

# Eksamen, Software Design H2021

*Christian Gregersen, Christoffer Østborg,  
Harry Liam, Roman Morso, Marie Stigen*

<b>Introduksjon</b>	<b>3</b>
Hvordan programmet fungerer	3
Hvordan programmet kjøres	4
<b>Prosess</b>	<b>5</b>
Idé, fase 1	5
Idé, fase 2	5
(prosjekt 1) Semestertracker	5
(prosjekt 2) Organisering	7
Diagrammer	7
Class Diagram	7
Sequence Diagram	8
ER Diagram	9
Features	10
Utvikling	11
<b>Pensum</b>	<b>14</b>
SOLID-prinsippene	14
UNIT-testing	15
Multithreading	15
Design Patterns	15
GIT, Parprogrammering & Versjonskontroll	16
Github Actions & Continuous integration	18
EF Core	18
Teknisk Gjeld & Refactoring	19
SQL Injection	21
Lagdeling	21
<b>Utfordringer</b>	<b>22</b>
EF Core	22
Scope	22
Deadline	22
Planlegging	22
<b>Features</b>	<b>23</b>
Automatisk oppdatering	23
Rekursjon	23
<b>Retrospektiv</b>	<b>23</b>
<b>Bugs</b>	<b>24</b>
<b>Kilder</b>	<b>24</b>

# Introduksjon

## Hvordan programmet fungerer

Hei og velkommen til Semestertracker™!

Dette programmet skal hjelpe deg med å holde oversikt over hvordan studiet ditt går så langt. Du vil få oversikt over progresjonen i de forskjellige emnene dine, og du kan markere oppgaver som gjort, og forelesninger som sett. Du vil ha full oversikt over hvilke emner du kanskje ligger litt bak i, og hvilke emner som er ferdige!

Hvordan bruker du programmet, tenker du kanskje - ikke fortvil!

Du logger inn med ditt egen epost og passord, pass på å ikke dele det med noen andre.

**Epost:** [kim@bruun.no](mailto:kim@bruun.no)

**Passord:** daarligpassord

Når du har logget inn har du oversikt over semesterene dine. Du går inn i et semester ved å trykke på tallet som hører til - hvis du trykker på 1, så vil du komme til første semester. Når du har kommet inn i semesteret, vil du se en oversikt over hvordan semesteret går så langt. # vil si at du har gjort oppgaver og sett forelesninger, = vil si at det er publisert, og - vil si at det ikke er lagt ut enda. Du kan også se status på hvert fag og semester i menyen, sånn at du ser hvor du bør legge litt ekstra innsats!

For å gå tilbake trykker du på **0**, og du kommer da til nivået over.

For å avslutte programmet, trykker du på **E**, deretter på **J**

For å sette en oppgave til gjort, et emne til bestått, eller en forelesning til sett, trykker du på **F**

# Hvordan programmet kjøres

## For å migrere EFCore må disse kommandoene kjøres fra terminalen

```
> dotnet tool install --global dotnet-ef
```

```
> dotnet ef database update
```

## Hvordan kjøre programmet

- Start programmet i visual studio eller standalone
- Logg inn med brukeren [kim@bruun.no](mailto:kim@bruun.no)
- Benytt passordet `daarlignpassord`

## NB:

Filene `tracker.db` og `progression-tracker-log.txt` blir lagret i

`C:/Users/<BRUKER>/AppData/Local/` (Windows)

# Prosess

## Idé, fase 1

Alle på gruppa skulle komme på 1-2 ideer hver, som de kunne tenke seg å jobbe med under eksamen. Vi satte deretter et ❤️ hver bak 1-2 oppgaver vi likte best. Oppgavene vi kom på var:

- **Gjøre om noen enkle brettspill/kortspill til PC, f.eks Exploding Kittens eller Sushi Go** ❤️
- **Tracking av skoleoppgaver / semester** ❤️❤️❤️
- Oppskrifter og allergener i mat - hver sin profil med allergier der man kan invitere andre brukere for å se hva man må ta hensyn til når det kommer til matlaging
- **NAV saksbehandlingssystem (hovedfokus på design patterns og brukerhåndtering i grupper)** ❤️
- Socketprogrammering - f.eks stein saks papir
- Rasevelger for hund - hva passer deg best basert på en undersøkelse
- Hva skal jeg lage til middag basert på det jeg har i kjøleskapet?
- **Prosjekt Tracker for gruppeprosjekt med steg. "Steg 1. Alle skriver ned ting de har lyst til å gjøre, alle stemmer på en ting hver, disse går videre til neste.."** ❤️
- Mario Party-opplegg med minigames/små programmer
- **Organisering av stasj. Kan ha forskjellige kategorier på hva det skal være og hvordan det fungerer.** ❤️❤️

## Idé, fase 2

De to ideene som fikk flest stemmer går videre til fase 2. Disse skal man raskt sette opp en beskrivelse for å få en oversikt over scopet og implementering av oppgavene.

### (prosjekt 1) Semestertracker

#### **Vi endte med å gå for denne idéen**

Det lages en oversikt over semesteret. Det er flere nivåer som benyttes. Man ser status over fagene i semesteret og hvordan det ligger an i forhold til semesterplanen. F.eks at et emne har hatt 6/12 forelesninger og er 50% ferdig.

Man kan deretter gå inn på et emne og se hvordan du ligger an i forhold til emnet. Kanskje du har sett 4/6 forelesninger og er 66% up to speed, men du har kanskje bare løst 40% av oppgavene som har kommet ut til nå.

Videre kan du gå inn i oppgavelista, og se hvilke oppgaver som er ute og hva du har gjort. Du får oversikt over hvor langt du har kommet på de forskjellige oppgavene og hvor mye du har igjen.

