

Timebox1 – Database Struktur

Oversigt

OpgaveNavn	Template structure for database		
Implementering af krav	Delvis implementering af krav WEB-1, WEB-2, WEB-3, WEB-4, WEB-5, WEB-6, WEB-7, WEB-8, WEB-9, WEB-10, WEB-11 og WEB-13		
Udført af	Marc og Jan	Dato	20-09-2021
Timebox	1	Udv.omr.	Website

Contents

INTRODUKTION.....	1
ANALYSE.....	2
DESIGN.....	3
IMPLEMENTERING	3
VERIFIKATION	5
TESTRESULTAT.....	6
KONKLUSION	6
REFERENCER	7

Introduktion

Dette dokument beskriver databasestrukturens opbygning, og implementering.

Skrivning af filnavn er således: *models.py*.

Engelske ord blive notere med situationstegn således: " Launch Phase".

Analyse

Undersøgelserne fra "Launch Phase", viser at der var brug for en tabel med bruger oplysninger, se Tabel 1, med størrelsen af de forskellige "field types".

Tabel 1 Bruger tabel

User	
80, CHAR [1]	Name
30, CHAR	UserType
120, CHAR	Email
20, CHAR [2]	Phone number
120, CHAR	Company
200, CHAR	Password

Der skal laves en tabel med facilitet information, se Tabel 2.

Tabel 2 Facilitet tabel

Facility	
80, CHAR	Name
120, CHAR	Location
80, CHAR	Owner
120, CHAR	Key

Database kommer til at indeholde en log, se Tabel 3.

Tabel 3 Log

Log	
80, CHAR	User Name
80, CHAR	Facility Name
DateTime	Dato og tid

Der anvendes "surrogate key" som "primary key", keys som er unikke og bliver genereret automatisk. Begrundelsen er at "surrogate key" som "primary key" er nemmere at implementere og fylder typisk ikke mere end 4 bytes [3].

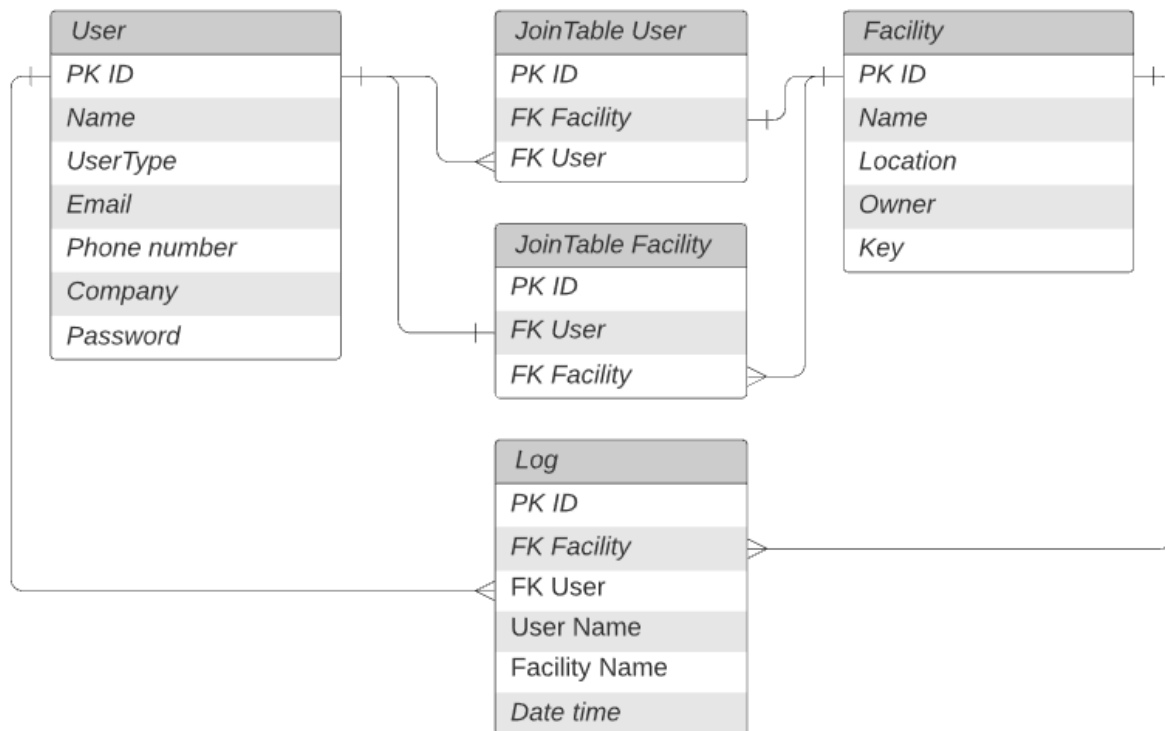
Opsætning af "primary keys" og "foreign keys" er i næste afsnit.

