ESILVision

Anouk Leyris, Wally Faye, Elias Ferreira, Mohamed Rakai

Github Link : https://github.com/anoukleyris/ESILVision_Project

Abstract

The ESILVision project emerges as a crucial evolution for the smooth functioning of ESILV. As part of a large-scale initiative, ESILV is occasionally affected by administrative issues that significantly impact the educational progress of students. Faced with these challenges, the creation of a new website, more reliable and attentive to the concerns of administrative staff and students, has proven to be indispensable. Designed and deployed on Visual Studio Code with the integrationof Streamlit, this portal offers a comprehensive array of features.

Key functionalities include the complete visualization of the academic profile of each student, a messaging system fostering communication between administration and students, as well as total autonomy for the administration in managing students, teachers, and courses.

1. Introduction

ESILVision is the new portal specifically designed for ESILV students, aiming to centralize essential information for the smooth functioning of the school. The portal encompasses comprehensive set of features, ranging from individual student grades to messaging facilitating effective communication between administration and students. What sets ESILVision apart is its commitment to personalized adaptation for each user. The primary goal revolves around improving satisfaction levels among both administrative staff and students through enhanced management and transparent communication.

2. Jira

2.0. Invitation in Jira

Jira Link :

https://esilvision.atlassian.net/jira/software/projects/SCRUM/boards/1

2.1. Structure of Jira

We used Jira to organize 3 sprints, each consisting of 10 user stories. Each sprint has a duration of 2 weeks, and we commenced on 29/11/2023, leading us to 11/01/2024. As of today, we have completed the entire first sprint and accomplished 7 out of the 10 user stories in the second sprint. The remaining three user stories in the second sprint pose a certain level of difficulty with the use of Streamlit, and additional time will be needed to finalize them.

On our Jira board, you can view all our sprints and user stories, each with a detailed description. We have also created tickets to document encountered issues and possible improvements.

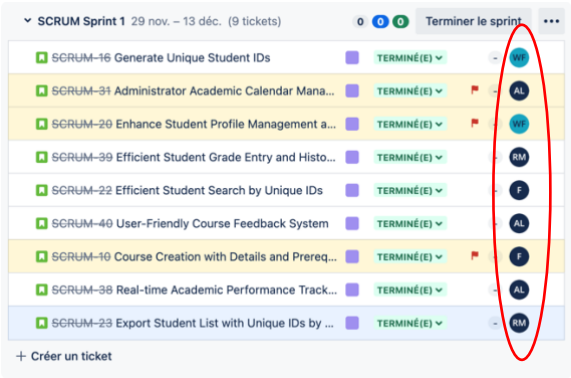
A detail description for the user-story Calendar

A Ticket for the user-story Calendar

2.2. Distribution of tasks

The task distribution was easily achieved. For each sprint, each team member was assigned between two to three tasks to ensure equity among team members.

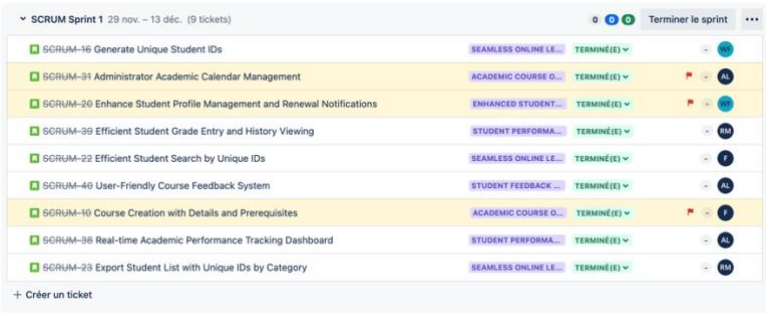
Regarding roles, we all took on the role of a developer, and Anouk additionally assumed the role of Scrum Master during the project.



Task distribution for Sprint 1

2.3. Success and failures

The first sprint was a success as we were able to address all user stories, and our entire code functions correctly. Some improvements were identified during the ticket creation process (visible on user stories "Sprint 31," "Sprint 20," and "Sprint 10"). The major challenge encountered in this sprint was that it was the initial one, requiring the setup of all elements and becoming familiar with new tools. Additionally, the user stories involved numerous steps, representing a substantial amount of work.



Sprint 1

The second sprint had a slightly different approach. The completion of the first sprint provided us access to many lines of code that we could reuse. However, this sprint involved creating new documents and numerous links between different account types. Moreover, three user stories could not be completed (the reasons and solutions are visible in our Jira tickets).

Nevertheless, the sprint remains a success, considering that the other seven user stories were fully completed, adhering entirely to the requirements.