Du får også oversikt over en timeline med arbeidskrav og eksamen

-----august----- -----september----- -----oktober----- -----november-----	
=====X-----	
=====X-----	PGR203 - forelesning
=====X-----	PGR203 - sett
====X-----	PGR203 - oppg
=====X-----	PG4200 - forelesning
=====X-----	PG4200 - sett
=====X-----	PG4200 - oppg
=====X-----	PG3302 - forelesning
=====X-----	PG3302 - sett
====X-----	PG3302 - oppg

### MVP

- Trenger ikke nødvendigvis under-oppgaver på en oppgave
- Basic UI
- Konsoll
  - Farger / differansiering av progresjon i hele emnet, og progresjon i emnet hittil
- Kun studentperspektiv
- All avhengig data (forelesninger, emner under studieprogram etc) som skal registreres av forelesning/administrasjon, vil hardkodes inn i databasen via en config-fi e.l.
- Se oppgaver tilhørende emner
- se forelesninger og oppgaver
- Oppdatere egen progresjon i forelesninger/oppgaver
- Lagre endringer gjort i applikasjonen

### Nice to have

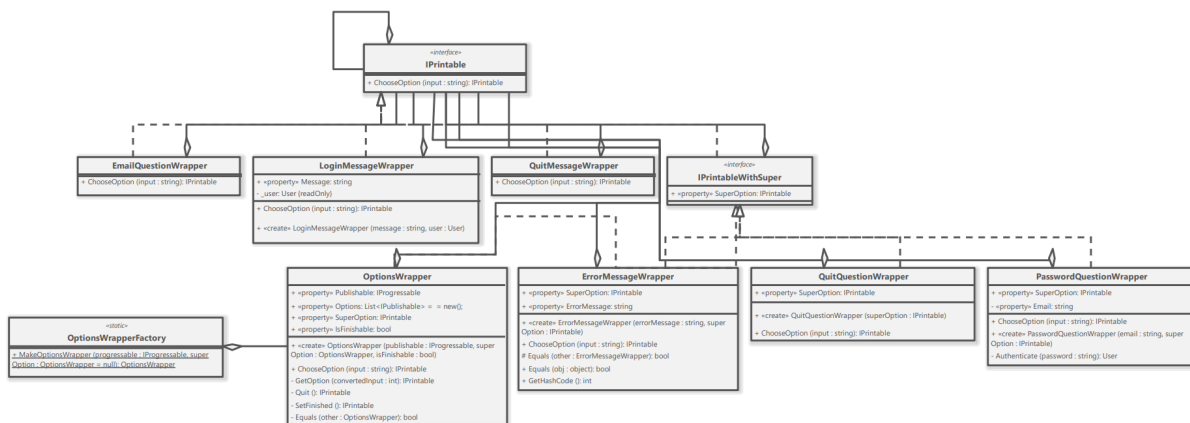
- Legge til egne oppgaver i et emne
- Opprette konto
- Foreleser og/eller admin-bruker
- Velge valgemne i semesteret det er relevant
- Mulighet til å bytte studieprogram, men beholde emnene man har bestått
- Mer enn ett studieprogram
- Automatisk oppmeldt i obligatoriske emner i registrert studieprogram
- Flere visninger
- Arbeidskrav/obligatoriske oppgaver

## (prosjekt 2) Organisering

En side du kan velge å organisere forskjellige collections av ting. Det kommer til å være noen kategorier som vil gi deg et valg av måter å sortere ting, og kanskje også gi noen tips basert på ting du har i kolleksjon din. Dette kan f.eks være at du har en platesamling og du ønsker å få tips basert på sjanger, eller om du har oppskrifter der du vil ha de som inneholder løk.