Design

På Figur 1 er designet for databasestrukturen, hvor der er to "one-to-one" [5] og resten er "one-to-many" [4].

Mellem "User" og "JoinTable User" har vi en "one-to-many" relation, da vi har en bruger som kan have adgang til flere faciliteter. Samme relation er der mellem "Facility" og "JoinTable Facility", hvor en facilitet kan have flere brugere.

"Log" har relationen "many-to-one" til både "User" og "Facility" da en bruger kan have flere logs med samme facilitet og vice versa, og en facilitet kan have flere forskellige brugere.

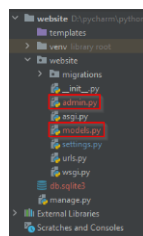


Figur 1 Databasestruktur

Implementering

Implementeringen er gjort ved at starte et standard Django projekt op, med en enkelt main app i, kaldet websitet.

Efter projektet er startet op, er der blevet tilføjet 2 ekstra filer til hoved appen: *models.py* og *admin.py*, som set på Figur 2.



Figur 2 Django projekt struktur

Strukturen til database er beskrevet i *models.py*, hvor det er sat op i klasser.

De forskjellige klasser kan ses på Figur 3 til og med Figur 6, og passer overens med analyse afsnittets Figur 1.

```
"For Users:"
class Users(models.Model):
    ID = models.AutoField(primary_key=True)

    Name = models.CharField(max_length=80)
    Email = models.CharField(max_length=120)
    PhoneNumber = models.CharField(max_length=20)
    Company = models.CharField(max_length=120)
    Password = models.CharField(max_length=200)
    UserType = models.CharField(max_length=30, default="Field user")

    def __str__(self):
        return self.Name
```

Figur 3 DB "for users"

```
"For facilities:"
class Facilities(models.Model):
    ID = models.AutoField(primary_key=True)

    Name = models.CharField(max_length=80)
    Location = models.CharField(max_length=120)
    Owner = models.CharField(max_length=80)
    Key = models.CharField(max_length=200)

    def __str__(self):
        return self.Name
```

Figur 4 DB "for facilities"

```
"Join tables:"
class JoinTableUser(models.Model):
    ID = models.AutoField(primary_key=True)
    Facility = models.OneToOneField(Facilities, on_delete=models.CASCADE, blank=True, null=True)
    User = models.ForeignKey('Users', on_delete=models.CASCADE, blank=True, null=True)

class JoinTableFacility(models.Model):
    ID = models.AutoField(primary_key=True)
    User = models.OneToOneField(Users, on_delete=models.CASCADE, blank=True, null=True)
    facility = models.ForeignKey('Facilities', on_delete=models.CASCADE, blank=True, null=True)
```

Figur 5 DB "join tables"

```
"log"
class Logs(models.Model):
    ID = models.AutoField(primary_key=True)
    Facility = models.ForeignKey('Facilities', on_delete=models.DO_NOTHING, blank=True, null=True, db_constraint=False)
    User = models.ForeignKey('Users', on_delete=models.DO_NOTHING, blank=True, null=True, db_constraint=False)
    DateTime = models.DateTimeField()
    UserName = models.CharField(max_length=80)
    FacilityName = models.CharField(max_length=80)
```

Figur 6 "log"

