```python
# This is all the form classes, they are imported and intansiated in the routes.py file
# and used to build form objects in the flask app which are validated on submission
# of the forms by the user.


from flask_wtf import FlaskForm, Form
from flask_wtf.file import FileField, FileAllowed
from flask_login import current_user
from wtforms import StringField, PasswordField, SubmitField, BooleanField, TextAreaField, DateField, IntegerField,
SelectField, FormField, FieldList
from wtforms.validators import DataRequired, Optional, Length, Email, NumberRange, EqualTo, ValidationError
from wtforms.fields.html5 import EmailField, DateField
from projectCode.models import User, Class, Course
import datetime
import re


# Generic search form for students, topics, users, posts e.t.c.
class SearchForm(FlaskForm):
    search_query = StringField('Search', validators=[DataRequired()])
    submit = SubmitField('đ')


# Registration form for new users.
class RegistrationForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')

    # Checking that the username entered is unique.
    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError('That username is taken. Please choose a different one.')

    # Checking that email is not taken by another user.
    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError('That email is taken. Please choose a different one.')


class LoginForm(FlaskForm):
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember = BooleanField('Remember Me') # Check if the user wants a cookie for login.
    submit = SubmitField('Login')


class UpdateAccountForm(FlaskForm):
    username = StringField('Username',
                           validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    picture = FileField('Update Profile Picture', validators=[FileAllowed(['jpg', 'png'])])
    submit = SubmitField('Update')

    # Checking that the username entered is unique.
    def validate_username(self, username):
        if username.data != current_user.username:
            user = User.query.filter_by(username=username.data).first()
            if user:
                raise ValidationError('That username is taken. Please choose a different one.')

    # Checking that email is not taken by another user.
    def validate_email(self, email):
        if email.data != current_user.email:
            user = User.query.filter_by(email=email.data).first()
            if user:
                raise ValidationError('That email is taken. Please choose a different one.')


class PostForm(FlaskForm):
    title = StringField('Title', validators=[DataRequired(), Length(max=100)])
    content = TextAreaField('Content', validators=[DataRequired()])
    submit = SubmitField('Post')


class RequestResetForm(FlaskForm):
    email = StringField('Email',
                        validators=[DataRequired(), Email()])
    submit = SubmitField('Request Password Reset')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user is None:
            raise ValidationError('There is no account with that email. You must register first.')


class ResetPasswordForm(FlaskForm):
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password',
                                     validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Reset Password')


class ClassForm(FlaskForm):
    class_name = StringField('Class Name', validators=[DataRequired(), Length(max=50)])
    course_id = SelectField('Add To Course', coerce=int)
    class_starting_date = DateField('Academic Start Date', validators=[DataRequired("Date before year 3000 is required")])
    submit = SubmitField('Submit')


class StudentForm(FlaskForm):
    class_id = SelectField('Add To Class', coerce=int, validators=[DataRequired()])
    name = StringField('Student Name', validators=[DataRequired(), Length(max=50)])
    email = EmailField('Student Email', validators=[DataRequired(), Email()])
    address = TextAreaField('Student Address', validators=[DataRequired()])
    parent_phone = StringField('Parent Phone Number', validators=[DataRequired()])
    predicted_grade = StringField('Predicted Grade', validators=[DataRequired()])
    submit = SubmitField('Submit')

    def validate_parent_phone(self, parent_phone):
        rule = re.compile(r'^(07[\d]{8,12}|447[\d]{7,11})$')
        if not rule.search(parent_phone.data):
            msg = "Invalid mobile number."
            raise ValidationError(msg)


class AddStudentToClass(FlaskForm):
    class_id = SelectField('Add To Class', coerce=int, validators=[DataRequired()])
    submit = SubmitField('Submit')


class RemoveStudentFromClass(FlaskForm):
    class_id = SelectField('Remove From Class', coerce=int, validators=[DataRequired()])
    submit = SubmitField('Submit')


class AddStudentsToClass(FlaskForm):
    students = SelectField('Add Students', coerce=int, validators=[DataRequired()])
    submit = SubmitField('Submit')


class TopicForm(FlaskForm):
    name = StringField('Topic Name', validators=[DataRequired(), Length(max=50)])
    course_id = SelectField('Add To Course', coerce=int)
    begin_date = DateField('Begin Date', validators=[DataRequired("Date before year 3000 is required")])
    end_date = DateField('End Date', validators=[DataRequired("Date before year 3000 is required")])
    submit = SubmitField('Submit')


class HomeworkForm(FlaskForm):
    name = StringField('Homework Name', validators=[DataRequired()])
    max_mark = IntegerField('Maximum Mark (Numbers Only)', validators=[DataRequired()])
    due_date = DateField('Due Date', validators=[DataRequired()])
    submit = SubmitField('Submit')

    def validate_max_mark(self, mark):
        if mark.data > 10000:
            raise ValidationError("Mark must be bellow 10000.")


class TestForm(FlaskForm):
    name = StringField('Test Name', validators=[DataRequired()])
    max_mark = IntegerField('Maximum Mark (Numbers Only)', validators=[DataRequired()])
    date = DateField('Date', validators=[DataRequired()])
    submit = SubmitField('Submit')


class ExamForm(FlaskForm):
    name = StringField('Exam Name', validators=[DataRequired(), Length(max=50)])
    date = DateField('Date', validators=[DataRequired("Date before year 3000 is required")])
    max_mark = IntegerField('Maximum Mark (Numbers Only)', validators=[DataRequired("Integer Required")])
    submit = SubmitField('Submit')

class CommentForm(FlaskForm):
    comment = TextAreaField('Leave A Comment', validators=[DataRequired()])
    submit = SubmitField('Submit')


class CourseForm(FlaskForm):
    name = StringField('Course Name', validators=[DataRequired(), Length(max=100)])
    start_date = DateField('Start Date', validators=[DataRequired("Date before year 3000 is required")])
    year_num = IntegerField('Course Length (In Years)', validators=[DataRequired("Must enter valid integer")])
    grade_system = SelectField('Grading System', coerce=int)
    submit = SubmitField('Submit')

    def validate_year_num(self, year_num):
        if year_num.data > 50:
            raise ValidationError("Year number cannot exceed 50 years.")


class HomeworkMarkForm(FlaskForm):
    marks = FieldList(StringField('Mark'), min_entries=5)
    submit = SubmitField('Submit')


class TestMarkForm(FlaskForm):
    marks = FieldList(StringField('Mark'), min_entries=5)
    submit = SubmitField('Submit')
```

```python
class ExamMarkForm(FlaskForm):
    marks = FieldList(StringField('Mark'), min_entries=5)
    submit = SubmitField('Submit')
```