SCRUM Sprint 2 13 déc. – 27 déc. (10 tickets)			
SCRUM-30 Administrator Announcement and Messaging Feature	ACADEMIC COURSE O...	TERMINÉ(E)	AL
SCRUM-17 Assign Teachers to Courses with Roles and Hours	ACADEMIC COURSE O...	TERMINÉ(E)	F
SCRUM-43 Enhancing Course Content with Additional Resources	ACADEMIC COURSE O...	À FAIRE	AL
SCRUM-14 Diploma Management and Certificate Generation	STUDENT FEEDBACK ...	TERMINÉ(E)	WF
SCRUM-46 Peer Feedback for Informed Course Selection	STUDENT FEEDBACK ...	TERMINÉ(E)	AL
SCRUM-21 Track Course Enrollments and Student Progress	EFFICIENT STUDENT ...	À FAIRE	WF
SCRUM-41 Automated Weekly Absence Reports	EFFICIENT STUDENT ...	TERMINÉ(E)	F
SCRUM-24 Online Document Submission for Students	ENHANCED STUDENT...	TERMINÉ(E)	WF
SCRUM-42 Custom Document Request Forms	ENHANCED STUDENT...	À FAIRE	RM
SCRUM-27 Parent Access to Academic Info	STUDENT PERFORMA...	TERMINÉ(E)	RM

+ Créer un ticket

Sprint 2

The last sprint is yet to be completed during the remaining two weeks.

3. Tool use for ESILVision website

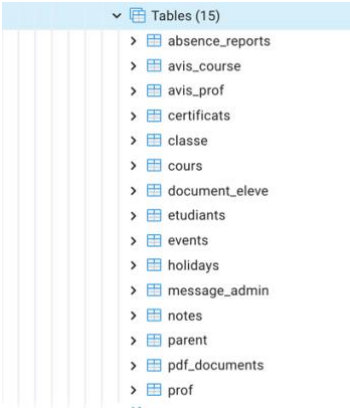
3.1. Streamlit

As all four of us are beginners in website creation, we looked for a framework that would allow us to showcase our computer skills, especially in Python. That's when we discovered Streamlit, described as an open-source Python framework for machine learning and data science teams.

```
import streamlit as st
```

3.2. PostgreSQL

An indispensable tool for us was PostgreSQL for creating a database. We created 15 tables and alors sequences, functions and triggers.



Tables created for the backend of ESILVision

The detail of all tl tgreSQL_cretion.pdf.

For each section and page created on the ESILVision site, it is necessary to establish a connection with the database, which is done as follows:

```
import psycopg2

# Etablir la connexion à la base de données PostgreSQL
def create_db_connection():
    connection = psycopg2.connect(
        host="localhost",
        database="ESILVision",
        user="postgres",
        password="zaL2520"
    )
    return connection
```

~~SCRUM-16~~ Generate Unique Student IDs

We addressed this user story right from the creation of the database since it was during the creation of the student table that we generated IDs automatically using the sequence:

3.3. HTML

We also had to use some HTML to create the calendar with holidays and important events. Resorting to HTML greatly assisted us because none of the previously used methods resulted in the calendar as we desired.

The function generate_calendar_html creates an HTML representation of a monthly calendar, with days highlighted based on events and holidays. It utilizes an HTML table with rows for the days of the week, and each day is a table cell. The function determines the background color of each cell based on the category of the day.

```python
def generate_calendar_html(year, month, events, holidays):
    cal = calendar.monthcalendar(year, month)
    days_of_week = list(calendar.day_abbr)

    calendar_html = "<table class='calendar'>\n" #création d'un calendrier
    calendar_html += "<tr>" + "".join(f"<th class='calendar-header'>{day}</th>" for day in days_of_week) + "</tr>\n" #permet de créer une en-tête pour chaque jour

    for week in cal: #pour chaque semaine :
        calendar_html += "<tr>" #rajoute une ligne pour cahque semaine
        for day in week: #pour chaque jour de la semaine :
            day_content = day if day != 0 else ""
            day_category = get_day_category(day, month, year, events, holidays) if day_content else "normal" #utilise la fonction get_day_category() pour connaître la "nature du jour"
            if day_category == 'event': #les évènements seront en rose
                color = 'violet'
            elif day_category == 'holiday': #les vacances seront en vertes
                color = 'lightgreen'
            else: #les autres jours ne sont pas colorés
                color = 'white'
            calendar_html += f"<td class='calendar-day' style='background-color: {color};'>{day_content}</td>" #associe à chaque jour sa couleur
        calendar_html += "</tr>\n"

    calendar_html += "</table>"
    return calendar_html
```

