API

peter.de.winne

November 2021

1 Introduction

De opdracht voor deze taak was om een systeem te ontwerpen dat toeliet om te registreren wie er allemaal aanwezig was tijdens een klas. De voorwaarden waren dat de studenten zich kunnen registreren via een unieke code, bijvoorbeeld een qr-code. Daarnaast moest de leerkracht de mogelijkheid krijgen om te kijken wie er allemaal aanwezig is/was tijdens de klas en moet hij/zij klassen kunnen aanmaken. Tenslotte moet er een administrator zijn die leerkrachten en studenten kan toevoegen aan een cursus en zelf cursussen kan aanmaken. Voor deze toepassing moesten we een databank, API en html pagina's ontwerpen.

2 Databank

Als eerste opdracht werd gevraagd een databank te ontwerpen. Het uiteindelijke resultaat is te zien in deze figuur1. In de databank zijn er vijf entiteiten. De user entiteit houdt alle belangrijke informatie bij van de gebruiker. Deze info is een userid, het e-mail adres, de voornaam, de achternaam en of de gebruiker al dan niet een administrator is. Deze entiteit is met de userid verbonden aan de entiteit van USERCOURSE deze entiteit verbind de gebruikers met alle

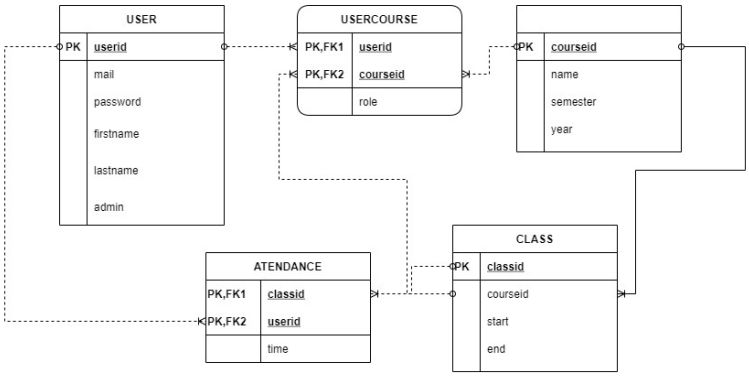

Figure 1: Databank ontwerp

cursussen. In deze tabel word ook bijgehouden of de gebruiker een student of leerkracht is voor de cursus. Bij de entiteit cursus wordt de naam, semester en jaar bijgehouden. Per cursus kan je verschillende klassen hebben. Van deze entiteit word bijgehouden van welke cursus ze is. Het startuur en einduur. Tenslotte is er de entiteit attendance. Dit verbind een gebruiker met een klas. Er wordt hierin ook opgeslagen wanneer de gebruiker zich registreert.

3 API

Het tweede deel van deze opdracht was een application interface program te maken. Dit is een programma dat toelaat om twee applicaties met elkaar te laten communiceren.

3.1 Forms

Voor ik uitleg wat de main.py module doet leg ik uit wat forms.py doet. Deze module gebruik om het overzicht te bewaren. In deze module staat alle info voor de forms die ik gebruik in de website.

3.1.1 Libraries

De bibliotheken die ik gebruik in deze module zijn FlaskForm, wtforms, de validators van wtforms en sqlite3. De FlaskForm module wordt gebruikt zodat de klasse van de desbetreffende form kan gebruikt worden in flask. Bij Wtforms worden de juiste input velden gekozen, bijvoorbeeld de passwordfield om een wachtwoord in te voeren of Booleanfield voor een checkbox. De validators controleren of de informatie die in de inputs wordt gegeven wel inorde is.

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField,

BooleanField, DateTimeField, SelectField, TimeField, HiddenField
from wtforms.validators import DataRequired, Length, Email,
EqualTo, ValidationError
import sqlite3
```

3.1.2 SelectField inputs

Om de SelectFields van gegevens te voorzien maken we deze hier aan.

```
days = (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
23,24,25,26,27, 28,29,30,31)
months = [(1,'january'),(2,'february'),(3,'march'),(4,'april'),
(5,'may'),(6,'june'),(7,'july'),(8, 'august'),(9, 'september'),
(10, 'october'),(11,'november'),(12, 'december'),]
years = [x for x in range(2021,2100)]
```

```
hours = [x for x in range(0,24)]
minutes = [x for x in range(0,60)]
```

3.1.3 Classes

Voor alle inputforms is er een aparte klasse die zegt welke input op de pagina welke soort input moet zijn en controleert ook of alle nodige inputs voldoen aan de voorwaarden.

