Project Files

./config.py

```
# CSRF form protection
WTF_CSRF_ENABLED = False
# Database connection
SQLALCHEMY_DATABASE_URI = 'sqlite:///database.db'
# Secret key for encryption
SECRET_KEY = 'A0Zr98j/3yX R~XHH!jmN]LWX/,?RT'

# Email settings
MAIL_SERVER = 'smtp.googlemail.com'
MAIL_PORT = 465
MAIL_USE_TLS = False
MAIL_USE_SSL = True
MAIL_USERNAME = 'testapp545545'
MAIL_PASSWORD = 'January28'
```

./run.py

```
import os
from app import app

if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    app.debug = True
    app.run(host='0.0.0.0', port=port)
```

./app/__init__.py

```
from flask import Flask
from flask_login import LoginManager
from flask_sqlalchemy import SQLAlchemy


from flask_mail import Mail

# Create app and initialize flask
app = Flask(__name__)

#load configuration options from config.py
app.config.from_object('config')

# Initialize flask-login and set up login manage
login_manager = LoginManager()
login_manager.init_app(app)
# The view which flask login redirects to if user is not logged in and trys to access restricted
view
login_manager.login_view = 'user.login'

# Initialize database with sqlalchemy
db = SQLAlchemy(app)

# Initialize flask-mail
mail = Mail(app)

# Import and register all blueprints
from .routes.matrix import matrix_blueprint
from .routes.questions import questions_blueprint
from .routes.loci import loci_blueprint
from .routes.user import user

# Blueprints used to break up larger app into smaller modules
app.register_blueprint(matrix_blueprint)
app.register_blueprint(loci_blueprint)
app.register_blueprint(questions_blueprint)
```

```
app.register_blueprint(user)

# Import all other views and database models
from app import models, views
```

./app/forms.py

```
from flask_wtf import FlaskForm
from wtforms import (BooleanField, PasswordField, SelectMultipleField,
                     SubmitField, TextField, validators, widgets)
from wtforms.fields.html5 import EmailField

from .pyscripts.question_dict import QUESTIONS


class RegisterForm(FlaskForm):
    """Form to register a new user (student only)"""

    fname = TextField('First Name', [validators.Required()])
    lname = TextField('Last Name', [validators.Required()])
    password = PasswordField('Password', [validators.Required()])
    confirm_password = PasswordField('Confirm Password', [validators.Required(
    ), validators.EqualTo('password', message='Passwords do not match')])
    email = EmailField('Email Address', [
                        validators.DataRequired(), validators.Email()])


class LoginForm(FlaskForm):
    """Form for a user to log in"""

    email = TextField('Username', [validators.Required()])
    password = PasswordField('Password', [validators.Required()])
    remember = BooleanField('Remember')


class RequestPasswordChangeForm(FlaskForm):
    """Form to request password change email to be sent"""

    email = TextField('Email', [validators.Required(), validators.Email()])


class ChangePasswordForm(FlaskForm):
    """Form for changing password (coming from email)"""

    password = PasswordField('Password', [validators.Required()])
    confirm_password = PasswordField('Confirm', [validators.Required(
    ), validators.EqualTo('password', message='Passwords do not match')])


class TeacherLinkForm(FlaskForm):
    """Form for students account page to link to teachers"""

    link_code = TextField('Link Code', [validators.Required()])
    link_submit = SubmitField('Go')


class MultiCheckboxField(SelectMultipleField):
    """Field for SetTaskForm (multiple select with checkboxes)"""
    widget = widgets.ListWidget(prefix_label=False)
    option_widget = widgets.CheckboxInput()


class SetTaskForm(FlaskForm):
    """Form for teachers account page to set students tasks"""
    student_select = MultiCheckboxField('Students',
                                        [validators.Required()], coerce=int)
    task_select = MultiCheckboxField('Tasks',[validators.Required()], coerce=int,
choices=[(q['id'], q['topic'] + '  ' + q['name']) for q in QUESTIONS])
    set_submit = SubmitField('Go')

    def __init__(self, selection_choices):
```

```
        """Override init so form can be initialized with custom choices."""
        super(SetTaskForm, self).__init__()
        self.student_select.choices = selection_choices


class ChangeDetailsForm(FlaskForm):
    """Form for teacher and student account page for changing name(s), email."""

    fname = TextField('First Name', [validators.Required()])
    lname = TextField('Last Name', [validators.Required()])
    email = EmailField('Email Address',
                       [validators.DataRequired(), validators.Email()])
    password = PasswordField('Password', [validators.Required()])
    change_submit = SubmitField('Go')


class ChangePasswordForm1(FlaskForm):
    """Form for changing password from account page."""

    old_password = PasswordField('Old Password', [validators.Required()])
    password = PasswordField('New Password', [validators.Required()])
    confirm_password = PasswordField('Confirm New Password', [validators.Required(),
validators.EqualTo('password', message='Passwords do not match')])
    pw_submit = SubmitField('Go')
```

./app/models.py

```
from app import db
import datetime
from werkzeug.security import generate_password_hash, check_password_hash
import random
import string


class User(db.Model):
    """User class for flask-login and storing user data"""
    user_id = db.Column('user_id', db.Integer, primary_key=True)
    fname = db.Column(db.String(80), unique=False)
    lname = db.Column(db.String(80), unique=False)
    email = db.Column(db.String(120), unique=True)
    password = db.Column(db.String(120), unique=False)
    authenticated = db.Column(db.Boolean, default=False)
    confirmed = db.Column(db.Boolean)
    role = db.Column(db.String(50))
    # One to many (one user - teachers and students - has many graphs)
    graphs = db.relationship('Graph', backref="user",
                             cascade="all, delete-orphan", lazy="dynamic")

    __mapper_args__ = {
        'polymorphic_identity': 'user',
        'polymorphic_on': role
    }

    def __init__(self, fname, lname, email, password, role,
                 auth=False, conf=False):
        self.fname = fname
        self.lname = lname
        self.email = email
        self.role = role
        self.password = generate_password_hash(password)
        self.authenticated = auth
        self.confirmed = conf

    def check_pw(self, password):
        return check_password_hash(self.password, password)

    def is_authenticated(self):
        return self.authenticated

    def is_active(self):
        return True
```

```
    def is_anonymous(self):
        return False

    def get_id(self):
        return self.email


class Teacher(User):
    """Teacher model inheriting from User."""

    __tablename__ = 'teacher'
    teacher_id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.user_id'))
    code = db.Column(db.String(7))
    # Many to many
    students = db.relationship('Student', secondary='links',
                               backref=db.backref('teachers', lazy='dynamic'),
                               lazy='dynamic')
    # One to many (One teacher sets many tasks)
    tasks = db.relationship('Task', backref="teacher",
                            cascade="all, delete-orphan", lazy="dynamic")

    def __init__(self, fname, lname, email, password, role):
        super().__init__(fname, lname, email, password, role)
        #Create random 7 character code
        self.code = ''.join(random.choice(string.ascii_letters)
                            for x in range(7))

    def add_student(self, student):
        """Return none id student already added, else add student to teacher
            and return new teacher object"""

        if not self.has_student(student):
            self.students.append(student)
            return self

    def remove_student(self, student):
        """Return none is student already removed, else remove student and
            return new teacher object"""
        if self.has_student(student):
            self.students.remove(student)
            return self

    def has_student(self, student):
        """ Check if teacher already has student by performing query with student id"""
        return self.students.filter(links.c.student_id == student.student_id).count() > 0

    __mapper_args__ = {
        'polymorphic_identity': 'teacher',
    }


class Student(User):
    """Student model inheriting from User."""

    __tablename__ = 'student'
    student_id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.user_id'))
    marks = db.relationship('Mark', backref="student",
                            cascade="all, delete-orphan", lazy="dynamic")
    tasks = db.relationship('Task', backref="student",
                            cascade="all, delete-orphan", lazy="dynamic")

    __mapper_args__ = {
        'polymorphic_identity': 'student',
    }

# Many-to-many intermediate table, linking teacher and student id
links = db.Table('links',
                 db.Column('teacher_id', db.Integer,
                           db.ForeignKey('teacher.teacher_id')),
                 db.Column('student_id', db.Integer,
                           db.ForeignKey('student.student_id'))
```

```
                    )


class Mark(db.Model):
    """Mark model."""

    mark_id = db.Column('mark_id', db.Integer, primary_key=True)
    score = db.Column(db.Integer)
    out_of = db.Column(db.Integer)
    date = db.Column(db.DateTime)
    # The id for the type of question (dictionary stored in QUESTION_DICT.py)
    question_id = db.Column(db.Integer)
    # Many to one (many marks to each student)
    student_id = db.Column('student_id', db.Integer,
                           db.ForeignKey('student.student_id'))
    # One to one
    # When a task is completed, a mark (id) is linked to it which is the mark the
    # student got on that task
    task = db.relationship('Task', uselist=False, back_populates='mark')

    def __init__(self, score, out_of, q_id, student_id):
        self.score = score
        self.out_of = out_of
        self.question_id = q_id
        self.student_id = student_id
        self.date = datetime.date.today()


class Task(db.Model):
    """Model for task which is set by a teacher."""

    task_id = db.Column('task_id', db.Integer, primary_key=True)
    completed = db.Column(db.Boolean)
    question_id = db.Column(db.Integer)
    student_id = db.Column('student_id', db.Integer,
                           db.ForeignKey('student.student_id'))
    teacher_id = db.Column('teacher_id', db.Integer,
                           db.ForeignKey('teacher.teacher_id'))
    mark_id = db.Column('mark_id', db.Integer, db.ForeignKey('mark.mark_id'))
    # One to one
    # When a task is completed, a mark (id) is linked to it which is the mark the
    # student got on that task
    mark = db.relationship('Mark', back_populates='task')

    def __init__(self, q_id, student_id, teacher_id):
        self.question_id = q_id
        self.student_id = student_id
        self.teacher_id = teacher_id
        self.completed = False


class Graph(db.Model):
    """Model for graph from loci plotter."""

    graph_id = db.Column('graph_id', db.Integer, primary_key=True)
    title = db.Column(db.String,nullable=False)
    # Description is not necessary, so nullable is true
    description = db.Column(db.String, nullable=True)
    # Graph data from desmos, very large string
    desmosdata = db.Column(db.String)
    # HTML for expressions table
    exprlist = db.Column(db.String)
    date = db.Column(db.DateTime)
    # Screenshot image data
    image_url = db.Column(db.String)
    user_id = db.Column('user_id', db.Integer, db.ForeignKey('user.user_id'))

    def __init__(self, desmosdata, exprlist, user_id, title, desc, image_url):
        self.desmosdata = desmosdata
        self.exprlist = exprlist
        self.user_id = user_id
        self.title = title
        self.description = desc
        self.image_url = image_url
```

```
        self.date = datetime.date.today()
```

./app/views.py

```
from flask import render_template

from app import app, login_manager
from app.models import User


@login_manager.user_loader
def user_loader(email):
    """Return user object for Flask-Login."""
    return User.query.filter_by(email=email).first()


@app.route('/')
def main():
    return render_template('index.html')


@app.route('/help')
def help():
    return render_template('help.html')


@app.route('/ttt')
def ttt():
    return render_template('ttt.html')


@app.route('/cttt')
def cttt():
    return render_template('cttt.html')
```

./app/pyscripts/__init__.py

```
from . import *
```

./app/pyscripts/base_question.py

```
class BaseQuestion(object):
    """Base question which matrix and complex question inherit from."""

    def __init__(self, question, answer, q_type):
        """Constructor for simple question object."""
        self.question = question
        self.answer = answer
        self.question_type = q_type

    def get_q(self):
        """Return question as string."""
        return str(self.question)

    def get_answer(self):
        """Return answer."""
        return self.answer
```

./app/pyscripts/complex_loci.py

```
from sympy import symbols, re, im, sqrt, atan, sympify, latex

# TODO instead of subbing Abs(), find sqrt(re(x)+im(x)) instead.


def parse_mod(inner):
    """Convert expression inside modulus to x-y equation."""
    # Set up sympy variables and convert inner expression to sympy object
```

```
    x, y = symbols('x y', real=True)
    locs = {'x': x, 'y': y}
    in_eq = sympify(inner, locs)
    # Return string version of formula with real and imaginary parts substituted
    return str(sqrt(im(in_eq)**2 + re(in_eq)**2))


def parse_arg(inner):
    """Convert expression inside arument function to x-y equation."""
    # Set up sympy variables and convert inner expression to sympy object
    x, y = symbols('x y', real=True)
    locs = {'x': x, 'y': y}
    in_eq = sympify(inner, locs)
    # Return string version of formula with real and imaginary parts substituted
    return str(2 * atan((sqrt(re(in_eq)**2 + im(in_eq)**2) - re(in_eq)) / im(in_eq)))


def parse(eq):
    """Return string of manipulated equation."""
    eq = eq.replace('z', 'Z').replace('^', '**')
    # Make the string a list for easier manipulation
    eq_list = list(eq)
    # Sympy recognises uppercase I as sqrt(-1)
    # Any isolated i's (not within another word like pi) should be converted to uppercase
    eq_list = ['I' if ch == 'i' and eq_list[n - 1]
               not in ['p', 'P', 's', 'S'] else ch for n, ch in enumerate(eq_list)]
    nums = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
    # Insert a * after a number and before a letter, bracket or modulus sign
    # Allows users enter 2z to mean 2 * z
    for n, ch in enumerate(eq_list):
        if (ch in ['Z', 'I', 'A', 'a', 'p', 'P', '(']) and (eq_list[n - 1] in nums) and n > 0:
            eq_list.insert(n, '*')
    # Convert back into string
    eq = ''.join(eq_list)
    # Substitute z for x+yi
    eq = eq.replace('Z', '(x+y*I)')

    # While there are still modulus lines
    while '|' in eq:
        # Find first modulus line
        a = eq.find('|')
        # Find matching line
        b = eq.find('|', a+1)
        # Parse everything between modulus lines according to formula
        eq = eq[:a] + parse_mod(eq[a + 1:b]) + eq[b + 1:]

    # While there is still an argument funcion in the equation
    while 'arg(' in eq:
        # Find occurence of function
        # a is the index of the start of the inner expression
        # Also initialise b to this value
        a, b = eq.find('arg(') + 4, eq.find('arg(') + 4
        # Find correct enclosing bracket
        found = 0
        while found >= 0:
            # Add 1 if open bracket, subtract 1 if close bracket
            found += 1 if eq[b] == '(' else 0
            found -= 1 if eq[b] == ')' else 0
            # Increment counter
            b += 1
            # When found is less than 0, then the correct close bracket is found
        # Parse everything between modulus lines according to formula
        eq = eq[:a - 4] + parse_arg(eq[a:b - 1]) + eq[b:]
    return eq

def parse_inequality(eq):
    """Return type of (in)equality as string"""
    if '<=' in eq:
        return '<='
    elif '>=' in eq:
        return '>='
    elif '>' in eq:
        return '>'
    elif '<' in eq:
```

```
        return '<'
    else:
        return '='

def get_implicit(eq, latx=False):
    """Return implicit equation in x and y"""
    # Get the type of (in)equality the expression is
    op = parse_inequality(eq)
    # Split it up depending on the type of expression
    lhs, rhs = eq.split(op)

    # Set up symbols
    x, y = symbols('x y', real=True)
    locs = {'x': x, 'y': y}
    # Parse both the left and right hand side
    # Convert to sympy objects based on set up variables
    lhs = parse(lhs)
    lhs = sympify(lhs, locals=locs)

    rhs = parse(rhs)
    rhs = sympify(rhs, locals=locs)

    # Put equation in the form f(x,y)=0 and simplify
    eq = lhs - rhs
    eq = eq.simplify()
    if latx:
        # Convert to format desmos understands (latex)
        return ((latex(eq)) + ' ' + op + ' 0').replace('atan', 'arctan')
    else:
        return str(eq) + op + '0'


if __name__ == '__main__':
    print(parse('arg(z+1)-arg(z-1)'))
```