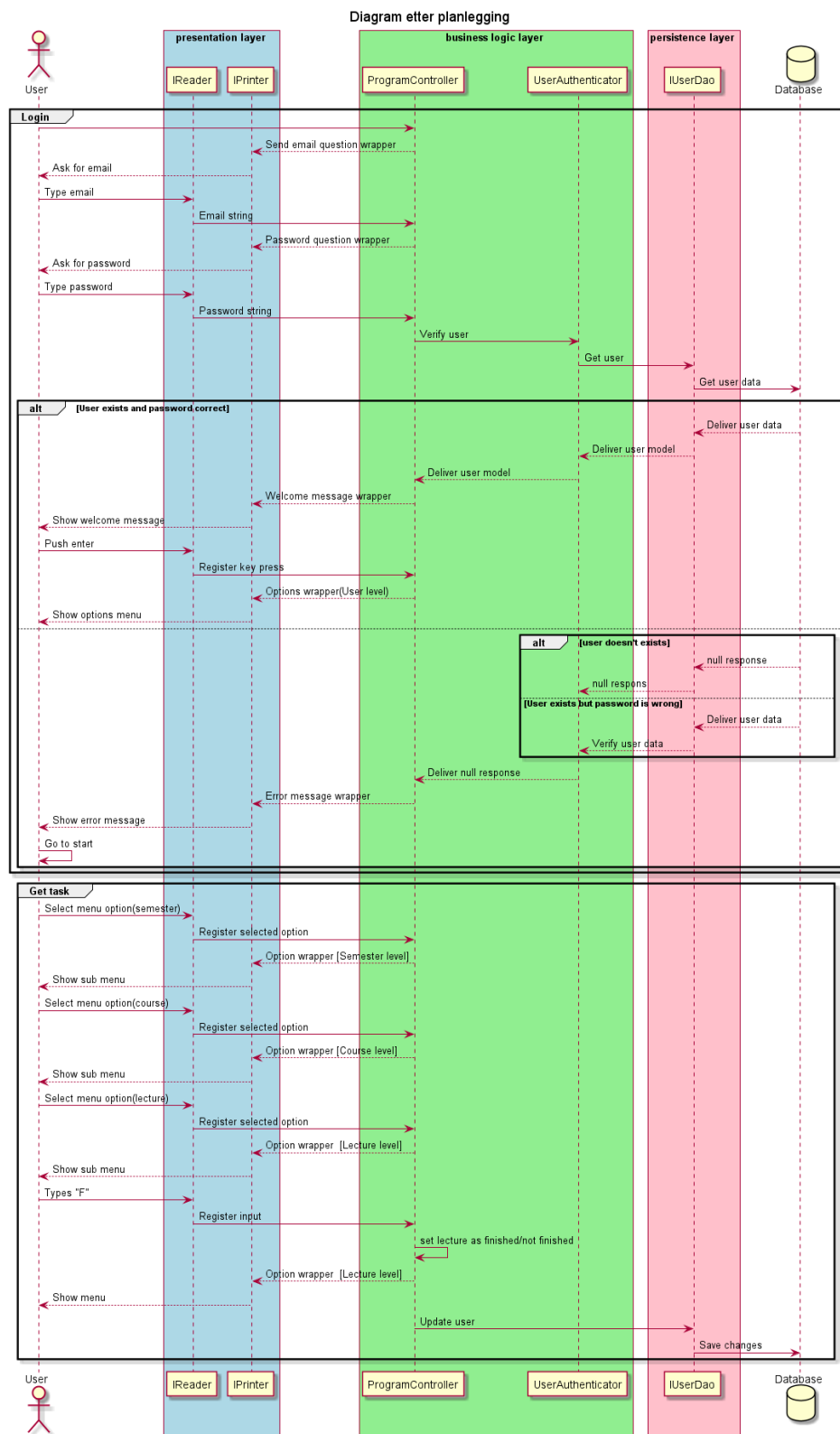
## Diagrammer

### Class Diagram



UML-diagrammet ble såpass stort at vi ikke får plass i dette dokumentet - for det fulle diagrammet, se documentation > UML-diagram, eller README-filen

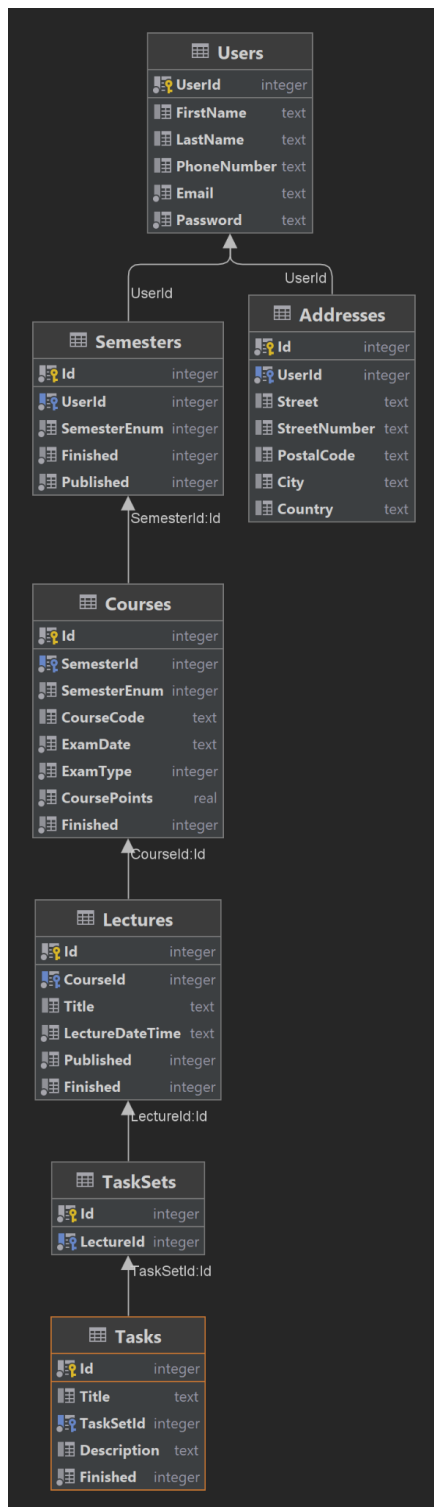
# Sequence Diagram



(Du finner større versjon i "documentation"-folderen, og README-filen)



# ER Diagram



(Du finner større versjon i “documentation”-folderen, og i README-filen)

# Features

Prios:

**MVP** - *must be in Minimal Viable Product*

**NTH** Nice to have - *features som ville gjort ting bedre, men som ikke er helt nødvendige*

Feature	Description	Prio
Se oversikt over semester	En bruker skal kunne se oversikt og status for semesteret de er i	MVP
Se oversikt over fag	En bruker skal kunne velge et fag for å se oversikt over status	MVP
Se oversikt over studieløp	En bruker skal kunne se oversikt over hele studiet - både fullført og ikke startet på	NTH
Gå inn på en forelesning	En bruker skal kunne gå inn på en spesifikk dag/forelesning i et fag	MVP
Gå inn på oppgaver i forelesning	En bruker skal kunne gå inn på oppgavesettet fra forelesningen	MVP
Sette en forelesning til sett/ikke sett og en oppgave til gjort/ikke gjort	En bruker skal kunne si at en forelesning er sett, eller en oppgave er gjort	MVP
Velge valgemne	En bruker skal kunne velge / få opp valgemne i tillegg til fastsatte fag	NTH
Legge til ekstra fag	En bruker skal kunne legge til fag utenfor studieretningen	NTH
Lage lærer-bruker	Lage en ny brukertype som kan publisere oppgaver og forelesninger, samt få oversikt over de som tar emner	NTH
Lage admin-bruker	Mulighet til å endre alle databaser, god-mode	NTH
Hente inn data fra API	Databasen genererer innhold fra API (canvas?)	NTH