```
class RegisterForm(FlaskForm):
        email = StringField('Email',validators=[DataRequired(),Email()])
        first = StringField('First',validators=[DataRequired()])
        last = StringField('Last', validators=[DataRequired()])
        password = PasswordField('Password',validators=[DataRequired(),
        EqualTo('confirm', message='Passwords not the same')])
        confirm = PasswordField('Confirm password',validators=[DataRequired()])
        remember = BooleanField('Remember Me')
        submit = SubmitField('Register')

class Create_classForm(FlaskForm):
        day = SelectField('Day', choices=days,validators=[DataRequired()])
        month = SelectField('Month', choices=months,validators=[DataRequired()])
        year = SelectField('Year', choices=years, validators=[DataRequired()])
        submit = SubmitField('Create class')
        start_h = SelectField('Start', choices=hours, validators=[DataRequired()])
        start_m = SelectField('', choices=minutes, validators=[DataRequired()])
        end_h = SelectField('End', choices=hours, validators=[DataRequired()])
        end_m = SelectField('',choices=minutes,validators=[DataRequired()])

class Register4classForm(FlaskForm):
        submit = SubmitField('Register')

class Create_Course(FlaskForm):
        name = StringField('Name',validators=[DataRequired()])
        semester = SelectField('Semester', choices=(1,2),validators=[DataRequired()])
        year = SelectField('Year', choices=years,validators=[DataRequired()])
        submit = SubmitField('Create class',validators=[DataRequired()])
```

3.1.4 Validators

Bij deze functies controleren we of de input voldoet aan de eisen

```
def validate_email(self, email):
        conn = sqlite3.connect('database.db')
        curs = conn.cursor()
        curs.execute("SELECT mail FROM user where mail = (?)",[email.data])
        valemail = curs.fetchone()
```

```
        if valemail is None:
                raise ValidationError('This Email ID is not registered.
                Please register before login')
```

3.2 Main.py

In deze module staat alles wat de API uitvoert.

3.2.1 Libraries

Voor de API maak ik gebruik van verschillende bibliotheken. De voornaamste is Flask. daarnaast maak ik gebruik van flask-bcrypt om wachtwoorde te hashen, flaks-qrcode om qrcodes te maken. Daarnaast gebruik ik flask-login voor alles wat te maken heet met sessies. Verder worde de forms uit de Forms module gebruikt. Tenslotte gebruik ik nog secrets voor een geheime code en datatime voor de tijdsnotaties.

```
from flask_bcrypt import Bcrypt
from flask_qrcode import QRcode
from flask import Flask
from flask import render_template, url_for, flash, request,
redirect, Response
import sqlite3
from flask_login import LoginManager, UserMixin, login_required,
login_user, logout_user, current_user
from forms import LoginForm, RegisterForm, Create_classForm,
Register4classForm, Create_Course
from datetime import datetime as dt
import secrets
```

3.2.2 Configuration

De secret key is een geheime sleutel voor de sessies. Op de tweede lijn staat waar de API de html templates moet halen. Verder wordt de bcrypt, login-manager en qrcode generator toegevoegd aan de app. Tenslotte wordt de SECRET-KEY voor de applicatie gekozen. De BASE staat nu leeg maar wordt later gebruikt bij de qrcode om de remote adres in op te slaan.

```
secret_key = secrets.token_hex(16)
app = Flask(__name__,template_folder='templates')
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = "login"
QRcode(app)
app.config['SECRET_KEY'] = secret_key
BASE = ''
```

3.2.3 CLASS USER

Deze klasse wordt opgeroepen elke keer een gebruiker inlogt. Met deze klasse kunnen we de gebruiker verder gebruiken in andere functies.

```
class User(UserMixin):
     def __init__(self, id, email, password, firstname, lastname, admin):
          self.id = id
          self.email = email
          self.password = password
          self.firstname = firstname
          self.lastname = lastname
          self.adm = admin
     def is_active(self):
         return self.is_active()
     def is_anonymous(self):
         return False
     def is_authenticated(self):
         return self.authenticated
     def is_active(self):
         return True
     def get_id(self):
         return self.id
     def get_admin(self):
          if self.adm == 1:
               return True
          else:
               return False
```

3.2.4 Login-manager

Met deze functie kunnen we een gebruiker aanmaken met de behulp van de user klasse

```
@login_manager.user_loader
def load_user(user_id):
   conn = sqlite3.connect('database.db')
   curs = conn.cursor()
   curs.execute("SELECT * from user where userid = (?)",[user_id])
   lu = curs.fetchone()
   conn.close()
   if lu is None:
      return None
   else:
      return User(int(lu[0]), lu[1], lu[2], lu[3], lu[4], lu[5])
```

3.2.5 @app.route

Telkens als de server een request krijgt dan controleert hij of deze request voldoet aan bepaalde voorwaarden en handelt deze af zoals geprogrameerd.

login De eerste route is de login route. Hierbij maakt hij een .html aan met daarin de loginform. In deze form wordt de e-mail en wachtwoord gevraagd. Deze gegevens controleert hij. Als de gegevens correct zijn dan logt hij de gebruiker aan en stuurt hij hem door naar de home pagina of naar de pagina waar de gebruiker voor de log in achter vroeg.