Utilization of HTML for calendar

4. Creation of ESILVision website

4.1. Authentification

In this project, as multiple perspectives were considered, we had to begin by enabling authentication for each role. In sprints 1 and 2, it concerns the administrative staff, students, and parents. Each can log in by adhering to specific criteria: administrative employees all have the same username and password. For students and parents, the username is the last name, and the password is automatically generated in the database.

```python
if role == "Administrateur": #si le rôle est administraeur
    if username == "admin" and password == "passwordadmin123": #et que les informations entrées sont bonnes
        # Authentification réussie pour l'administrateur
        st.success("Authentification réussie en tant qu'administrateur.")
        st.session_state['authenticated'] = {"role": "admin"} #authentification en tant qu'admin
    else:
        st.warning("Identifiants incorrects pour l'administrateur.")
```

Connection as admin

```python
elif role == "Etudiant": #si le rôle est étudiant
    cursor.execute("""
        SELECT id, mot_de_passe
        FROM public.etudiants
        WHERE nom = %s AND mot_de_passe = %s;
    """, (username, password)) #on récupère les infos de l'exécution de la requête dans la base de données
    result = cursor.fetchone() #on récupère la première ligne de résultat (il ne peut que y en avoir une car le mot de passe est unique)

    if result: #si on a récupéré une ligne
        student_id, _ = result
        st.success("Authentification réussie en tant qu'étudiant.")
        st.session_state['authenticated'] = {"role": "etudiant", "id": student_id} #authentification en tant qu'étudiant et on récupère l'id qui nous sera utile
    else:
        st.warning("Identifiants incorrects pour l'étudiant.")
```

Connection as student

Remark : The connection as parent follows exactly the same logic than the one for student

4.2. Admin account

To begin with, a certain number of created functions are, in fact, the execution of SQL queries. They all operate in the same way, and it is only necessary to be proficient in SQL to be able to implement them. Here is an example:

~~SCRUM-31~~ Administrator Academic Calendar Management

```python
# Fonction pour récupérer les événements
def get_events(year, month):
    connection = create_db_connection()
    query = """
    SELECT title, start_date, end_date, description
    FROM public.events
    WHERE EXTRACT(YEAR FROM start_date) = %s AND EXTRACT(MONTH FROM start_date) = %s;
    """
    events_df = pd.read_sql(query, connection, params=(year, month))
    connection.close()
    return events_df
```

Here, we establish a connection with the database, retrieve the result of the SQL query, close the connection, and return the result of the SQL query.

~~SCRUM-22~~ Efficient Student Search by Unique IDs

To retrieve only the information of a specific student based on their ID, we needed to fetch the data associated with the student's ID using an SQL query. Then, we created a section on the website where the user can input the desired ID, a button to perform the search, and displayed the desired row using an st.table().

```python
recherche = st.text_input('Recherche par ID')
add_button = st.button('Rechercher')

if add_button:
    if recherche:
        result_data = get_student_data_by_id(recherche)
        st.table(result_data)
    else:
        st.warning("Veuillez entrer un ID pour effectuer la recherche.")
```

~~SCRUM-17~~ Assign Teachers to Courses with Roles and Hours
~~SCRUM-23~~ Export Student List with Unique IDs by Category

To accomplish these two tasks, we used the same logic as retrieving information for a specific student based on their ID. For instance, to display the list of students by class, we created a function with the corresponding SQL query, and then we added the set of text box, button, and display to our main() function.

~~SCRUM-10~~ Course Creation with Details and Prerequisites

This part involves the utilization of SQL queries, and we decided to adopt a similar approach for both students and teachers. Indeed, an admin can add or delete students or teachers, just as they can add or delete courses.

~~SCRUM-38~~ Real-time Academic Performance Tracking Dashboard

On the student profile page, there are three steps. Firstly, we display the personal information of the student, as seen earlier. Next is the list of paths to the documents. All the documents for each student ID are stored in the database in the "pdf_document" table, so displaying their list for a given student relies on the same logic as displaying the list of students for a given class. Finally, there is the visualization of grades through graphs and tables. This involves plotting the grades against the control number for each subject and displaying the grades per subject using an st.table().

```
course_data = notes_data[notes_data['cours_title'] == course]

fig, ax = plt.subplots()
ax.plot(course_data['numero_controle'], course_data['note'], marker='o', linestyle='-', color='b')
ax.set_xlabel('Numéro de Contrôle')
ax.set_ylabel('Note')
ax.set_title(f'Notes de l\'élève pour le cours {course}')

# Définir les numéros de contrôle comme ticks sur l'axe des abscisses
ax.set_xticks(course_data['numero_controle'])

st.pyplot(fig)
```

Creation of graphs per subject to track the evolution of a student's grades.

```
course_data = course_data.sort_values(by='numero_controle')

# Afficher les données dans un tableau
st.table(course_data[['numero_controle', 'note', 'coefficient']].reset_index(drop=True))
```

Creation of tables per subject to track the evolution of a student's grades.