./app/pyscripts/complex_questions.py

```
from sympy import im, re
import random
from cmath import phase
from app.pyscripts.base_question import BaseQuestion


class ComplexQuestion(BaseQuestion):
    """Class for creating matrix questions, inherits from BaseQuestion."""

    @staticmethod
    def get_question(q_type):
        """Return a question object given a question type."""
        if q_type == 'add_sub':
            question, answer = add_sub_question()
        elif q_type == 'mult':
            question, answer = mult_question()
        elif q_type == 'div':
            question, answer = div_question()
        elif q_type == 'mod_arg':
            question, answer = mod_arg_question()
        else:
            raise ValueError
        return ComplexQuestion(question, answer, q_type)

    def is_mod_arg(self):
        """Return true if the object is a modulus argument question."""
        if self.question_type == 'mod_arg':
            return True
        else:
            return False


def add_sub_question():
    """Return random question and answer pair for addition and subtraction."""
    # Generate 2 random complex numbers
```

```
    a = random.randint(1, 10) + random.randint(1, 10) * 1j
    b = random.randint(1, 10) + random.randint(1, 10) * 1j
    # Choose random operatot
    rand_op = random.choice(['+', '-'])
    # Calculate answer and create question based on which opertor was chosen
    if rand_op == '+':
        answer = a+b
        question = "Calculate  `" + str(a).replace('j', 'i') + "` + `"+str(b).replace('j', 'i')
+ "`"
    else:
        answer = a-b
        question = "Calculate  `" + str(a).replace('j', 'i') + "` - `"+str(b).replace('j', 'i')
+ "`"
    return (question, answer)


def mult_question():
    """Return random question and answer pair for multiplication."""
    # Generate 2 random complex numbers
    a = random.randint(1, 10) + random.randint(1, 10) * 1j
    b = random.randint(1, 10) + random.randint(1, 10) * 1j
    # Calculate answer and generate question with the random numbers
    answer = a * b
    question = "Calculate  `" + str(a).replace('j', 'i') + "` * `"+str(b).replace('j', 'i') +
"`"
    return (question, answer)


def div_question():
    """Return random question and answer pair for division."""
    # Generate 2 random complex numbers
    a = random.randint(1, 10) + random.randint(1, 10) * 1j
    b = random.randint(1, 10) + random.randint(1, 10) * 1j
    # Calculate answer and generate question with the random numbers
    ans = a / b
    # Round both parts to 2 dp
    answer = round(im(ans), 2) * 1j + round(re(ans), 2)
    question = "Calculate `" + str(a).replace('j', 'i') + " / "+str(b).replace('j', 'i') + "` (2
decimal places)"
    return (question, answer)


def mod_arg_question():
    """Return random question and answer pair for modulus and argument."""
    # Generate 2 random complex numbers
    a = random.randint(1, 10)+random.randint(1, 10) * 1j
    # Calculate answer and generate question with the random numbers
    mod = round(abs(a), 2)    # Round both to two dp
    arg = round(phase(a), 2)
    answer = (mod, arg)
    question = 'Find, to two decimal places, the modulus and argument of  `'+str(a).replace('j',
'i')+'`'
    return (question, answer)
```

./app/pyscripts/matrices.py

```
import fractions


class MatrixError(Exception):
    """An exception class for Matrix."""

    pass


class Matrix(object):
    """Class for matrix operations."""

    def __init__(self, rows):
        """Return Matrix object given 2D list."""
        self.y = len(rows)
        self.x = len(rows[0])
```

```
        for row in rows:
            if len(row) != self.x:
                # All rows have to be the same row for a valid matrix
                raise MatrixError
        else:
            self.rows = [list(row) for row in rows]

    def __add__(self, other):
        """Override add operator to add matrix objects."""
        # Can only add matrices to matrices
        if type(other) != Matrix:
            raise TypeError
        # Both matrices have to be the same size to add
        elif self.get_dimensions() != other.get_dimensions():
            raise MatrixError
        else:
            # Create empty result list
            result = []
            for y in range(self.y):
                # Add empty row to result list
                result.append([])
                for x in range(self.x):
                    # Add values and add to row
                    result[y].append(self.rows[y][x] + other.rows[y][x])
            return Matrix(result)

    def __sub__(self, other):
        """Override subtract operator to subtract matrix objects."""
        # Can only subtract matrices from matrices
        if type(other) != Matrix:
            raise TypeError
        # Both matrices have to be the same size to subtract
        elif self.get_dimensions() != other.get_dimensions():
            raise MatrixError
        else:
            # Create empty result list
            result = []
            for y in range(self.y):
                # Add empty row to result list
                result.append([])
                for x in range(self.x):
                    # Subtract values and add to row
                    result[y].append(self.rows[y][x] - other.rows[y][x])
            return Matrix(result)

    def __mul__(self, other):
        """Override multiply operator to multiply matrix objects."""
        # Can only multiply a matrix by an integer, float or another matrix
        if type(other) not in [Matrix, int, float]:
            raise TypeError
        # Can only multiply by another matrix if width of one is height of other
        elif type(other) == Matrix and self.get_dimensions()[1] != other.get_dimensions()[0]:
            raise MatrixError
        # If multiplying by a int or float, multiply each element in matrix by the int or float
        if type(other) == int or type(other) == float:
            result = [[0 for y in range(self.get_dimensions()[1])]
                      for x in range(self.get_dimensions()[0])]
            for y in range(self.y):
                for x in range(self.x):
                    result[y][x] = self.rows[y][x] * other
        # If multiplying by another matrix, more complex
        else:
            # Empty result 2-d array with zeroes with correct dimensions
            result = [[0 for y in range(other.get_dimensions()[1])]
                      for x in range(self.get_dimensions()[0])]
            # Get data from 2 matrix objects
            a = self.rows
            b = other.rows
            for row in range(len(a)):
                for column in range(len(b[0])):
                    tot = 0
                    for x in range(len(a[0])):
                        tot += a[row][x] * b[x][column]
                    result[row][column] = tot
        return Matrix(result)
```

```
def tostr(self):
     """Return Matrix with all elements converted to strings."""
     rows = self.rows
     return Matrix(
          [[str(rows[y][x]) for x in range(self.x)] for y in range(self.y)]
               )

def get_dimensions(self):
     """Return dimensions of the matrix as a tuple."""
     return (self.y, self.x)

def get_rows(self):
     """Get matrix data."""
     return self.rows

def transpose(self):
     """Returned transposed matrix object."""
     rows = self.rows
     # List comprehension to flip matrix
     return Matrix(
          [[rows[x][y] for x in range(self.x)] for y in range(self.y)]
               )

def determinant(self):
     """Return determinant of matrix as a float or int."""
     rows = self.rows
     if self.x != self.y:
          raise MatrixError
     if self.x == 2:
          # For a 2x2 matrix [[a,b],[c,d]], the determinant is a*d-b*c
          det = ((rows[0][0]) * (rows[1][1])) - (rows[0][1] * rows[1][0])
          return det
     else:
          # For larger matrices
          # Get the top row of the matrix
          top_row = rows[0]
          # Initialize det variable
          det = 0
          # Loop over each item in the top row
          for x in range(len(top_row)):
               # Find the inner matrix
               # (the matrix got when you delete the row and column the item is in)
               inner_mat = [[b for a, b in enumerate(
                    i) if a != x] for i in rows[1:]]
               inner_mat = Matrix(inner_mat)
               # Find the determinant of the inner matrix
               # Multiply by the value in the row
               # Add or subtract depending on where in the row it is
               det += (-1)**x * top_row[x] * inner_mat.determinant()
          return det

def display(self):
     """Print Matrix."""
     for row in self.rows:
          for value in row:
               print(value, end=" ")
          print('\n', end='')

def triangle(self):
     """Return Matrix in triangle form."""
     n = self.x
     rows = self.rows
     # Loop from 0 to width-1
     for i in range(n - 1):
          if rows[i][i] == 0:
               for j in range(i + 1, n):
                    if rows[j][i] == 0:
                         # If all elements in the column are 0, do not swap rows
                         continue
                    else:
                         # Swap rows numbered j and i
                         rows[j], rows[i] = rows[i], rows[j]
          else:
```

```
                for k in range(i + 1, n):
                    # Get ratio between item in row i and row k (under i)
                    ratio = fractions.Fraction(rows[k][i], rows[i][i])
                    for r in range(i, n, 1):
                        # Subtract this row from row above * the ratio
                        # Means there is a 0 in first column(s)
                        rows[k][r] -= ratio * rows[i][r]
        return Matrix(rows)

    def cofactors(self):
        """Return cofactor matrix object."""
        co_mat = self.rows
        for y in range(len(co_mat)):
            for x in range(len(co_mat[0])):
                inner_mat = [[b for a, b in enumerate(
                    j) if a != x] for i, j in enumerate(co_mat) if i != y]
                inner_mat = Matrix(inner_mat)
                co_mat[y][x] = (-1) ** (x + y) * inner_mat.determinant()
        return Matrix(co_mat)

    def adjoint(self):
        """Return adjoint of matrix."""
        return self.cofactors().transpose()

    def inverse(self):
        """Return inverse of matrix."""
        # Can only find inverse of a square matrix
        if self.x != self.y:
            raise MatrixError
        det = self.determinant()
        # Can only find inverse if determinant is not 0
        if det == 0:
            raise MatrixError
        # Find the transpose of the cofactor matrix (the adjoint)
        c_t = self.adjoint().get_rows()
        # Divide every item in c_t by the determinant
        for x in range(len(c_t)):
            for y in range(len(c_t[0])):
                c_t[x][y] = fractions.Fraction(c_t[x][y] / det).limit_denominator()
        return Matrix(c_t)
```

./app/pyscripts/matrix_questions.py

```
from app.pyscripts.matrices import Matrix
from app.pyscripts.base_question import BaseQuestion
import random


class MatrixQuestion(BaseQuestion):
    """Class for creating matrix questions, inherits from BaseQuestion."""

    @staticmethod
    def get_question(q_type):
        """Return matrix question object given question type."""
        if q_type == 'add_sub':
            question, answer = add_sub_question()
        elif q_type == 'mult':
            question, answer = mult_question()
        elif q_type == 'inv':
            question, answer = inv_question()
        elif q_type == 'det':
            question, answer = det_question()
        else:
            raise ValueError
        return MatrixQuestion(question, answer, q_type)

    def get_answer(self):
        """Override get_answer as matrix type answers need to also call get_rows."""
        if self.question_type != 'det':
            return self.answer.get_rows()
        else:
            return self.answer
```

```
    def is_mat_ans(self):
        """Return True if answer is a matrix."""
        if type(self.answer) != Matrix:
            return False
        else:
            return True

    def get_ans_dim(self):
        """Return dimensions if answer is a matrix, otherwise return 0."""
        if type(self.answer) == Matrix:
            return self.answer.get_dimensions()
        else:
            return 0


def mult_question():
    """Return random question and answer pair for multiplication."""
    # Choose a random size for first matrix
    y1 = random.randint(1, 2)
    x1 = random.randint(2, 3)
    # Set height of second to width of first
    y2 = x1
    # Set random width of second matrix
    x2 = random.randint(1, 2)
    # Populate both matrices with random numbers (between 0 and 10)
    m1 = [[random.randint(1, 10) for x in range(x1)] for y in range(y1)]
    m2 = [[random.randint(1, 10) for x in range(x2)] for y in range(y2)]
    # Create matrix objects from the 2D lists
    mat1 = Matrix(m1)
    mat2 = Matrix(m2)
    # Calculate answer and generate question
    question = 'Calculate `' + str(m1) + '` X `' + str(m2)+'`'
    answer = (mat1 * mat2)
    return (question, answer)


def add_sub_question():
    """Return random question and answer pair for addition and subtraction."""
    x, y = random.randint(2, 3), random.randint(2, 3)
    m1 = Matrix(
        [[random.randint(1, 10) for x in range(x)] for y in range(y)]
            )
    m2 = Matrix(
        [[random.randint(1, 10) for x in range(x)] for y in range(y)]
            )
    rand_op = random.choice(['+', '-'])
    # Calculate answer and create question based on which opertor was chosen
    if rand_op == '+':
        answer = m1 + m2
        question = 'Calculate `'+str(m1.get_rows())+' + '+str(m2.get_rows())+'`'
    else:
        answer = m1 - m2
        question = 'Calculate `' + str(m1.get_rows())+' - '+str(m2.get_rows()) + '`'
    return (question, answer)


def det_question():
    """Return random question and answer pair for finding the determinant."""
    # Generate randomly 3x3 matrix
    mat = Matrix(
        [[random.randint(1, 10) for x in range(3)] for y in range(3)]
            )
    # Calculate answer and generate question
    answer = mat.determinant()
    question = 'Find the Determinant of `'+str(mat.get_rows())+'`'
    return (question, answer)


def inv_question():
    """Return random question and answer pair for finding the inverse."""
    # Generate random matrices until one with an inverse is found
    while True:
        # Create random 3x3 matrix
        mat = Matrix(
```

```
                [[random.randint(1, 10) for x in range(3)] for y in range(3)]
                )
            if mat.determinant() != 0:
                # If Matrix with valid inverse is created, break
                break
        # Calculate answer and generate question
        question = 'Find the Inverse of `'+str(mat.get_rows())+'`'
        answer = mat.inverse().tostr()
        return (question, answer)

if __name__ == '__main__':
        q = MatrixQuestion.get_question('inv')
        print(q.get_q())
```

./app/pyscripts/question_dict.py

```
QUESTIONS = [
        {'id': 0, "name": 'Addition and Subtraction', "topic": 'Matrix', "q_type": "add_sub"},
        {'id': 1, 'name': 'Multiplication', "topic": 'Matrix', "q_type": "mult"},
        {'id': 2, 'name': 'Determinant', "topic": 'Matrix', "q_type": "det"},
        {'id': 3, 'name': 'Inverse', "topic": 'Matrix', "q_type": "inv"},
        {'id': 4, 'name': 'Addition and Subtraction', "topic": 'Complex', "q_type":  "add_sub"},
        {'id': 5, 'name': 'Multiplication', "topic": 'Complex', "q_type": "mult"},
        {'id': 6, "name": 'Division', "topic": 'Complex', "q_type": "div"},
        {'id': 7, "name": 'Argument and Modulus', "topic": 'Complex', "q_type": "mod_arg"},
        ]

MATRIX_QUESTIONS = {
        "add_sub": 0,
        "mult": 1,
        "det": 2,
        "inv": 3
}

COMPLEX_QUESTIONS = {
        "add_sub": 4,
        "mult": 5,
        "div": 6,
        "mod_arg": 7
}
```