For at kunne føre modellerne ind i databasen, skal der i Django projektets *settings.py*, sættes at modellerne skal tages med i databasen, hvilket gøres i filens "INSTALLED_APPS" afsnit, som kan ses på Figur 7.

```
INSTALLED_APPS = [
    'website',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Figur 7 "INSTALLED_APPS"

Herefter kan databasestrukturen migreres ind i database, ved hjælp af Djangos CMD-kommandoer:

- \$python manage.py makemigrations
 - Laver migrations filer for hvad der skal flyttes, udefra ændringer i filerne.
- \$python manage.py migrate
 - Migrerer strukturen/ændringer til database, baseret på den sidste generet migrerings fil.

Verifikation

For at kunne verificere at de forskellige tabeller er inde i databasen, er der blevet oprettet ene generisk Django admin bruger, hvor de forskellige tabeller er sat synlige.

For at oprette admin brugeren, er der blevet brugt standard Django CMD-kommandoer:

- \$python manage.py createsuperuser

Efter admin brugeren er oprettet, er der gennem *admin.py* blevet sat at brugeren kan se alle de forskellige tables, som kan ses på Figur 8.

```
from django.contrib import admin
from .models import Users, UserLogs, Facilities, FacilityLog, JoinTableFacility, JoinTableUser

admin.site.register(Users)
admin.site.register(UserLogs)
admin.site.register(Facilities)
admin.site.register(FacilityLog)
admin.site.register(JoinTableUser)
admin.site.register(JoinTableFacility)
```

Figur 8 admin.py

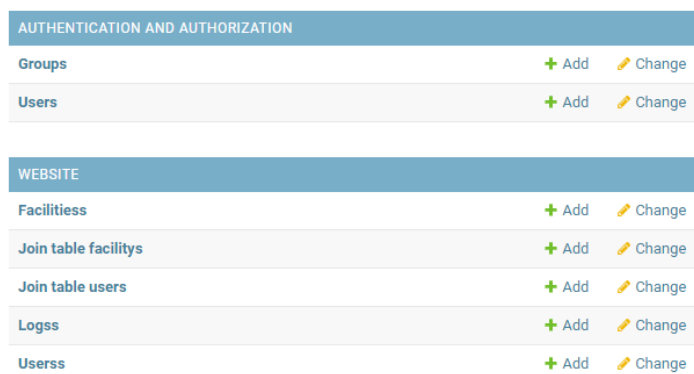
Efter admin brugeren er oprettet kan hjemmesiden starts op, ved hjælp af standard Django CMD-kommandoer:

- \$python manage.py runserver

Hvor der så kan gås ind på hjemmesidens admin url:

- 127.0.0.1:8000/admin

Og logges ind med den nye admin bruger, hvorefter de forskellige databasetabeller kan ses, som vist på Figur 9.



AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change
WEBSITE		
Facilities	+ Add	Change
Join table facilities	+ Add	Change
Join table users	+ Add	Change
Logss	+ Add	Change
Userss	+ Add	Change

Figur 9 "url/admin"

Tabel 4: Tests til verifikation af opgave

Test	Test Steps	Pre-requisites	Pass-betingelser	Ref. til data	Resultat
Test "insert"	1. login som admin. 2. gå ind på website's forskellige tabeller. 3. udfyld felterne 2 gange. 4. tryk på save. 5. slet alle entries	Modeller er migreret til databasen. Der er oprettet en admin bruger. Django hjemmesiden er oppe og køre	Ingen fejl bliver givet, og de forskellige indsætninger kan ses i databasen, gennem admin siden.		Bestået

	Grunden til nummer 3, er for at kunne teste ManyToOne relationer				
--	--	--	--	--	--

Testresultat

Der kunne sættes entries ind i databasen, og relationerne virker, samt kan alle entries slettes igen.

Konklusion

Database er efter analyse, design og test udviklet og klar til at blive anvendt. Dokumentet beskriver fremgangsmåden og de vigtigste elementer af databasen.

Referencer

[1]	Maximum længde af navn (top-10) https://largest.org/people/names (sidst besøgt 20-09-2021)
[2]	Maximum længde af telefonnummer https://support.telesign.com/s/article/what-is-the-maximum-length-of-any-phone-number (sidst besøgt 20-09-2021)
[3]	Genlæsning af "keys" https://www.sqlshack.com/sql-database-design-choosing-primary-key/ (sidst besøgt 20-09-2021)
[4]	One-to-many https://en.wikipedia.org/wiki/One-to-many_(data_model) (sidst besøgt 20-09-2021)
[5]	One-to-one https://en.wikipedia.org/wiki/One-to-one_(data_model) (sidst besøgt 20-09-2021)