DESIGNING AN ALGORITHM AND DEVELOPING A COMPUTER APPLICATION FOR AUTOMATING THE DOCUMENT FLOW REQUIRED FOR DOMAIN PRACTICE ACTIVITY

IVAN Emanuel, GHEORGHIȚĂ Vlad
Faculty: IIR, Specialization: IAII, Year of Study: 4, Email: emanuelivan18@gmail.com

ABSTRACT: Obtaining the necessary documents for university processes is often a complex and time-consuming task. Automating the processes of obtaining and managing the documents required for university processes, as well as developing web applications that facilitate these processes, represents a major innovation in the field of education, addressing the increasing needs of educational institutions, students, and the personnel involved in these processes.

KEYWORDS: domain practice, administration, web service, database.

1. Introduction

The documents required for the domain practice process involve completing certain documents without which the process cannot begin. These include a collaboration agreement and a practice protocol between the faculty and a company willing to accept students for practice, as well as a framework agreement concluded between the same company, student, and faculty. Completing these documents involves working with confidential data of the student, faculty, and company, which is why the security level of the application necessary to fulfill its intended purpose needs to be high.

The documents required to conclude the practice process encompass a collection of documents, including a report, a portfolio, and a Gantt chart. These can be completed during the practice period and serve as a demonstration of the activities carried out within the company. At the end of the practice, the student receives a certificate from the company attesting to their completion of the practice within the company.

This paper aims to automate the generation of documents necessary for the domain practice within the IIR faculty by involving student interaction and the involvement of faculty staff through a web application.

Web applications are typically built with a layered architecture known as the layered architectural model. This refers to the separation of business logic, data presentation, and user interaction into distinct layers. This approach helps create applications that are easier to maintain and scale. Typically, these layers include the following:

- Presentation layer: This layer deals with the presentation of data to users and their interaction with it. This part of the web application is usually implemented using HTML, CSS, and JavaScript.
- Application layer: This layer contains the business logic of the application. It is responsible for data processing and interaction with the database. This layer is often implemented using a backend programming language such as Python, Ruby, PHP, or Node.js.
- Data layer: This layer manages the database and allows access to it. This layer can be implemented using a Database Management System (DBMS) such as MySQL, PostgreSQL, MongoDB, or Access.
- Infrastructure layer: This is the layer that handles infrastructure-related aspects such as server management, security, and scalability of the web application. This layer can be implemented using tools such as Docker, Kubernetes, or AWS. [1]

These layers can communicate with each other, enabling the application to function efficiently. Additionally, there are other architectural approaches, such as service-based architecture or event-driven architecture, which can be used based on the specific needs of the application. [2]