## Utvikling

Klasse	Ansvarsområder	Kommentarer
IController	Skal brukes av kontrollere	
ProgramController	Holder hovedlogikken for å kjøre programmet. Her kjører program-loopen som viser menyer basert på hvor du er. I tillegg blir endringer lagret i databasen herfra.	Implementerer IController
IProgressionHandler	Interface for ProgressionHandlers, brukes til Composite Pattern, returnerer et ProgressionWrapper-objekt	
OptionsWrapper	Genererer menyalternativer utifra string-input. Implementerer IPrintable	
OptionsWrapperFactory	Genererer OptionsWrapper-objekter, og fungerer som implementasjon av Factory Pattern	
ProgressionBarHandler	Genererer en progresjons-bar av et ProgressionWrapper-objekt	
ProgressionHandlerComposite	Composite-delen av Composite Pattern-implementasjon av IProgressionHandler	
ProgressionHandlerFactory	Genererer objekter av klasser som implemeterer IProgressionHandler-interfacet. Fungerer som implementasjon av Factory Pattern.	Genererer alle Children for ProgressionHandler-Composite rekursivt
ProgressionHandlerLeaf	Leaf-delen av Composite Pattern-implementasjon av IProgressionHandler	Brukes på Course-, Lecture-, og Task-nivå, for å se om disse som bestått/sett/gjort

ProgressionWrapper	Holder på prosentverdier for om noe er publisert og ferdig	
Address	Et adresse-objekt, har informasjon om en adresse.	Blir ikke vist i programmet, men vil oppdateres i databasen ved registrering av brukeren.
Course	Et fag-objekt, har informasjon om et fag	
IFinishable	Sier om noe er ferdig eller ikke. Implementerer IPublishable	
IProgressable	Sier om noe kan ha en progresjon eller ikke	
IPublishable	Sier om noe er publisert eller ikke. Implementerer IProgressable	
Lecture	Et forelesnings-objekt, har informasjon om en forelesning	
Semester	Et semester-objekt, har informasjon om et semester	
SemesterEnum	En Semester-enum, har seks semestre (bachelor)	Kan enkelt utvides til 10 om man vil ha med master
Task	Et oppgave-objekt, har informasjon om en oppgave	
TaskSet	En oppgaveliste-objekt. Har flere oppgaver (task)	
User	En bruker (student), har informasjon om en bruker samt studiet brukeren går på	Dette er topp-noden i en trestruktur som holder på all informasjon nødvendig for programmet
ICrudDao	Levere en kontrakt om basisfunksjonalitet for DAO.	
AbstractDao	Implementerer ICrudDao	
IUserDao	Kontrakt om ekstra funksjonalitet i DAO til	

	User-modellen	
UserDao	Implementerer AbstractDao og IUserDao. Håndterer data-laget i applikasjonen	
TrackerContext	Nødvendig for opprettelse av tabeller med EF Core. Definerer også connection string og hvilken DBMS som benyttes	
ConfigFile	ConfigFile har metoder som kan brukes til å legge til start-data osv	
IPrintable	Interface for alt som skal kunne printes til konsoll	
IPrintableWithSuper	Interface for alt som har en overordnet IPrintable man skal kunne komme tilbake til. Implementerer IPrintable	
Program	Her ligger Main metoden. Har kun ansvar for å sjekke om databasen inneholder data og evt legge inn dummy data dersom den ikke har det. Deretter delegerer den arbeidet til ProgramController.	Tanken er at alt som skjer her skal enkelt kunne byttes ut med noe annet dersom applikasjonen for eksempel skal kjøres i web eller annet.
UserAuthenticator	Har ansvar for å hente ut bruker basert på epost og sjekker da om passord og epost stemmer overens. Returnerer et User-objekt dersom infoen stemmer.	Her bruker vi BCrypt hashing for å lagre passordet i databasen.
EmailQuestionWrapper	Ber bruker om epost	Implementerer IPrintable
ErrorMessageWrapper	Sender feilmelding	Implementerer IPrintableWithSuper
LoginMessageWrapper	Sender velkomstmelding ved login	Implementerer IPrintable
PasswordQuestionWrapper	Ber bruker om passord	Implementerer IPrintableWithSuper

QuitQuestionWrapper	Spør om bruker vil avslutte	Implementerer IPrintableWithSuper
QuitMessageWrapper	Sender avslutningsmelding ved avslutning	Implementerer IPrintable
IReader	Interface for brukerinput	
Reader	Leser input fra konsoll	Implementerer IReader

## Pensum

### SOLID-prinsippene

#### Single-responsibility principle

*Hver klasse har ett ansvarsområde, og det skal aldri være mer enn en grunn for en klasse å endre seg.*

I prosjektet har vi, så godt det lar seg gjøre, skrevet koden med Single Responsibility i tankene. Vi har laget hjelpeklasser, interfaces, enums, og andre klasser for å holde klassene så punktlig som mulig. Vi har også eksempler av factory-klasser, hvis eneste ansvar er å lage en instans av en annen klasse.

#### Open-closed principle

*Hver klasse skal kunne utvides, men ikke endres.*

Vi har i stor grad benyttet modularitet i prosjektet. Dette har vi blant annet gjort ved bruk av interfaces.

#### Liskov substitution principle

*En child-klasse skal kunne erstatte en parent-klasse*

C# oppfyller Liskov substitution principle så og si automatisk, så dette er ikke noe vi har tatt ekstra hensyn til.

### Interface segregation principle

*Et interface skal kun ha properties som er relevant for alle klasser som implementerer den.*

Vi har delt opp Dao-interface til flere mindre interfaces. Vi har et parent interface(ICrudDao) som har metodesignaturer som er relevant for alle Dao-klassene, og vi har et child interface for hver av Dao-klassene, med metodesignaturer som er spesifikk for klassen. På denne måte blir vi ikke tvunget til å implementere unødvendige metoder og det vil gjøre det enklere å vedlikeholde koden.

### Dependency inversion principle

*En klasse skal heller implementere en (eller flere) interfaces, enn å opprette nye objekter. Det gjør at en klasse får samme funksjonalitet uten å være bundet til et spesifikt objekt.*

Brukt mye i implementasjon av Composite Pattern, og klasser som gir menyalternativer som kan printes i konsoll (IPrintable)

## UNIT-testing

Vi har skrevet unit tester for enkelte klasser og prøvd oss litt på testdrevet utvikling (TDD). Det var nyttig å kunne teste at komponentene hadde den forventede atferden, og det hjalp oss å avdekke bugs, spesielt etter en refaktorering.

