DAT 109 – OBLIG 4

Modelldrevet utvikling

GRUPPE 13

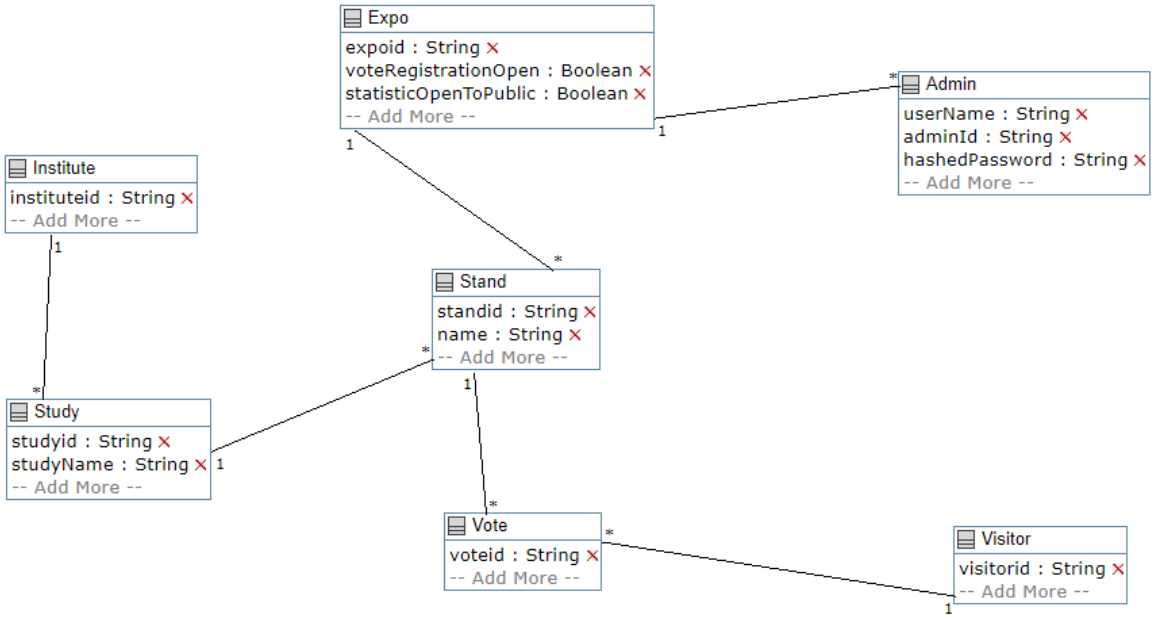
Adrian R. J. Mortensen

Kristoffer Nome

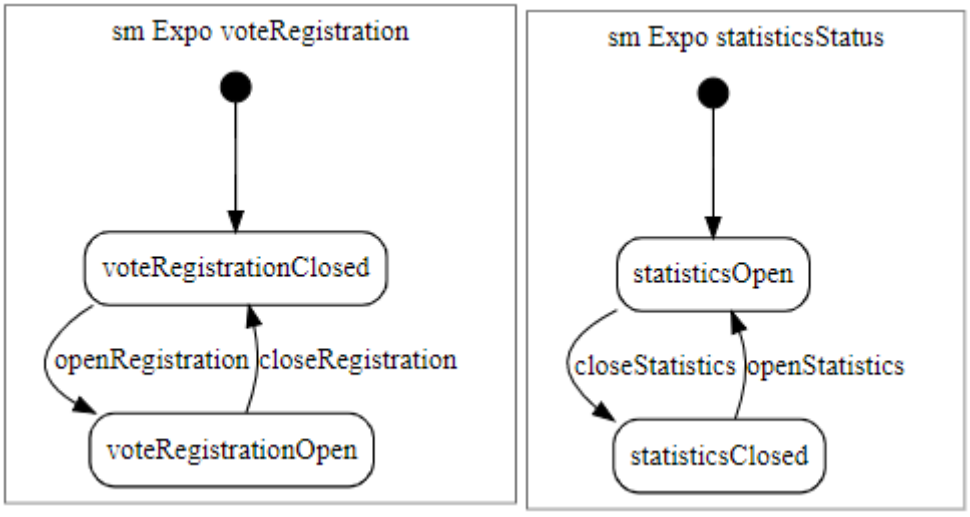
Anders Simonsen