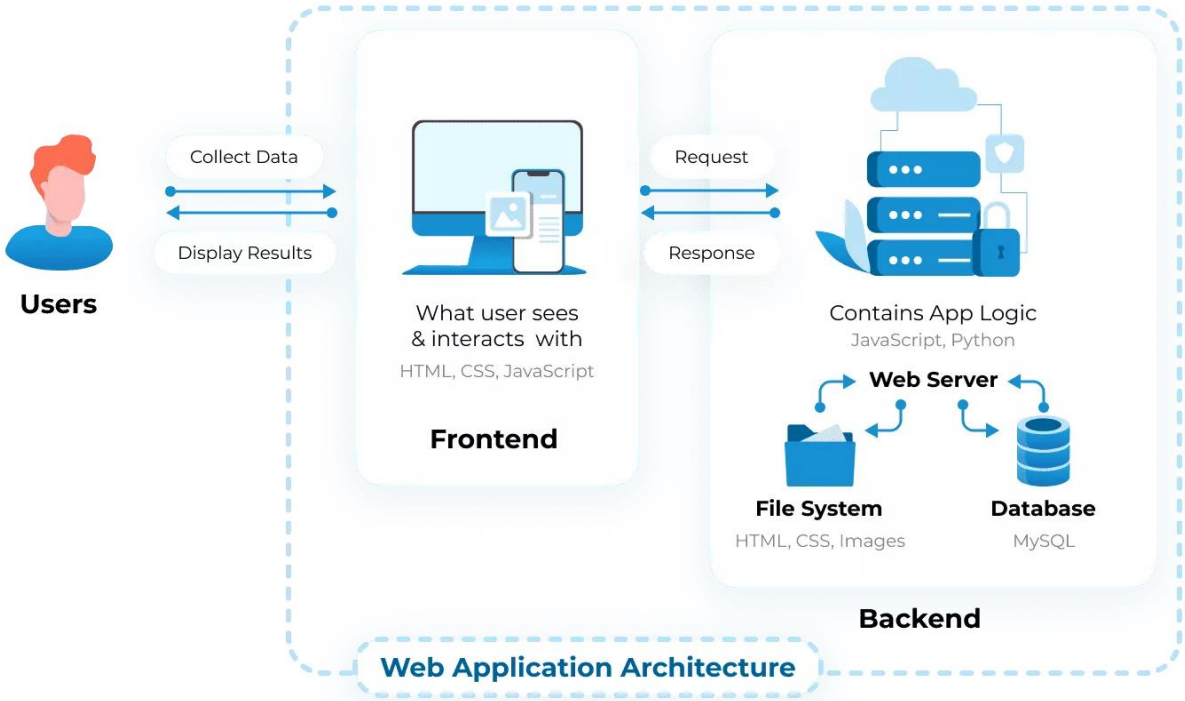


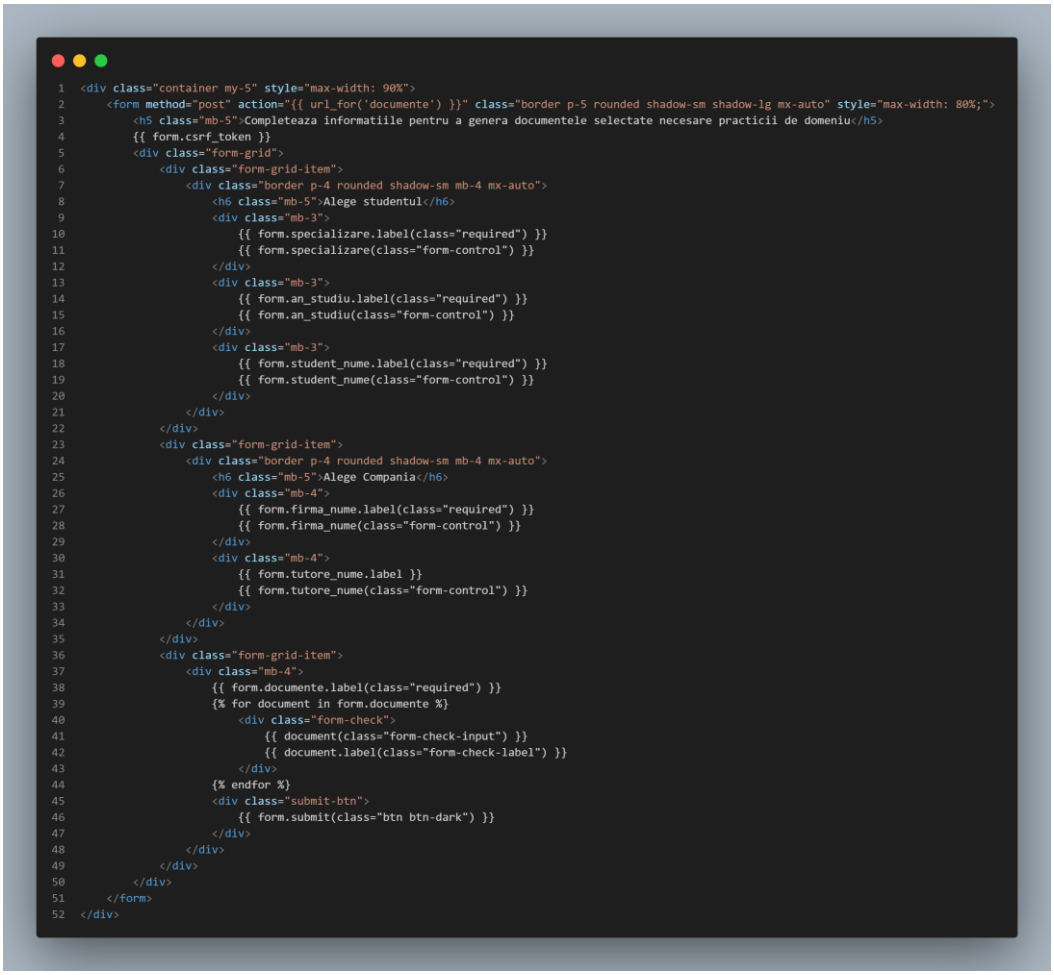
Figure 1. General architecture of web applications [3]