./app/routes/__init__.py

```
from . import *
```

./app/routes/loci.py

```
from flask import render_template, request, jsonify, abort, Blueprint
from flask_login import login_required, current_user
import json
from ..pyscripts.complex_loci import get_implicit

from ..models import Graph, User
from app import db

from sympy import sympify, re, im
import html
# Initialise blueprint
loci_blueprint = Blueprint('loci', __name__, template_folder='templates')


@loci_blueprint.route('/loci-plotter')
def loci():
        if current_user.is_authenticated:
                # If user is logged in, load and send saved graph data
                user = User.query.get(current_user.user_id)
                user_graphs = user.graphs.all()
        else:
                user_graphs = None
        return render_template('loci.html', user_graphs=user_graphs)
```

```python
@loci_blueprint.route('/_plot')
def plot():
    # Get input equation
    eq = request.args.get('eq', 0, type=str)
    try:
        # Modify equation with function in complex_loci.py
        line = get_implicit(eq, latx=True)
        print(line)
        if ' i ' in line:
            # A separated i means that there is a complex number in the output
            # This means the input equation was invalid
            raise TypeError
        return jsonify(result=line, eq=html.escape(eq))
    except Exception as e:
        print(e)
        # Abort if there is an error (causes error message on client-side)
        abort(500)


@loci_blueprint.route('/_addgraph', methods=['GET', 'POST'])
@login_required
def addgraph():
    if request.method == 'POST':
        # POST request means user is saving a graph
        try:
            # Get data from form
            data1 = request.form.get('desmosdata', None)
            data2 = request.form.get('exprlist', None)
            title = request.form.get('title', "")
            desc = request.form.get('description', "")
            image_url = request.form.get('image', "")
            user_id = current_user.user_id

            exists = Graph.query.filter_by(
                title=title, user_id=user_id).first()
            if title.replace(' ', '') == '':
                return jsonify(status="error", error="Please enter a title")
            if exists:
                # Prevent same graph getting saved more than once
                return jsonify(status="error", error="Graph already exists")
            else:
                # Add graph data to database and link to current user
                g = Graph(data1, data2, user_id, title, desc, image_url)
                db.session.add(g)
                db.session.commit()
                graph_id = g.graph_id  # has to be after commit
                return jsonify(id=graph_id, title=title, image_url=image_url,
                               desc=desc, status="ok", error=None)
        except Exception as e:
            # Return error message and error status if there is an error
            # Causes error popup on client-side
            return jsonify(status="error", error="Error saving Graph")
    if request.method == 'GET':
        # GET request means the user is loading a graph
        try:
            graph_id = request.args.get('graph_id', None)
            g = Graph.query.get(graph_id)
            return jsonify(desmosdata=g.desmosdata, exprlist=g.exprlist)
        except Exception as e:
            # Abort if there is an error (causes error message on client-side)
            return abort(500)


@loci_blueprint.route('/operations-argand')
def operations():
    return render_template('operations.html')


@loci_blueprint.route('/_addcalc', methods=['GET'])
def addcalc():
    # Get equation requested
    eq_str = request.args.get('eq', None)
```

```
    letters = json.loads(request.args.get('letters', None))
    # Convert to sympy object
    eq = sympify(eq_str)
    # Get all the variables in the expression
    vars_ = eq.free_symbols
    correct_variables = set({str(v) for v in vars_}).issubset(set(letters ))
    if not correct_variables:
        return abort(500)
    # get real and imaginary parts of expression
    real = str(re(eq).expand(complex=True))
    imag = str(im(eq).expand(complex=True))
    # Loop over the variables
    for v in vars_:
        # Replace im(variable) with variable.Y()
        # Replace re(variable) with variable.X()
        # This is how JSXGraph allows points based on other points
        real = real.replace('im('+str(v)+')', str(v)+'.Y()')
        real = real.replace('re('+str(v)+')', str(v)+'.X()')
        imag = imag.replace('im('+str(v)+')', str(v)+'.Y()')
        imag = imag.replace('re('+str(v)+')', str(v)+'.X()')
    # Relace functions for javascript
    real = real.replace('**', '^').replace('sin', 'Math.sin').replace('cos',
'Math.cos').replace('atan2', 'Math.atan')
    imag = imag.replace('**', '^').replace('sin', 'Math.sin').replace('cos',
'Math.cos').replace('atan2', 'Math.atan')
    # Return the real and imaginary parts of calculated points as JSON
    print(real,imag)
    return jsonify(x=real, y=imag)
```

./app/routes/matrix.py

```
from flask import request, render_template, Blueprint
from ..pyscripts.matrices import Matrix
from fractions import Fraction

# Initialise blueprint
matrix_blueprint = Blueprint('matrix_blueprint', __name__,
                             template_folder='templates')


@matrix_blueprint.route('/matrix', methods=['GET', 'POST'])
def matrix():
    result = None
    # If the form is posted, it has been submitted
    if request.method == 'POST':
        try:
            # Get operator pressed (+,-,*)
            # If no operator was pressed, op = None
            op = request.form.get('submit', None)
            # Get operation requested if a button under matrix A is pressed
            acalc = request.form.get('a-submit', None)
            # Get operation requested if a button under matrix A is pressed
            bcalc = request.form.get('b-submit', None)
            if acalc:
                # If operation on Matrix A requested
                letter = 'A'
                calc = acalc
            elif bcalc:
                # If operation on Matrix B requested
                letter = 'B'
                calc = bcalc
            else:
                # Initialise matrix A list
                mata = []
                # Get matrix data from form
                for x in range(3):  # TODO make a function def to get a matrix
                    mata.append([])  # def get_mat(letter,x,y):
                    for y in range(3):
                        string = 'A' + str(x) + str(y)
                        mata[x].append(Fraction(request.form[string]))
                # Initialise matrix B list
                matb = []
                # Get matrix data from form
```

```
                for x in range(3):
                    matb.append([])
                    for y in range(3):
                        string = 'B' + str(x) + str(y)
                        matb[x].append(Fraction(request.form[string]))
                # Create both matric objects
                a = Matrix(mata)
                b = Matrix(matb)
                # Check which operator button pressed
                # Assign result to matresult depending on operator
                if op == 'X':
                    matresult = a * b
                if op == '-':
                    matresult = a - b
                if op == '+':
                    matresult = a + b
                # Convert all items in matrix to strings and return page
                result = matresult.tostr().rows
                return render_template('matrix.html', matrix_result=result,
                                       det_result=None, Error=None)
            # Initialise Matrix list
            mat = []
            for x in range(3):
                mat.append([])
                for y in range(3):
                    # Get data from form for correct matrix
                    # Depends on whcih button was pressed
                    string = letter + str(x) + str(y)
                    mat[x].append(Fraction(request.form[string]))
            # Create matrix object based on this
            m = Matrix(mat)
            # Calculate result depending on which button pressed
            # Return page with relevant result
            if 'Determinant' in calc:
                result = str(m.determinant())
                return render_template('matrix.html', matrix_result=None,
                                       det_result=result, Error=None)
            elif 'Inverse' in calc:
                result = m.inverse().tostr().rows
                return render_template('matrix.html', matrix_result=result,
                                       det_result=None, Error=None)
            elif 'Transpose' in calc:
                result = m.transpose().tostr().rows
                return render_template('matrix.html', matrix_result=result,
                                       det_result=None, Error=None)
            elif 'Triangle' in calc:
                result = m.triangle().tostr().rows
                return render_template('matrix.html', matrix_result=result,
                                       det_result=None, Error=None)
            else:
                return render_template('matrix.html', matrix_result=None,
                                       det_result=None, Error=None)
        except Exception as e:
            print(e)
            # If there was an error, return page with an error message
            error = 'Invalid Matrix, Try again'
            return render_template('matrix.html', matrix_result=None,
                                   det_result=None, Error=error)
    # Return basic page if GET request (no form submitted yet)
    return render_template('matrix.html', matrix_result=result)
```

./app/routes/questions.py

```
import ast

from flask import (Blueprint, abort, flash, jsonify, render_template,
                   request, session)
from flask_login import current_user

from app import db

from ..models import Mark, Student
from ..pyscripts.complex_questions import ComplexQuestion
```

```python
from ..pyscripts.matrix_questions import MatrixQuestion
from ..pyscripts.question_dict import COMPLEX_QUESTIONS, MATRIX_QUESTIONS

# Initialise Blueprint
questions_blueprint = Blueprint('questions', __name__,
                                template_folder='templates')


@questions_blueprint.route('/questions')
def questions():
    """Questions home page."""
    return render_template('questions/questions.html')


@questions_blueprint.route('/questions/<topic>/<q_type>')
def show_questions(topic, q_type):
    """Show page with requested question type."""
    q_number = request.args.get('n', 10)
    if topic == 'matrix':
        # Generate question objects
        questions = [MatrixQuestion.get_question(q_type) for x in range(q_number)]
        # Get list of answers from question objects
        answers = [q.get_answer() for q in questions]
        # Check whether the answers are of type matrix or not
        matans = questions[0].is_mat_ans()
        # List of question strings from question objects
        # List of answer strings from question objects
        # Save both in server side session
        session['questions'] = [q.get_q() for q in questions]
        session['answers'] = [str(q.get_answer()) for q in questions]
        return render_template('questions/mat_questions.html',
                               questions=enumerate(questions), answers=answers,
                               mat_ans=matans, q_type=q_type, topic=topic)
    elif topic == 'complex':
        questions = [ComplexQuestion.get_question(q_type) for x in range(q_number)]
        answers = [q.get_answer() for q in questions]
        session['questions'] = [q.get_q() for q in questions]
        session['answers'] = [str(q.get_answer()) for q in questions]
        return render_template('questions/complex_questions.html',
                               questions=enumerate(questions), answers=answers,
                               q_type=q_type, topic=topic)
    else:
        # If topic is invalid, return 404 page not found
        abort(404)


@questions_blueprint.route('/questions/_answers/<topic>/<q_type>')
def answers(topic, q_type):
    """Return scores and marked questions given topic, question type and list
    of input answers from questions form"""
    if topic == 'matrix':
        # Get question_id from dictionary
        question_id = MATRIX_QUESTIONS[q_type]
        # Get answers from session
        answers = session['answers']
        if q_type == 'det':
            # For determinant questions (non-matrix answers)
            # Get all input answers from form
            inputs = [request.args.get(str(x), 0) for x in range(10)]
            # Convert all inputs to ints
            inputs = [int(x) if x else 0 for x in inputs]
            # Initialise empty scores array
            scores = []
            # Loop over all answers
            for n, x in enumerate(answers):
                # check corresponding input answer
                # Add one to scores array if input answer matches
                if int(x) == inputs[n]:
                    scores.append(1)
                else:
                    scores.append(0)
            # Calculate percentage score
            percent = sum(scores)*100//len(answers)
        else:
```

```
            # Convert each answer in answers list to a 2-d list
            # Also convert all elements in matrix to string
            answers = [[[str(i) for i in j] for j in ast.literal_eval(a)] for a in answers]
            # Initialize input answers list
            inputs = []
            for n, a in enumerate(answers):
                inputs.append([])
                for x in range(len(a)):
                    inputs[n].append([])
                    for y in range(len(a[0])):
                        # Get form input and add to matrix
                        # Add 0 if there is no form input
                        i = request.args.get(str(n) + str(x) + str(y), 0)
                        if i:
                            inputs[n][x].append(i)
                        else:
                            inputs[n][x].append('0')
            # Initialise scores list
            scores = []
            # Check all input answers against stored answers
            for n, x in enumerate(answers):
                if x == inputs[n]:
                    scores.append(1)
                else:
                    scores.append(0)
            # Calculate percent score
            percent = sum(scores) * 100 // len(answers)
        questions = session['questions']
        # Convert items in answers and inputs to strings
        # so they can be displayed in score page
        answers = [str(a).replace("'", "") for a in answers]
        inputs = [str(i).replace("'", "") for i in inputs]

    elif topic == 'complex':
        # Get question id (for database)
        question_id = COMPLEX_QUESTIONS[q_type]
        answers = session['answers']
        if q_type == 'mod_arg':
            # For modulus and argument questions (non-complex answers)
            # Initialise input answer list
            inputs = []
            for x in range(len(answers)):
                # Get form inputs with names beginning mod and arg
                # Put into tuple (modulus,answer) and add to inputs list
                inputs.append((
                    request.args.get(str(x) + 'mod', 0),
                    request.args.get(str(x) + 'arg', 0)
                    ))
            # Change answers stored in session to correct type
            answers = [(str(i), str(j)) for i, j in [ast.literal_eval(a) for a in answers]]
            # Initialise scores list
            scores = []
            # Check answers, add 1 to scores if answers match else add 0
            for n, x in enumerate(answers):
                if x == inputs[n]:
                    scores.append(1)
                else:
                    scores.append(0)
            # Calculate percent score
            percent = sum(scores) * 100 // len(answers)
            pass
        else:
            # Initialise input answers list
            inputs = []
            for x in range(len(answers)):
                # Get form inputs
                # Add string representation of complex number answer to inputs
                re = str(request.args.get(str(x)+'re', 0).replace(' ', ''))
                im = str(request.args.get(str(x)+'im', 0).replace(' ', ''))
                if '-' in im:    # If imaginary part is negative
                    inputs.append(re+im+'j')
                else:    # If imaginary part is positive, use '+'
                    inputs.append(re+'+'+im+'j')
            # Initialise scores list
            scores = []
```

```
            # Loop over answers
            for n, x in enumerate(answers):
                # Convert both input answer and stored answer to complex number
                # Check if both answers match and add 1 or 0 to scores
                print(x,inputs[n])
                if complex(x) == complex(inputs[n]):
                    scores.append(1)
                else:
                    scores.append(0)
            # Calculate percent score
            percent = sum(scores)*100//len(answers)
        questions = session['questions']
    else:
        # If topic is not matrix or complex, return 404
        abort(404)
    if current_user.is_authenticated and current_user.role == 'student':
        # Only add a mark to database if user is a logged in student
        mark = Mark(sum(scores), len(scores), question_id,
                    current_user.user_id)
        db.session.add(mark)
        # Get student by user id
        s = Student.query.filter_by(user_id=current_user.user_id).first()
        # Get all of the students tasks
        ts = s.tasks.all()
        # Find out whether there is an active task with the same question_id
        # as the task that is being completed
        task = None
        # Loop over all tasks
        for t in ts:
            if t.question_id == question_id and t.completed is False:
                task = t
                # Break once relevant task found
                break
        # If there is no task, task = None
        # If there is such a task
        if task:
            # Link task with corresponding mark
            task.mark_id = mark.mark_id
            # Set task as completed
            task.completed = True
            # Add to database
            db.session.add(task)
            flash('Task Completed!')
        # Update database with all changes
        db.session.commit()
    # Return data in JSON form for javascript to display
    return jsonify(answers=answers, inputs=inputs, questions=questions,
                    percent=percent, scores=scores)
```

