og muligheten til å reversere feil.

For å sikre god kvalitet ble verltøyene Checkstyle og SonarLint brukt og enhetstester kjørt, slik at jeg tidlig kunne oppdage feil og sikre en robust kode.

3.2 Verktøy

Verktøy:	Versjon:	Brukt til:
IntelliJ IDEA Community Edition	2025.2.1	Brukt til utvikling av Java-koden.
Java Development Kit (JDK)	24.0.2	Kjøring og kompilering av Java-programmet.
PlantUML (IntelliJ plugin)	7.13.0-IJ2023.2	Tegning av use-case-, aktivitets-, klasse- og sekvensdiagram
diagrams.net (draw.io)	Nettversjon	Tegning av UML-diagrammer manuelt
Git	2.51.0.windows.1	Lokal versjonskontroll, commit, branching og styring av prosjektets historikk.
GitHub	2025	Lagring av prosjektet i versjonskontrollen.
CheckStyle	5.114.0	Sikrer at koden har fulgt faste standarder.
SonarLint	11.5.0.82678	Statisk kodeanalyse som avdekket feil, duplikatkode, sikkerhetsadvarsler og "code smells". Sikrer kodekvalitet.
Windows Terminal / IntelliJ Terminal	-	Kjøring og testing av programmet fra kommandolinjen.
ChatGPT	-	Hjelp med forklaringer og som sparringspartner.

Tabell 4. Verktøy

3.3 Bruk av KI verktøy

Se KI-deklarasjon.

4 RESULTAT

4.1 Teknisk Design

Dette systemet er en enkel konsollbasert applikasjon som kjører lokalt på brukerens maskin. Data lagres i minnet så lenge programmet kjører, gjennom Java-objekter, og programmet bruker ingen database, filsystem eller nettverk.

Siden det ikke var noe behov for lagring mellom kjøring eller bruk av eksterne tjenester ble valget å lage en enkel konsollbasert arkitektur, med lokalt minne. Et slikt system er lettere å implementere som et øvingsprosjekt i grunnleggende objektorientert programmering.

Systemets arkitektur er delt inn lagvis. Presentasjonslaget som er brukergrensesnitt (DairyUi), hådterer input, output og presenterer menyvalget for brukeren. Logikklaget med registerklasser DiaryEntryRegister og AuthorRegister, skiller logikk fra brukergrensesnittet og behandler data ved å telle, søke og filtrere. Domenelag med dataklasser som Author og DiaryEntry, representerer data som brukeren plotter inn.

Designet ble valgt for å opprettholde et system med godt vedlikehold. For å enklere utvide systemet var det ønskelig med høy kohesjon. Dermed ble det opprettet egne registerklasser for å samle funksjonalitet for å lagre og behandle dagbokinnlegg og forfattere. UI og logikk ble holdt adskilt med separate klasser for å opprettholde separasjon av bekymringer.

Ved hjelp av et klassediagram som viser struktur med enhetsklasser, registerklasser som håndterer logikk og et brukergrensesnitt som kobler alt sammen får jeg visualisert relasjoner og avhengigheter i det tekniske designet.

4.2 Implementasjon

Den tekniske implementasjonen av dagboksystemet beskriver hvordan løsningen er bygget opp i Java og hvilke komponenter som er brukt. Her får man et innblikk i teknologien og alle klassene som er implementert i programvaren. Hvordan de ulike klassene fungerer for å skape en god funksjonalitet forklares i en tabell, mens systemarkitekturen og samhandling er illustrert gjennom UML-diagrammer.

4.2.1 Oversikt over alle klasser som er brukt i programmet

Klasse:	**Rolle/Ansvar:**
Author	Representerer en forfatter med navn og e-post.
DiaryEntry	Representerer et dagbokinnlegg, med id, tittel, innhold, dato og forfatter.
DiaryEntryRegister	Lagrer og håndterer dagbokinnlegg, som søker, sletter og filtrer.
AuthorRegister	Lagrer forfatter og teller forfatter per innlegg.
DiaryUi	Brukergrensesnitt, som viser meny, leser input og styrer programflyten.
DiaryApplication	Startpunkt for programmet.
AuthorPostCount	Representerer antall innlegg per forfatter.

