itucsdb Documentation

Release 1.0

ITUCSDB 12

Dec 22, 2017

CONTENTS

Team ITUCSDB 12

Members

- Oğuzhan Kocatürk
- Muhammed İşiyok

Project development is a hard and expensive process. If a project is small and a couple of people work on it, there will be no communication problem between them. But when the projects get larger and larger, communication between its members also getting hard. In order to solve this communication problem and make project creation process easier we created a web application named ProjectEasy. This project aims to create a great communication between project members and deal with arrangement of project tasks.

Contents:

CHAPTER

ONE

USER GUIDE

1.1 Parts Implemented by Oguzhan Kocaturk

Welcome to the user guide. In this guide I will show you how to use ProjectEasy pages. When you first enter to the website you will see the login screen(Fig 1.1). There is no "Sign Up" because system admin can only add companies and after that companies log in to the system and they create their own employees.

Fig. 1.1: Login Page.

If you want to create a company you need admin account to do that job. You can reach create company menu only after you logged in as admin After logged in, you can click create company menu on the topbar and reach the creation page(Fig 1.2).

When you fill the form and click submit, company will be created. But also a user with the type of the company will be created. Its username will be the name of the company, and its password will be the password that you have just entered. If you want to see all companies you can click on the "List Companies" button(Fig 1.3).

If you want to delete or alter a company you can click "Alter company" menu(Fig 1.4). You need to write company name to change its properties or delete it. After you write the name of the company you can go update page by clicking update button or you can delete it from the database by clicking delete button.

Finally you can click to "Logout" button to logout from the system.

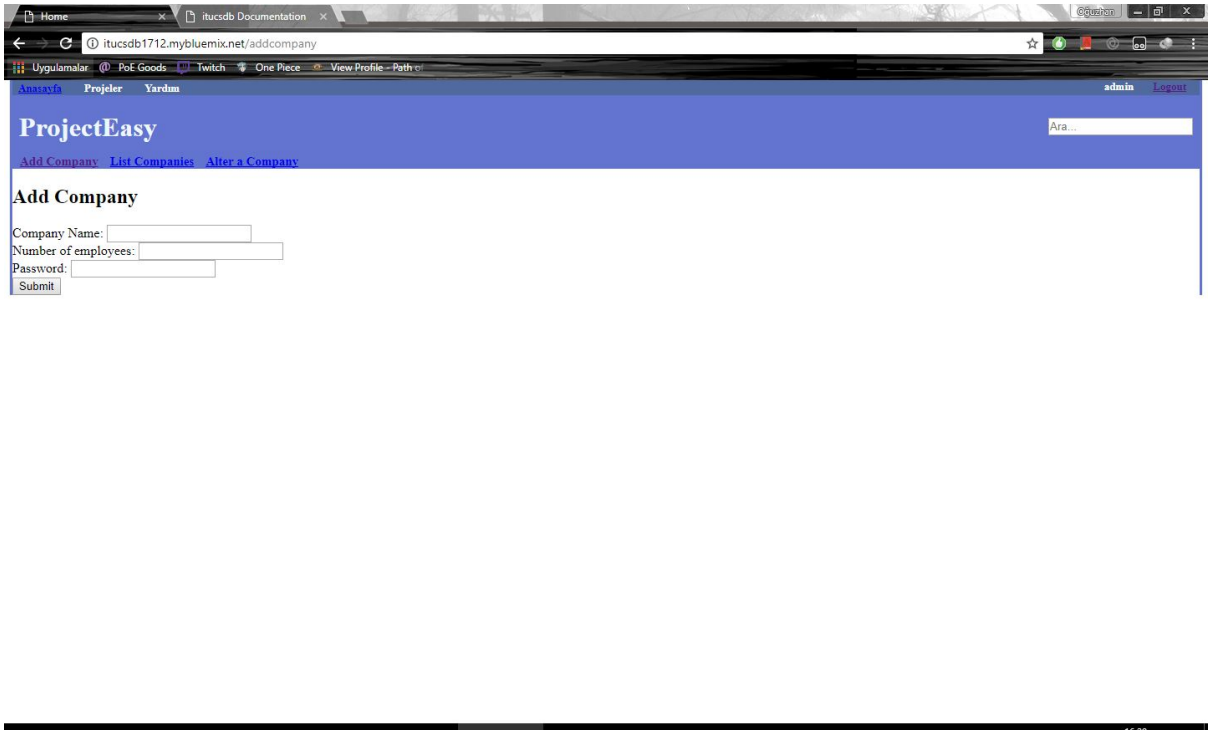

Fig. 1.2: Create Company Page.