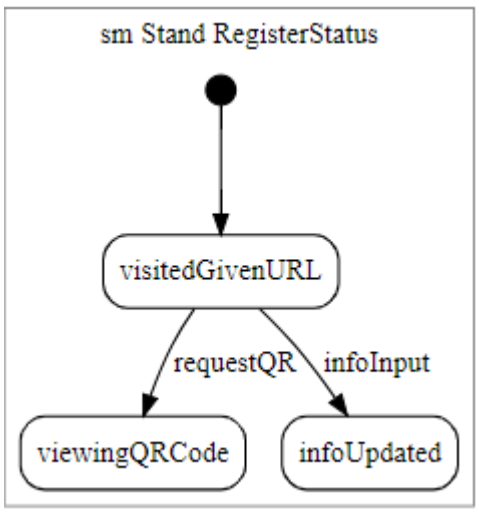
Kode for UMPLE er vedlagt i /UMPLE/Expo.ump

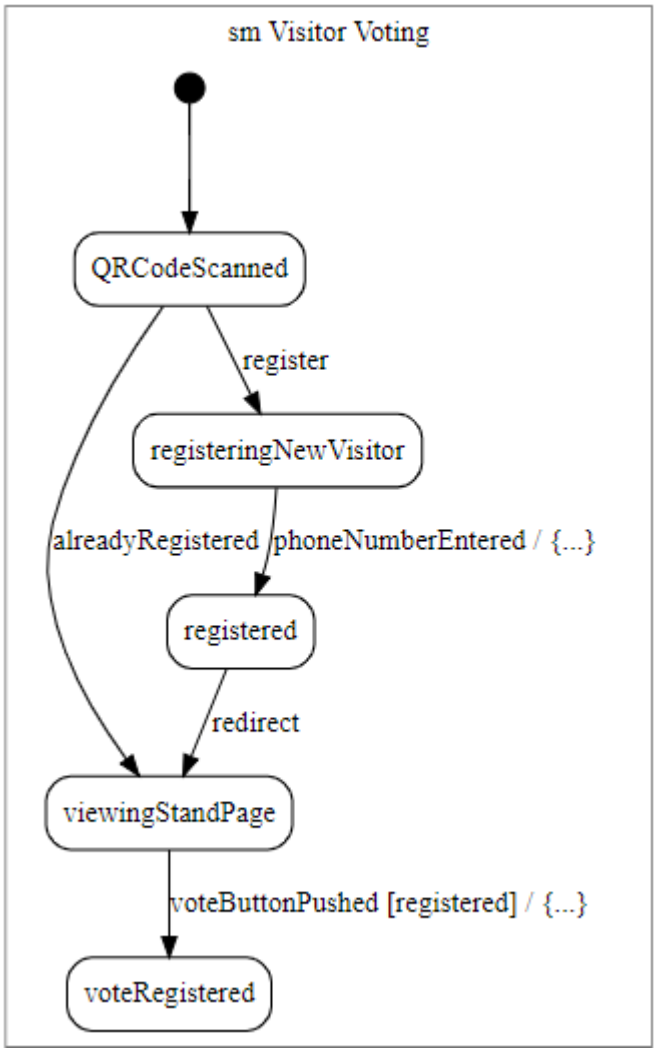
1. Lag en domenemodell for din EXPO applikasjon i UMPLE