Tabell 5. Oversikt over klasser

4.2.2 En oversikt over alle biblioteker og API som er brukt i programmet

Bibliotek/API:	**Hvordan og hvorfor det er brukt i systemet:**
java.util.ArrayList	Lagring av forfatter og dagbokinnlegg. Arrayliste ble brukt for at innleggene skal bli vist i den rekkefølgen de er lagt inn. Det gir også enkel iterering og rask tilgang via indeks.
java.util.Scanner	Leser brukerinput i konsollen, fra tastaturet. Den er enkel å bruke og egner seg godt til konsollbaserte Java-programmer.
java.util.LocalDateTime	Håndterer dato og tid for dagbokinnlegg. Den gir enkel håndtering av tidspunkt uten behov for tidsoneinformasjon.
java.util.logging.Logger	Logging for informasjon og feilmeldinger. Den brukes til å logge hendelser i systemet, som feil og viktige operasjoner. Dette gjør det enklere å feilsøke og følge med på hva som skjer i programmet.

Tabell 6. Bibliotek/API

Utover dette er det ingen eksterne biblioteker som blir brukt, så systemet forholder seg til JDK standardbibliotek.

4.2.3 Sortering av dagbokinnlegg:

Det var et behov for at brukeren skulle få en sortert liste av dagbokinnlegg i konsollen.

Jeg satte dermed inn:

results.sort(Comparator.comparing(DiaryEntry::getCreatedAt).reversed());

Comparator brukes for å bestemme hvordan objekter skal sorteres. Her sammenligner den dagbokinnlegg basert på datoen den ble opprettet, ved å sette getCreatedAt. Ved å bruke reversed() sorterer den fra nyest til eldst ved å snu rekkefølgen fra hve den vanligvis gjør fra eldst til nyest. Koden ble laget i samarbeid med en venn.

4.2.4 UML-diagram

UML-diagrammer brukt i prosjektet viser blant annet:

- **klassestruktur og relasjoner**,
- **flyten gjennom programmet fra start til stopp**,
- **samhandling ved opprettelse av nye innlegg**.

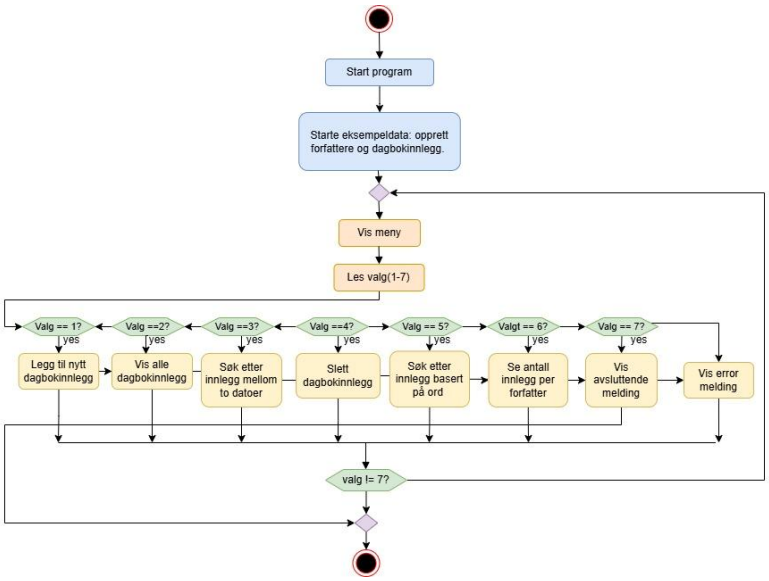
Diagrammene fungerte som støtte i designfasen og bidro til å holde strukturen klar, men gjentas ikke i detalj her for å unngå dobbel forklaring.