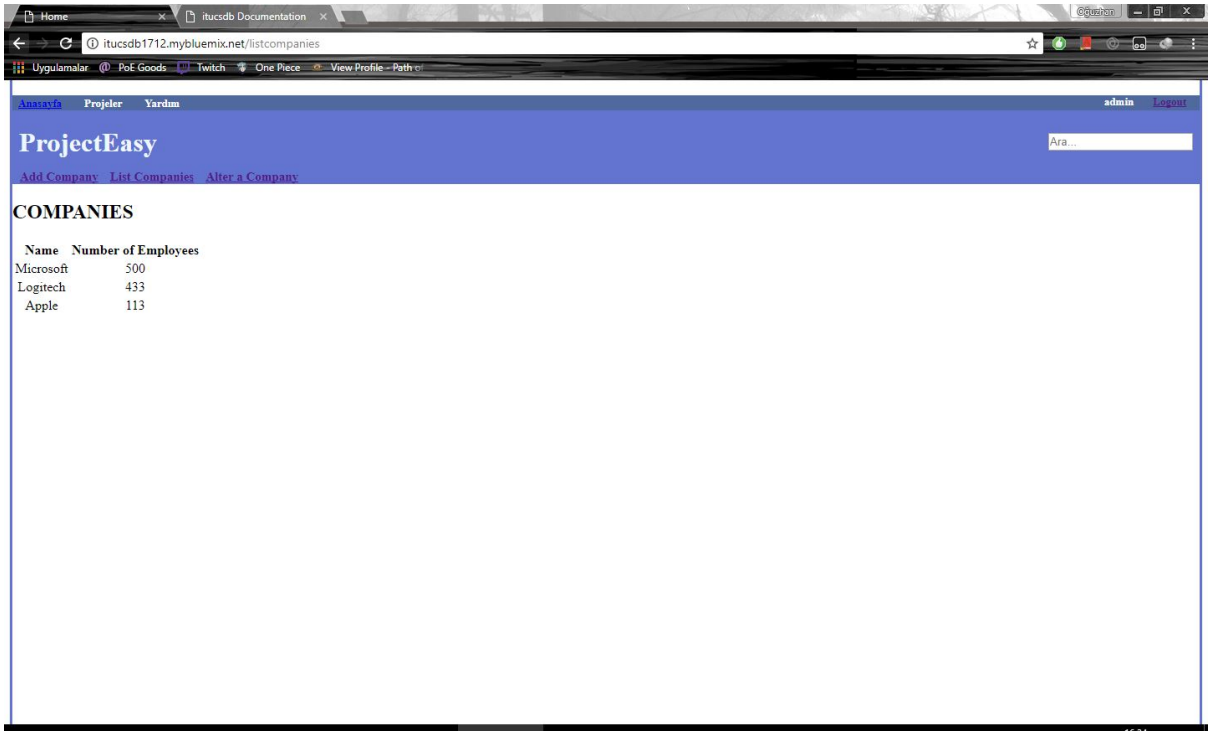

Fig. 1.3: List Company Page

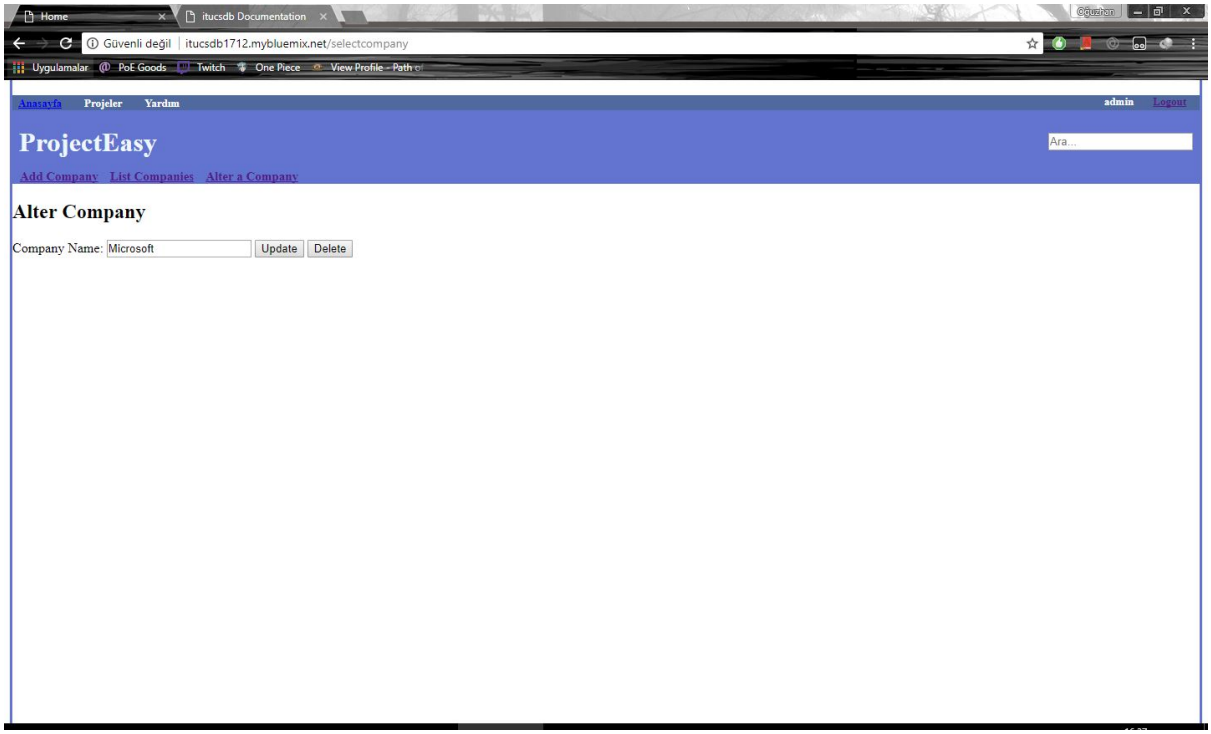

Fig. 1.4: Alter Company Page

You can now login to the system as a company that you have created before. When you log in as a company you can add employee accounts to the system through “Add Employee” menu(Fig 1.5). When you fill the form and click submit button system will create a username and password for the employee so employee can be log in to the system by using those username and password.

You can also change user information through “List Employee” menu. If you click delete next to employee that you want to delete, the employee will be deleted from the system. You can also click to the update button next to employee. After that you can alter the employee by the form next to it. If you want to include an employee into a project, you can select a project from the dropdown list. If you do not include him in a project at the creation, you can include it by update menu later(Fig 1.6).