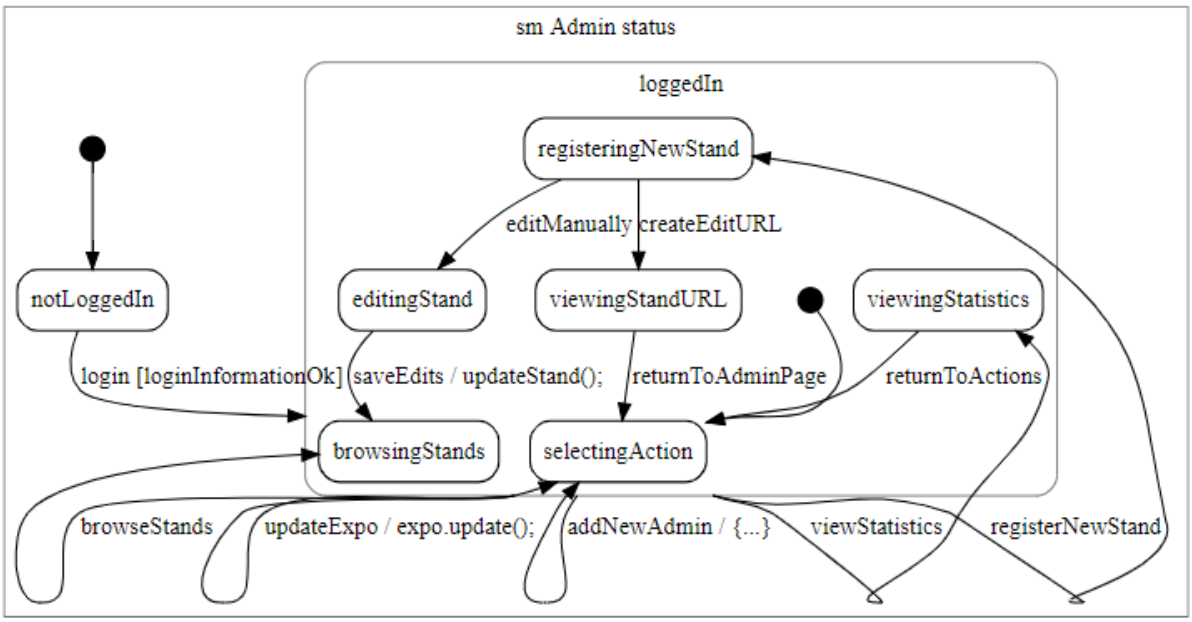


2. Lag et tilstandsdiagram for din EXPO applikasjon i UMPLE




sm Stand RegisterStatus
visitedGivenURL
requestQR
infoInput
viewingQRCode
infoUpdated


sm Visitor Voting
QRCodeScanned
register
registeringNewVisitor
alreadyRegistered
phoneNumberEntered / {...}
registered
redirect
viewingStandPage
voteButtonPushed [registered] / {...}
voteRegistered


sm Admin status
loggedIn
registeringNewStand
editManually
createEditURL
notLoggedIn
editingStand
viewingStandURL
viewingStatistics
login [loginInformationOk]
saveEdits / updateStand();
returnToAdminPage
returnToActions
browsingStands
selectingAction
browseStands
updateExpo / expo.update();
addNewAdmin / {...}
viewStatistics
registerNewStand

