# **Eksamen A**

**i fagene**

**Programmering & Teknik**

**2020 januar 09.00 - 14.00**  
**Let tilpasset til prøve-eksamen, foråret 2020**

Alle skriftlige materialer, pc'er, bærbare computere og internet ressourcer er tilladt til eksamen.  
  
Mobiltelefoner og kommunikation med andre personer (inkl. chatbots), bortset fra kommunikation med eksaminator og censor, er forbudt.

Du må ikke benytte/gemme din løsning på eksterne netværksdrev/værter som Github, Facebook o. lign. Overtrædelse af dette vil medføre bortvisning fra eksamen og passende sanktioner vil senere blive tildelt både til afsender/uploader og modtager.

Ved eksamens afslutning skal du lægge din løsning op i Wiseflow.

Eksamen varer 4 timer og efterfølges af 1 times evaluering. I disse tidsrum kan den studerende kun forlade eksamenslokalet med henblik på toilet besøg. Rygning er ikke tilladt.   
  
Eksamenssættet består af 8 opgaver.  
Ud over disse opgaver, kan du kan blive bedt at besvare nogle ekstra spørgsmål omkring dine svar og eventuelle andre valg.

**Kontroller at dette sæt indeholder 6 sider inklusive forsiden.**

**Kørselsregnskab**

I denne opgave skal du arbejde med Library, Test, TCP, REST, Azure, DB og SSL.

*Udover at programmere dette system, er der også teoretiske spørgsmål undervejs, hvor du bliver bedt om at redegøre for overvejelser og nedskrive forklaringer. Dette kan du med fordel samle i et pdf-dokument, som uploades til Wiseflow sammen med dine programmer i en zip fil.*

**Domæne beskrivelse**

Transportfirmaet FastFup har en række nye og ældre lastbiler. FastFup er med på den grønne bølge og er begyndt at blive mere miljøbevidste. De har derfor besluttet at følge brændstofforbruget for hver transportdag ift. hver chauffør og lastbil for at se om kørselsmønsteret skal ændres og/el. lastbilen skal udskiftes. *Hver chauffør har netop én lastbil på én dag.*

Mere specifikt skal du programmere et system, der kan registre lastbilens gennemsnitsforbrug af brændstof målt som antal liter per kørt 100 km.

Oversigten kunne f.eks. fremgå af nedenstående informationer:

|  |  |  |  |
| --- | --- | --- | --- |
| Lastbil | Dato yyyy-mm-dd | Antal km. per dag | Gennemsnit liter per 100 km |
| A | 2019-12-01 | 1500 | 7.05 |
| B | 2019-12-01 | 1050 | 8.15 |
| C | 2019-12-01 | 976 | 7.92 |
| D | 2019-12-01 | 1278 | 6.34 |
| E | 2019-12-01 | 778 | 8.23 |
| A | 2019-12-05 | 700 | 7.11 |
| B | 2019-12-05 | 1104 | 8.34 |

*Der er selvfølgelig flere lastbiler og mange flere datoer, men i denne opgave ser vi kun på 5 lastbiler*

I tabellen er vist det samlede antal km per dag for alle transporter. Hvis en lastbil har haft flere forskellige transporter på samme dag, skal det samlede antal km derfor kunne udregnes før det gemmes i tabellen.  
Summen af 5 transporter (km1, km2 …. km5) udregnes ved følgende formel:

total = km1 + km2 + ……. + km5

Hvis der f.eks. er tre transporter på hhv. 155, 45 og 150 km er:

total = 155+245+150= 450 km

Brændstofforbruget udregnes ved følgende formel:

gennemsnit = (antalLiter/antalKm) x 100

F.eks. for 300 km og et forbrug på 22.5 liter er

gennemsnit = (22.7 / 300) x 100 = 7.67 (afrundet)

For en transport for én lastbil på samme dag gælder: at: 0 < antalKm <= 2000. (kører også i Tyskland!)

**Opgave 1: Class Library**

Opret et nyt projekt, FupLib af typen ”Class Library (.Net core)” med en klasse TransportCalc

1. Opret flg. *static* metoder i TransportCalc:

* double Average(int antalKm, double antalLiter)  
  Returnerer gennemsnittet ud fra antalKm og antalLiter iht. ovenstående formel.
* int Total(string transporter)  
  Strengen, *transporter*, indeholder to eller flere heltal (kilometer) adskilt af mellemrum (*space*).  
  Metoden opdeler først transporter i de enkelte heltal-strings, som så konverteres til heltalsværdier, og derefter returneres summen af tallene.

1. Giv beskrivende kommentarer til dine metoder.
2. Generer en DLL-fil.

**Opgave 2: Test dit Class Library**

Tilføj et testprojekt, som kan tester metoderne i dit Class Library.

1. Det er tilstrækkeligt først at teste metoden *Total*.
2. Udvid derefter din anden metode *Average*, så der kastes en passende Exception, hvis antal km falder uden for det tilladte interval: 0 - 2000 (se domænebeskrivelsen).
3. Udvid så din test til at vise, at der bliver kastet den forventede Exception, hvis antal km er udenfor det tilladte interval.
4. Kør testen og overvej din tests ”Code Coverage”.

**Opgave 3: TCP Server**  
  
Du skal nu gøre brug af en TCP Server, som kan tilbyde to udregninger (G*ennemsnit* og *TotalKm*).  
Udregningen, ”Gennemsnit”, beregner gennemsnittet ud fra *antalKm* og *antalLiter*.  
Udregningen, ”TotalKm”, beregner samlet antal km (summen) ud fra en række km-tal.