./app/routes/user.py

```
from flask import (Blueprint, Flask, abort, flash, jsonify, redirect,
                   render_template, request, url_for)
from flask_login import current_user, login_required, login_user, logout_user
from flask_mail import Message
from itsdangerous import URLSafeTimedSerializer
from werkzeug.security import generate_password_hash

from app import app, db, mail

from ..forms import (ChangeDetailsForm, ChangePasswordForm,
                     ChangePasswordForm1, LoginForm, RegisterForm,
                     RequestPasswordChangeForm, SetTaskForm, TeacherLinkForm)
from ..models import Student, Task, Teacher, User, Graph
from ..pyscripts.question_dict import QUESTIONS

# Initialise blueprint
user = Blueprint('user', __name__, template_folder='templates')

# Create serializer object -- used to create tokens for emails
serializer = URLSafeTimedSerializer(app.config["SECRET_KEY"])
```

```
def send_email(address, subject, html):
    """Sends email to address given the html content of email."""
    msg = Message(subject, sender="testapp545545@gmail.com",
                  recipients=[address])
    msg.html = html
    mail.send(msg)


@user.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        # already logged in user shouldnt go to login page
        return redirect(url_for('main'))
    loginform = LoginForm()
    if loginform.validate_on_submit():
        user = User.query.filter_by(email=loginform.email.data.lower()).first()
        # If user exists and password enters is valid
        if user and user.check_pw(loginform.password.data):
            # Login user and update database
            user.authenticated = True
            db.session.add(user)
            db.session.commit()
            login_user(user)
            # Go back to home page
            return redirect(url_for('main'))
        else:
            flash('Incorrect Credentials')
    # Return empty login page if no form submitted or errors on form validation
    return render_template('user/login.html', loginform=loginform)


@user.route('/logout')
@login_required
def logout():
    # Load current user
    user = current_user
    # Unauthenicate
    user.authenticated = False
    # Update database
    db.session.add(user)
    db.session.commit()
    # Log out user and return to home page
    logout_user()
    return redirect('main')


@user.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        # Already logged in user shouldnt go to register page
        return redirect(url_for('main'))
    regform = RegisterForm()
    if regform.validate_on_submit():
        if not User.query.filter_by(email=regform.email.data.lower()).first():
            # Only students have to register
            # So create student object with form data
            u = Student(regform.fname.data.lower(), regform.lname.data.lower(),
                        regform.email.data.lower(), regform.password.data,
                        'student')
            # Add student object to database.
            db.session.add(u)
            db.session.commit()

            subject = "Email Confirmation"
            #Create token for confimation link
            token = serializer.dumps(u.email, salt='email-confirm-key')
            # Create confirmation url from token
            confirm_url = url_for('user.confirm_email', token=token, _external=True)
            # Render and send confirmation email
            html = render_template('emails/confirm_email.html', confirm_url=confirm_url)
            send_email(address=u.email, subject=subject, html=html)
            # Log in the user and redirect to homepage
            login_user(u)
            flash("Confirmation Email Sent")
```

```
            return redirect(url_for('main'))
        else:
            # Cant have two users with the same email
            flash('Email already exists')
    # Return signup page if GET request (no form submitted)
    return render_template('user/signup.html', regform=regform)

@user.route('/confirm/<token>')
def confirm_email(token):
    try:
        # Try to decode the token in the url with the given salt
        # Reject if token is more than an hour old
        # The token decodes to the user's email address
        email = serializer.loads(token, salt="email-confirm-key", max_age=86400)
    except:
        # If the token is invalid and an error is thrown, return 404 error code
        abort(404)
    # Get User from database based on email (from decoded token)
    user = User.query.filter_by(email=email).first_or_404()
    # Set confirmed status
    user.confirmed = True
    # Update database
    db.session.add(user)
    db.session.commit()
    #Log in user and redirect to home page
    login_user(user)
    flash('Email Confirmed')
    return redirect(url_for('main'))


@user.route('/reset', methods=['GET', 'POST'])
def reset():
    #Load form
    form = RequestPasswordChangeForm()
    if form.validate_on_submit():
        #Get user from input email
        user = User.query.filter_by(email=form.email.data).first()
        # If there is no user, display error message and return to same page
        if not user:
            flash('Email address does not exist')
            return render_template('user/reset.html', form=form)
        # If user not confirmed, display error message and return to same page
        elif not user.confirmed:
            flash('Email address not confirmed')
            return render_template('user/reset.html', form=form)
        # If there is a user with that email address, create email
        subject = "Password Reset"
        # Create token based on users email
        token = serializer.dumps(user.email, salt='recover-key')
        # Generate url with the token
        recover_url = url_for('user.reset_with_token', token=token, _external=True)
        # Render and send the email
        html = render_template('emails/recover_email.html', recover_url=recover_url)
        send_email(address=user.email, subject=subject, html=html)
        # Display success message and redirect to home page
        flash('Password reset email sent')
        return redirect(url_for('main'))
    return render_template('user/reset.html', form=form)


@user.route('/reset/<token>', methods=['GET', 'POST'])
def reset_with_token(token):
    try:
        # Try to decode the token in the url with the given salt
        # Reject if token is more than an hour old
        # The token decodes to the user's email address
        email = serializer.loads(token, salt='recover-key', max_age=86400)
    except:
        # If the token is invalid and an error is thrown, return 404 error code
        abort(404)
    # Load form
    form = ChangePasswordForm()
    if form.validate_on_submit():
        # Get User from database based on email (from decoded token)
```

```
            user = User.query.filter_by(email=email).first_or_404()
            print(user)
            # Change user's password
            user.password = generate_password_hash(form.password.data)
            # Add updated user to database
            db.session.add(user)
            db.session.commit()
            flash("password updated successfully")
            # Redirect to login page so user logs in with new password
            return redirect(url_for('user.login'))
        return render_template('user/reset_with_token.html', form=form, token=token)


@user.route('/_delete_teacher', methods=['POST'])
def delete_teacher():
    # Get user id
    user_id = current_user.user_id
    # Get student from user_id
    s = Student.query.filter_by(user_id=user_id).first()
    # Get teacher id to be deleted from student and delete link
    teacher_id = request.form.get('teacher_id', None)
    t = Teacher.query.filter_by(teacher_id=teacher_id).first()
    a = t.remove_student(s)
    db.session.add(a)
    db.session.commit()
    return jsonify()  # return nothing (no error)


@user.route('/_delete_student', methods=['POST'])
def delete_student():
    # Get user id
    user_id = current_user.user_id
    # Get teacher from user_id
    t = Teacher.query.filter_by(user_id=user_id).first()
    # Get student id to be deleted from teacher and delete link
    student_id = request.form.get('student_id', None)
    s = Student.query.filter_by(student_id=student_id).first()
    a = t.remove_student(s)
    db.session.add(a)
    db.session.commit()
    return jsonify()  # return nothing (no error)


@user.route('/_delete_graph', methods=['POST'])
def delete_graph():
    # Get id of graph to be deleted
    graph_id = request.form.get('graph_id', None)
    # Query database and delete graph
    g = Graph.query.filter_by(graph_id=graph_id).delete()
    db.session.commit()
    return jsonify()  # return nothing (no error)


@user.route('/_delete_task', methods=['POST'])
def delete_task():
    pass


@user.route('/account', methods=['GET', 'POST'])
@login_required
def account():
    # Get current user data
    user_id = current_user.user_id
    u = User.query.get(user_id)
    if u.role == 'student':
        # Load all forms
        changeform = ChangeDetailsForm(obj=u)
        linkform = TeacherLinkForm()
        pwform = ChangePasswordForm1()
        if linkform.link_submit.data and linkform.validate_on_submit():
            # If the links form is submitted, try to get the teacher with input code
            t = Teacher.query.filter_by(code=linkform.link_code.data).first()
            if t:
                # If teacher with link exists
```

```
                    # Load student object
                    s = Student.query.filter_by(user_id=user_id).first()
                    # Try to add this student to the teacher
                    a = t.add_student(s)
                    if not a:
                         # If add_student returns none, then student is already linked
                         flash('Already linked to this teacher')
                         # Go back to account page
                         return render_template('user/student_account.html',
                              student=u, qs=QUESTIONS, linkform=linkform, changeform=changeform,
pwform=pwform)
                    # If student not already linked, commit link to database
                    db.session.add(a)
                    db.session.commit()
                    flash('Successfully linked')
                    # Go back to account page
                    return render_template('user/student_account.html',
                         student=u, qs=QUESTIONS, linkform=linkform,
                         changeform=changeform, pwform=pwform)
               else:
                    flash('No teacher with that code')
                    # Go back to account page with appropriate message
                    return render_template('user/student_account.html',
                         student=u, qs=QUESTIONS, linkform=linkform,
                         changeform=changeform, pwform=pwform)
          if changeform.change_submit.data and changeform.validate_on_submit():
               # If user is changing details, check password is correct
               if u.check_pw(changeform.password.data):
                    # Get data from form and change attributes for user
                    u.fname = changeform.fname.data
                    u.lname = changeform.lname.data
                    u.email = changeform.email.data
                    # Update user in database.
                    db.session.add(u)
                    db.session.commit()
                    flash('Details changed successfully')
                    # Go back to account page with message
                    return render_template('user/student_account.html',
                         student=u, qs=QUESTIONS, linkform=linkform,
                         changeform=changeform, pwform=pwform)
               else:
                    flash('Incorrect password')
                    # Go back to account page with message
                    return render_template('user/student_account.html',
                         student=u, qs=QUESTIONS, linkform=linkform,
                         changeform=changeform, pwform=pwform)
          if pwform.pw_submit.data and pwform.validate_on_submit():
               # Make sure old password was input correctly
               if u.check_pw(pwform.old_password.data):
                    # Get form data and update users password
                    u.password = generate_password_hash(pwform.password.data)
                    # Update user in database
                    db.session.add(u)
                    db.session.commit()
                    flash('Password changed successfully')
                    # Go back to account page with message
                    return render_template('user/student_account.html',
                         student=u, qs=QUESTIONS, linkform=linkform,
                         changeform=changeform, pwform=pwform)
               else:
                    flash('Incorrect password')
                    # Go back to account page with message
                    return render_template('user/student_account.html',
                         student=u, qs=QUESTIONS, linkform=linkform,
                         changeform=changeform, pwform=pwform)
          return render_template('user/student_account.html', student=u,
                                 qs=QUESTIONS, linkform=linkform,
                                 changeform=changeform, pwform=pwform)
     elif u.role == 'teacher':
          # Get all the teachers linked students from the database
          students = u.students.all()
          # List of tuples (student_id,student_name) for each student
          # Used in SetForm, when a teacher chooses which students to set which tasks
          choices = [(s.student_id, s.fname+' '+s.lname) for s in students]
          # Load forms
```

```
        setform = SetTaskForm(choices)
        changeform = ChangeDetailsForm(obj=u)
        pwform = ChangePasswordForm1()
        if setform.set_submit.data and setform.validate_on_submit():
            # If setform submitted (form for setting tasks)
            # Get teacher object from user_id
            t = Teacher.query.filter_by(user_id=user_id).first()
            teach_id = t.teacher_id
            # Loop over each student selected
            for s_id in setform.student_select.data:
                # Loop over each tasks selected
                for q_id in setform.task_select.data:
                    # Set task to student
                    t = Task(q_id, s_id, teach_id)
                    # Add to database
                    db.session.add(t)
            # Commit database changes
            db.session.commit()
            # Go back to account page with success message
            flash('Tasks set successfully')
            return render_template('user/teacher_account.html', teacher=u, students=students,
setform=setform, qs=QUESTIONS, pwform=pwform, changeform=changeform)
        if changeform.change_submit.data and changeform.validate_on_submit():
            if u.check_pw(changeform.password.data):
                # Get data from form and change attributes for user
                u.fname = changeform.fname.data
                u.lname = changeform.lname.data
                u.email = changeform.email.data
                # Update user in database
                db.session.add(u)
                db.session.commit()
                # Go back to account page with message
                flash('Details changed successfully')
                return render_template('user/teacher_account.html',
                    teacher=u, students=students, setform=setform,
                    qs=QUESTIONS, pwform=pwform, changeform=changeform)
            else:
                flash('Incorrect password')
                return render_template('user/teacher_account.html',
                    teacher=u, students=students, setform=setform,
                    qs=QUESTIONS, pwform=pwform, changeform=changeform)
        if pwform.pw_submit.data and pwform.validate_on_submit():
            if u.check_pw(pwform.old_password.data):
                # Get form data and update users password
                u.password = generate_password_hash(pwform.password.data)
                db.session.add(u)
                db.session.commit()
                # Go back to account page with success message
                flash('Password changed successfully')
                return render_template('user/teacher_account.html',
                    teacher=u, students=students, setform=setform,
                    qs=QUESTIONS, pwform=pwform, changeform=changeform)
            else:
                # Go back to account page with error message
                flash('Incorrect password')
                return render_template('user/teacher_account.html',
                    teacher=u, students=students, setform=setform,
                    qs=QUESTIONS, pwform=pwform, changeform=changeform)
        return render_template('user/teacher_account.html',
            teacher=u, students=students, setform=setform, qs=QUESTIONS,
            pwform=pwform, changeform=changeform)
    else:
        return abort(500)
```

./app/static/scripts/loci_plot.js

```
var elt = document.getElementById('calculator');
// Options for desmos grpah plotter
var options = {
    expressions:true,
    expressionsCollapsed:true,
    keypad:false,
    settingsMenu:false
}
```

```
// Initialize plotter
var calculator = Desmos.Calculator(elt,options);
calculator.setGraphSettings({xAxisLabel:'Re',yAxisLabel:'Im'})
// Create reset state (for when the user resets the plotter)
var reset = calculator.getState();

var plots = -1

function addplot(){
// Get input from equation input field
var eq = $('input[name="in"]').val()
// Send GET request to server at /_plot with 'eq' variable
$.getJSON($SCRIPT_ROOT + '/_plot', {
    eq: eq
}, function(data) {  //function carried out on receiving data from server
    // Increment plots counter
    // Allows each plot to have an unique id
    plots += 1;
    // Add html for a row in the expressions table
    // Includes the expression, show/hide checkbox, delete button
    $('#expressions tbody').append(
        '<tr id="row'+plots+'">'+
            '<td>'+
                '`'+data.eq+'`'+
            '</td>'+
            '<td>'+
                '<input type="checkbox" name="plot" id="'+plots+'" checked>'+
            '</td>'+
            '<td>'+
                '<input type="button" class="btn btn-block" name="del" id="del'+plots+'"
value="X">'+
            '</td>'+
        '</tr>'
    );
    // Typeset math
    MathJax.Hub.Queue(["Typeset",MathJax.Hub,"expressions"]);
    // Get result from received data
    var result = data.result;
    console.log(result)
    // Add expression to desmos plot
    calculator.setExpression({id:plots,latex:result});

}).fail(function(){  //Display error message if server returns an error
    $('#eq_in').blur()
    alert('Please enter a valid equation')
    $('#eq_in').focus()
});
return false;
}


$(document).ready(function() {
    $('#graph-select').imagepicker({show_label:true})
    var queryDict = {};
    // Get all parameters
    location.search.substr(1).split("&").forEach(function(item) {queryDict[item.split("=")[0]] =
item.split("=")[1]});
    // If there is an id parameter, load graph with that id
    if (queryDict['id']) {
        $.getJSON($SCRIPT_ROOT+'/_addgraph',{
            graph_id: queryDict['id']
        },function(data){
            $('#load-modal').modal('hide')
            console.log(data.exprlist)
            $('#expressions').html(data.exprlist)
            calculator.setState(data.desmosdata)
        }).fail(function(){
            alert('Error Loading Graph')
        });
    }
    // If enter pressed while in the input box, add the plot to the graph
    $('#eq_in').on('keydown', function(e) {
        if (e.keyCode===13) {
            addplot();
```

```
        }});
    // CLicking the go button also adds a plot to the graph
    $('#go').on('click', addplot);
    // Function for clear all button
    $('#clear').on('click', function() {
        // Delete all rows in expressions table
        $('#expressions tbody > tr').remove();
        // Reset plot counter
        plots = -1;
        // Reset calculator
        calculator.setState(reset);
    });
    // Button for deleting individual plots
    $('#expressions').on('click','[type=button]',function(){
        // Get the id of button that was clicked - corresponds to the plots id
        var plot_no = $(this).attr('id').replace('del','');
        // Delete relevant row in expressions table
        $('#row'+plot_no).remove();
        // Delete relevant plot on graph
        calculator.removeExpression({id:parseInt(plot_no)})
    });
    // Show/hide line(s)
    $('#expressions').on('click','[type=checkbox]',function(){
        // Get id of checkbox clicked
        var plot_no=$(this).attr('id')
        // See whether the checkbox is checked or not
        var checked = $(this).is(':checked')
        // Set hidden attribute of the expression the opposite of checked
        calculator.setExpression({id:plot_no,hidden:!checked})
    });
    // Function for saving a graph
    $('#submit-graph').on('click',function(){
        // Send POST request to server at '/_addgraph' with data:
        // Graph title, graph data from desmos, the html for the expressions table,
        // Graph description, image data from a screenshot of graph
        $.post('/_addgraph',{
            title:$('#title').val(),
            desmosdata:JSON.stringify(calculator.getState()),
            exprlist:$('#expressions').html(),
            description:$('#desc').val(),
            image: calculator.screenshot({width:100,height:100,targetPixelRatio:2})
        },function(data){
            if (data.status === 'ok') {
                // If there was no error
                // Hide the save-graph dialog
                $('#save-modal').modal('hide')
                // Add option to load-graph dialog
                $('#graph-select').append('<option value="'+data.id+'" data-img-
src="'+data.image_url+'">'+data.title+'</option>')
                alert("Successfully saved graph")
                $('#graph-select').imagepicker({show_label:true})
            } else {
                // If there was an error, display the error message in a pop-up
                alert(data.error)
                $('#eq_in').focus()
            }
            });
    });
    // Function for loading a saved graph
    $('#load-graph').on('click',function(){
        console.log($("#graph-select").val())
        // Get the id of the graph selected
        id = $("#graph-select").val()
        // Send GET request to server with graph_id
        $.getJSON($SCRIPT_ROOT+'/_addgraph',{
            graph_id: id
        },function(data){  //Function carried out when data recieved
            // Hide the load-graph modal is not already hidden
            $('#load-modal').modal('hide')
            // Put exressions data into the table (from recieved data)
            $('#expressions').html(data.exprlist)
            // Set the calculator state
            calculator.setState(data.desmosdata)
        }).fail(function(){
            // Display error message on server error
```

```
                alert('Error Loading Graph')
            });
        })
        //modal made in jinja at start, then graphs added into html when one is saved
});
```

