INFOB318: Projet individuel

'SuppléezMoi' Programmer's guide

Author:
Ulrich Touji Nana

Client:
Babette di Guardia

12 avril 2024

Contents

1 Introduction

1.1 What is "SuppléezMoi"?

1.1.1 Objective

The application aims to optimize the submission process for substitute requests from University of Namur professors. It simplifies the procedure, ensuring efficient input of substitute requests data by lecturers.

1.1.2 Functionality Highlights

Professors have the capability to input information for the requests through a form provided within the application. Subsequently, the application conducts validation procedures to ensure the completeness of the entered data. The administrative personnel hold the authority to oversee and manage any deficient data, while also supervising the allocation of substitutes through the application.

1.1.3 Intended Users

The target audience includes people familiar with the academic and administrative processes of the Faculty of Informatics at the University of Namur, such as professors, researchers, and the administration.

1.1.4 User License (Open-Source License)

The software is distributed under an MIT open-source license, allowing users to view, modify, and redistribute the source code in accordance with specified terms. Refer to `https://fr.wikipedia.org/wiki/Licence_MIT` for detailed information on associated rights and restrictions.

1.2 General Purpose of the Guide

The aim of this guide is to provide clear and concise documentation to enable an external developer to understand the architecture and operation of the web application. It should help the developer quickly become familiar with the code and be able to work independently on the application.

2 Prerequisites

2.1 Application Dependencies and Their Roles

We have utilized specific versions of various packages for the design of this application, including `django-jazzmin 2.6.0`, `django-select2 8.1.2`, `python-dateutil 2.8.2`, `python-json-logger 2.0.7`, `Python 3.10.11`, `Django 5.0.2`, `asgiref 3.7.2`, `sqlparse 0.4.4`, `typing-extensions 4.9.0`, `tzdata 2024.1`, and `openpyxl 3.1.2`. However, it's possible to use later versions of these packages which include all the features of the earlier versions. It's important to note the role of each package in the development of the web application:

- **Django (5.0.2)** and **django-jazzmin (2.6.0)**: Provide the main framework for application development, with enhanced user interface through Jazzmin.
- **django-select2 (8.1.2)**: Adds advanced selection features for forms.
- **python-dateutil (2.8.2)**: Offers robust and flexible date and time handling.
- **python-json-logger (2.0.7)**: Enables advanced logging in JSON format, facilitating integration with monitoring and analysis systems.
- **Python (3.10.11)**: The core programming language for development.
- **asgiref (3.7.2)**: Ensures asynchronous compatibility with Django, necessary for modern web applications.
- **sqlparse (0.4.4)**: Used for parsing and splitting SQL statements, useful in database migration scenarios or handling complex SQL queries.
- **typing-extensions (4.9.0)**: Provides additional typing extensions, useful for clearer code and error prevention.
- **tzdata (2024.1)**: Offers the latest timezone information for accurate time management.
- **et-xmlfile (1.1.0)** and **openpyxl (3.1.2)**: Enable reading and writing Excel files, useful for data import and export.

2.2 Installation Recommendations

While it is possible to directly install the required packages on your computer, we strongly recommend using a virtual environment (venv). Because of:

- **Isolation**: A virtual environment keeps dependencies required by different projects separate by creating isolated python virtual environments for them. This is one of the easiest and most effective ways to manage project dependencies.
- **Version Control**: With venv, it is easier to manage the versions of the packages that your project relies on. It helps ensure that your project remains compatible with the versions of the libraries you develop with, regardless of updates or changes made to these libraries in the future.
- **Simplicity**: Virtual environments make it easy to set up multiple development environments if you are working on several projects. Each project can have its own dependencies, regardless of what dependencies every other project has.
- **No Administrator Rights Required**: Installing packages globally may require administrator rights. Using a virtual environment allows you to install new packages and tools without these rights, making it more convenient, especially on shared systems.
- **Replicability**: It makes your projects more replicable since it's easier to recreate an environment with specific versions of dependencies.

3 How to Get the App ?

3.1 For Use Purpose

No specific prerequisites are required. Simply ensure you are using a modern and updated web browser. The application is accessible via the following website: `https://suppleezmoi.unamurcs.be/`.