Men det tar tid å skrive tester, og det var tilfeller hvor vi måtte skrive om testene etter en refaktorering. Testene tester kun komponentene i isolasjon, som gjør at de nødvendigvis ikke avdekker bugs som oppstår når de er samspill med andre komponenter. Hvis vi skulle videreutvikle og vedlikeholde programmet, ville vi vurdert å skrive flere tester.

## Multithreading

Siden oppgaven vår tar utgangspunkt i én bruker, så er det ikke noe behov for multithreading, da forskjellige brukere ikke kan endre på eller kalle metoder samtidig. Vi har også planlagt projektet på en måte der det ikke er nødvendig, da ingen brukere skal ha mulighet til å endre andres data. Det er ingen ytelsesforbedringer i vårt tilfelle, men kan føre til mange unødvendige bugs. Vi har også frontend og backend i samme applikasjon.

## Design Patterns

### Facade

*Facade er et pattern som lager en enkel interface mot brukeren, som videre utfører brukerens ønske i koden. Dette gjør det lettere for en bruker å kjøre programmet - og programmet vet hva den skal kunne håndtere fra en brukerside.*

Siden oppgaven vi lager i stor grad består av et konsoll-UI som en bruker skal kunne navigere seg i, lønner det seg å benytte Facade som design pattern. Det gjør at vi kan gi brukeren tilgang på de tingene man skal ha tilgang på, og lage programmet på en måte som gjør det enkelt å svare på brukerens requester.

### Composite

*Composite er et pattern som lar kode kjøres på alle "children" av interfacet når koden executer. Dette gjør det lettere å execute kode på tvers av klasser og instanser.*

I oppgaven skal vi sjekke om forskjellige ting er gjort, eller ikke gjort - for så å returnere en totalverdi av mengden som er- eller ikke er fullført. Ved å bruke et composite-pattern kan vi lett finne alle klasser som har en gjort/ikke gjort funksjon, og telle det uten å kjøre koden flere steder.

### Factory

*Factory er et pattern som lager nye objekter når man kaller på en klasse, i stedet for å lage nye objekter selv.*

Vi har brukt Factory til å generere menyalternativer basert på hvilket menynivå du befinner deg i, og for å generere ProgressionHandlers rekursivt på det aktuelle nivået

### Singleton

*Singleton er et pattern som passer på at det kunn eksisterer én instans av en klasse på tvers av hele koden.*

I oppgaven ønsker vi å ha en instans som logger feilmeldinger eller meldinger til brukeren. Denne ønsker vi det bare skal eksistere en instans av som skal være tilgjengelig globalt. Vi lagde derfor denne klassen som en singleton.

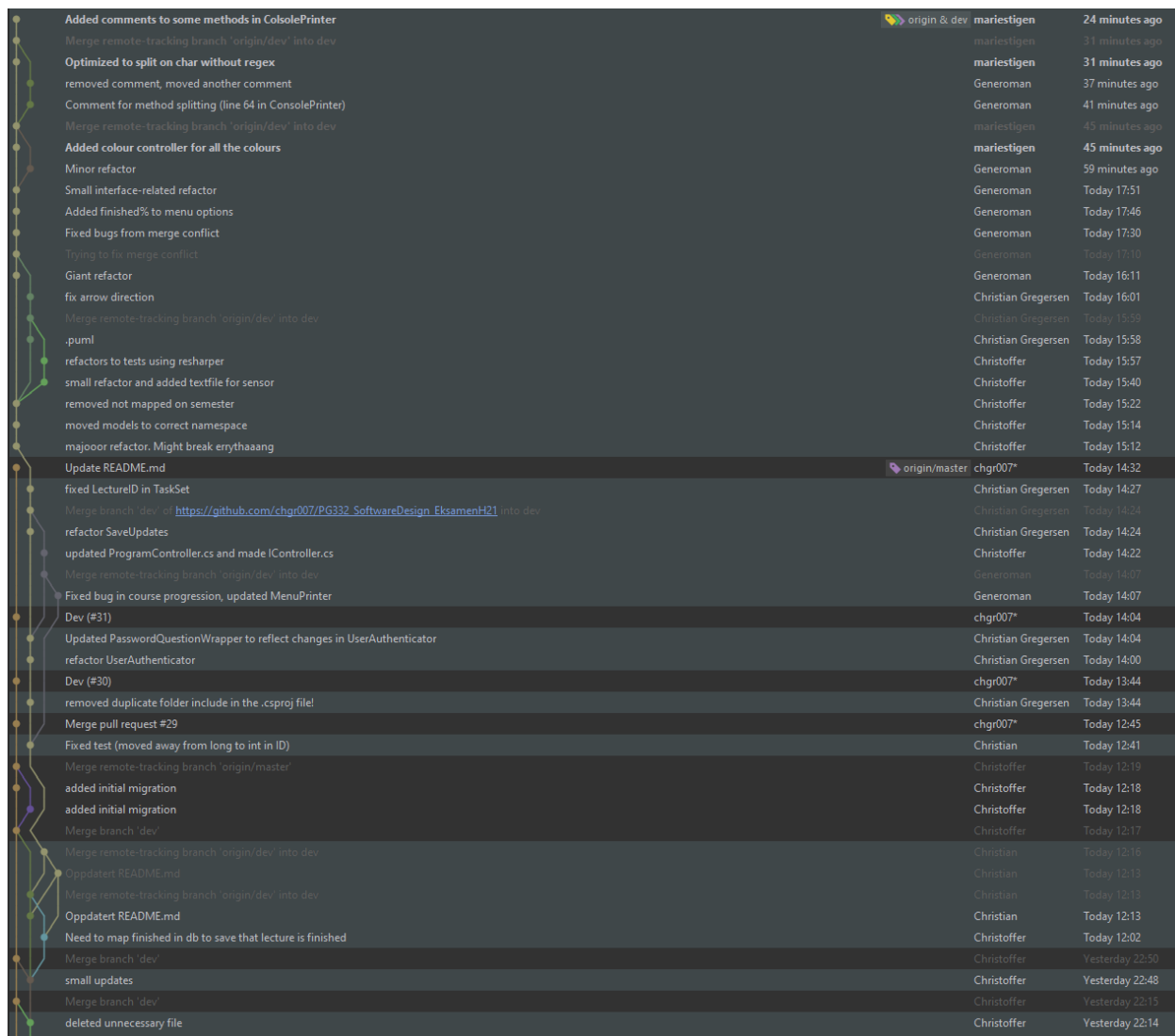
## GIT, Parprogrammering & Versjonskontroll