2. Current Stage

Currently, the application offers the possibility to generate and download several documents required for the completion and initiation of the practice process by making queries based on the information stored in the database. These functionalities are accessible through an interface built with HTML, CSS (using the Bootstrap framework), and JavaScript, running within a Python application using the Flask library.

By navigating to a predefined endpoint "/documents" of the web application, which loads the "documents.html" page, the personnel involved in validating the practice process can generate the necessary documents for each student based on the selections made in the form. The HTML page is a template populated with Flask's template tags ("{{", "}}", "{%", "%}"), which allow integrating the logic created in Python with HTML using the Jinja2 library. The page can be loaded using the "render_template('convention.html')" function at the endpoint.

The form includes validations both on the presentation and logic side, providing data about the student who wants to start the domain practice, the name of the company involved, and a selectable list of documents that can be generated. Figure 2 shows the form, which includes a security token "{{ form.csrf_token }}" generated by the HTML form. This token protects the application against Cross-Site Request Forgery (CSRF) attacks, which can lead to the execution of unwanted actions by an unauthorized user.



```
<div class="container my-5" style="max-width: 90%">
    <form method="post" action="{{ url_for('documente') }}" class="border p-5 rounded shadow-sm shadow-lg mx-auto" style="max-width: 80%;">
        <h5 class="mb-5">Completeaza informatiile pentru a genera documentele selectate necesare practicii de domeniu</h5>
        {{ form.csrf_token }}
        <div class="form-grid">
            <div class="form-grid-item">
                <div class="border p-4 rounded shadow-sm mb-4 mx-auto">
                    <h6 class="mb-5">Alege studentul</h6>
                    <div class="mb-3">
                        {{ form.specializare.label(class="required") }}
                        {{ form.specializare(class="form-control") }}
                    </div>
                    <div class="mb-3">
                        {{ form.an_studiu.label(class="required") }}
                        {{ form.an_studiu(class="form-control") }}
                    </div>
                    <div class="mb-3">
                        {{ form.student_nume.label(class="required") }}
                        {{ form.student_nume(class="form-control") }}
                    </div>
                </div>
            </div>
            <div class="form-grid-item">
                <div class="border p-4 rounded shadow-sm mb-4 mx-auto">
                    <h6 class="mb-5">Alege Compania</h6>
                    <div class="mb-4">
                        {{ form.firma_nume.label(class="required") }}
                        {{ form.firma_nume(class="form-control") }}
                    </div>
                    <div class="mb-4">
                        {{ form.tutore_nume.label }}
                        {{ form.tutore_nume(class="form-control") }}
                    </div>
                </div>
            </div>
            <div class="form-grid-item">
                <div class="mb-4">
                    {{ form.documente.label(class="required") }}
                    {% for document in form.documente %}
                        <div class="form-check">
                            {{ document(class="form-check-input") }}
                            {{ document.label(class="form-check-label") }}
                        </div>
                    {% endfor %}
                    <div class="submit-btn">
                        {{ form.submit(class="btn btn-dark") }}
                    </div>
                </div>
            </div>
        </div>
    </form>
</div>
```