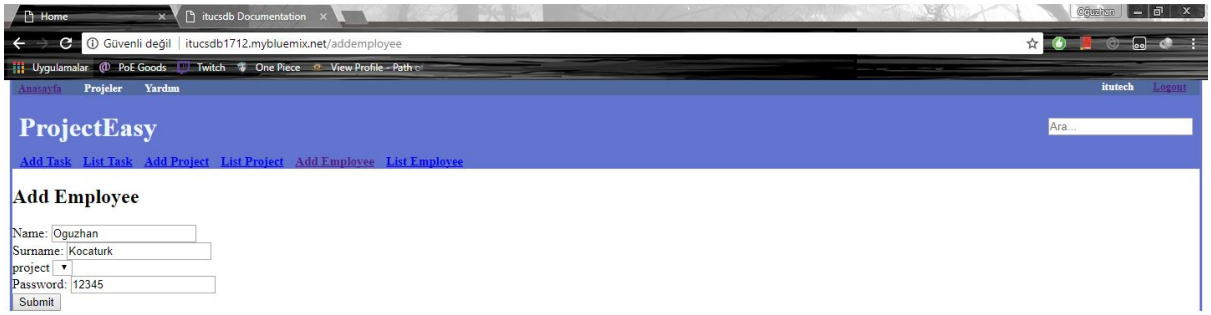

Fig. 1.5: Add Employee Page

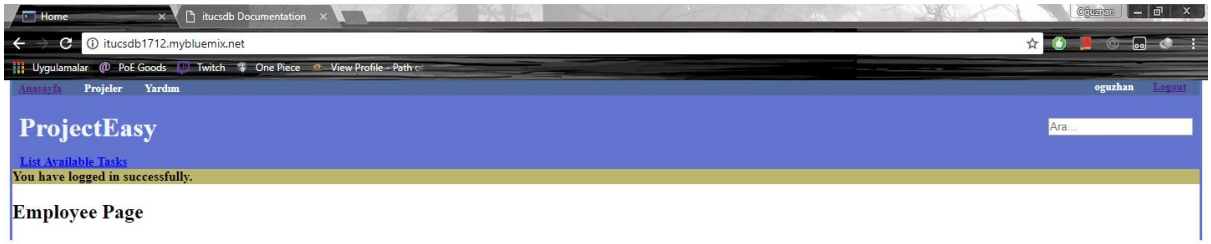

Fig. 1.6: Employee Page

1.2 Parts Implemented by Muhammed Isiyok

CHAPTER

TWO

DEVELOPER GUIDE

2.1 Database Design

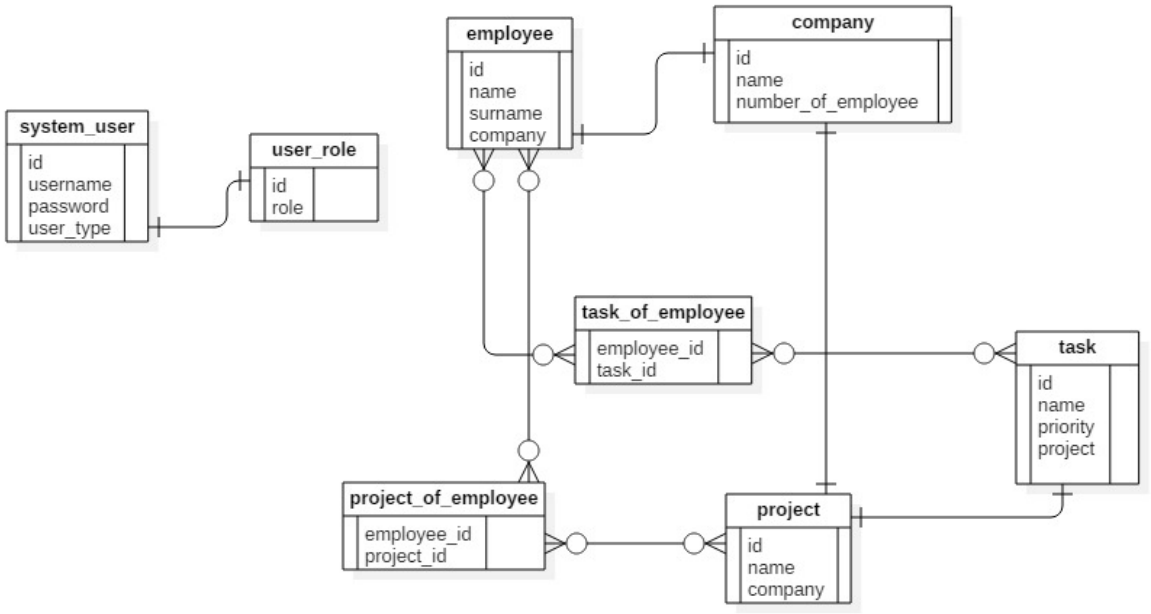

Fig. 2.1: E/R Diagram of the Database.

2.1.1 Parts Implemented by Oguzhan Kocaturk

In this guide, I will explain source codes and their function.

server.py

```
    @app.route('/', methods=['GET', 'POST'])
    def login_page():
session['username'] = None
session['logged_in'] = False
session['user_type'] = 4
form = LoginForm()
if form.validate_on_submit():
    username = form.username.data
    password = form.password.data
    information = returnCompany(username)
```

```
    hash = getUserPwHash(username)
    if hash is None:
        flash('You have entered wrong username or password.')
        return render_template('login.html', form = form, username = session.get(
↪'username', None))
    if hash[0] == password:
        session['logged_in'] = True
        flash('You have logged in successfully.')
        session['username'] = username
        if hash[1] == 1:
            session['user_type'] = 1
            return render_template('homepage_admin.html', username = session.get(
↪'username', None))
        if hash[1] == 2:
            session['user_type'] = 2
            session['company_number'] = information[0][0]
            return render_template('homepage_company.html', username = session.get(
↪'username', None))
        if hash[1] == 3:
            session['user_type'] = 3
            return render_template('homepage_employee.html', username = session.
↪get('username', None))
    else:
        flash('You have entered wrong username or password.')
        session['username'] = None
return render_template('login.html', form = form, username = session.get('username
↪', None))
```