Klassediagram:



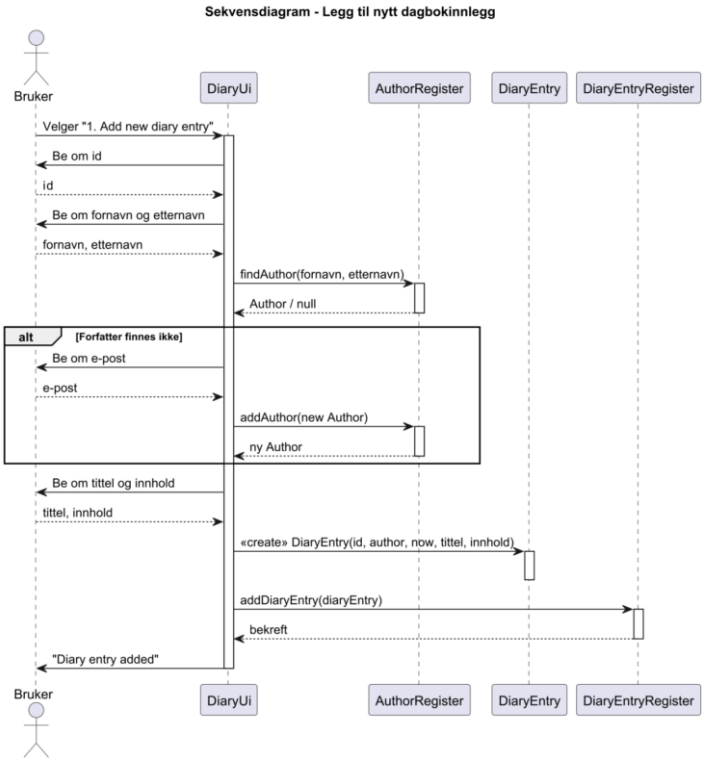
Figur 2. Klassediagram som viser oversikt over klassene som er brukt i programmet.

Aktivitetsdiagram:



Figur 3. Aktivitetsdiagram som viser funksjonaliteten

Sekvensdiagram:



Figur 4. Sekvensdiagram som viser interaksjon mellom bruker og klasser.

4.3 Testing

4.3.1 Testklasser

Teststrategien var å fokusere på domenelogikken og sikre at registerklassene håndterte lagring, søk og validering korrekt. Det ble skrevet enhetstester for alle klasser med domenelogikk, og testene dekket både normale tilfeller, grensetilfeller og feilinput. Brukergrensesnittet ble testet manuelt gjennom brukertesting.

Testklasser:	**Beskrivelse:**	**Strategi:**
DiaryEntryTest	Tester at et dagbokinnlegg blir opprettet riktig. Tester også setter-metoder og validering av tekstfelt.	Testet både gyldige og ugyldige verdier. Testet at IllegalArgumentException kastes ved feil input. Verifiserte at get-metoder returnerer riktig data.
DiaryEntryRegisterTest	Tester lagring av dagbokinnlegg, søk mellom datoer, søk på nøkkelord og sletting av innlegg.	Opprettet flere testinnlegg og sjekket at registeret returnerer korrekt liste for ulike søkescenarier. Testet slettefunksjon og at elementer faktisk fjernes. Testet både funn og ikke-funn.
AuthorTest	Tester opprettelse av forfatter, validering av fornavn, etternavn og e-post, samt at get-metodene fungerer som forventet.	Testet gyldige og ugyldige verdier, inkludert null, blanke strenger og ugyldige e-poster. Sjekket at IllegalArgumentException kastes ved feileverdier.
AuthorRegisterTest	Tester lagring av forfattere, søk på navn og telling av antall innlegg per forfatter.	Tester lagring av forfattere, søk på navn og telling av antall innlegg per forfatter.

Tabell 7. Testklasser

4.3.2 Brukertesting

Programmet ble brukertestet av 2 medstudenter. De fikk i oppgave å bruke menyen og utføre vanlige handlinger som å registrere, søke og slette innlegg. Testingen avdekket problemer med inputvalidering og uklare feilmeldinger, som ble justert etter tilbakemeldingene.