./app/static/scripts/operations.js

```
// Get width and height of box in pixels
var w = $('#box').width();
var h = $('#box').height();
// Make sure the axes have the same scale
var x_ax=w/150;
var y_ax=h/150;
var board_atts = {
            boundingbox: [-x_ax, y_ax, x_ax, -y_ax],
            axis: true,
            showCopyright:false,
            showNavigation:false,
            pan: {
                enabled: true,
                needshift: false,
                needTwoFingers: true
            },
            zoom: {
                factorX: 1.25,
                factorY: 1.25,
                wheel: true,
                needshift: false
            }
}
// Initialize board
board = JXG.JSXGraph.initBoard('box',board_atts);
// Create hidden origin point
var org = board.create('point', [0,0], {style:10,visible:false,fixed:true,name:' '});
// Initialise points and lines objects, and start letter at 'a'
var points = {};
var org_lines = {}
var letter = 'a';

$(document).ready(function(){
    $('#addpoint').on('click',function(){
            // Function for adding a new moveable point
            //  Increment letter
            letter = String.fromCharCode(letter.charCodeAt()+1);
            // Add new points at (1,1)
            points[letter] =
board.create('point',[1,1],{style:4,color:'red',strokeColor:'red',name:letter});
            // Add line from origin to point
            org_lines[letter] = board.create('arrow',[org,points[letter]],{strokeColor:'blue'})
            // Add row to table
            $('#expressions tbody').append('<tr id="row'+letter+'">'+
                '<td><b>'+letter+'</b></td>'+
                '<td>'+
                    '<input id="re'+letter+'" style="width:50px"
value="'+points[letter].X().toFixed(2)+'">+'+
                    '</td><td><input id="im'+letter+'" style="width:50px"
value="'+points[letter].Y().toFixed(2)+'">i'+
                '</td>'+
                '<td>'+
                    '<input type="checkbox" name="plot" id="show'+letter+'" checked>'+
                '</td>'+
                '<td>'+
                    '<input type="checkbox" name="plot" id="line'+letter+'" checked>'+
                '</td>'+
                '<td>'+
                    '<input type="button" class="btn btn-block" name="del" id="del'+letter+'"
value="X">'+
                '</td>'+
            '</tr>');
    });
    $('#addcalc').on('click',function(){
            // Function for adding calculated point
            // Get form input
```

```
        calc = $('#calc_in').val()
        // Send data to server as GET request at /_addcalc
        letters_list = []
        for(var letter in points){letters_list.push(letter)};
        console.log(letters_list);
        console.log(JSON.stringify(letters_list));
        $.getJSON($SCRIPT_ROOT + '/_addcalc',{
            eq:calc,
            letters:JSON.stringify(letters_list)
        }, function(data){ // Function for recieved data
            // Increment letter
            letter = String.fromCharCode(letter.charCodeAt()+1);
            // Add points with coordinates from recieved data
            points[letter] =
board.create('point',[data.x,data.y],{style:4,color:'blue',strokeColor:'blue',name:letter});
            // Add line from origin to point
            org_lines[letter] = board.create('arrow',[org,points[letter]],{strokeColor:'red'})
            // Add row to table
            $('#expressions tbody').append('<tr id="row'+letter+'">'+
                '<td><b>'+letter+'</b></td>'+
                '<td colspan="2" id="label'+letter+'">'+

'`'+points[letter].X().toFixed(2)+'+'+points[letter].Y().toFixed(2)+'i'+'`'+
                '</td>'+
                '<td>'+
                    '<input type="checkbox" name="plot" id="show'+letter+'" checked>'+
                '</td>'+
                '<td>'+
                    '<input type="checkbox" name="plot" id="line'+letter+'" checked>'+
                '</td>'+
                '<td>'+
                    '<input type="button" class="btn btn-block" name="del" id="del'+letter+'"
value="X">'+
                '</td>'+
            '</tr>');
        }).fail(function(){
            // Error message if there if function fails
            alert('Invalid calculation')
        });
    });
    $('#box').on('click change mouseup mousedown',function(){
        // When box is clicked, update the table which displays the points
        // The box being clicked means a points is moved
        console.log('aa');
        for (var letter in points) {
            // get x and y of all points

            var x = parseFloat(points[letter].X())
            var y = parseFloat(points[letter].Y())
            console.log(x,y)
            if ($('#label'+letter).length>0) {
                // Change table cell of calculated points
                $('#label'+letter).html(x.toFixed(2)+'+'+y.toFixed(2)+'i')
            } else {
                // Change value of inputs for moveable points
                $('#re'+letter).val(x.toFixed(2))
                $('#im'+letter).val(y.toFixed(2))
            }
        }
    });
    $('#clear').on('click', function() {
        // Delete table
        $('#expressions tbody > tr').remove();
        // Reset variables and recreate board
        points = {}
        lines = {}
        letter = 'a'
        JXG.JSXGraph.freeBoard(board);
        board = JXG.JSXGraph.initBoard('box', board_atts);
    });
    $('#expressions').on('click','[type=button]',function(){
        // Get id of button clicked, and get letter which it corresponds to
        var id = $(this).attr('id').replace('del','');
        // Remove point with the id
        board.removeObject(points[id]);
```

```
        // delete from points object
        delete points[id];
        // remove row from table
        $('#row'+id).remove();
    });
    $('#expressions').on('click','[type=checkbox][id*=show]',function(){
        // Get id of checkbox clicked, and get letter which it corresponds to
        var id=$(this).attr('id').replace('show','')
        // See whether point is visible
        point_vis = points[id].getAttribute('visible');
        // Flip visibility of point and corresponding line
        points[id].setAttribute({visible:!point_vis})
        org_lines[id].setAttribute({visible:!point_vis})
    });
    $('#expressions').on('click','[type=checkbox][id*=line]',function(){
        // Get id of checkbox clicked, and get letter which it corresponds to
        var id=$(this).attr('id').replace('line','')
        // See whether line is visible or not
        vis = org_lines[id].getAttribute('visible');
        // Flip hidden attribute
        org_lines[id].setAttribute({visible:!vis})
    });
    $('#expressions').on('keyup',function(){
        for (var letter in points) {
            if ($('#re'+letter).length>0){

points[letter].setPosition(JXG.COORDS_BY_USER,[parseFloat($('#re'+letter).val()),parseFloat($('#
im'+letter).val())])
            }
        }
        board.fullUpdate()
    });
});
```

./app/static/scripts/questions.js

```
$('#submit').on('click', function() {
    // When submit clicked, run function submit_answers
    submit_answers()
});

function submit_answers() {
    // Get data from questions form as a dictionary
    var data = $('#questions').serializeArray().reduce(function(obj, item) {
        obj[item.name] = item.value;
        return obj;
    }, {});
    // send GET request to server to analyse questions and calculate scores
    $.getJSON($SCRIPT_ROOT + '/questions/_answers/' + topic + '/' + q_type,
        data,
        function(data) {  //Function carried out when data is received
            // Create outline for table of scores
            html = '<table id="answers" class="table"> <thead> <th> Question</th> <th>Actual
Answer</th><th>Your Answer</th><th></th></thead><tbody>'
            //Get returned data
            questions = data.questions
            answers = data.answers
            inputs = data.inputs
            scores = data.scores
            percent = data.percent
            console.log(inputs, answers)
            // For each question/answer which is sent back
            for (var i = 0; i < data.questions.length; i++) {
                // Add row to table with question, correct answer, your answer and a colored
                html += '<tr><td>' + questions[i] + '</td><td>' + `` + answers[i] + `` +
'</td><td>' + `` + inputs[i] + ``
                if (scores[i] === 1) {
                    // Add green coloured square if correct
                    html += '<td bgcolor="#00FF00">'
                } else {
                    // Add red coloured square if wrong
                    html += '<td bgcolor="#FF0000">'
                }
```

```
                }
                html += '</tbody></table>'
                // Remove questions form from page, leaving almost empty page
                $('#questions').remove();
                // Add the table to the page
                $(html).appendTo('#main')
                // Typeset maths
                MathJax.Hub.Queue(["Typeset", MathJax.Hub, "answers"]);
            }).fail(function() {
            // An error means that there was invalid input
            alert('Please check your answers')
        })
}
```

./app/static/scripts/student_account.js

```
// js for sidebar formatting
$('#nav').affix({
     offset: {
          top: $('#nav').offset().top
     }
});
$('#nav').affix({
     offset: {
          bottom: ($('footer').outerHeight(true) +
                   $('.application').outerHeight(true)) +
                   40
     }
});
// Initialise imagepicker
$('#graph-select').imagepicker({show_label:true})
$('#links-table').on('click','[type=button]',function(){
  if (confirm('Are you sure')) {
    // Send data of student to be deleted to server
    teacher_id = $(this).attr('id');
    console.log(teacher_id);
    $.post($SCRIPT_ROOT + '/_delete_teacher',{
      teacher_id:teacher_id
    },function(data){
      // Remove teacher from account page
      $('#link-row'+teacher_id).remove()
    }).fail(function(){
      // Show error message if there is a server error
      alert('Error deleting teacher')
    });
  }
});
$('input[type="button"][id*="del-graph"]').on('click',function(){
  if (confirm('Are you sure')) {
    graph_id = $(this).attr('id').replace('del-graph','');
    // Send data of graph to be deleted to server
    $.post($SCRIPT_ROOT + '/_delete_graph',{
      graph_id:graph_id
    },function(data){
      // Remove graph from account page
      $('option[value="'+graph_id+'"]').remove()
      $('#graph-select').imagepicker({show_label:true})
    }).fail(function(){
      // Show error message if there is a server error
      alert('Error Deleting Graph')
    });
  }
});
```

./app/static/scripts/teacher_account.js

```
$('#nav').affix({
     offset: {
          top: $('#nav').offset().top
     }
});
$('#nav').affix({
```

```
    offset: {
        bottom: ($('footer').outerHeight(true) +
                 $('.application').outerHeight(true)) +
                 40
    }
});
$('#graph-select').imagepicker({show_label:true})
$('#links-table').on('click','[type=button]',function(){
  if (confirm('Are you sure')) {
    student_id = $(this).attr('id');
    // Send data of student to be deleted to server
    $.post($SCRIPT_ROOT + '/_delete_student',{
      student_id:student_id
    },function(data){
      // Delete all other references to this student on account page
      $('#link-row'+student_id).remove()
      $('#task-row'+student_id).remove()
      $('input[type="checkbox"][value="'+student_id+'"]').parents('tr').remove()
    }).fail(function(){
      // Show error message if there is a server error
      alert('Error deleting Student')
    });
  }
});
$('input[type="button"][id*="del-graph"]').on('click',function(){
  if (confirm('Are you sure')) {
    graph_id = $(this).attr('id').replace('del-graph','');
    // Send data of student to be deleted to server
    $.post($SCRIPT_ROOT + '/_delete_graph',{
      graph_id:graph_id
    },function(data){
      $('option[value="'+graph_id+'"]').remove()
      $('#graph-select').imagepicker({show_label:true})
    }).fail(function(){
      // Show error message if there is a server error
      alert('Error Deleting Graph')
    });
  }
});
```

./app/static/styles/account.css

```
.affix {
    top: 20px;
    width: 213px;
}

section {
    border: 1 px solid black;
    width: 100%
}

.title {
    text-align: center;
    font-size: 30pt;
}

@media (min-width: 1200px) {
    .affix {
        width: 263px;
    }
}

.affix-bottom {
    position: absolute;
    width: 213px;
}

@media (min-width: 1200px) {
    .affix-bottom {
        width: 263px;
    }
}
```

```
/*** custom checkboxes ***/

input[type=checkbox] { display:none; } /* to hide the checkbox itself */
input[type=checkbox] + label:before {
  font-family: FontAwesome;
  display: inline-block;
}

input[type=checkbox] + label:before { content: "\f096"; } /* unchecked icon */
input[type=checkbox] + label:before { letter-spacing: 10px; } /* space between checkbox and
label */

input[type=checkbox]:checked + label:before { content: "\f046"; } /* checked icon */
input[type=checkbox]:checked + label:before { letter-spacing: 5px; } /* allow space for check
mark */
```

./app/static/styles/complex_q.css

```
.out_question{
    width: 100%;
    padding: 10px;
    text-align: center;
}

.input{
    width: 50px;
    height: 30px;
    text-align: center;
    padding: 5px;
}

.answer{
    display:inline-block;
    width:50%;
    margin:0 auto;
    padding: 10px;
}

.question{
    padding: 10px;
}
```

./app/static/styles/main.css

```
html {
  position: relative;
  min-height: 100%;
}
body {
  margin-bottom: 60px;
}
.footer {
  position: absolute;
  bottom: 0;
  width: 100%;
  height: 60px;
  background-color: #f5f5f5;
  padding: 0 20px;

}
.content {
  width: 100%;
  padding: 0 15px;
  overflow: hidden;
  min-height: calc(100vh - 70px);

}
.container .text-muted {
  margin: 20px 0;
}
```

```
.dropdown-submenu {
    position: relative;
}

.dropdown-submenu>.dropdown-menu {
    top: 0;
    left: 100%;
    margin-top: -6px;
    margin-left: -1px;
    -webkit-border-radius: 0 6px 6px 6px;
    -moz-border-radius: 0 6px 6px;
    border-radius: 0 6px 6px 6px;
}

.dropdown-submenu:hover>.dropdown-menu {
    display: block;
}

.dropdown-submenu>a:after {
    display: block;
    content: " ";
    float: right;
    width: 0;
    height: 0;
    border-color: transparent;
    border-style: solid;
    border-width: 5px 0 5px 5px;
    border-left-color: #ccc;
    margin-top: 5px;
    margin-right: -10px;
}

.dropdown-submenu:hover>a:after {
    border-left-color: #fff;
}

.dropdown-submenu.pull-left {
    float: none;
}

.dropdown-submenu.pull-left>.dropdown-menu {
    left: -100%;
    margin-left: 10px;
    -webkit-border-radius: 6px 0 6px 6px;
    -moz-border-radius: 6px 0 6px 6px;
    border-radius: 6px 0 6px 6px;
}
```

./app/static/styles/matrix.css

```
.in {
    width: 50px;
    height: 50px;
    text-align: center;
}

.pure-button {
    width: 180px;
    margin-top: 5px;
}

.op-btn {
    width: 40px;
}

.clear-btn {
    color: red;
    width: 100px;
}

.matrix {
```

```
    position: relative;
}

.matrix:before, .matrix:after {
    content: "";
    position: absolute;
    top: 0;
    border: 1px solid #000;
    width: 6px;
    height: 100%;
}

.matrix:before {
    left: -6px;
    border-right: 0px;
}

.matrix:after {
    right: -6px;
    border-left: 0px;
}

.matrix td {
    padding: 5px;
    text-align: center;
}

#outer {
    display: table;
    width: 100%;
    height: 100%;
}

#inner {
    display: table-cell;
    vertical-align: middle;
    text-align: center;
}

#matrices {
    display: inline-block;
    margin-left: auto;
    margin-right: auto;
    margin-top: 10px;
}

#answers {
    float: bottom;
    width: 100%;
}

#A-input {
    float: left;
    margin-right: 20px;
}

#B-input {
    float: left;
}

#operators {
    float: left;
    margin-right: 20px;
    margin-top: 100px;
    margin-bottom: 100px;
}
```

./app/static/styles/questions.css

```
.out_question{
    width: 100%;
    padding: 10px;
```

```
    text-align: center;
}

.input{
    width: 50px;
    height: 50px;
    text-align: center;
}

.answer{
    display:inline-block;
    width:50%;
    margin:0 auto;
    padding: 10px;
}

.question{
    padding: 10px;
}
```