This code piece define the login page and its functions. It checks whether username is in the database or not. If its in the database, function gets the user type information and write it into the session's user_type. If login information is correct, function calls home_page function. If login information is false then it renders the login page again.

```
    @app.route('/home')
    def home_page():
if session.get('user_type', None) == 1:
    return render_template('homepage_admin.html', username = session.get('username
↪', None))
if session.get('user_type', None) == 2:
    return render_template('homepage_company.html', username = session.get(
↪'username', None))
if session.get('user_type', None) == 3:
    return render_template('homepage_employee.html', username = session.get(
↪'username', None))
if session.get('user_type', None) == 4:
    return redirect(url_for('login_page'))
```

According to the user type, this function and route returns proper homepage.

```
    @app.route('/logout')
    def logout():
session['logged_in'] = False
session['username'] = None
flash('You have been logged out successfully.')
return redirect(url_for('login_page'))
```

This logout function removes session information and returns the login page.

company_view.py

```
    @company_app.route('/addcompany', methods=['GET', 'POST'])
    def add_company():
form = AddCompanyForm()
if form.validate_on_submit():
    company_name = form.company_name.data
    number_of_employees = form.number_of_employees.data
    password = form.company_account_pw.data
    user_type = 2
    addUserToDb(company_name, password, user_type)
    addCompanyToDb(company_name, number_of_employees)
    flash('Company added successfully.')
    return render_template('homepage_admin.html', form=form)
return render_template('addcompany.html', form=form, username = session.get(
↪'username', None))
```

In this part, function indentify form variable and send it into the html. Read html form data and send this information into the function that handles database operation. Then return homepage.

```
    @company_app.route('/listcompanies', methods=['GET'])
    def list_companies():
information = listCompanies()
return render_template('listcompanies.html', informations = information, username␣
↪= session.get('username', None))
```

This part, function calls another database function to get database information about companies. Then it renders the information on the html page.

```
    @company_app.route('/selectcompany', methods=['GET', 'POST'])
    def select_company():
form = SelectCompanyForm()
if form.validate_on_submit():
    global company_name_global
    company_name_global = form.company_name.data
    if form.submitUpdate.data is True:
        return redirect(url_for('company_app.update_company'))
    if form.submitDelete.data is True:
        return redirect(url_for('company_app.delete_company'))
    return render_template('selectcompany.html', form = form)
return render_template('selectcompany.html', form = form, username = session.get(
↪'username', None))
```

The role of this route is to select a company for delete or update operation. According to the selection it renders proper route to handle the operation.

```
    @company_app.route('/updatecompany', methods=['GET', 'POST'])
    def update_company():
information = returnCompany(company_name_global)
form = AddCompanyForm(company_name=information[0][1], number_of_
↪employees=information[0][2])
company_id = information[0][0]
if form.submit.data is True:
    if form.validate_on_submit():
        name = form.company_name.data
        number_of_employees = form.number_of_employees.data
        print(name)
        print(number_of_employees)
        updateCompany(company_id, name, number_of_employees)
        flash('Company updated successfully.')
        return render_template('adminpage.html')
return render_template('updatecompany.html', form = form)
```

This function get information about company from database and fill the blank form places with that information. If you make any changes and click submit button it will call update function that applies the changes to the database.

```
    @company_app.route('/deletecompany', methods=['GET', 'POST'])
    def delete_company():
information = returnCompany(company_name_global)
company_id = information[0][0]
username = information[0][1]
print(username)
deleteCompany(company_id)
deleteUser(username)
flash('Company deleted successfully.')
return render_template('homepage_admin.html')
```

This route get the username from the form and searches it in the database. If it finds the information, it deletes the proper information from the database.

employee_view.py

```
    @employee_app.route('/addemployee', methods=['GET', 'POST'])
    def add_employee():
form = EmployeeForm()
information = returnAllProjects(session.get('company_number', None))
form.project.choices = information
if form.validate_on_submit():
    name = form.name.data
    surname = form.surname.data
    password = form.password.data
    project = form.project.data
    addEmployeeToDb(name, surname, session.get('company_number', None))
    employee_id = returnEmployeeId(name)
    createProjectEmployeeRelation(employee_id, project)
    user_type = 3
    addUserToDb(name, password, user_type)
    flash('Employee has been added successfully.')
    return render_template('homepage_company.html', username = session.get(
↪'username', None))
return render_template('addemployee.html', form = form, username = session.get(
↪'username', None))
```