Dertil skal der designes en brugerdefineret protokol. Protokollen skal kunne skelne mellem de to udregninger, og protokollen skal overføre de nødvendige data for de to typer af udregninger. Protokollen skal derfor kunne:

1. Modtage og Bestemme hvilken udregning der ønskes: ” Gennemsnit” eller ”TotalKm”.  
   Så modtage de nødvendige data som: antalKm, antalLiter, el. transporter med flere km-tal  
   *Derefter*
2. Udregne gennemsnit eller totalt antal kørte km, afhængig af den ønskede udregning.
3. Sende resultatet af udregningen tilbage til klienten.

*Det kan anbefales først at lave en meget simpel TCP Server (denne opgave – Opgave 3), som kan testes med en meget simpel TCP Client (Opgave 4). Herefter kan programmet (programmerne) udvides sideløbende. Din TCP Server må meget gerne fra start af udnytte/udstille dine metoder i Library, men det er ikke et krav i første omgang.*

1. På denne baggrund, overvej og design en meget simpel protokol, dvs. hvordan du adskiller om udregningen er ”Gennemsnit” eller ”TotalKm”, samt hvordan data overføres.
2. Lav et simpel Console Application projekt, der gør brug af TcpListener og TcpClient sockets, og som lytter (listening) på en port f.eks. 7000.  
   Udvid programmet til at:

• Læse beskederne (defineret ved din protokol) fra klienten og udskrive beskeden.  
• Fortolke beskederne og udregne det ønskede resultat.  
• Sende et svar med det udregnede resultat tilbage til klienten.

1. Hvis du ikke har programmeret en multi-trådet (*concurrent*) TCP-server, skal du nu redegøre for hvordan dette kunne gøres.   
   Udvid derefter programmet, så TCP-serveren kan håndtere hver klient i en separat task/tråd.
2. Din TCP-server skal kunne tilgå dit Library, og derfor skal du skal nu udstille disse metoder i dette projekt. Du skal bruge din DLL-fil fra opgave 1.  
   Udnyt derefter metoderne i dit Library til udregningen af resultatet, hvis du ikke allerede har gjort det.

**Opgave 4: TCP Client**

Lav et simpelt Console Application projekt, som kan forbinde til din TCP Server og som kan sende beskeder til serveren.

1. Vis hvordan man kan sende beskeder til serveren, modtage og udskrive serverens svar.
2. Forklar formatet og indholdet af et TCP-segment?  
   Du må gerne forklare vha. TCP-segmenter opfanget af Wireshark el.lign.

**Opgave 5: Sikker (Secure) TCP Service**

*Du skal forberede skriftlige svar på nedenstående spørgsmål:*  
Forklar følgende to grundlæggende elementer i netværks sikkerhed:

* Confidentiality (Fortrolighed)
* Encryption (Kryptering)

1. Hvordan kan asymmetriske nøgler (public/private keys) anvendes til at sikre
   * Fortrolighed af en besked?
   * Autencitet af en afsender?

**Udvidet domænebeskrivelse**  
Transportoplysninger kan registreres i en database, som vist i nedenstående tabel:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Id | Lastbil | Chauffor | Dato | AntalKm | Gennemsnit |
| 1 | A | MarkJet | 2019-12-01 | 1500 | 7.05 |
| 2 | B | MikeWolf | 2019-12-01 | 1050 | 8.15 |
| 3 | C | AndyRap | 2019-12-01 | 976 | 7.92 |
|  | ----- |  |  |  |  |
| 47 | A | AndyRap | 2019-12-05 | 700 | 7.11 |
| 48 | B | Jammer | 2019-12-05 | 1104 | 8.34 |
|  | --- |  |  |  |  |
| 1178 | C | PeteBlack | 2019-12-18 | 550 | 7.96 |

*Der er mange flere transporter, men de er ikke vist her.*   
  
  
**Opgave 6: Rest Service med in memory database**

FastFup ønsker som sagt at registrere data om alle kørslerne for hver dag. Derfor skal du nu anvende en service som bruger a SQL-Server in-memory database i skyen (cloud Microsoft Azure) til at opbevare data om kørselsoplysningerne. Der bruges model først approach til at oprette in-memory databasen.

1. Lav et ASP.Net Core Service Web Application projekt/provider, FastFupService.
2. Opret en ny model class, TransportTab, med information om

* Id, unik identifikation af data
* Lastbil
* Chauffor, et unikt chaufførnavn
* Dato
* AntalKm
* Gennemsnit

svarende til den viste tabel.   
Sørg for at modellen indeholder properties med passende datatyper.

1. Opret dit data context som indeholder dataset property
2. Register the database context i *configurservices* methoden i startup.cs
3. Generere (ved brug af entity framework) en ny controller, TransportController, som benytter modelklassen og som tilbyder metoderne:
   1. GetAll()  
      Returnerer alle dagstransporter.
   2. GetId(int id)  
      Returnerer en bestemt transport med identifikationen, *id*.
   3. InsertData(Transport nyTransport)  
      Indsætter information om en ny transport, nyTransport, i databasen.

For hver metode bedes du overveje:

Hvilken HTTP metode skal benyttes?

URI (Route) stien?

1. Eksekver din service lokalt på din computer og fremvis den i en browser,
2. Benyt Postman eller Fiddler til at afprøve din service.

**Opgave 7: Web-applikation**

1. Lav en web-applikation, som kan udnytte din Restful web service.  
   Web-applikationen skal bruge HTML, TypeScript og Vue.js

1. Vis hvordan der kan hentes og fremvises:

* Alle transporter

1. Vis endvidere hvordan:

* En ny transport kan tilføjes.

**Opgave 8: UI-test af web-applikation**

Du skal lave en UI-test af din web-applikation. Testen skal anvende Selenium og MS Test.

Din test behøver ikke være udtømmende: Du skal kun teste en del af din web-applikation.