~~SCRUM-41~~ Automated Weekly Absence Reports

For absences, we proceeded in two steps. Initially, we have access to the summary table with details of each absence. Subsequently, for better visualization, we created a graph on which absences are marked with red crosses, and hovering over them reveals the details of the absence.

~~SCRUM-39~~ Efficient Student Grade Entry and History Viewing

To address this user story, we displayed the grades per subject using SQL queries, andwe also used SQL to modify or delete the grades based on their ID.

```
fig = go.Figure() #création d'un emplacement pour le graphe

for index, row in absences_data.iterrows():
    fig.add_trace(go.Scatter(
        x=[row['report_date']], #en absisse : les dates
        y=[1], #en ordonnée : présent ou absent
        mode='markers', # sytème de crois pour marquer les absences
        marker=dict(color='red', symbol='cross'),
        hovertext=[f"Absent le {row['report_date']} - Raison: {row['absence_reason']}"], #légende pour chaque point
        hoverinfo='text',
    ))

fig.update_layout( #ajout des titres et des légendes
    title=f'Absences de {nom_prenom}',
    xaxis_title='Date',
    yaxis_title='Présence',
    yaxis=dict(tickvals=[0, 1], ticktext=['Présent', 'Absent']),
    showlegend=False
)

st.plotly_chart(fig) # Affichage du graphique
```

~~SCRUM-30~~ Administrator Announcement and Messaging Feature

To facilitate communication between the administration and students, we decided to create multiple tables, each capable of storing the entirety of the messages. This way, with an INSERT INTO, a message can be sent, and with a SELECT, the message can be read.

Here, it's about the user story that requests the ability to communicate messages to all students of a class. Using SQL queries, we insert a message into the message_admin table for a specific class. Then, a student, if they belong to the class, retrieves it through a SELECT.

```python
message = st.text_input('MESSAGE:')
classe = st.text_input('CLASSE:')
add_button_cours = st.button('Envoyer')

if add_button_cours:
    new_data = pd.DataFrame({
        "MESSAGE": [message],
        "CLASSE": [classe]
    })
    add_message(new_data)
    st.success(f"Message envoyé à la classe {classe}!")
    st.experimental_rerun()
```

Writing the message by the admin in the "message" text area for the recipient "class."

```python
def get_messages(classe):
    connection = create_db_connection()
    query = f"SELECT message, date_envoi FROM message_admin WHERE classe = '{classe}' ORDER BY date_envoi DESC;"
    messages = pd.read_sql_query(query, connection)
    connection.close()
    return messages

def main(id):
    st.title("Messages de l'administration")

    classe_etudiant = get_student_classe(id)

    messages = get_messages(classe_etudiant)
```

Retrieval of messages sent to the student's class

~~SCRUM-14~~ Diploma Management and Certificate Generation


Diplôme d'ingénieur spécialisé en DIA5

Monsieur Pascal Pinot, directeur de l'Ecole supérieure d'ingénieurs Léonard De Vinci, certifie que Clara Martin est désormais diplômé ingénieur généraliste de l'ESILV

Fait à Paris, le 28/12/2023

Creating pre-filled diplomas wasn't very practical with Streamlit. Ultimately, we generated this type of diploma (here for the student Clara Martin).

For the user stories related to parent and student accounts, since all the necessary logics have already been discussed, we have decided to demonstrate the equivalences using tables.

4.3. Student's accounts

User story	Same logic as
~~SCRUM-40~~ User-Friendly Course Feedback System	~~SCRUM-30~~ Administrator Announcement and Messaging Feature
~~SCRUM-46~~ Peer Feedback for Informed Course Selection	SQL queries (as the example of calendar)
~~SCRUM-24~~ Online Document Submission for Students	~~SCRUM-38~~ Real-time Academic Performance Tracking Dashboard The part with documents of the student
~~SCRUM-20~~ Enhance Student Profile Management	~~SCRUM-38~~ Real-time Academic Performance Tracking Dashboard The part with documents of the student

4.4. Parents' accounts

User story	Same logic as
~~SCRUM-27~~ Parent Access to Academic Info	~~SCRUM-38~~ Real-time Academic Performance Tracking Dashboard Same page but in parents' account, we cannot select a student with its id, there is juste the

5. Conclusion

In conclusion, the ESILVision project has significantly improved administrative processes and user experience at ESILV. With effective communication and optimized data management, the portal has achieved its main goals, enhancing interaction between administration and students.