```
@app.route("/login", methods=['GET','POST'])
def login():
    if current_user.is_authenticated:
        return redirect(request.args.get("next") or url_for("index"))
    form = LoginForm()
    if form.validate_on_submit():
        try:
            conn = sqlite3.connect('database.db')
            curs = conn.cursor()
            curs.execute("SELECT * FROM user where mail = (?)",[form.email.data])
            user = curs.fetchone()
            Us = load_user(user[0])
            conn.close()
            if form.email.data == Us.email
            and bcrypt.check_password_hash(Us.password, form.password.data):
                login_user(Us, remember=form.remember.data)
                name = Us.firstname + ' ' + Us.lastname
                flash('Logged in successfully '+ name)
                return redirect(request.args.get("next") or url_for("index"))
            else:
                flash('Password or email wrong.')
        except:
            flash('You don not have a acount.')
    return render_template('login.html',title='Login', form=form)
```

Logout Bij deze functie wordt de user gewoon uitgelogd

```
@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('You have successfully logged yourself out.')
    return redirect(url_for('index'))
```

Index Bij deze route zijn er twee functies. Eén functie stuurt de gebruiker naar de home-pagina als hij is ingelogd of naar de login indien dit nog niet was

gebeurt. De tweede, de BASE functie haalt het ip-adres en poort op van de server

```
@app.route("/")
def index():
     if current_user.is_authenticated:
          return redirect(url_for('home'))
     else: return redirect(url_for('login'))
def Base():
    return request.remote_addr
```

Register Bij deze functie probeert hij een gebruiker te registreren. Hij controleert of de email al in de databank staan.

```
@app.route("/register", methods=['GET','POST'])
def register():
     form = RegisterForm()
     if form.validate_on_submit():
          try:
               conn = sqlite3.connect('database.db')
               curs = conn.cursor()
               curs.execute("INSERT INTO user (mail, password, firstname,
               lastname, role) values (?,?,?,?,0)",(form.email.data,
               bcrypt.generate_password_hash(form.password.data),
               form.first.data, form.last.data))
               conn.commit()
               conn.close()
               flash("registration succesfull")
               return redirect(url_for('home'))
          except:
               flash("E-mail already registerd")
     return render_template('register.html',title='register', form=form)
```

Home Dit is de home pagina en renderd een de home.html

```
@app.route('/home')
@login_required
def home():
     return render_template("home.html")
```

Courses Hierbij haalt de functie alle courses waarvoor de gebruiker is ingeschreven, stel dat de gebruiker een leerkracht is voor de cursussen dan kan hier hierop verder klikken. Stel dat de gebruiker een admin is, dan wordt hij/zij doorgestuurd naar de pagina voor admins.

```
@app.route('/courses')
@login_required
```

```
def courses():
     user = current_user
     if current_user.get_admin():
          return redirect(url_for('admin'))
     con = sqlite3.connect('database.db')
     con.row_factory = sqlite3.Row
     cur = con.cursor()
     cur.execute("SELECT course.courseid, course.name, course.semester, course.year,
     usercourse.role FROM course INNER JOIN USERCOURSE ON usercourse.courseid =
     course.courseid WHERE usercourse.userid = ? AND usercourse.role =
     'student'",[int(user.get_id())])
     students = cur.fetchall()
     cur.execute("SELECT course.courseid, course.name, course.semester, course.year,
     usercourse.role FROM course INNER JOIN USERCOURSE ON usercourse.courseid =
     course.courseid WHERE usercourse.userid = ? AND usercourse.role =
     'teacher'",[int(user.get_id())])
     teachers = cur.fetchall()
     con.close()
     return render_template("courses.html", students=students,
     teachers = teachers, title='courses')
```

Course Deze functie laadt een webpagina die alle gebruikers weergeeft die zijn ingeschreven en alle klassen weergeeft. Op de klassen kan je verder klikken om meer info over de klas te krijgen. Mocht de gebruiker geen leerkracht zijn van de course dan wordt hem de toegang ontzegd en krijgt hij een mooie 403 pagina te zien.