Vi valgte å bruke GIT som versjonshånderingsverktøy. Vi lagde et felles repo, der vi lagde en ny Dev-branch som vi jobbet mot. Hvis man skulle jobbe på en spesifikk oppgave uavhengig av



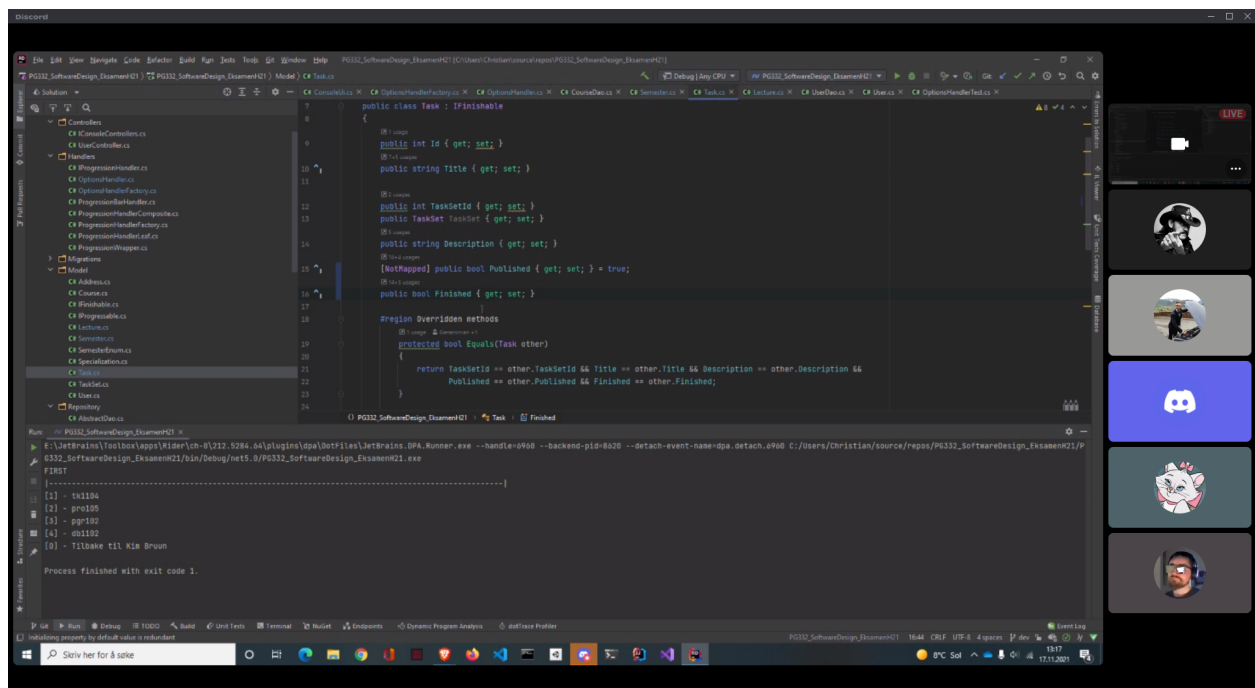
noen lagde vi av og til en egen branch for å passe på at seksjonen ble merget riktig inn i dev-branchen.

Det å bruke versjonshåndtering av oppgaven gjør at vi har mulighet til å aktivt teste nye ting uten å være redd for å ødelegge hele prosjektet, og alle har tilgang på nyeste oppdatering til en hver tid. Noen av problemene kan være at vi gjør veldig store commits som fører til store konflikter, eller at man glemmer å pushe kode som andre trenger senere.



Added comments to some methods in ConsolePrinter	origin & dev	mariestigen	24 minutes ago
Merge remote-tracking branch 'origin/dev' into dev		mariestigen	31 minutes ago
Optimized to split on char without regex		mariestigen	31 minutes ago
removed comment, moved another comment		Generoman	37 minutes ago
Comment for method splitting (line 64 in ConsolePrinter)		Generoman	41 minutes ago
Merge remote-tracking branch 'origin/dev' into dev		mariestigen	45 minutes ago
Added colour controller for all the colours		mariestigen	45 minutes ago
Minor refactor		Generoman	59 minutes ago
Small interface-related refactor		Generoman	Today 17:51
Added finished% to menu options		Generoman	Today 17:46
Fixed bugs from merge conflict		Generoman	Today 17:30
Trying to fix merge conflict		Generoman	Today 17:10
Giant refactor		Generoman	Today 16:11
fix arrow direction		Christian Gregersen	Today 16:01
Merge remote-tracking branch 'origin/dev' into dev		Christian Gregersen	Today 15:59
.puml		Christian Gregersen	Today 15:58
refactors to tests using resharper		Christoffer	Today 15:57
small refactor and added textfile for sensor		Christoffer	Today 15:40
removed not mapped on semester		Christoffer	Today 15:22
moved models to correct namespace		Christoffer	Today 15:14
majoor refactor. Might break erythaaang		Christoffer	Today 15:12
Update README.md	origin/master	chgr007*	Today 14:32
fixed LectureID in TaskSet		Christian Gregersen	Today 14:27
Merge branch 'dev' of <a href="https://github.com/chgr007/PG332-SoftwareDesign-EksamenH21">https://github.com/chgr007/PG332-SoftwareDesign-EksamenH21</a> into dev		Christian Gregersen	Today 14:24
refactor SaveUpdates		Christian Gregersen	Today 14:24
updated ProgramController.cs and made IController.cs		Christoffer	Today 14:22
Merge remote-tracking branch 'origin/dev' into dev		Generoman	Today 14:07
Fixed bug in course progression, updated MenuPrinter		Generoman	Today 14:07
Dev (#31)		chgr007*	Today 14:04
Updated PasswordQuestionWrapper to reflect changes in UserAuthenticator		Christian Gregersen	Today 14:04
refactor UserAuthenticator		Christian Gregersen	Today 14:00
Dev (#30)		chgr007*	Today 13:44
removed duplicate folder include in the .csproj file!		Christian Gregersen	Today 13:44
Merge pull request #29		chgr007*	Today 12:45
Fixed test (moved away from long to int in ID)		Christian	Today 12:41
Merge remote-tracking branch 'origin/master'		Christoffer	Today 12:19
added initial migration		Christoffer	Today 12:18
added initial migration		Christoffer	Today 12:18
Merge branch 'dev'		Christoffer	Today 12:17
Merge remote-tracking branch 'origin/dev' into dev		Christian	Today 12:16
Oppdatert README.md		Christian	Today 12:13
Merge remote-tracking branch 'origin/dev' into dev		Christian	Today 12:13
Oppdatert README.md		Christian	Today 12:13
Need to map finished in db to save that lecture is finished		Christoffer	Today 12:02
Merge branch 'dev'		Christoffer	Yesterday 22:50
small updates		Christoffer	Yesterday 22:48
Merge branch 'dev'		Christoffer	Yesterday 22:15
deleted unnecessary file		Christoffer	Yesterday 22:14