4.4 Utrulling til sluttbruker (deployment)

Dagbokinnlegget kjøres direkte fra en versjonskontroll som støtter Java-applikasjoner. Systemet er en enkel konsollbasert Java-applikasjon som ikke inkluderer eksterne grafiske komponenter slik at det ikke er noe behov for webserver, installasjon eller pakkefiler. Brukeren kan dermed starte applikasjonen ved å kjøre main-metoden i klassen DiaryApplicaton, som starter systemet og viser menyen i terminalen.

5 DRØFTING

5.1 Drøfting av løsning/design

I henhold til kravspesifikasjonen har jeg implementert funksjoner for å registrere, søke, telle og slette dagbokinnlegg. Jeg valgte nivå 2, som betyr at brukeren også kan registrere forfattere og søke i forfatterlister. Brukeren kan søke etter innlegg basert på dato eller nøkkelord, og antall innlegg per forfatter vises i en egen tabell.

Logging er implementert med *java.util.logging* for å vise informasjon, advarsler og feil. Checkstyle anbefalte bruk av logger fremfor System.out.println, noe som ga bedre kontroll og et mer profesjonelt system. Inputvalidering håndteres med try/catch og IllegalArgumentException for blant annet id, dato og tekstfelter. Likevel kunne valideringen vært bedre, for eksempel ved å hindre tomme strenger, sjekke gyldig e-post eller unngå duplikate og negative id-verdier. Systemet lagrer heller ikke data til fil eller database, slik at all informasjon forsvinner når programmet avsluttes. Loggingen er konfigurert gjennom *logging.properties*-filen for å gi en mer strukturert output.

Flere av domeneklassene, som *DiaryEntry* og *Author*, har høy kohesjon og følger enkeltansvarsprinsippet (SRP) godt ved å kun representere én type informasjon. *AuthorRegister* har også et tydelig ansvar knyttet til forfatterlogikk. Registerklassene bryter derimot SRP noe ved å inneholde utskrift, som egentlig burde håndteres av brukergrensesnittet. Dette ville gjort dem enklere å teste og mer fleksible.

Når det gjelder separasjon av bekymringer, er lagdelingen delvis god, men *DiaryUi* får for mye ansvar siden den håndterer både brukerinteraksjon, validering og kontrollflyt. Dette svekker modulariteten og gjør det vanskeligere å utvide systemet. En tydeligere lagdeling eller flere hjelpeklasser ville gitt bedre struktur.

UML-diagrammene var nyttige i planleggingsfasen for å få oversikt over ansvar og relasjoner, men designet utviklet seg underveis og diagrammene ble derfor ikke alltid fullt representative. Dette viser at UML fungerer best som planleggingsverktøy og ikke nødvendigvis som en endelig modell. Selv om designet gir et greit grunnlag for videreutvikling, må ansvarsfordelingen forbedres dersom systemet skal utvides med fil-lagring, database eller grafisk brukergrensesnitt.

5.2 Drøfting av prosess

Underveis i utviklingen fulgte jeg i stor grad den iterative og inkrementelle metoden fra metodedelen, men måtte tilpasse prosessen da oppgaven ble publisert i tre deler. Jeg startet som planlagt med del 1, men valgte å jobbe parallelt med del 2 og 3 senere. Dette gjorde arbeidet mer fleksibelt, men førte også til at UML-planleggingen ble vanskeligere å holde oppdatert.

Det som fungerte best var jevnlig testing og hyppige Git-commits, som ga god kontroll på versjoner og gjorde feilretting enklere. Verktøy som Checkstyle, SonarLint og logger bidro til kvalitetssikring gjennom hele prosessen. Både medstudenter og ChatGPT fungerte som gode sparringspartnere, spesielt ved feilsøking og designvurderinger, selv om slike diskusjoner noen ganger brøt opp arbeidsflyten.

Det som fungerte mindre godt var strukturen i DiaryUi, som burde vært delt opp i mindre hjelpeklasser for bedre oversikt og vedlikehold. UML-diagrammene burde også vært oppdatert oftere når nye krav kom. Inputvalideringen ble prioritert sent, noe som førte til ekstra arbeid i ettertid.