Figure 2. Document Generation and Download Form

The form validation is done at the class level, specifically the “DownloadForm” class, which uses the parent class “FlaskForm” as shown in Figure 3. The “FlaskForm” class does not allow the form to be submitted in an intermediate state of completion. The class allows for initializing default values for the fields using “default” and validating the choices made using “validators”.

```
class DownloadForm(FlaskForm):
    specializare = SelectField('Specializare', validators=[DataRequired()], default='3')
    an_studiu = SelectField('An studiu', validators=[DataRequired()],
                            choices=[('2', 'An 2'), ('3', 'An 3')],
                            default='2')
    student_nume = SelectField('Student', validators=[DataRequired()])

    firma_nume = SelectField('Firma', validators=[DataRequired()])
    tutore_nume = SelectField('Tutore')

    documente = MultipleCheckboxField('Selecteaza documente de generat',
                                      choices=[
                                          (1, 'Conventie_cadru_2022_2023.docx'),
                                          (2, 'Raport_de_practica_2022-2023.docx'),
                                          (3, 'Portofoliu_de_practica_2022-2023.docx'),
                                          (4, 'Grafic_Gantt_2022-2023.docx'),
                                          (5, 'Adeverinta_de_practica_2022-2023.docx')],
                                      coerce=int,
                                      validators=[DataRequired()])

    submit = SubmitField('Genereaza')
```

Figure 3. Document Generation and Download Form Class

The value list of the “Student” field is dynamic, updating based on the selections made in the “Specialization” and “Year of Study” fields. It also offers the possibility of searching within the selection list due to its “select2” attribute (Figure 4). Data updating is achieved by integrating a JavaScript function into the “documents.html” document. The “updateStudentNameOptions” method calls an AJAX function to the endpoint stored in the “getStudentUrl” variable, which retrieves the student values from the database based on the specialization and year of study criteria. These values are then loaded into the selector, and the selector is re-initialized with the default value by calling the “initializeSelect2” function.

```
$(document).ready(function() {
    initializeSelect2();
    updateStudentNameOptions();

    $("#specializare").change(updateStudentNameOptions);
    $("#an_studiu").change(updateStudentNameOptions);

    function updateStudentNameOptions() {
      var academicYear = $("#an_studiu").val();
      var facultySpecialization = $("#specializare").val();

      makeAjaxCall(getStudentsUrl, { an_studiu: academicYear, specializare: facultySpecialization },
        function(data) {
          var studentName = $("#student_nume");
          studentName.empty();

          $.each(data, function(key, value) {
            var option = $('<option>').val(value[0]).text(value[1]);
            studentName.append(option);
          });

          initializeSelect2();
        },
        function(xhr, status, error) {
          console.error(error);
        }
      );
    }

    function initializeSelect2() {
      $('#student_nume').select2({
          width: '100%',
          allowClear: true,
          theme: 'bootstrap',
          placeholder: {
              id: '',
              text: 'Cauta...',
              selected: true
          }
      });
  }
```

Figure 4. Student Field Update Function

Based on the selections made in the form on the “documents.html” page and after validating it, a data dictionary is generated by invoking the “get_context” function. This involves calling a series of functions based on the student's ID, company, and supervisor to generate the completed documents selected by the designated person. The function calls the “get_student” method to retrieve the student's data, the “get_supervisor” method for the designated department supervisor's data, the “get_company” method for the company's data, and the “get_company_representative” and “get_company_tutor” methods for the company representative and the tutor chosen for the student within the company. Each of these helper functions returns a dictionary of values containing data extracted based on executed SQL queries using the “pyodbc” module. Thus, personal information will not be exposed to the web page, only the ID, which can be encrypted and decrypted as needed. The helper dictionaries that make up the template context are passed through a function called “replace_empty_values_with_dash,”

which standardizes the appearance of the values in each dictionary regardless of their initial appearance.

```python
def get_context(self, student_id, company_id, tutor_id):
    # Student information
    student = self._instance.get_student(student_id)

    # University information
    supervisor = self._instance.get_supervisor(student_id)

    # Company information
    company = self._instance.get_company_details(company_id)
    representative = self._instance.get_company_representative(company_id)
    tutor = self._instance.get_company_tutor(tutor_id)

    # Building the context
    self._instance.context = {
        'STUDENT': student,
        'FIRMA': {
            'DETALII': company,
            'REPREZENTANT': representative,
            'TUTORE': tutor,
        },
        'UNIVERSITATE': {
            'NUME': 'Polithnica Bucuresti',
            'FACULTATE': 'Inginerie Industriala si Robotica',
            'SUPERVIZOR': supervisor,
        },
        'DATA': datetime.now().strftime('%d.%m.%Y')
    }
    return self._instance.context
```