Vi satt ofte å programmerte sammen, der en delte skjerm på Discord, og vi hjalp hverandre med å løse problemer. Dette gjorde at vi hadde oversikt over prosjektet, og visste hva alle holdt på med. Vi hadde også fysiske oppmøter der vi jobbet sammen med å løse problemer eller logikk. Vi delte oss også opp i mindre grupper som satt og løste et problem sammen, og gikk deretter tilbake til de andre gruppemedlemmene med løsningsforslaget på problemet.



## Github Actions & Continuous integration

Vi prøvde å benytte Github Actions CI for å alltid ha status på tester i pull requests, og for å enkelt kunne se på github om noe var feil med bygging og tester. Dette viste seg imidlertid vanskelig å få til med EF Core og vår forhåndskunnskap. Selv om SQLite er dependency styrt, og ikke krever installasjon av DBMS på vertsmaskinen, måtte noen kommandoer kjøres i terminalen for å migrere EF Core. Dette fikk vi ikke til å fortelle Github Actions. Selv om vi syns dette er nyttig funksjonalitet å bruke i et prosjekt, valgte vi å ikke bruke mer tid på å finne ut av det, da det ligger utenfor pensum.

## EF Core

Vi har brukt EF Core (Entity Framework Core) for å forenkle arbeidet med å jobbe mot en database. Vi valgte en code-first approach som innebærer vi først kodet modell-klassene, og som på migrering automatisk blir opprettet som ferdige tabeller i databasen. Vi gikk for denne approachen, da vi tenkte vi har mer erfaring med programmering sammenlignet med databasemodellering, og at vi sannsynligvis kom til å gjøre mye endringer underveis.

## Teknisk Gjeld & Refactoring

Under hele prosjektet prøvde vi å tilrettelegge for ekstra funksjonalitet og forhindre at vi skapte unødvendig teknisk gjeld. Vi prøvde å lage en oversikt over “Must have”- og “Nice to have”-features - der vi gjorde “must have”s, tilrettela for å legge til “nice to have”, og generelt prøvde å bygge skalerbar kode for ting som dukker opp senere.

Vi har refactored jevnt gjennom prosjektet, og flyttet og endret navn på klasser der vi følte det ga mening, samt gjort generaliseringer for å unngå “DRY”.

### Eksempel: Utdrag av kode før refactoring:

```
public class UserDao : IUserDao
{

    public void Update(User m)
    {
        using TrackerContext trackerContext = new TrackerContext();
        trackerContext.Update(m);
        trackerContext.SaveChanges();
    }

    public void Save(User m)
    {
        using TrackerContext trackerContext = new TrackerContext();
        trackerContext.Add(m);
        trackerContext.SaveChanges();
    }

    public User RetrieveById(long id)
    {
        using TrackerContext trackerContext = new TrackerContext();
        return trackerContext.Find<User>(id);
    }
}
```

## Refactores og generaliseres i abstrakt klasse:

```
public abstract class AbstractDao<TModel> : ICrudDao<TModel> where TModel : class
{
    public void Update(TModel m)
    {
        using TrackerContext trackerContext = new TrackerContext();
        trackerContext.Update(m);
        trackerContext.SaveChanges();
    }

    public void Save(TModel m)
    {
        using TrackerContext trackerContext = new();
        trackerContext.Add(m);
        trackerContext.SaveChanges();
    }

    public TModel RetrieveById(long id)
    {
        using TrackerContext trackerContext = new();
        return RetrieveDbSet(trackerContext).Find(id);
    }

    protected abstract DbSet<TModel> RetrieveDbSet(TrackerContext trackerContext);

    public List<TModel> ListAll()
    {
        using TrackerContext trackerContext = new();
        return RetrieveDbSet(trackerContext).ToList();
    }
}
```

## Implementering i UserDao etter refactoring:

```
public class UserDao : AbstractDao<User>, IUserDao
{
    protected override DbSet<User> RetrieveDbSet(TrackerContext trackerContext)
    {
        return trackerContext.Users;
    }

    public User RetrieveByEmail(string email)
    {
        return RetrieveOneByField(u => u.Email == email);
    }

    public User RetrieveByLastName(string lastName)
    {
        return RetrieveOneByField(u => u.LastName == lastName);
    }
}
```

## SQL Injection

Vi har gjennom bruk av EF Core rammeverket sikret oss mot SQL Injections. Dersom vi hadde gjort SQL-spørringene direkte i programmet uten EF Core, hadde vi brukt statement parameter, slik at input-strenger ikke settes direkte inn i spørringen.

