# Overvejelser

En **monitorunit** kan have flere sensorer tilsluttet -> En pool kan derfor kun have 1 MU.

Ved flere sensortilslutninger: Løsninger overvejes i hardware og embedded software.

**Addresse** skalar-strengen fjernes -> name bruges til både poolnavn og adresse (hvis brugeren ønsker dette)

**MU-Serienumre** gemmes i en ”fri” tabel eller skal der skabes binding mellem MU og pool?

PIN kode udgår idet MU tildeles en bool der beskriver om en MU er registreret eller ej.

Skal fysisk MU have plug anf play funktionalitet eller kommer den as is?

* PNP funktionalitet. Sensor porte assignes til sensortyper. Dette giver mulighed for senere udvidelse med flere typer sensorer.
* MU i db har en liste af sensorer (sensorentity) - (sensor porte er nullable).

# Designovervejelser

Skal PoolAccess tager imod en User eller en Email når den skal hente en bruger ud af DB. Eller skal PoolAccess have en reference til UserAccess klassen? Tredje mulighed er at lave metoden i SmartpoolDB klassen og på den måde have funktionalitet længere oppe i hierakiet.

Løsningen indtil videre bliver at vi lader PoolAcces og UserAccess have en reference til et interface af den anden type. På den måde kende de ikke nogen implementering og er lavt koblet.

Som implementeringen ser ud nu, så bruger PoolAccess kun FindUserByEmail fra UserAccess klassen, så det overvejes om der skal laves et interface kun med denne methode, som PoolAccess så kan bruge.

Grundet lange søgetider i pool databasen, kan vi arkivere data eksempelvi per kvartal. Herved opnår vi et maksimum for søgetider i nyere data, hvorimod meget gammelt data kan tage længere tid.

Tupler bruges i stedet for dictionaries som data containers.

## Test impact

Vi kan ikke bruge Nsubstitute, da dette ikke reelt lægger en User i databasen Når der herefter forsøges at smide en pool ind i databasen fås en fejl, idet der ikke er en user at binde den til.

Startet med DDS lite, gået over til EF

# Proces

Vi har kørt TDD, og har tit oplevet at læbe med hovedet mod en mur, idet det ofte har vist sig at det ikke er programmet, men i stedet vores tests der ikke fungerer.

# Fejl og problemer

AddPool tager både en User og et User.Id, hvilket er et problem. Her er det kun User.Id der skal bruges. Hvis man også giver den en User med, oprettes en ny user i databasn udenom vores CheckForsameUser metode.

Cultureinfo skal fixes – Flere muligheder. Set på host PC eller tving gennem kode.

Som systemet er designet nu, lægger respektive sensordata for alle pools i deres enkelte tabel.

* Kan optimeres væk (allerede gjort med faktor 4), ved at oprette nye tabeller for hver pool (kan ikke gøres uden at bruge for meget tid).

# Rapport og Dokumentation

Brug en eller to User Stories som en rød tråd gennem rapporten.

Med hensyn til dokumentation beskrives hvert modul fra enden til anden i hvert sit afsnit.

# Klasse og metodebeskrivelser for data access + datakontekst

Klassebeskrivelser

DataAccess:

DataAccess implementerer IdataAccess og står for at skrive og læse data records til/fra databasen.

IDataAccess refereres af klassen SmartPoolDB

---Insert class diagram---

PoolAccess:

PoolAccess implementerer IpoolAccess og står for at skrive og læse pool records til/fra databasen. PoolAccess indeholder administrator funktionalitet til at slette alle pools på databasen. Herudover står PoolAcces for data editing af pools samt validering af pooldata før det gemmes.

---Insert class diagram---

UserAccess:

UserAccess implementerer IuserAccess og står for at skrive og læse user records til/fra databasen.

UseraAcces indeholer administrator funktionalitet til at slette alle users på databasen. Herudover står UserAcces for data editing af users samt validering af userdata før det gemmes.

---Insert class diagram---

SmartPoolDB:

SmartpoolDB implementerer IsmartPoolDB og står for at initialisere DataAccess, UserAccess og PoolAccess gennem deres interfaces. SmartpoolDB kan da bruge af klienter når data access funktionalitet skal bruges.

---Insert class diagram---

DataBaseContext : DbContext

DbContext er en del af Entity Frameworket. DatabaseContext arver fra DbContext og udgør da bindeleddet fra koden til databasen.

Entities er defineret i klassen som public virtual DbSet<T> hvor typen er den spcifikke entity.

DatabaseContext objektet skal oprettes når der laves en query på databasen eller sættes data ind.

På en DbSet property kan der kaldes Add(), som tilføjer en entity i den kontekst der kaldes i. Herefter skal DbContext’s SaveChanges() metode kaldes for at ændringerne skrives i databasen.