This function returns a html with form. After that, it gets the information from form and send it into the function that modifies database.

```
    @employee_app.route('/listemployee', methods=['GET', 'POST'])
    def list_employee():
form = EmployeeForm(request.form)
information = returnAllProjects(session.get('company_number', None))
form.project.choices = information
tasks = getEmployeeFromDb(session.get('company_number', None))
return render_template('listemployee.html', tasks = tasks, form = form, username =␣
↪session.get('username', None))
```

This route first get database information about employees and then send the information into the html.

```
    @employee_app.route('/deleteemployee/<employee_id>', methods=['GET', 'POST'])
    def delete_empoloyee(employee_id):
deleteEmployeeFromDb(employee_id)
return redirect(url_for('employee_app.list_employee'))
```

Calls the delete function with the information of the employee comes from form data.

```
    @employee_app.route('/updateemployee/<employee_id>', methods=['GET', 'POST'])
    def update_employee(employee_id):
form = EmployeeForm(request.form)
information = returnAllProjects(session.get('company_number', None))
form.project.choices = information
name = form.name.data
surname = form.surname.data
project = form.project.data
form.name.data = ''
form.surname.data = ''
form.project.data = ''
updateEmployeeInDb(name, surname ,employee_id)
createProjectEmployeeRelation(employee_id, project)
return redirect(url_for('employee_app.list_employee'))
```

Get information from the form data and update the user comes from the route. This function calls another database function with the information it has.

```
    @employee_app.route('/listemployeetask', methods=['GET', 'POST'])
    def list_employee_task():
information = returnAllTasks(session.get('username', None))
form = EmployeeForm()
return render_template('listemployeetask.html', tasks = information, form = form,
                       username = session.get('username', None))
```

This function calls a function that returns tasks of the employee. Then it sends the information to the html file.

database.py

```
    def initdb(dsn):
try:
    global connection
    connection = dbapi2.connect(dsn)
    print("Connected to database.")
except:
    print("Database connection failed.")
try:
    cursor = connection.cursor()
    statement = """CREATE TABLE IF NOT EXISTS company (
                id SERIAL,
                name VARCHAR(20),
                number_of_employees INTEGER,
                PRIMARY KEY (id))"""
    cursor.execute(statement)

    statement = """CREATE TABLE IF NOT EXISTS project (
                        id SERIAL PRIMARY KEY,
                        name VARCHAR(20),
                        company INTEGER REFERENCES company ON DELETE CASCADE)"""
    cursor.execute(statement)

    statement = """CREATE TABLE IF NOT EXISTS task (
                        id SERIAL PRIMARY KEY,
                        name VARCHAR(20),
                        priority INTEGER,
                        project INTEGER REFERENCES project ON DELETE CASCADE)"""
    cursor.execute(statement)
    statement = """CREATE TABLE IF NOT EXISTS user_role (
                id SERIAL,
                role VARCHAR(10),
                PRIMARY KEY(id))"""
```

```
    cursor.execute(statement)
    statement = """CREATE TABLE IF NOT EXISTS system_user (
                id SERIAL,
                username VARCHAR(20),
                password VARCHAR(100),
                user_type INTEGER REFERENCES user_role ON DELETE CASCADE,
                PRIMARY KEY (id))"""
    cursor.execute(statement)
    statement = """CREATE TABLE IF NOT EXISTS employee (
                id SERIAL PRIMARY KEY,
                name VARCHAR(20),
                surname VARCHAR(20),
                company INTEGER REFERENCES company ON DELETE CASCADE)"""
    try:
        cursor.execute(statement)
    except:
        print("employee table cannot be created.")
    finally:
        connection.commit()

    statement = """CREATE TABLE IF NOT EXISTS project_of_employee (
                employee_id INTEGER REFERENCES employee ON DELETE CASCADE,
                project_id INTEGER REFERENCES project ON DELETE CASCADE,
                PRIMARY KEY(employee_id, project_id))"""
    try:
        cursor.execute(statement)
    except:
        print("project_of_employee table cannot be created.")
    finally:
        connection.commit()

    statement = """CREATE TABLE IF NOT EXISTS task_of_employee (
                task_id INTEGER REFERENCES task ON DELETE CASCADE,
                employee_id INTEGER REFERENCES employee ON DELETE CASCADE,
                PRIMARY KEY(task_id, employee_id))"""
    try:
        cursor.execute(statement)
    except:
        print("task_of_employee table cannot be created.")
    finally:
        connection.commit()
    connection.commit()
except:
    print("Failed to create cursor.")
    connection.commit()
    cursor = None
finally:
    if cursor is not None:
        cursor.close()

try:
    cursor = connection.cursor()

    statement = """INSERT INTO user_role (id, role)
                VALUES (1, 'admin')"""
    cursor.execute(statement)

    statement = """INSERT INTO user_role (id, role)
                VALUES (2, 'company')"""
    cursor.execute(statement)

    statement = """INSERT INTO user_role (id, role)
                VALUES (3, 'employee')"""
```

```
    cursor.execute(statement)

    statement = """INSERT INTO system_user (username, password, user_type)
                        VALUES ('admin', 'itucsdb1712', 1)"""
    cursor.execute(statement)

    connection.commit()
except:
    print("User Roles already exists. Skip this stage")
    connection.commit()
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

This function is always called at the start of the web application. It creates tables of the database, and fill the tables with the required information. User types table is initialized with this function. Also admin username and its password inserted into the user table by this function. If these informations already in the database at the program start, then exceptions are triggered and handled.