## Lagdeling

UI / Presentasjonslaget
<i>UI-laget er det laget brukeren ser og interagerer med.</i>
Konsollen tar seg av det brukeren kan se, og lar brukeren skrive inn input. Reader leser input, og ConsolePrinter printer til konsoll.

Business
<i>Business-laget er laget som videresender data til riktig plattform (f.eks konsoll, plattformspesifikke applikasjoner o.l.)</i>
Program-klassen tar seg av hovedarbeidet her. Her genereres progresjonsinformasjon utifra data som hentes i persistence-laget, samt menyalternativer utifra hva brukeren gjør.

Persistence-laget
<i>Laget som håndterer DAOs og API'er</i>
UserDao ligger her. Vi benytter oss ikke av APIs.

Databaselaget
<i>Det tekniske laget er laget som tar seg av lagring og håndtering av informasjon.</i>
Her har vi brukt SQLite som databasesystem, som UserDao henter informasjon fra, og lagrer informasjon i.

# Utfordringer

## EF Core

En utfordring som meldte seg ved å benytte EF Core og et helt nytt rammeverk i et slikt prosjekt, var å få til koblingsentiteter i databasen representert i code first objekter på en ryddig måte. Da vi gikk vekk fra flere brukere i MVP, var EF Core stort sett en drøm å jobbe med.

## Scope

Vi valgte tidlig å bygge et prosjekt som var enkelt å skalere - hovedsaklig fordi vi visste det er lett å overvurdere hva man rekker å gjøre på en eksamensoppgave. Vi endte med å nedskalere prosjektet, da vi mener det er viktigere i dette emnet å levere kode med noe mindre funksjonalitet som er godt planlagt og uttenkt, i forhold til å “hacke” sammen et program med mye funksjonalitet, men mye teknisk gjeld. Vi brukte veldig mye tid på planlegging og diskusjoner om designatterns, diagrammer, og features, og koden er satt opp på en måte der det er lett å videreutvikle prosjektet til standarden vi ønsket.

## Deadline

Parallelt med eksamen i Software Design har det pågått gruppeeksamen i Avansert Java. Dette har gjort at det har vært utfordrende å vite hva man skal prioritere - og hvor mye tid man bør sette av til hver eksamen. Med tanke på at Avansert Java utføres i andre grupper enn Software Design har det også vært utfordrende å finne tid som passer for alle, samtidig som det har vært vanskelig for alle å bidra like mye i hele prosessen.

En del av gruppa har hatt C# i Unity utvikling tidligere, mens for andre var det et helt nytt språk. Da vi måtte kode C# med .NET samtidig som vi hadde Java-eksamen der vi også måtte benytte JavaScript, HTML og CSS, ble det fort kluss i kodespråkene. Det var også litt komplisert å sette seg inn i Visual Studio, og mange endte opp med å bruke Rider i stedet for, da det ligger nærmere IDE-et IntelliJ som vi alle er godt kjent med.

## Planlegging

Det å vite hvor mye tid man skal sette av til planlegging har også vært en utfordring. Når eksamen går ut på å planlegge og gjennomføre et prosjekt, men mye av pensum ligger på hvordan man planlegger prosjektet - er det lett å bli så opphengt i alle små detaljer at man ikke får nok tid til selve utviklingen. Det var utfordrende å vite når det var nok planlegging, og når man kanskje kunne brukt litt mer.

# Features

## Automatisk oppdatering

Databasen og koden oppdateres automatisk og real time når brukeren gjør endringer. Databasen oppdateres også hver gang når brukeren går inn i en ny meny - dette kunne vi endret til å kun oppdateres om det har vært gjort endringer, eller oppdatere når det avsluttes, men i dette prosjektet ville det nok brukt mer ressurser enn å oppdatere det hver gang.

## Rekursjon

Informasjon om progresjon blir generert rekursivt av ProgressionHandlerFactory. Dette vil si at for hvert semester en bruker har, blir det generert informasjon om progresjon. Denne progresjonsprosenten blir delt på antall semestre. På semesternivå skjer det samme for hvert emne, og hver forelesning for hvert emne, etc.

## Retrospektiv

Alt i alt er vi veldig fornøyd med eksamensukene. Det var noen ganger det gikk litt fort i svingene på refakturering, og det kunne være vanskelig å henge med på alt som skjedde, men alt i alt fikk vi til det vi ønsket av en MVP for prosjektet. Vi ser i ettertid at vi har vært litt dårlige på kommentarer og regions i koden, men føler klassenavnene og metodene er såpass små at de er relativt selvforklarende. Vi skrev også informasjon om ansvarsområdene til klassene i tabellen "klasser" sånn at vi hadde oversikt over hva som skjedde hvor.

Vi burde nok ha brukt mer tid sammen i starten for å lage det fundamentale i programmet, sånn at alle lettere kunne "slenge seg på" i oppgavene. Vi kunne også vært flinkere til å dele ut oppgaver og jobbe opp mot kanban eller lignende verktøy, samt kommunisere klarere hva man jobber med. Det var også utfordrende, men gøy, at alle hadde forskjellige måter man liker å jobbe med prosjekter og oppgaver på.

Vi har lært mye under eksamen, men føler det hjalp oss å sette Software Design som fag, inn i praktisk bruk på en god måte. Alt i alt synes vi det har vært en spennende, litt stressende, men morsom prosjekt.

# Bugs

Det er ingen kjente bugs i koden. Programmet er veldig kompakt, og ved bruk av facade så har brukeren lite tilgang på det som skjer “behind the scenes”.

# Kilder

<https://refactoring.guru/design-patterns>

Singleton:

[Singleton Design Pattern In C# \(c-sharpcorner.com\)](#)

*EFCore:*

<https://docs.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>