```
@app.route('/course/<course_id>')
@login_required
def course(course_id):
     us = current_user
     con = sqlite3.connect('database.db')
     con.row_factory = sqlite3.Row
     cur = con.cursor()
     cur.execute("SELECT role FROM usercourse WHERE userid= ? AND courseid =
     ?",[int(us.get_id()),int(course_id)])
     role = cur.fetchone()
     if role[0] != 'teacher':
          return render_template("403.html")
     else:
          con = sqlite3.connect('database.db')
          con.row_factory = sqlite3.Row
          cur = con.cursor()
          cur.execute("SELECT user.firstname, user.lastname, user.mail,
          usercourse.role FROM user INNER JOIN usercourse ON user.userid =
          usercourse.userid WHERE usercourse.courseid=?",[course_id])
```

```
        users = cur.fetchall()
        cur.execute("SELECT classid, start, end FROM class WHERE
        courseid=?",[course_id])
        dates = cur.fetchall()
        con.close()
        return render_template("course.html", users = users,dates = dates,
        course_id = course_id, role = role)
```

Create-Class Op de vorige pagina kan je klassen aanmaken. Op de pagina kan je kiezen wanneer de klas begint en eindigt. Als de klas succesvol is aangemaakt dan keert hij terug naar de course pagina.

```
@app.route('/create_class/<course_id>', methods=['GET','POST'])
@login_required
def create_class(course_id):
    us = current_user
    con = sqlite3.connect('database.db')
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    cur.execute("SELECT role FROM usercourse WHERE userid= ? AND courseid = ?",
    [int(us.get_id()),int(course_id)])
    role = cur.fetchone()
    if role[0] != 'teacher':
        return render_template("403.html")
    else:
        print("help")
        form = Create_classForm()
        if form.validate_on_submit():
            start_str = form.year.data + '-'+ form.month.data + '-' +
            form.day.data + ' ' +  form.start_h.data + ':' +
            form.start_m.data
            end_str = form.year.data + '-'+ form.month.data + '-' +
            form.day.data + ' ' +  form.end_h.data + ':' +
            form.end_m.data
            start = dt.strptime(start_str, '%Y-%m-%d %H:%M')
            end = dt.strptime(end_str, '%Y-%m-%d %H:%M')
            if end > start:
                try:
                    conn = sqlite3.connect('database.db')
                    curs = conn.cursor()
                    curs.execute("INSERT INTO class (courseid, start, end)
                    values (?,?,?)",(course_id, start, end))
                    conn.commit()
                    conn.close()
                    flash("registration succesfull")
                    return redirect(url_for('course', course_id = course_id))
```

```
                    except:
                        flash("Didn't create class")
                else: flash ("End before start")
    return render_template('create_class.html', form = form)
```

Remove class Als je bij de course pagine klikt om de klas teverwijderen, dan controleert hij eerst of de gebruiker een leerkracht is en verwijdert en als het correct is, verwijdert hij de klas. Daarbij verwijdert hij ook alle entries in de Attendance Entiteit

```
@app.route('/remove_class/<class_id>')
@login_required
def remove_class(class_id):
    us = current_user
    con = sqlite3.connect('database.db')
    con.row_factory = sqlite3.Row
    cur = con.cursor()
    cur.execute("SELECT usercourse.role FROM USERCOURSE INNER JOIN CLASS ON
    USERCOURSE.courseid = CLASS.courseid WHERE class.classid = ? AND
    usercourse.userid = ?",[str(class_id),str(us.get_id())])
    role = cur.fetchone()
    con.close()
    if role[0] == 'teacher':
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        cur.execute("SELECT courseid FROM CLASS WHERE classid = ?",[str(class_id)])
        course_id = cur.fetchone()[0]
        cur.execute("DELETE FROM class WHERE Classid = ?", [str(class_id)])
        con.commit()
        cur.execute("DELETE FROM attendance WHERE Classid = ?", [str(class_id)])
        con.commit()
        con.close()
        return redirect(url_for('course', course_id = course_id))
    else:
        return render_template("403.html")
```

Class Op deze pagina staat de qr-code van de klas en alle aanwezigen

admin Deze functie lijst alle courses op en zorgt er voor dat je hier op kan verder klikken. je kan de course ook verwijderen of aanmaken.