UMPLEKODE:

```
class Expo
{
  expoid;
  Boolean voteRegistrationOpen;
  Boolean statisticOpenToPublic;

  voteRegistration
  {
      voteRegistrationClosed
      {
      openRegistration -> voteRegistrationOpen;
      }

      voteRegistrationOpen
      {
      closeRegistration -> voteRegistrationClosed;
      }
  }

  statisticsStatus
  {
      statisticsOpen
      {
      closeStatistics -> statisticsClosed;
      }

   statisticsClosed
   {
      openStatistics -> statisticsOpen;
    }
  }
}


class Stand
{
  standid;
  name;
  *--1 Expo;

  RegisterStatus
  {
    visitedGivenURL
    {
      infoInput -> infoUpdated;
      requestQR -> viewingQRCode;
    }
    infoUpdated{}
    viewingQRCode{}
  }
}

class Study
{
  studyid;
  studyName;
  1--* Stand;
}
```

```
class Institute
{
  instituteid;
  1--* Study;
}

class Vote
{
  voteid;
  *--1 Stand;
  *--1 Visitor;
}

class Visitor
{
  visitorid;

  Voting
  {
    QRCodeScanned
    {
      entry/{checkIfRegistered();}
      register -> registeringNewVisitor;
      alreadyRegistered -> viewingStandPage;
    }

    registeringNewVisitor
    {
      phoneNumberEntered -> /{registerNewVisitor();} registered;
    }

    registered
    {
      redirect -> viewingStandPage;
    }

    viewingStandPage
    {
      voteButtonPushed [registered] -> /{registerNewVote();}
voteRegistered;
    }

    voteRegistered{}
  }
}

class Admin
{
  userName;
  adminId;
  hashedPassword;

  *--1 Expo;

  status
  {
    notLoggedIn
    {
      login [loginInformationOk] -> loggedIn;
    }
```

```
    loggedIn
    {
      selectingAction{}

      addNewAdmin -> /{addNewAdmin(new Admin(userName, password));}
selectingAction;

      updateExpo -> /{Expo.update();} selectingAction;

      registerNewStand -> registeringNewStand;

      browseStands -> browsingStands;

      viewStatistics -> viewingStatistics;

      registeringNewStand
    {
      entry/{createNewStand();}
      editManually -> editingStand;
      createEditURL ->  viewingStandURL;
    }

    viewingStandURL
    {
      returnToAdminPage -> selectingAction;
    }

    editingStand
    {
      saveEdits -> /{updateStand();} browsingStands;
    }

    browsingStands
    {

    }

    viewingStatistics{returnToActions -> selectingAction;}

    }


  }
}
```

3. Generer kode til minst 3 forskjellige språk, for eksempel Java, SQL, PHP, JSON, osv, for diagrammene i 1 og 2

Generert Java-kode finnes i /Java

Generert SQL-kode finnes i /SQL

Generert JSON-kode finnes i /JSON

Generert PHP-kode finnes i /PHP

4. Diskutér hvordan den genererte koden er i forhold til koden som din gruppe implementerte i prosjektoppgaven. For eksempel, hvilke klasser/tabeller er like/forskjellige og hvordan, osv.

Det velges å generere kode til Java, SQL, JSON og PHP.

Tabeller og entiteter er stort sett lik koden vår gruppe implementerte – med unntak av at Expo i for eksempel SQL har to attributter av type BIT, i stedet for boolean – dette har liten betydning, siden boolean i flere databaser implementeres som 0 eller 1 – altså, en bit.

Den største forskjellen fra den genererte koden i forhold til koden vår gruppe implementerte i prosjektoppgaven, finnes i at alle entitetene som har en tilhørende tilstandsmaskin – Expo, Stand, Visitor og Admin – har egne enum-attributter/variabler med de forskjellige mulige tilstandene. Eksplisitte tilstander og statuser er generelt noe vår implementasjon tok lite eller ingen hensyn til.

I versjonen vår gruppe implementerte hadde en Administrator mulighet til å åpne eller stenge for avstemming, og å gjøre statistikk tilgjengelig/ikke tilgjengelig for besøkende – etterfulgt av at Expo-objektet ble oppdatert i databasen ved hjelp av JPA. I modellen som er laget ved hjelp av Umple er det laget en metode for updateExpo(), men det er ingen tydelig måte å registrere hva som eventuelt er endret, og det har ikke lykkes å få Umple til å gjenkjenne at en Admin forsøker å oppdatere eller endre et Expo-objekt – selv om det ble opprettet en relasjon mellom de to entitetene som ikke eksisterte i vår opprinnelige implementasjon, og selv om det i tilstandsmaskinen opprettes et eksplisitt kall på ‘expo.update()’ genereres det ikke en metode i den genererte Expo-klassen som heter update(). Denne oppdateringen måtte sannsynligvis bli eksplisitt definert i Java-koden, eller bli implementert på en annen måte.