Figure 5. Data Dictionary (Context) Generation

Based on this data dictionary, a temporary file is created in memory to save all the selected documents from the form. This is achieved by calling the “save_docx_file” function for each document, which invokes the “render” function from the “DocxTemplate” class. The “render” function takes the previously generated dictionary as an input parameter. This function replaces the values of the keys found in the docx template (Figure 7) with the corresponding values from the context generated for each student (Figure 5).

```python
def save_docx_file(self, temporary_directory, docx_template):
    docx_template = self.docx_templates_directory / docx_template

    if not docx_template.is_file():
        print(f'Template file not found at {docx_template}')
        return None

    try:
        doc = DocxTemplate(docx_template)
        doc.render(self.context)

        docx_file_name = f'{docx_template.stem}_{self.student_name}_{self.company_name}_{self.now}.docx'
        docx_file_path = temporary_directory / docx_file_name

        doc.save(docx_file_path)
        return docx_file_name
    except (PackageNotFoundError, PermissionError) as e:
        print(f'An error occurred: {e}')
    except Exception as e:
        print(f'An error occurred: {e}')

    return None
```

Figure 6. Generation of a docx Document Based on the Dictionary and docx Template

The template can contain both static and dynamic values, allowing for logical operations ("if", "while", "for") to be performed within it, given the presence of the labels shown in Figure 2.

The values in the context can be accessed in the template using "{{DICTIONARY_KEY}}", and it is possible to navigate within it using the dot notation.

Societatea comercială, *instituția centrală* ori *locală*, *persoana juridică* {{FIRMA.DETALII.DENUMIRE}} (denumită în continuare ***Partener de practică***), reprezentată de (numele și calitatea) dl/dna {{FIRMA.REPREZENTANT.NUME}} {{FIRMA.REPREZENTANT.PRENUME}} – {{FIRMA.REPREZENTANT.FUNCTIE}}, adresa partenerului de practică: {{FIRMA.DETALII.ADRESA_SEDIU}}
adresa unde se va desfășura stagiul de practică {{FIRMA.DETALII.ADRESA_PRACTICA}} tel. {{FIRMA.DETALII.TELEFON}}, fax {{FIRMA.DETALII.FAX}}, email: {{FIRMA.DETALII.EMAIL}},

şi

Student {{STUDENT.NUME}} {{STUDENT.PRENUME}} (denumit în continuare **Practicant**) CNP {{STUDENT.CNP}}, data nașterii {{STUDENT.DATA_NASTERII}}, locul nașterii {{STUDENT.LOCUL_NASTERII}}, cetățean {{STUDENT.CETATENIE}}, pașaport (dacă este cazul) {{STUDENT.PASAPORT}}, permisul de ședere (dacă este cazul) {{STUDENT.PERMIS_SEDERE}}, adresa de domiciliu {{STUDENT.ADRESA_DOMICILIU}}, adresa unde va locui pe durata desfășurării stagiului de practică {{STUDENT.ADRESA_LOCUINTA_PRACTICA}}, înscris în anul universitar 2022-2023, Universitatea {{UNIVERSITATE.NUME}}, Facultatea {{UNIVERSITATE.FACULTATE}}, seria {{STUDENT.SERIE}}, grupa {{STUDENT.GRUPA}}, email: {{STUDENT.EMAIL}}, telefon: {{STUDENT.TELEFON}}.

Figure 7. Example docx Template

3. Conclusion

I consider the automation of generating the necessary documents for field practice to be an important contribution to the field of business process automation and web application development. By using these solutions, significant benefits in terms of efficiency and reduction of human errors can be achieved, enabling better time and resource management. However, there are still technical and research challenges that need to be addressed, such as ensuring the security of access endpoints and developing new functionalities to meet the evolving needs of the university.

4. Bibliography

[1]. Kumar, Bimal. (2015). Layered architecture for mobile web based applications. Asia-Pacific World Congress on Computer Science and Engineering, APWC on CSE 2014. 10.1109/APWCCSE.2014.7053865.
[2]. Corcho, Oscar & Arco, Jose & Arias Fisteus, Jesus. (2006). Web Services: Introduction and State of the Art.
[3]. Deremuk I. (2021). "Modern Web Application Architecture".