```
@app.route('/admin')
@login_required
def admin():
    if current_user.get_admin():
```

```
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        cur.execute("SELECT * FROM  COURSE ORDER BY year DESC",)
        courses = cur.fetchall()
        con.close()
        return render_template("adcourse.html", courses = courses, title="Admin courses")
    else:
        return render_template("403.html")
```

Admin Course Dit is de pagina waarbij de admin de details ziet van de course en gebruikers kan toevoegen en verwijderen.

```
@app.route('/admin/<course_id>')
@login_required
def adminCourse(course_id):
    if current_user.get_admin():
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        cur.execute("SELECT DISTINCT user.userid, user.firstname, user.lastname,
        user.mail FROM USER WHERE USER.userid NOT IN (SELECT DISTINCT USER.userid
        FROM  USER INNER JOIN USERCOURSE on user.userid = usercourse.userid WHERE
        usercourse.courseid=?)", [course_id])
        users = cur.fetchall()
        cur.execute("SELECT DISTINCT user.userid, user.firstname, user.lastname,
        user.mail, usercourse.role FROM  USER INNER JOIN USERCOURSE on user.userid
        usercourse.userid WHERE usercourse.courseid=?", [course_id])
        students = cur.fetchall()
        con.close()
        return render_template("aduser2course.html",
        title="Admin courses", users=users, course_id=course_id,
        students=students)
    else:
        return render_template("403.html")
```

Remove from course

```
@app.route('/remove4course')
@login_required
def remove4course():
    if current_user.get_admin():
        course_id = request.args.get('course_id', None)
        user_id = request.args.get('user_id', None)
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row
```

```
            cur = con.cursor()
            cur.execute("DELETE FROM USERCOURSE WHERE userid = ? AND courseid=?",
            [user_id,course_id])
            con.commit()
            con.close()
            return redirect(url_for('adminCourse', course_id = course_id))
      else:
            return render_template("403.html")
```

Add to course

```
@app.route('/add2course')
@login_required
def ad2course():
      if current_user.get_admin():
            course_id = request.args.get('course_id', None)
            user_id = request.args.get('user_id', None)
            role = request.args.get('role', None)
            con = sqlite3.connect('database.db')
            con.row_factory = sqlite3.Row
            cur = con.cursor()
            cur.execute("INSERT INTO usercourse (userid, courseid, role) VALUES
            (?,?,?)", [user_id,course_id,role])
            con.commit()
            con.close()
            return redirect(url_for('adminCourse', course_id = course_id))
      else:
            return render_template("403.html")
```

Add course Hier maakt de API een course aan. Hij controleert of de course nog niet bestaat en als dat in orde is, maakt hij de course aan.

```
@app.route('/addCourse', methods=['GET','POST'] )
@login_required
def addCourse():
      if current_user.get_admin():
            form = Create_Course()
            if form.validate_on_submit():
                  name = form.name.data
                  semester = form.semester.data
                  year = form.year.data
                  con = sqlite3.connect('database.db')
                  con.row_factory = sqlite3.Row
                  cur = con.cursor()
                  cur.execute("SELECT courseid FROM course where name = ? AND
                  semester = ? AND year = ?", [name, semester, year])
```

```
                course = cur.fetchone()
                con.close()
                if course == None:
                    con = sqlite3.connect('database.db')
                    con.row_factory = sqlite3.Row
                    cur = con.cursor()
                    cur.execute("INSERT INTO COURSE (name, semester, year) VALUES
                    (?,?,?)", [name, semester, year])
                    con.commit()
                    con.close()
                    return redirect(url_for('admin'))
                else: flash("Course already exists")
            return render_template("addCourse.html", form=form)
        else:
            return render_template("403.html")
```

Remove course

```
@app.route('/removeCourse/<course_id>')
@login_required
def removeCourse(course_id):
    if current_user.get_admin():
        con = sqlite3.connect('database.db')
        con.row_factory = sqlite3.Row
        cur = con.cursor()
        cur.execute("DELETE FROM course WHERE courseid = ?", [course_id])
        con.commit()
        cur.execute("DELETE FROM usercourse WHERE courseid = ?", [course_id])
        con.commit()
        cur.execute("DELETE FROM class WHERE courseid = ?", [course_id])
        con.commit()
        con.close()
        return redirect(url_for('admin'))
    else:
        return render_template("403.html")
```

3.3 403 en 404

404 Als de server een bepaalde request niet kan verwerken omdat het onmogelijk is, krijgt de gebruiker een 404 pagina.

```
@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404
```

403 Als de gebruiker geen toegang heeft tot bepaalde pagina's maar toch probeert via een url of dergelijke dan krijgt hij een 403 pagina.

```
@app.errorhandler(403)
def page_not_found(e):
    return render_template('403.html'), 404
```