I fremtidige prosjekter vil jeg holde meg til engelsk språk i koden fra start for å unngå omfattende refaktorering. Jeg vil også oppdatere dokumentasjon og UML mer fortløpende og jobbe enda mer iterativt mellom deloppgaver for å få bedre struktur og flyt.

5.3 Drøfting av bruken av KI-verktøy

I denne oppgaven brukte jeg ChatGPT som sparringspartner, først og fremst for å få forklaringer på konsepter og forslag til hvordan jeg kunne strukturere deler av koden, men også til å forstå hvordan ulike Java-funksjoner burde brukes. KI var til god hjelp når jeg trengte å forstå deler av oppgaveteksten som for eksempel kravene til feilhåndtering.

Jeg fikk også enkelte kodeforslag, som datoformatering og en enkel kolonnebasert liste med forfattere og antall innlegg. Disse løsningene måtte jeg tilpasse og kvalitetssikre selv for å fungere i prosjektet.

Selv om jeg fikk god hjelp, oppdaget jeg at KI kunne ha manglende forståelse for hele prosjektet da den ikke hadde hele informasjonen, og at jeg kun ba om hjelp til deler av koden. Dermed kunne den forslå kode som ikke passet eller krevde omskriving. Den kunne også gi forslag med kode som så riktig ut, men som innehold logiske feil. Jeg måtte derfor være svært kritisk og teste all KI-generert innhold.

KI-verktøyet var derfor nyttig i form av ideutvikling og som læringsstøtte, men alle løsninger måtte tilpasses før bruk i koden.

5.4 Konklusjon – erfaring

Applikasjonen oppfyller hovedpunktene i kravspesifikasjonen gjennom funksjonalitet for registrering, visning, søk og sletting av dagbokinnlegg, i tillegg til håndtering av forfattere og telling av deres innlegg. Systemet fungerer som forventet og dekker de sentrale målene for oppgaven.

Dersom jeg skulle startet prosjektet på nytt, ville jeg planlagt lagdelingen av klassene tydeligere, spesielt med tanke på å skille brukergrensesnitt, logikk og validering. Jeg ville også oppdatert UML-diagrammene fortløpende og prioritert inputvalidering tidligere for å redusere behovet for refaktorering senere i prosessen.

Siden systemet er konsollbasert og lagrer data kun i minnet, har det naturlige begrensninger både i funksjonalitet og brukervennlighet. Strukturen i enkelte klasser gjør også videre utvidelser mer krevende enn nødvendig.

Fremtidig arbeid kan derfor fokusere på fil- eller databaselagring, en tydeligere moduldeling og et grafisk brukergrensesnitt. I tillegg vil forbedret validering og mer konsekvent bruk av logging styrke robustheten og kvaliteten i løsningen.

REFERANSER

[1] GeeksforGeeks, «Coupling and Cohesion - Software Engineering,» GeeksforGeeks, 24 04 2025. [Internett]. Available: https://www.geeksforgeeks.org/software-engineering/software-engineering-coupling-and-cohesion/. [Funnet 26 11 2025].

[2] GeeksforGeeks, «Single Responsibility in SOLID Design Principle,» GeeksforGeeks, 23 07 2025. [Internett]. Available: https://www.geeksforgeeks.org/system-design/single-responsibility-in-solid-design-principle/. [Funnet 28 11 2025].

[3] GeekforGeeks, «Separation of Concerns (SoC),» GeekforGeeks, 13 02 2024. [Internett]. Available: https://www.geeksforgeeks.org/software-engineering/separation-of-concerns-soc/. [Funnet 28 11 2025].

[4] GeekforGeeks, «Unified Modeling Language (UML) Diagrams,» GeeksForGeeks, 28 08 2025. [Internett]. Available: https://www.geeksforgeeks.org/system-design/unified-modeling-language-uml-introduction/. [Funnet 28 11 2025].