./app/templates/index.html

```
{% extends "layout.html" %}

{% block title %}Home Page{% endblock %}

{% block head %}
{{ super() }}
{% endblock %}

{% block nav %}
{# home page does not need navigation bar, all the links are on this page#}
{% endblock %}

{% block main %}
  <div class="col-lg-12" style="margin-top:15px;">
    {% if current_user.is_authenticated %}
        Logged in as {{ current_user.fname }} {{ current_user.lname }}
    {% endif %}

    <!--Links to all other pages-->
    <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('loci.loci') }}">
      Loci Plotter</a>
    <br>
    <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('loci.operations') }}">
      Complex Number Operations</a>
    <br>
    <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('matrix_blueprint.matrix') }}">
      Matrix Calculator</a>
    <br>
    <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('questions.questions') }}">
      Questions</a>

    <div>
    {# Show log in buttons if user isnt logged in, else show other buttons #}
    {% if not current_user.is_authenticated %}
      <div class = "col-sm-6">
        <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('user.login') }}">
          Log In
        </a>
      </div>
      <div class = "col-sm-6">
        <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('user.register') }}">
          Sign Up
        </a>
      </div>
    {% else %}
      <div class = "col-sm-6">
        <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('user.logout') }}">
          Log Out
        </a>
      </div>
      <div class = "col-sm-6">
```

```
      <a class="btn btn-primary btn-lg btn-block" href="{{ url_for('user.account')
}}">Account</a>
      </div>
    {% endif %}
    </div>
    {# show all flashed messages #}
    {% with messages = get_flashed_messages() %}
        {% if messages %}
            {% for message in messages %}
            <div class="alert alert-info alert-dismissable fade in" style="margin-top:55px">
              <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
              {{ message }}
            </div>
            {% endfor %}
        {% endif %}
    {% endwith %}

  </div>
{% endblock %}
```

./app/templates/layout.html

```
<!DOCTYPE html>
{# Jinja 2 Template which all other templates will inherit from #}
{# Contains blocks which are overridden by child templates #}
{# Child blocks call super() to get the common elements #}

<html>
<head>
  <!-- load all common libraries and css in head, can be overidden -->
  {% block head %}
  <meta charset="utf-8">
  <title>{% block title %} A2 Project Website{% endblock %}</title>
  <!-- load jquery -->
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js">
  </script>
  <!-- bootstrap css and js -->
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles/main.css') }}">
  <!-- All pages have the same favicon -->
  <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
  {% endblock %}
</head>
<body>
  {% block nav %}
  <nav class="navbar navbar-default">
    <div class="container-fluid">
      <!-- Brand and toggle get grouped for better mobile display -->
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-
target="#bs-example-navbar-collapse-1" aria-expanded="false">
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="{{ url_for('main') }}">A2 Project</a>
      </div>

      <!-- Collect the nav links, forms, and other content for toggling -->
      <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
        <ul class="nav navbar-nav">
          <li><a href="{{ url_for('loci.loci') }}">Loci Plotter</a></li>
          <li><a href="{{ url_for('matrix_blueprint.matrix') }}">Matrix Calculator</a></li>
          <li><a href="{{ url_for('loci.operations') }}">Complex Number Operations</a></li>
          <li>
            <a href="#" class="dropdown-toggle" data-toggle="dropdown">Questions <b
class="caret"></b></a>
            <!-- dropdown link list -->
            <ul class="dropdown-menu multi-level">
              <li class="dropdown-submenu">
                <a href="{{ url_for('questions.questions') }}" class="dropdown-toggle" data-
```

```
toggle="dropdown">Matrix</a>
                  <ul class="dropdown-menu">
                    <li><a href="{{
url_for('questions.show_questions',topic='matrix',q_type='add_sub') }}">Addition and
Subtraction</a></li>
                    <li><a href="{{
url_for('questions.show_questions',topic='matrix',q_type='mult') }}">Multiplication</a></li>
                    <li><a href="{{
url_for('questions.show_questions',topic='matrix',q_type='inv') }}">Inverse</a></li>
                    <li><a href="{{
url_for('questions.show_questions',topic='matrix',q_type='det') }}">Determinant</a></li>
                  </ul>
                </li>
                <li class="dropdown-submenu">
                  <a href="{{ url_for('questions.questions') }}"  class="dropdown-toggle" data-
toggle="dropdown">Complex numbers</a>
                  <ul class="dropdown-menu">
                    <li><a href="{{
url_for('questions.show_questions',topic='complex',q_type='add_sub') }}">Addition and
Subtraction</a></li>
                    <li><a href="{{
url_for('questions.show_questions',topic='complex',q_type='mult') }}">Multiplication</a></li>
                    <li><a href="{{
url_for('questions.show_questions',topic='complex',q_type='div') }}">Division</a></li>
                    <li><a href="{{
url_for('questions.show_questions',topic='complex',q_type='mod_arg') }}">Modulus and
Argument</a></li>
                  </ul>
                </li>
              </ul> <!--end left links-->
            </li>
              <!--right links-->
              <ul class="nav navbar-nav navbar-right">
                {% if current_user.is_authenticated %}
                <li><a href="{{ url_for('user.account') }}">Account</a></li>
                <li><a href="{{ url_for('user.logout') }}">Log Out</a></li>
                {% else %}
                <li><a href="{{ url_for('user.login') }}">Log In</a></li>
                <li><a href="{{ url_for('user.register') }}">Register</a></li>
                {% endif %}
              </ul><!--end right links-->
            </li>
          </ul>
        </div><!-- /.navbar-collapse -->
      </div><!-- /.container-fluid -->
    </nav>
    {% endblock %}

    <div class="container-fluid content">
      {% block main %}
      {% endblock %}
    </div>
    <footer class="footer">
      <div class="container">
      <!-- Same footer for all pages, but can be overridden -->
      {% block footer %}
        <p class="text-muted"> Anik Roy 2017 </p>
      {% endblock %}
      </div>
    </footer>
    <!-- Block for other javascript at end of document -->
    {% block endscripts %}
    {% endblock %}
</body>

</html>
```

./app/templates/loci.html

```
{% extends "layout.html" %}

{% block title %} Loci Plotter {% endblock %}
```

```
{% block head %}
{{ super() }}
<link rel="stylesheet" href="{{ url_for('static',filename='image-picker/image-picker.css') }}">
<!--Load desmos and image-picker javascript-->
<script
src="https://www.desmos.com/api/v0.7/calculator.js?apiKey=dcb31709b452b1cf9dc26972add0fda6"></sc
ript>
<script src="{{ url_for('static',filename='image-picker/image-picker.min.js') }}"></script>

<script>
    // Initialize MathJax and typesetting
    window.MathJax = {
        AuthorInit: function() {
            MathJax.Hub.Register.StartupHook('End', function() {
                MathJax.Hub.processSectionDelay = 0
                var eq_in = document.getElementById('eq_in')
                var MathPreview = document.getElementById('MathPreview')
                var math = MathJax.Hub.getAllJax('MathPreview')[0]
                eq_in.addEventListener('input', function() {
                    MathJax.Hub.Queue(['Text', math, eq_in.value])
                })
            })
        }
    }
</script>
<!--Load MathJax library-->
<script src=https://cdn.mathjax.org/mathjax/latest/MathJax.js?config=AM_HTMLorMML-full.js>
</script>
{% endblock %}

{% block main %}
<row>
  <div class="col-lg-12">
    <!-- equation input group -->
    <div class="input-group" style="margin-top:10px;margin-bottom:10px">
      <span class="input-group-btn">
        <button class="btn btn-primary" type="button" data-toggle="modal" data-target="#help-
modal">Help</button>
      </span>
      <input type="text" class="form-control" placeholder="Enter Equation..." id='eq_in'
name="in">
      <!-- inline button to right of text input-->
      <span class="input-group-btn">
        <button id="go"  class="btn btn-success" type="button">Go</button>
      </span>
    </div>
    <!-- end input-group -->
  </div>
</row>
<div class="col-sm-4">
  <!-- math typesetting preview box-->
  <div>
    <div id=MathPreview style="padding: 3px; width:100%; margin-top:5px;">``</div>
    <div id="MathBuffer" style="padding: 3px; width:100%; margin-top:5px; visibility:hidden;
position:absolute; top:0; left: 0"></div>
  </div>
  <div>
    <!-- empty table to be filled in by javascript-->
    <table id="expressions" class="table">
      <thead>
        <tr>
          <th>Plot</th>
          <th></th>
          <th></th>
        </tr>
      </thead>
      <tbody> </tbody>
    </table>
  </div>
  <div>
    <button type="button" class="btn btn-block" id="clear">Clear All</button>
  </div>
</div>

<div class="col-sm-8">
```

```
  <div id="calculator" style="width:100%; height:600px;"></div>
  <div>
    {# only show save/load graphs if user is logged in #}
    {% if current_user.is_authenticated %}
      <button type="button" class="btn btn-block" id="save-btn" data-toggle="modal" data-
target="#save-modal">Save Graph</button>
      <button type="button" class="btn btn-block" id="load-btn" data-toggle="modal" data-
target="#load-modal">Load Graph</button>
    {% endif %}
  </div>
  <!-- save-modal-->
  <div id="save-modal" class="modal fade" role="dialog">
    <div class="modal-dialog">
      <!-- Modal content-->
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal">&times;</button>
          <h4 class="modal-title">Save Graph</h4>
        </div>
        <!-- form for saving graphs -->
        <form class="form" id="save-form">
          <div class="modal-body">
            <label for="title">Title</label>
            <input id="title" class="form-control" type="text">
            <label for="desc">Description</label>
            <textarea id="desc" class="form-control" form="save-form" ></textarea>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-default" data-dismiss="modal">Cancel</button>
            <button type="button" id="submit-graph" class="btn btn-primary">Save</button>
          </div>
        </form>
        <!-- end form -->
      </div>
      <!-- end modal content-->
    </div>
  </div>
  <!-- end save-modal-->
  <!--load-modal-->
  <div id="load-modal" class="modal fade" role="dialog">
    <div class="modal-dialog">
      <!-- Modal content-->
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal">&times;</button>
          <h4 class="modal-title">Load Graph</h4>
        </div>
        <form class="form" id="load-form">
        <div class="modal-body">
          <label for="title">Select Graph</label>
          <select size="5" class="image-picker show-html" id="graph-select">
            {# load all user_graphs into gallery for users to select from #}
            {% if user_graphs %}
            {# only do this is the user has any graphs at all #}
              {% for g in user_graphs %}
                {# create option in select box for each graph with appropriate image url #}
                <option data-img-src="{{ g.image_url }}" value="{{ g.graph_id }}">{{ g.title
}}</option>
              {% endfor %}
            {% endif %}
          </select>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-default" data-dismiss="modal">Cancel</button>
          <button type="button" id="load-graph" class="btn btn-primary">Load</button>
        </div>
        </form>
      </div>
      <!-- end modal content-->
    </div>
  </div>
  <!--end load-modal-->
  <!-- help modal -->
  <div class="modal fade" id="help-modal" tabindex="-1" role="dialog" aria-
labelledby="exampleModalLabel" aria-hidden="true">
```

```
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal">&times;</button>
          <h4 class="modal-title">Load Graph</h4>
        </div>
        <!-- help modal content -->
        <div class="modal-body">
          <h5>Loci Plotter</h5>
          <p>
            <b>You must use the variable z for equations </b><br>
            Use the | symbol for modulus<br>
            Use arg(...) for the argument function<br>
            You can use pi by typing 'pi' into the input bar<br>
            You must enter an equation or an inequality<br>
            Press the Go button to plot the equation<br>
            You can also save graphs (only if logged in) by pressing the button under the
graph<br>
            Examples:
              |z|=5
              |z+3+i|=2
              arg(z-i)=pi/2
          </p>
        </div>
        <!-- end content -->
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
        </div>
      </div>
    </div>
  </div>
</div>
<!-- end help modal -->
{% endblock %}

{% block endscripts %}
    <script type=text/javascript>
        $SCRIPT_ROOT = {{ request.script_root | tojson | safe }};
    </script>

    <script type="text/javascript" src="{{ url_for('static',filename='scripts/loci_plot.js')
}}">
    </script>
{% endblock %}
```

./app/templates/matrix.html

```
{% extends "layout.html" %}

{% block title %}Matrix Calculator{% endblock %}

{% block head %}
{{ super() }}
<!-- load css -->
<link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.6.0/pure-min.css">
<link rel="stylesheet" media="screen" type="text/css" href="{{
url_for('static',filename='styles/matrix.css') }}">
{% endblock %}

{% block main %}
<div id="outer">
  <div id="inner">
    <div align="center" id="matrices">
      <form class="pure-form" method="post" action="matrix">
        <div id="A-input">
        {# Create 3x3 matrix of inputs with names corresponding to their position #}
          {% for x in ['0','1','2'] %}
            {% for y in ['0','1','2'] %}
              <input type="text" name="{{  'A'+x+y  }}" size="3" class="in" value="{{
request.form['A'+x+y] }}">
            {% endfor %}
            <br>
          {% endfor %}
```

```
            <input type="submit" value="Determinant" name="a-submit" class="pure-button">
            <br>
            <input type="submit" value="Inverse" name="a-submit" class="pure-button">
            <br>
            <input type="submit" value="Transpose" name="a-submit" class="pure-button">
            <br>
            <input type="submit" value="Triangle" name="a-submit" class="pure-button">
            <br>
          </div>

          <div id="operators">
            <input type="submit" value="X" name="submit" class="pure-button op-btn">
            <br>
            <input type="submit" value="+" name="submit" class="pure-button op-btn">
            <br>
            <input type="submit" value="-" name="submit" class="pure-button op-btn">
            <br>
            <input type="button" value="Clear All" onclick="window.location.reload()"
class="pure-button clear-btn">
          </div>

          <div id="B-input">
          {# Create 3x3 matrix of inputs with names corresponding to their position #}
            {% for x in ['0','1','2'] %}
              {% for y in ['0','1','2'] %}
                <input type="text" name="{{  'B'+x+y  }}" size="3" class="in" value="{{
request.form['B'+x+y] }}">
              {% endfor %}
              <br>
            {% endfor %}

            <input type="submit" value="Determinant" name="b-submit" class="pure-button">
            <br>
            <input type="submit" value="Inverse" name="b-submit" class="pure-button">
            <br>
            <input type="submit" value="Transpose" name="b-submit" class="pure-button">
            <br>
            <input type="submit" value="Triangle" name="b-submit" class="pure-button">
            <br>
          </div>
        </form>
      </div>
              <!-- anwer display div-->
      <div id="answer" align="center">
      {# div displays different things depending on result passed to template #}
          {% if det_result %}
            <p id="det"> Determinant: {{ det_result }} </p>
          {% endif %}

          {% if Error %}
            <p id="error"> {{ Error }} </p>
          {% endif %}

          {% if matrix_result %}
            <p>Result:</p>
            <table class="matrix">
              {% for row in matrix_result %}
              <tr>
                  {% for value in row %}
                    <td>{{ value }}</td>
                  {% endfor %}
                </tr>
                {% endfor %}
            </table>
          {% endif %}
      </div>
    </div>
</div>
{% endblock %}
```

./app/templates/operations.html

```
{% extends "layout.html" %}

{% block title %} Argand Diagram {% endblock %}

{% block head %}
{{ super() }}
<link rel="stylesheet" type="text/css" href="http://jsxgraph.uni-
bayreuth.de/distrib/jsxgraph.css" >

<script type="text/javascript" src="http://jsxgraph.uni-bayreuth.de/distrib/jsxgraphcore.js">
</script>
{% endblock %}

{% block main %}
  <div class="col-sm-4">
    <div>
      <!--empty table to be filled in by javascript-->
      <table id="expressions" class="table">
        <thead>
          <tr>
            <th></th>
            <th colspan="2">Point</th>
            <th>Show</th>
            <th>Line</th>
            <th></th>
          </tr>
        </thead>
        <tbody>
          <!--empty tbody-->
        </tbody>
      </table>
    </div>
    <!--add moveable point button-->
    <div>
      <button id="addpoint" type="button" class="btn btn-block btn-primary">
        Add Moveable point
      </button>
    </div>
    <div>
      <!--add calculated point input-->
      <div class="input-group">
        <input id="calc_in" type="text" class="form-control" placeholder="Enter Calculation">
        <span class="input-group-btn">
          <button id="addcalc"  class="btn btn-primary" type="button">
            Add Calculated point
          </button>
        </span>
      </div> <!-- end input group-->
    </div>
    <!--clear all button-->
    <div>
      <button type="button" class="btn btn-block" id="clear">
        Clear All
      </button>
    </div>
  </div> <!--end left column-->
  <div class="col-sm-8"> <!--right column-->
    <!--graphing area-->
    <div id="box" class="jxgbox" style="width:100%; height:600px;"></div>
    <button class="btn btn-primary btn-block" type="button"
    data-toggle="modal" data-target="#help-modal">Help</button>
  </div>

  <!-- help modal -->
  <div class="modal fade" id="help-modal" tabindex="-1" role="dialog" aria-
labelledby="exampleModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal">&times;</button>
          <h4 class="modal-title">Help</h4>
          </div>
          <!-- help modal content -->
          <div class="modal-body">
```

```
              Add a moveable point on argand diagram<br>
              Add calculated point by inputting calculation<br>
              You can change the co-ordinates of moveable points by inputting values into the
input
              boxes in the point list<br>
              You can use the following operations:<br>
              <ul>
              <li>Add Complex Numbers</li>
              <li>Subtract Complex Numbers</li>
              <li>Multiply Complex Numbers</li>
              <li>Divide Complex Numbers</li>
              <li>Raise a Complex Number to an integer power</li>
              </ul>
            </div>
            <!-- end content -->
            <div class="modal-footer">
              <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
            </div>
          </div>
        </div>
      </div><!-- end help modal -->
    </div>
{% endblock %}

{% block endscripts %}
      <script type=text/javascript>
          $SCRIPT_ROOT = {{ request.script_root | tojson | safe }};
      </script>

      <script type="text/javascript" src="{{ url_for('static', filename='scripts/operations.js')
}}"></script>
{% endblock %}
```

