**Webapplikasjoner**

**ITPE3200-1**

**Oppgave 1 – Diagnose**

Muhammed Ateeb Salam **-s351951-k259**

Mohamed Alsaied Modathir **-s351901 k115**

**INNHOLDSFORTEGNELSE**

Hvorfor vi valgte Symptom til diangose kalkulator...........................................

Planlegging og diagrammer ..............................................................................4

Use-Case diagram.....................................................................................8

Fremgangsmåten fra koding til produkt........................................................10

Fra diagram til kode ...................................................................................10

Valg av API.........................................................................................................10

Første stegene mot en Database........................................................................

**Introduksjon**

På denne prosjektoppgaven valgte vi oppgave nummer tre, diagnose kalkulator. Vi iverksatte en løsning med en database hvor man kan velge forhåndsdefinerte symptomer og utfallet skal fortelle brukeren sin diagnose. Det ble tatt i bruk enkle eksempler om symptomer og diagnoser. Løsningen ble gjennomført i .Net Core. 3.1. Løsningen har et grensesnitt som bruker interaktive nettside egenskaper. Databasen som er inkludert i løsningen har funksjoner som gir en bruker tilgang til å opprette, lese, oppdatere og slette data

Vi hadde i oppgave å velge i mellom tre alternativer, «Aksjehandel side», «Offentlig UFO observasjoner» og den siste som vi bestemte oss for «Symtom til Diagnose Kalkulator». UFO oppgaven hadde potensial, samme med aksjehande, men siden det var bare en av oss med forkunnskaper om aksjer, så valgte vi oss å holde oss unna den aksjehandel, itilegg til det hørte vi at det var oppgaven som de fleste dro for. Til slutt satt vi igjen med to alternativer, hvor vi valgte «Symtom til Diagnose Kalkulat» Grunnen bak hvorfor vi valgte det tredje alternative, var på grunn av den virka mest gunstig for oss, hvor vi begge kjente oss igjen og kunne komme med masse innspill. Dermed starta planlegginga, hvor Mohamed startet allerede med fornt-end delen og satt opp skisse av hvordan start siden skal se ut, med tanke på brukergrensesnitt og «minimal viable products».

Overblikk og Diagrammer

Diagnose kalkulator Løsningen starter med å opplyse en bruker om at ingen data vil blir lagret eller brukt for noe annet formal f.eks. en undersøkelse. Neste lysbilde har inngangsfelter der brukeren oppgir standard informasjon som brukes for det meste på offentlige nettsider. Informasjon som blir oppgitt på denne fase vil bli framstilt på databasen for å illustrere bruk av «Entity Framework». Den tredje fasen har spørsmål om hvilke symptomer brukeren eller eventuelt en pasient har. Ut fra svarene som oppgis skal løsningen analysere og komme til en nøyaktig diagnose. Den fjerde og sluttfasen skal angi brukeren en oppsummering på prosessen og mulighet for å lese mer om diagnosen med henvising til anerkjente kilder som Helse Norge.

Mohamed hadde allerede startet med nettsiden, dermed startet jeg med hvordan databasen skal se ut, hvilken attributter, antall entiteter og koblinger imellom dem. Vi skrev det ned. Etterhvert oppstod det problemer, om hvordan vi skal forme vår database, skal vi lage vår egen data, eller hva? Med tanke på det kom ideen om å implementere en API, som har allered de 100 symptomene og tilhørende Diagnosene. Da bestemte Ateeb å lære seg å bruke API, som han fikk til etter tid med litt «trial and error». Vi kommunsierte ofte på skolen, om hva vi skal gjøre next, som for eksempel, når vi skal lage de forkjellige UML-Diagrammene, og hvem som skal ta seg av hva. Vi prøvde vårt beste å balansere det ut mest mulig, jeg gjorde litt css, html og han gjorde litt databaser. Men vi begge hadde klare hovedoppgaver, Mohamed lagde selve nettsiden, som ble redigert over tid av begge, og Ateeb lagde databasen og kobla til API-en, hvor Mohamed hjalp til da Ateeb satt fast. Mohamed lagde klassediagramm, og Ateeb aktivitetsDiagram. Use-case diagrammet ble lagd av begge.

Mening med Diagrammene var for å hjelpe oss med å lage oversikt, over hvordan vi skal lagre og hente data-ene fra databasen. Altså det dannet et bildet for oss av hvordan prosessen i prosjektet skulle bli gjennomført. Vi valgte å lage tre forkjellige Diagrammer, som vi klarte etter litt oppsumering av emnet «Systemutvilking» fra ifjor året. KlasseDiagramm, Use-case og AktivitetsDiagramm.

Aktivitetsdiagram var det første som ble lagd, siden vi ville danne et bildet av hvordan nettsiden vår burde se ut, og hvordan en bruker skal komme seg fra A til B. Som i vårt tilfelle fra symptom til en diagnose. Under på Vedlegg2 kan dere se hvordan vi visualiserte oss nettsiden skulle mota en bruker.