[ref: <https://msdn.microsoft.com/en-us/data/jj729737.aspx>]

---insert class diagram---

Funktionsbeskrivelser

UserAccess.cs

bool AddUser(string fullname, string email, string password);

Står for at indsætte en user i databasen. Metoden kalder IsEmailInUse samt ValidateName før der oprettes et database kontekst objekt af typen DataBaseContext. På Kontekstobjektets DbSet<User> property kaldes Add metoden med den nye User som parameter. Herefter gemmes den nye User i databasen ved at kalde SaveChanges metoden på kontekstobjektet. Adduser returnerer false hvis der er fejl i enten check på email eller fullname. Hvis det lykkes at tilføje brugeren til databasen returneres true.

-------------------------------------------------------------------------------------

User FindUserByEmail(string email);

Søger i databasen efter en User med den email der angives som parameter. Det må ikke eksistere flere brugere med samme email i databasen. Søgningen udføres med et LINQ statement på databasekonteksten.Metoden returnerer den fundne User med mindre der kastes en exception.

using (var db = new DatabaseContext())

{

var searchByEmail = from search in db.UserSet

where search.Email.Equals(email)

select search;

if (searchByEmail.Count() > 1) throw new MultipleOccourencesOfEmailWasFoundException();

if (searchByEmail.Count() == 0) throw new UserNotFoundException();

foundUser = searchByEmail.First();

}

Exceptions:

MultipleOccourencesOfEmailWasFoundException() Sikkerhedsforanstaltning ved flere emails

UserNotFoundException() Der finds ikke en burger med angivne email addresse I databasen.

-------------------------------------------------------------------------------------

bool IsEmailInUse(string email);

Laver en LINQ query på databasekontekst objektet og checker om der findes en mail adresse i databasen der matcher den som er givet som parameter. Metoden returnerer false hvis mail addressen ikke er i brug og true hvis den er.

-------------------------------------------------------------------------------------

bool ValidatePassword(string email, string password);

bool ValidateName(string fullname)

Da et Fullname skal indeholde mere end et navn, checkes der her for et minimum af to navne. Hvis navnet kan godkendes returneres true, eller false.

void RemoveUser(string email);

Checker først om en burger med email findes.

Laver en LINQ query på databasekontekst objektet. Og finder den pågældende bruger.

Brugeren fjernes ved at kalde remove på kontekstobjektets UserSet property.

db.UserSet.Remove(user);

-------------------------------------------------------------------------------------

void DeleteAllUsers();

Bruges kun til systemtest og under udvikling.

Sletter alle brugere i databasen. Dette gøres ved at eksekvere en SQL kommando direkte på databasen.

using (var db = new DatabaseContext())

{

db.Database.ExecuteSqlCommand("DELETE [UserSet]");

}

-------------------------------------------------------------------------------------

bool EditUserPassword(string email, string newPassword);

-------------------------------------------------------------------------------------

bool EditUserEmail(string email, string newEmail);

-------------------------------------------------------------------------------------

bool EditUserName(string email, string newName);

-------------------------------------------------------------------------------------

PoolAccess.cs

IUserAccess UserAccess { get; set; }

-------------------------------------------------------------------------------------

bool AddPool(string email, string name, double volume);

-------------------------------------------------------------------------------------

bool IsPoolNameAvailable(string email, string name);

-------------------------------------------------------------------------------------

Pool FindSpecificPool(string email, string name);

-------------------------------------------------------------------------------------

bool RemovePool(string email, string name);

-------------------------------------------------------------------------------------

void DeleteAllPools();

-------------------------------------------------------------------------------------

bool EditPoolName(string ownerEmail, string currentName, string newName);

-------------------------------------------------------------------------------------

bool EditPoolVolume(string ownerEmail, string name, double newVolume);

-------------------------------------------------------------------------------------

bool EditPoolUser(string currectOwnerEmail, string name, string newUserEmail);

-------------------------------------------------------------------------------------

List<Pool> FindAllPoolsOfUser(string ownerEmail);

-------------------------------------------------------------------------------------

dataAccess.cs

IPoolAccess PoolAccess { get; set; }

-------------------------------------------------------------------------------------

bool CreateDataEntry(string ownerEmail, string poolName, double chlorine, double temp,

-------------------------------------------------------------------------------------

double pH, double humidity);

-------------------------------------------------------------------------------------

void DeleteAllData();

-------------------------------------------------------------------------------------

List<Tuple<SensorTypes, double>> GetChlorineValues(string poolOwnerEmail, string poolName, int daysToGoBack);

-------------------------------------------------------------------------------------

List<Tuple<SensorTypes, double>> GetTemperatureValues(string poolOwnerEmail, string poolName, int daysToGoBack);

-------------------------------------------------------------------------------------

List<Tuple<SensorTypes, double>> GetPhValues(string poolOwnerEmail, string poolName, int daysToGoBack);

-------------------------------------------------------------------------------------

List<Tuple<SensorTypes, double>> GetHumidityValues(string poolOwnerEmail, string poolName, int daysToGoBack);

-------------------------------------------------------------------------------------