./app/templates/emails/confirm_email.html

```
<html>
  <body>
    <p>Hi</p>
    <p>Confirm Here: <a href="{{ confirm_url }}">{{ confirm_url }}</a></p>
  </body>
</html>
```

./app/templates/emails/recover_email.html

```
<html>
  <body>
    <p>Hi</p>
    <br>
    <p>Reset Password Here: <a href="{{ recover_url }}">{{ recover_url }}</a></p>
  </body>
</html>
```

./app/templates/questions/complex_questions.html

```
{% extends "layout.html" %}

{% block title %} Complex Numbers {% endblock %}

{% block head %}
{{ super() }}
<script src=https://cdn.mathjax.org/mathjax/latest/MathJax.js?config=AM_HTMLorMML-full.js>
</script>

<link rel="stylesheet" href="{{ url_for('static',filename='styles/questions.css') }}">
{% endblock %}

{% block main %}
  <div id="main">
    <form id="questions" class="form-inline">
        {# loop over all questions #}
```

```
          {% for n,q in questions %}
            <div class="out_question">
              <!--question-->
              <div class="question">
                {{ q.get_q().replace('(1i','(i').replace('+1i','+i') }}
              </div>
              <!--answer-->
              <div class="answer">
                {# Different formatting for modulus and argument question #}
                {% if not q.is_mod_arg() %}
                  <input type="text" name="{{ n|string+'re'}}" class="form-control input"> `+`
 
                  <input type="text" name="{{ n|string+'im'}}" class="form-control input"> `i`
                {% else %}
                  mod: <input type="text" name="{{ n|string+'mod'}}" class="form-control input">
                  arg: <input type="text" name="{{ n|string+'arg'}}" class="form-control input">
                {% endif %}
              </div>
            </div>
          {% endfor %}
          <!--Submit button-->
          <div class="row">
            <div class="col-lg-12">
              <input id="submit" class="btn btn-primary btn-lg btn-block submit" type="button"
value="Submit Answers">
              <br>
            </div>
          </div>
        </form>
    </div>
{% endblock %}

{% block endscripts %}
     <script type=text/javascript>
          // Variables for the js script
          $SCRIPT_ROOT = {{ request.script_root | tojson | safe }};
          q_type = {{ q_type | tojson | safe }}
          topic = {{ topic | tojson | safe }}
     </script>

     <script type="text/javascript" src="{{ url_for('static',filename='scripts/questions.js')
}}">
     </script>
{% endblock %}
```

./app/templates/questions/mat_questions.html

```
{% extends "layout.html" %}

{% block title %} Complex Numbers {% endblock %}

{% block head %}
{{ super() }}
<script src=https://cdn.mathjax.org/mathjax/latest/MathJax.js?config=AM_HTMLorMML-full.js>
</script>

<link rel="stylesheet" href="{{ url_for('static',filename='styles/questions.css') }}">
{% endblock %}

{% block main %}
  <div id="main">
    <form id="questions" class="form-inline">
      {% for n,q in questions %}
        <div class="out_question">
          <!--question-->
          <div class="question">
            {{ q.get_q().replace('[','(').replace(']',')') }}
          </div>
          <!--answer-->
          <div class="answer">
            {# Display correct type of inputs for the answer type #}
            {% if mat_ans %}
              {# Use answer dimensions to make correct size matrix of inputs #}
```

```
                {% for a in range(q.get_ans_dim()[0]) %}
                  {% for b in range(q.get_ans_dim()[1]) %}
                    <input name="{{ n|string+a|string+b|string }}" type="text" class="form-
control input" size=3>
                  {% endfor %}
                  <br>
                {% endfor %}
              {% else %}
                <input name="{{ n|string }}" type="text" class="form-control input" size=3>
              {% endif %}
            </div>
          </div>
          <br>
        {% endfor %}
        <!--Submit button-->
        <div class="row">
          <div class="col-lg-12">
            <input id="submit" class="btn btn-primary btn-lg btn-block submit" type="button"
value="Submit Answers">
            <br>
          </div>
        </div>
      </form>
    </div>
{% endblock %}

{% block endscripts %}
    <script type=text/javascript>
        // Variables used in javascript
        $SCRIPT_ROOT = {{ request.script_root | tojson | safe }};
        q_type = {{ q_type | tojson | safe}}
        topic = {{ topic | tojson | safe}}
    </script>

    <script type="text/javascript" src="{{ url_for('static',filename='scripts/questions.js')
}}">
    </script>
{% endblock %}
```

./app/templates/questions/questions.html

```
{% extends "layout.html" %}

{% block title %} Questions {% endblock %}

{% block head %}
{{ super() }}
{% endblock %}

{% block main %}
  <!--List of links to questions-->
  <div class="col-lg-6">
    <b>Matrix Questions</b>
    <a class="btn btn-primary btn-lg btn-block"
    href = "{{
      url_for('questions.show_questions',topic='matrix',q_type='add_sub') }}">
      Addition and Subtraction</a>
    <a class="btn btn-primary btn-lg btn-block" href="{{
      url_for('questions.show_questions',topic='matrix',q_type='mult') }}">
      Multiplication</a>
    <a class="btn btn-primary btn-lg btn-block"
    href="{{
      url_for('questions.show_questions',topic='matrix',q_type='det') }}">
      Determinant</a>
    <a class="btn btn-primary btn-lg btn-block"
    href="{{
      url_for('questions.show_questions',topic='matrix',q_type='inv') }}">
    Inverse</a>
  </div>

  <div class="col-lg-6">
    <b>Complex Number Questions</b>
    <a class="btn btn-primary btn-lg btn-block" href
```

```
    ="{{
      url_for('questions.show_questions',topic='complex',q_type='add_sub') }}">
      Addition and Subtraction</a>
    <a class="btn btn-primary btn-lg btn-block"
    href="{{
      url_for('questions.show_questions',topic='complex',q_type='mult') }}">
      Multiplication</a>
    <a class="btn btn-primary btn-lg btn-block"
    href="{{
      url_for('questions.show_questions',topic='complex',q_type='div') }}">
      Division</a>
    <a class="btn btn-primary btn-lg btn-block"
    href="{{
      url_for('questions.show_questions',topic='complex',q_type='mod_arg') }}">
      Argument and Modulus</a>
  </div>
{% endblock %}
```

./app/templates/user/login.html

```
{% extends "layout.html" %}

{% block title %} Log In {% endblock %}

{% block head %}
{{ super() }}
{% endblock %}

{% block main %}
  <div class="container">
    <div id="loginbox" style="margin-top:50px;" class="mainbox col-md-6 col-md-offset-3 col-sm-8
col-sm-offset-2">
      <div class="panel panel-info">
        <div class="panel-heading">
          <div class="panel-title">Sign In</div>
          <!--forgot password link-->
          <div style="float:right; font-size: 80%; position: relative; top:-10px"><a href="{{
url_for('user.reset') }}">Forgot password?</a></div>
        </div>
        <div style="padding-top:30px" class="panel-body">
          <div style="display:none" id="login-alert" class="alert alert-danger col-sm-
12"></div>
          <!--login form -->
          <form id="loginform" name="loginform" class="form-horizontal" role="form" action="{{
url_for('user.login') }}" method="post">
            {{ loginform.hidden_tag() }}
            <!-- email field -->
            <div style="margin-bottom: 25px" class="input-group">
              <span class="input-group-addon">
                <i class="glyphicon glyphicon-user"></i>
              </span>
              {{ loginform.email(class="form-control",placeholder="Username") }}
            </div>
            <!-- password field -->
            <div style="margin-bottom: 25px" class="input-group">
              <span class="input-group-addon"><i class="glyphicon glyphicon-lock"></i></span>
{{ loginform.password(class="form-control",placeholder="Password") }}
            </div>
            <!-- remember me checkbox-->
            <div class="input-group">
              <div class="checkbox">
                <label>
                  {{ loginform.remember(class="") }}
                  Remember Me
                </label>
              </div>
            </div>
            <div style="margin-top:10px" class="form-group">
              <!-- Submit button -->
              <div class="col-sm-12 controls">
                <input type="submit" value="Login" class="btn btn-success">
              </div>
            </div>
```

```
            <!-- link to register form page -->
            <div class="form-group">
              <div class="col-md-12 control">
                <div style="border-top: 1px solid#888; padding-top:15px; font-size:85%">
                  Don't have an account?
                  <a href="{{ url_for('user.register') }}">
                    Sign Up Here
                  </a>
                </div>
              </div>
            </div>
          </form>
          <!-- end form -->
          {# display form validation errors in list under form #}
          {# e.g. password not entered #}
          {% if loginform.errors %}
          <ul class="errors">
               {% for field_name, field_errors in loginform.errors|dictsort if field_errors %}
                    {% for error in field_errors %}
                      <li>
                        {{ loginform[field_name].label }}: {{ error }}
                      </li>
                    {% endfor %}
               {% endfor %}
          </ul>
          {% endif %}
          {# display other messages (success/failure) #}
          {# e.g. incorrect credentials #}
          {% with messages = get_flashed_messages() %}
               {% if messages %}
                    <ul class=flashes>
                         {% for message in messages %}
                              <li>{{ message }}</li>
                         {% endfor %}
                    </ul>
               {% endif %}
          {% endwith %}

        </div>
      </div>
    </div>
  </div>
{% endblock %}
```

./app/templates/user/reset.html

```
{% extends "layout.html" %}

{% block title %}Forgot Password{% endblock %}


{% block head %}
{{ super() }}
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-
awesome.min.css">
{% endblock %}


{% block main %}
<div class="form-gap" style="padding-top:70px"></div>
<div class="container">
  <div class="row">
    <div class="col-md-4 col-md-offset-4">
      <div class="panel panel-default">
        <div class="panel-body">
          <div class="text-center">
            <h3><i class="fa fa-lock fa-4x"></i></h3>
            <h2 class="text-center">Forgot Password?</h2>
            <p>Enter your email address</p>
              <div class="panel-body">
                   <!-- form -->
                <form action="{{ url_for('user.reset') }}" class="form" method="POST">
                  {{ form.hidden_tag() }}
```

```
                    <!--Email field-->
                    <div class="form-group">
                      <div class="input-group">
                        <span class="input-group-addon">
                          <i class="glyphicon glyphicon-envelope color-blue">
                        </i></span>
                        {{ form.email(class="form-control",placeholder="email
address",type="email") }}
                      </div>
                    </div>
                    <!-- Submit button -->
                    <div class="form-group">
                      <input name="recover-submit" class="btn btn-lg btn-primary btn-block"
value="Reset Password" type="submit">
                    </div>
                  </form>
                <!-- form -->
              </div>
            </div>

            {# Show flashed messages (for form validation errors) #}
            {% with messages = get_flashed_messages() %}
              {% if messages %}
                <ul class=flashes>
                  {% for message in messages %}
                    <li>{{ message }}</li>
                  {% endfor %}
                </ul>
              {% endif %}
            {% endwith %}

          </div>
        </div>
      </div>
    </div>
  </div>
{% endblock %}
```

./app/templates/user/reset_with_token.html

```
{% extends "layout.html" %}

{% block title %}Reset Password{% endblock %}

{% block head %}
  {{ super() }}
{% endblock %}

{% block main %}
<div class="container">
  <div id="resetbox" style="margin-top:50px" class="mainbox col-md-6 col-md-offset-3 col-sm-8
col-sm-offset-2">
    <!-- whole form is within a panel -->
    <div class="panel panel-info">
      <div class="panel-heading">
          <div class="panel-title">Change Password</div>
      </div>
      <div class="panel-body">
        <form action="{{ url_for('user.reset_with_token',token=token) }}" method="POST">
          <!-- password field -->
          <div class="form-group">
            <label class="col-md-3 control-label">Password</label>
            <div class="col-md-9">
              {{ form.password(class="form-control") }}
            </div>
          </div>
          <!-- confirm password field -->
          <div class="form-group">
            <label class="col-md-3 control-label">Confirm Password</label>
            <div class="col-md-9">
              {{ form.confirm_password(class="form-control") }}
            </div>
          </div>
```

```
            <div class="form-group">
              <!-- Submit button -->
              <div class="col-md-offset-3 col-md-9">
                <input type="submit" class="btn btn-info btn-block" value="Change Password">
              </div>
            </div>
          </form>
        </div>
        {% if form.errors %}
        <ul class="errors">
          {% for field_name, field_errors in form.errors|dictsort if field_errors %}
            {% for error in field_errors %}
              <li>{{ form[field_name].label }}: {{ error }}</li>
            {% endfor %}
          {% endfor %}
        </ul>
        {% endif %}
      </div>
    </div>
</div>
{% endblock %}
```

./app/templates/user/signup.html

```
{% extends "layout.html" %}

{% block title %} Sign Up {% endblock %}

{% block head %}
{{ super() }}
{% endblock %}


{% block main %}
<div class="container">
   <div id="signupbox" style="margin-top:50px" class="mainbox col-md-6 col-md-offset-3 col-sm-8
col-sm-offset-2">
     <div class="panel panel-info">
       <div class="panel-heading">
         <div class="panel-title">Sign Up</div>
       </div>
       <div class="panel-body">
         <form id="signupform" name="regform" class="form-horizontal" role="form" action="{{
url_for('user.register') }}" method="post">
           {{ regform.hidden_tag() }}
           <!-- email field-->
           <div class="form-group">
             <label for="email" class="col-md-3 control-label">
               Email
             </label>
             <div class="col-md-9">
               {{ regform.email(class="form-control",placeholder="Email") }}
             </div>
           </div>
           <!--first name field-->
           <div class="form-group">
             <label for="firstname" class="col-md-3 control-label">
               First Name
             </label>
             <div class="col-md-9">
               {{ regform.fname(class="form-control",placeholder="First Name") }}
             </div>
           </div>
           <!--last name field-->
           <div class="form-group">
             <label for="lastname" class="col-md-3 control-label">
               Last Name
             </label>
             <div class="col-md-9">
               {{ regform.lname(class="form-control",placeholder="Last Name",placeholder="Last
Name") }}
             </div>
           </div>
```

```
            <!--password field-->
            <div class="form-group">
                 <label for="password" class="col-md-3 control-label">
                   Password
                 </label>
                 <div class="col-md-9">
                      {{ regform.password(class="form-control",placeholder="Password") }}
                 </div>
            </div>
            <!--confirm password field-->
            <div class="form-group">
              <label for="password" class="col-md-3 control-label">
                Confirm Password
              </label>
              <div class="col-md-9">
                {{ regform.confirm_password(class="form-control",placeholder="Confirm Password")
}}
              </div>
            </div>
            <div class="form-group">
              <!-- Submit button -->
              <div class="col-md-offset-3 col-md-9">
                <input type="submit" value="Sign Up" class="btn btn-info">
              </div>
            </div>
            <!--Link to login page-->
            <div class="form-group">
              <div class="col-md-12 control">
                <div style="border-top: 1px solid#888; padding-top:15px; font-size:85%">
                  Have an account?
                  <a href="{{ url_for('user.login') }}">Sign In Here</a>
                </div>
              </div>
            </div>
          </form>
          {# Simple form validation errors get passed to this #}
          {# e.g. first name not entered #}
          {% if regform.errors %}
          <ul class="errors">
               {% for field_name, field_errors in regform.errors|dictsort if field_errors %}
                 {% for error in field_errors %}
                   <li>{{ regform[field_name].label }}: {{ error }}</li>
                 {% endfor %}
               {% endfor %}
          </ul>
          {% endif %}

          {# Flashed messages for other form validation errors #}
          {# e.g. user already exists #}
          {% with messages = get_flashed_messages() %}
            {% if messages %}
              <ul class=flashes>
                {% for message in messages %}
                  <li>{{ message }}</li>
                {% endfor %}
              </ul>
            {% endif %}
          {% endwith %}
        </div>
      </div>
    </div>
</div>
{% endblock %}
```