Diagram

Description automatically generated

Vedlegg 2

Her starter brukeren på hjemmesiden, hvor det skal være mulig å forstå klart å tydelig hva som fåregår. Altså alt er lett og tilgjengelig, han er her for å ta en diagnose så kan han se hvor han skal starte fra, eller hvis brukeren er bare en besøkende, så vil han lett forstå hva denne nettsiden innebærer og gjør. Brukeren intialiserer prosessen ved å trykke «Ta Diagnosetest», dermed vil han måtte oppgi personlig informasjon, alder, kjønn, navn og kontaktinformasjon som er nødvendig for kalkulatoren igjennom testen. Her vil han ha muligheten til å avlsutte, med en knapp som tydelig sier avbryt. Hvis brukeren fortsetter til neste steg vil han befinne seg i «oppgi symptom, hvor og hva», her vil kalkulatorne spørre om hvor smerten av symptomen befinner seg, og hvilken type symptom er det som oppstår. Her igjen vil brukeren ha muligheten til å avbryte, med samme knapp som før. Hvis brukeren fortsetter kommer vi oss fremm til slutten, hvor kalkulatoren bestemmer seg for mulige diagnoser utifra informasjonen brukeren nå har oppgitt igjennom stegene. Diagnosene vil bli listet basert på hvilket av diagnosene som høyest sansynlighet for brukeren. Han vil ha muligheten til å ta testen på nytt, hvis nei er det også mulighete for å se informasjonen listet i en tabell under «oversikt», hvor han kan velge å fjerne, eller endre opplysningene.

Neste UML-diagram vi trengte i prosjekte vårt var en KlasseDiagramm for et overblikk over hvordan databasen skulle bli settet opp. Her under er vedlegg3 som innholder klassediagrammet vårt.

API

Vi Valgte å bruke API’en Apimedic, som har sin egen nettside, og låner ut sine data til brukere, for å danne diagnose kalkulatorer. De deler ut informasjone om alle type symptomer og deres realsjoner til hvlken diagnose, og hvilken kroppsdel det omhandler. De gir også dypere data om Symptomen og Diagnosene, deres påvirkelse på dere, og beskrivelser på lignende symptomer og navn. Dett gir brukerne en enormt stor mulighet til å utforske og lage alle type Symptom til Diagnose kalkulator. Dette kan bli brukt til å studere hvordan man kan lage et reelt medisinks app, og det er allerede folk som har annonsert sucessfulle utgave også ved deres hjelp. Denne API’en ga oss en stort utgave av muligheter, men vi valgte bare å bruke bare et lite apsekt av det med tanke på MVP, og hvordan vi fortsatt er nye til det.

Forbedringer:

Løsning er kun en MVP (Minimum Viable product) som skal bevise at proposisjonen med fungerende tekniske muligheter kan bli til et ferdig produkt. Det er mange deler på applikasjonen som kan utvikles enda videre for en komplett løsning. Tjenester siden kan inneholde flere funksjoner. Databasen kan utformes slik at den kan håndtere flere mengder data. Universell design skal tas i hensyn som et grunnleggende prinsipp for del 2. Løsningen kan bli presentert for en gruppe brukere å få tilbakemelding på andre eventuelle faktorer. På den nåværende løsningen ble det brukt ferdig lagd logo, en forbedring til neste del vil være å design egen logo. Enda en forbedring kunne ha

KlassseDiagramm

entiteter

1. Diagnoser
2. Iddiagnose(autoIncrement)\_PK
3. Symptomer
4. diagnose
5. Bruker
6. Idbruker(auotincrement)\_PK
7. Fornavn
8. Etternavn
9. email
10. Kjønn
11. Alder
12. Telefon
13. Register
14. Iddiagnose\_(FK)
15. Idbruker\_(FK)

FK = Fremmednøkkel

PK = Primærnøkkel

Disse tre entiteter ble vi enige om som skulle danne vår database, vi lagde to classer, Diagnoser og Bruker, hvor begge hadde mange til mange relasjoner, en diagnose kan ha flere kunder, og flere kunder kan ha flere diagnoser. Dermed ble den tredje entiteten dannet, med fremmed nøkkelen fra de andre to sine primærnøkkeler.

Fremgangsmåten

Vi planlegget html, og css først, dermed bevegde vi oss mot databaser. Vi prioriterte oppgaven først og fremst før vi tenkte på noe ekstra. Altså vi valgte å legge mer vekt på funksjonaliteten og brukergrensesnitt.  starta med en enkel boks, med en enkel beskjed, som både informerte brukeren hva denne nettsiden er og hva brukeren selv skal gjøre. Det gjorde vi ved å fokusere alt på stor knapp, som sa “Start Diagonosering”. Enkel og gunstig.  dermed satt vi opp en aktivitets diagramm, til å lede oss

Databasen

Vi startet å lage databasen, med en plan i bakhodet, hva vi skal ha, hva vi kommer til å trenge og hvordan det skal fordeles. Slik at allerede fra starten vi ikke trengte å gjøre store endringer, vi fulgte en mal fra læreren, og brøt det ned til CROD.

Vi startet med å opprette klassen Bruker.cs, her har vi alle kolonnenen fra forhånd planlagt klassediagramm, som innhold følgende attributter og kolonner. Som vi gjorde ved hjelp av «get» og «sett» metoder. Hver gang kunden tok en diagnose ble det lagt inn i klasse Bruker.cs og Diagnose.cs, som er koblet sammen ved mange til mange relasjoner, dermed visste vi hvordan DiagnoseController.cs skulle se ut. Vi hadde tre metoder, lagre(), Slett(), og Endre(). Vi lagret ved å hentet informasjon fra client siden til server, hjelp av metoden «post» under:

$.post("bruker/Lagre", bruker, function (OK) {

Som videre sendte bruker-info til server, hvor den ble tatt imot og lagret som et objekt av klasse Diagnoser, og de ble senere hentet med en annen metode «get» under:

$.get("Bruker/HentAlleDiagnoser", function(data) {

Som hentet dataene fra list-en Diagnoser, og formeterte dem på client-siden.

Vi hadde også Dbinnit.cs som hadde fårhånd