```
    def addCompanyToDb(company_name, number_of_employees):
try:
    cursor = connection.cursor()
    statement = """INSERT INTO company (name, number_of_employees)
                VALUES (%s, %s)"""
    cursor.execute(statement, [company_name, number_of_employees])
    connection.commit()
except:
    print("Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

This function, get company_name and number_of_employees information and insert a company into the company table with these informations.

```
    def addUserToDb (username, password, user_type):
try:
    cursor = connection.cursor()
    statement = """INSERT INTO system_user (username, password, user_type)
                VALUES (%s, %s, %s)"""
    cursor.execute(statement, [username, password, user_type])
    connection.commit()
except:
    print("Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

This function is responsible for the login information insertions and it is called whenever a company or employee added to the system. It takes username, password and type of the user. With these information it creates a user.

```
    def listCompanies():
try:
    cursor = connection.cursor()
    statement = """SELECT name, number_of_employees FROM company"""
    cursor.execute(statement)
    information = cursor.fetchall()
    return information
except:
```

```
    print("Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

This function returns all the companies from the database.

```
    def returnCompany(company_name):
try:
    cursor = connection.cursor()
    statement = """SELECT * FROM company
                WHERE name = %s"""
    cursor.execute(statement, [company_name])
    information = cursor.fetchall()
    return information
except:
    print("returnCompany: Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

Selects a specific company from the database. It selects according to company name.

```
    def updateCompany(company_id, name, number_of_employees):
try:
    cursor = connection.cursor()
    statement = """UPDATE company SET name = %s, number_of_employees = %s
                WHERE (%s = id)"""
    cursor.execute(statement, [name, number_of_employees, company_id])
    connection.commit()
except:
    print("updateCompany: Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

Updates company with the new information provided.

```
    def deleteCompany(company_id):
try:
    cursor = connection.cursor()
    statement = """DELETE FROM company
                WHERE (%s = id)"""
    cursor.execute(statement, [company_id])
    connection.commit()
except:
    print("deleteCompany: Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

Deletes company from the database with the company_id value that it gets.

```
    def deleteUser(username):
try:
    cursor = connection.cursor()
    statement = """DELETE FROM system_user
                WHERE (%s = username)"""
```

```
    cursor.execute(statement, [username])
    connection.commit()
except:
    print("deleteCompany: Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

If a company is deleted from the system, this function is also called to delete its user information from the database.

```
    def getUserPwHash(username):
try:
    cursor = connection.cursor()
    statement = """SELECT password, user_type FROM system_user WHERE username = %s"
↪""
    cursor.execute(statement, [username])
    hash = cursor.fetchone()
    return hash
except:
    print("getUserPwHash: Failed to create cursor or wrong SQL Statement")
    cursor = None
finally:
    if cursor is not None:
        cursor.close()
```