./app/templates/user/student_account.html

```
{% extends "layout.html" %}

{% block title %} Account {% endblock %}

{% block head %}
{{ super() }}
```

```
<script src="{{ url_for('static',filename='image-picker/image-picker.min.js') }}">
</script>

<link rel="stylesheet" href="{{ url_for('static',filename='styles/account.css') }}">

<link rel="stylesheet" href="{{ url_for('static',filename='image-picker/image-picker.css') }}">
{% endblock %}

{% block main %}
<div id="title" class="row">
  <div class="col-sm-12">
    <div class="alert alert-info title">
      <strong>Your Account</strong>
    </div>
  </div>
</div>
<div data-spy="scroll" data-target=".scrollspy" style="position:relative">
<div class="col-md-3 scrollspy">
  <ul id="nav" class="nav hidden-xs hidden-sm affix-top" data-spy="affix">
    <li><a href="#links">Links</a></li>
    <li><a href="#set">Set Tasks</a></li>
    <li><a href="#graphs">Graphs</a></li>
    <li><a href="#scores">Scores</a></li>
    <li><a href="#settings">Settings</a>
      <ul id="nav">
        <li>
          <a href="#change-details">Details</a>
        </li>
        <li>
          <a href="#pwd">Password</a>
        </li>
      </ul>
    </li>
  </ul>
</div>

<div class="col-md-9">
  {# Show flashed messages #}
  {% with messages = get_flashed_messages() %}
    {% if messages %}
      {% for message in messages %}
      <div class="alert alert-info alert-dismissable fade in">
        <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
        {{ message }}
      </div>
      {% endfor %}
    {% endif %}
  {% endwith %}
  {% if changeform.errors %}
  <ul class="errors">
      {% for field_name, field_errors in changeform.errors|dictsort if field_errors %}
          {% for error in field_errors %}
              <li>{{ changeform[field_name].label }}: {{ error }}</li>
          {% endfor %}
      {% endfor %}
  </ul>
  {% endif %}

  {% if pwform.errors %}
  <ul class="errors">
      {% for field_name, field_errors in pwform.errors|dictsort if field_errors %}
          {% for error in field_errors %}
              <li>{{ pwform[field_name].label }}: {{ error }}</li>
          {% endfor %}
      {% endfor %}
  </ul>
  {% endif %}


  <section id="links">
  <!-- Form for students to link to teachers -->
    <h1>Links</h1>
    <h2>Add Teacher</h2>
    <form method="post" action="{{ url_for('user.account') }}">
```

```
    {{ linkform.hidden_tag() }}
    <div class="form-group">
      {{ linkform.link_code(class="form-control form-control-lg",placeholder="Enter Code...") }}
    </div>
    <div class="form-group">
      {{ linkform.link_submit(class="btn btn-primary") }}
    </div>
    </form>
    <h2>Existing Links:</h2>
    <table id="links-table" class="table">
      <thead>
        <th>Fname</th>
        <th>Lname</th>
        <th></th>
      </thead>
      <tbody>
        {% for t in student.teachers.all() %}
        <tr id="link-row{{ t.teacher_id }}">
          <td>{{ t.fname }}</td>
          <td>{{ t.lname }}</td>
          <td><input type="button" class="btn btn-danger" id="{{ t.teacher_id }}"
value="Delete"></td>
        </tr>
        {% endfor %}
      </tbody>
    </table>
  </section>

  <section id="set">
  <!-- Display tasks which are set-->
    <h1>Set Tasks</h1>
    {# Show tasks table if there any set tasks #}
    {% if student.tasks.first() %}
    <table class="table">
      <thead>
        <th>Topic</th>
        <th>Task Name</th>
        <th>Completed</th>
        <th>Mark</th>
        <th>Percent</th>
        <th>Date Completed</th>
        <th></th>
      </thead>
      <tbody>
      {# Loop though all tasks and render information in table #}
      {% for t in student.tasks.all() %}
        <tr>
          <td>{{ qs[t.question_id]['topic'] }}</td>
          <!-- Link to relevant questions -->
          <td><a href="{{ url_for('questions.show_questions',
topic=qs[t.question_id]['topic'].lower(), q_type=qs[t.question_id]['q_type']) }}">
          {{ qs[t.question_id]['name'] }}</a></td>
          <td>{{ t.completed }}</td>

          {% if t.mark %}
            <td>{{ t.mark.score }}/{{ t.mark.out_of }}</td>
            <td>{{ t.mark.score/t.mark.out_of*100 }}%</td>
            <td>{{ t.mark.date }}</td>
          {% endif %}
        </tr>
      {% endfor %}
      </tbody>
    </table>
    {% endif %}
  </section>

  <section id="graphs">
  <!-- Gallery shows saved graphs-->
  <!-- Each graph also link to the loci plotter page (form posts to that url with a graph id)-->
    <h1>Graphs</h1>
    <form action="{{ url_for('loci.loci') }}" method="get" id="graphform">
      <select size="5" class="image-picker show-html" id="graph-select" form="graphform"
name="id">
        {% for g in student.graphs.all() %}
        <!--option with buttons underneath-->
```

```
            <option data-img-label="{{ g.title }}<br>
              {{ g.description }}
              <br><button class='btn btn-block'>Load</button><br>
              <input type='button' value='Delete'
              id='del-graph{{ g.graph_id }}' class='btn btn-block'>"
              data-img-src="{{ g.image_url }}"
              value="{{ g.graph_id }}">
            </option>
          {% endfor %}
        </select>
      </form>
    </section>


    <section id="scores">
    <h1>Task Scores</h1>
    <!--Show scores on previous tasks and completed questions-->
      <table class="table">
        <thead>
          <th> Topic </th>
          <th> Score </th>
          <th> Percentage </th>
        </thead>
        <tbody>
          {% for mark in student.marks.all() %}
          <tr>
            <td>{{ qs[mark.question_id]["topic"] }} {{ qs[mark.question_id]["name"] }}</td>
            <td>{{ mark.score }}/{{ mark.out_of }}</td>
            <td>{{ mark.score/mark.out_of*100 }}%</td>
          </tr>
          {% endfor %}
        </tbody>
      </table>
    </section>

    <section id="settings">
      <!--Forms for changing details-->
    <h1>Settings</h1>
      <h2>Change Details</h2>
      <div id="change-details"></div>
      <form id="change-details" class="form-horizontal" method="post", action="{{
url_for('user.account') }}">
        {{ changeform.hidden_tag() }}
        <div class="form-group">
          {{ changeform.fname.label(class="col-xs-2 control-label") }}
          <div class="col-xs-10">
            {{ changeform.fname(value=student.fname,class="form-control") }}
          </div>
        </div>
        <div class="form-group">
          {{ changeform.lname.label(class="col-xs-2 control-label") }}
          <div class="col-xs-10">
            {{ changeform.lname(value=student.lname,class="form-control") }}
          </div>
        </div>
        <div class="form-group">
          {{ changeform.email.label(class="col-xs-2 control-label") }}
          <div class="col-xs-10">
            {{ changeform.email(value=student.email,class="form-control") }}
          </div>
        </div>
        <div class="form-group">
          {{ changeform.password.label(class="col-xs-2 control-label") }}
          <div class="col-xs-10">
            {{ changeform.password(class="form-control") }}
          </div>
        </div>
        <div class="form-group">
          <div class="col-xs-offset-2 col-xs-10">
            {{ changeform.change_submit(class="btn btn-primary") }}
          </div>
        </div>
      </form>
```

```
    <h2>Password change</h2>
    <form id="pwd" class="form-horizontal" method="post", action="{{ url_for('user.account')
}}">
      {{ pwform.hidden_tag() }}
      <div class="form-group">
        {{ pwform.old_password.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ pwform.old_password(class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        {{ pwform.password.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ pwform.password(class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        {{ pwform.confirm_password.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ pwform.confirm_password(class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        <div class="col-xs-offset-2 col-xs-10">
          {{ pwform.pw_submit(class="btn btn-primary") }}
        </div>
      </div>
    </form>
  </section>
</div>
</div>
</div>
{% endblock %}

{% block endscripts %}
<script>
    $SCRIPT_ROOT = {{ request.script_root | tojson | safe }};
</script>
<script src="{{ url_for('static',filename='scripts/student_account.js') }}">
</script>
{% endblock %}
```

./app/templates/user/teacher_account.html

```
{% extends "layout.html" %}

{% block title %} Account {% endblock %}

{% block head %}
{{ super() }}

<script src="{{ url_for('static',filename='image-picker/image-picker.min.js') }}">
</script>

<link rel="stylesheet" href="{{ url_for('static',filename='styles/account.css') }}">

<link rel="stylesheet" href="{{ url_for('static',filename='image-picker/image-picker.css') }}">

<link rel="stylesheet" href="http://netdna.bootstrapcdn.com/font-awesome/3.2.1/css/font-
awesome.css">
{% endblock %}

{% block main %}
<div id="title" class="row">
  <div class="col-sm-12">
    <div class="alert alert-info title">
      <strong>Your Account</strong>
    </div>
  </div>
</div>
<div data-spy="scroll" data-target=".scrollspy" style="position:relative">
<div class="col-md-3 scrollspy">
  <ul id="nav" class="nav hidden-xs hidden-sm affix-top" data-spy="affix">
```

```
      <li><a href="#links">Links</a></li>
      <li><a href="#set">Set Tasks</a></li>
      <li><a href="#graphs">Graphs</a></li>
      <li><a href="#settings">Settings</a>
        <ul id="nav">
          <li>
            <a href="#change-details">Details</a>
          </li>
          <li>
            <a href="#pwd">Password</a>
          </li>
        </ul>
      </li>
    </ul>
  </div>
  <div class="col-md-9">
      {% with messages = get_flashed_messages() %}
        {% if messages %}
          {% for message in messages %}
          <div class="alert alert-info alert-dismissable fade in">
            <a href="#" class="close" data-dismiss="alert" aria-label="close">&times;</a>
            {{ message }}
          </div>
          {% endfor %}
        {% endif %}
      {% endwith %}

      {% if changeform.errors %}
      <ul class="errors">
          {% for field_name, field_errors in changeform.errors|dictsort if field_errors %}
              {% for error in field_errors %}
                  <li>{{ changeform[field_name].label }}: {{ error }}</li>
              {% endfor %}
          {% endfor %}
      </ul>
      {% endif %}

      {% if pwform.errors %}
      <ul class="errors">
          {% for field_name, field_errors in pwform.errors|dictsort if field_errors %}
              {% for error in field_errors %}
                  <li>{{ pwform[field_name].label }}: {{ error }}</li>
              {% endfor %}
          {% endfor %}
      </ul>
      {% endif %}


    <section id="links">
      <h1>Links</h1>
      <p><strong>Your Code</strong>: {{ teacher.code }}</p>

      <table id="links-table" class="table">
        <thead>
          <th>Name</th>
          <th></th>
        </thead>
        <tbody>
          {% for s in teacher.students.all() %}
          <tr id="link-row{{ s.student_id }}">
            <td>{{ s.fname }} {{ s.lname }}</td>
            <td><input id="{{ s.student_id }}" type="button" class="btn btn-danger"
              value="Delete"></td>
          </tr>
          {% endfor %}
        </tbody>
      </table>
      <h2>Current Tasks</h2>
      {% for s in students %}
        <ul>
          <li id="task-row{{ s.student_id }}">{{ s.fname }} {{ s.lname }}
            {% if s.tasks.first() %}
              <table class="table">
                <thead>
```

```
                    <th>Topic</th>
                    <th>Task Name</th>
                    <th>Completed</th>
                    <th>Mark</th>
                    <th>Percent</th>
                    <th>Date Completed</th>
                  </thead>
                  <tbody>
                  {% for t in s.tasks.all() %}
                    <tr>
                      <td>{{ qs[t.question_id]['topic'] }}</td>
                      <td>{{ qs[t.question_id]['name'] }}</td>
                      <td>{{ t.completed }}</td>
                      {% if t.mark %}
                      <td>{{ t.mark.score }}/{{ t.mark.out_of }}</td>
                      <td>{{ t.mark.score/t.mark.out_of*100 }}%</td>
                      <td>{{ t.mark.date }}</td>
                      {% endif %}
                    </tr>
                  {% endfor %}
                  </tbody>
                </table>
                {% endif %}
              </li>
          </ul>
      {% endfor %}
    </section>

    <section id="set">
      <h1>Set Tasks</h1>
      <form method="post", action="{{ url_for('user.account') }}">
        <div class="col-xs-6">
          <h3>Students</h3>
          {% for student in setform.student_select %}
          <div class="form-check">
            <label class="form-check-label">
                {{ student }}
                {{ student.label }}
            </label>
          </div>
          {% endfor %}

        </div>
        <div class="col-xs-6">
          <h3>Tasks</h3>
          {% for task in setform.task_select %}
          <div class="form-check">
            <label class="form-check-label">
              {{ task }}
              {{ task.label }}
          </div>
          {% endfor %}
        </div>
        <div>
          {{ setform.set_submit(class="btn btn-primary",value='Set') }}
        </div>
      </form>
    </section>

    <section id="graphs">
      <h1>Graphs</h1>
      <form action="{{ url_for('loci.loci') }}" method="get" id="graphform">
        <select size="5" class="image-picker show-html" id="graph-select" form="graphform"
name="id">
          {% for g in teacher.graphs.all() %}
            <option data-img-label="{{ g.title }}<br>{{ g.description }}
              <br><button class='btn btn-block'>Load</button><br>
              <input type='button' value='Delete'
              id='del-graph{{ g.graph_id }}' class='btn btn-block'>"
              data-img-src="{{ g.image_url }}" value="{{ g.graph_id }}">
            </option>
          {% endfor %}
        </select>
      </form>
```

```
  </section>

  <section id="settings">
    <h1>Settings</h1>
    <h2>Change Details</h2>
    <form id="change-details" class="form-horizontal" method="post", action="{{
url_for('user.account') }}">
      {{ changeform.hidden_tag() }}
      <div class="form-group">
        {{ changeform.fname.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ changeform.fname(value=teacher.fname,class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        {{ changeform.lname.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ changeform.lname(value=teacher.lname,class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        {{ changeform.email.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ changeform.email(value=teacher.email,class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        {{ changeform.password.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ changeform.password(class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        <div class="col-xs-offset-2 col-xs-10">
          {{ changeform.change_submit(class="btn btn-primary") }}
        </div>
      </div>
    </form>

    <h2>Password change</h2>
    <form id="pwd" class="form-horizontal" method="post", action="{{ url_for('user.account')
}}">
      {{ pwform.hidden_tag() }}
      <div class="form-group">
        {{ pwform.old_password.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ pwform.old_password(class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        {{ pwform.password.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ pwform.password(class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        {{ pwform.confirm_password.label(class="col-xs-2 control-label") }}
        <div class="col-xs-10">
          {{ pwform.confirm_password(class="form-control") }}
        </div>
      </div>
      <div class="form-group">
        <div class="col-xs-offset-2 col-xs-10">
          {{ pwform.pw_submit(class="btn btn-primary") }}
        </div>
      </div>
    </form>
  </section>
</div>
</div>
{% endblock %}

{% block endscripts %}
<script>
```

```
    $SCRIPT_ROOT = {{ request.script_root | tojson | safe }};
</script>
<script src="{{ url_for('static',filename='scripts/teacher_account.js') }}">
</script>
{% endblock %}
```