This function is called whenever password authorization is required.

```
    def addEmployeeToDb(name, surname, company_id):
try:
    cursor = connection.cursor()
except:
    print('addEmployeeToDb: cursor creation has failed.')
    return
statement = """INSERT INTO employee (name, surname, company)
              VALUES (%s, %s, %s )"""
try:
    cursor.execute(statement, [name, surname, company_id])
except:
    print('addEmployeeToDb: SQL command failed.')
finally:
    connection.commit()
    cursor.close()
```

Adds employee to the database table with the information that is provided.

```
    def getEmployeeFromDb(company_id):
try:
    cursor = connection.cursor()
except:
    print('getEmployeeFromDb: cursor creation has failed.')
    return
statement = """SELECT * FROM employee
            WHERE company = %s"""
try:
    cursor.execute(statement, [company_id])
    information = cursor.fetchall()
except:
    print('getEmployeeFromDb: SQL command failed.')
finally:
    connection.commit()
    cursor.close()
return information
```

Gets all employees that belong to the company that have logged in.

```
    def deleteEmployeeFromDb(employee_id):
try:
    cursor = connection.cursor()
except:
    print('deleteEmployeeFromDb: cursor creation has failed.')
    return
statement = """DELETE FROM employee
            WHERE id = %s"""
try:
    cursor.execute(statement, [employee_id])
except:
    print('deleteEmployeeFromDb: SQL command failed.')
finally:
    connection.commit()
    cursor.close()
```

Deletes the employee with the "employee_id" id value.

```
    def updateEmployeeInDb(name, surname, employee_id):
try:
    cursor = connection.cursor()
except:
    print('updateEmployeeInDb: cursor creation has failed.')
    return
statement = """UPDATE employee SET name=%s, surname=%s
               WHERE (id = %s)"""
try:
    cursor.execute(statement, [name, surname, employee_id])
except:
    print('updateEmployeeInDb: SQL command failed.')
finally:
    connection.commit()
    cursor.close()
```

Update the specific employee with the new information provided.

```
    def returnEmployeeId(name):
try:
    cursor = connection.cursor()
except:
    print('returnEmployeeId: cursor creation has failed.')
    return
statement = """SELECT id FROM employee
            WHERE name = %s"""
try:
    cursor.execute(statement, [name])
    information = cursor.fetchone()
    return information
except:
    print('returnEmployeeId: SQL command failed.')
finally:
    connection.commit()
    cursor.close()
```

Return the id of the employee with the name "name" variable that is provided.

```
    def createProjectEmployeeRelation(employee_id, project):
try:
    cursor = connection.cursor()
```

```
except:
    print('createProjectEmployeeRelation: cursor creation has failed.')
    return
statement = """INSERT INTO project_of_employee VALUES (%s, %s)"""
try:
    cursor.execute(statement, [employee_id, project])
except:
    print('createProjectEmployeeRelation: SQL command failed. or passed')
finally:
    connection.commit()
    cursor.close()
```

Fills the relation table that implies the relation between employees and the projects.

forms.py

```
    class AddCompanyForm(FlaskForm):
company_name = StringField('Company Name: ', validators=[DataRequired()])
number_of_employees = IntegerField('Number of employees: ',
                                   validators=[DataRequired()])
company_account_pw = StringField('Password: ', validators=[DataRequired()])
submit = SubmitField('Submit')

    class SelectCompanyForm(FlaskForm):
company_name = StringField('Company Name: ', validators=[DataRequired()])
submitUpdate = SubmitField('Update')
submitDelete = SubmitField('Delete')

    class LoginForm(FlaskForm):
username = StringField('Username: ', validators=[DataRequired()])
password = StringField('Password: ', validators=[DataRequired()])
submit = SubmitField('Submit')

    class TaskForm(FlaskForm):
name = StringField('Task name', validators=[DataRequired()])
priority = IntegerField('priority[1-10]', validators=[DataRequired()])
projects = SelectField('project', coerce=int)
submit = SubmitField('Submit')

    class ProjectForm(FlaskForm):
name = StringField('Project name', validators=[DataRequired()])
submit = SubmitField('Submit')

    class EmployeeForm(FlaskForm):
name = StringField('Name: ', validators=[DataRequired()])
surname = StringField('Surname: ', validators=[DataRequired()])
project = SelectField('project', coerce=int)
password = StringField('Password: ', validators=[DataRequired()])
submit = SubmitField('Submit')
```

This python file includes class deceleration of the web forms that is used in html files.

2.1.2 Parts Implemented by Muhammed Isiyok