3.2 For Development Objectives

The source code of the application is available by following the link github: `https://github.com/UNamurCSFaculty/2324_INFOB318_SuppleezMoi.git`.

This is a private Github intended for students of the faculty of computer science but permission could be given to anyone wishing to participate in the improvement of the project.

If you are an internal member of the university, you Also have the option to view or directly modify certain features of the application through its source code deployed on the university-hosted VM. However, this requires a preliminary process. To obtain access keys to the VM where the application was developed, please make a request to the administration section responsible for the application. The file containing the access keys was provided with the application. To obtain these keys, please contact the IT department or the faculty of computer science secretary. They will provide additional information and guide you through the process of obtaining the necessary access keys.

4 Architecture

This Django application adheres to the MVC (Model-View-Controller) architecture:

Model: This layer is embodied by the data structure classes in our application, like the models for request and users, among others. These models are central to handling the application's data.

View: The View is essentially the user interface aspect of our application. It is realized through various class-based views that are responsible for presenting data to the user. This includes views for tasks like displaying a list of request, filling out a request, and showing results.

Controller: In this context, the Controller is manifested through various methods we have implemented to orchestrate the application's logic. This involves functionalities such as creating a new user, adding and removing request and other similar actions.

This structure ensures a clean separation of concerns, making our application more manageable and scalable.

4.1 Models

Models in Django are specified within the 'models.py' file. This file acts as a template, comprising various class templates that delineate the data schema for a web application. In essence, each defined model correlates to a database table, where the fields in the models correspond to the columns in the table. This setup forms the structure of the web application's data architecture.

Example of model :

```
class Demandes(models.Model):
    """
    A model representing substitution requests
    Attributes
    ----------
    STATUT_CHOICES : list of tuple
        choices for the status of the request
    OPTION_CHOICES : list of tuple
        choices for options lectures in the request
    CHOICES_ACCORD_SUPPLEANT : list of tuple
        choices for the agreement with the substitute
    professeur : ForeignKey
        foreign key to the UserApp model representing the professor making the request
    cours : ForeignKey
        foreign key to the Cours model representing the course for the request
    cours_optionnel : str
        option for optional course
    confirmation_Pour_cours_optionnel : str
        confirmation for an optional course
    suppleant : str
        substitute for the professor
```

```
22      Motif : TextField
23          reason for the request
24      imputation : ForeignKey
25          foreign key to the ImputationSalaire model representing the
    salary allocation
26      cpo : str
27          CPO for the request
28      Accord_suppleant : str
29          agreement with the substitute
30      remarque : TextField
31          remarks for the request
32      num_poste : str
33          post number for the request
34      salaire : DecimalField
35          salary for the request
36      Accord_CF : str
37          agreement with the CF
38      date_soumission : DateField
39          date of submission for the request
40      statut_demande : str
41          status of the request
42
43      Methods
44      -------
45      __str__(self):
46          Return a string representation of the request
47
48      Meta:
49          verbose_name_plural = "Demandes en cours"
50      """
51
52      STATUT_CHOICES = [
53          ('En cours de traitement', 'En cours de traitement'),
54          ('Approuv ', 'Approuv '),
55          ('Refus ', 'Refus '),
56      ]
57
58      OPTION_CHOICES = [
59          ('','S lectionnez une option'),
60          ('Oui', 'Oui'),
61          ('Non', 'Non'),
62      ]
63
64      CHOICES_ACCORD_SUPPLEANT = [
65          ('', 'S lectionnez une option'),
66          ('Oui', 'Oui'),
67          ('Non', 'Non')
68      ]
69
70      professeur = models.ForeignKey(UserApp, on_delete=models.CASCADE)
71      cours = models.ForeignKey(Cours, on_delete=models.CASCADE)
72      cours_optionnel = models.CharField(max_length=200, choices=
    OPTION_CHOICES, default='S lectionnez une option')
73      confirmation_Pour_cours_optionnel = models.CharField(max_length
    =200, choices=OPTION_CHOICES, default='S lectionnez une option')
74      suppleant = models.CharField(max_length=200)
```

```
75      Motif = models.TextField()
76      imputation = models.ForeignKey(ImputationSalaire, on_delete=models.
        CASCADE)
77      cpo = models.CharField(max_length=200, null=True, blank=True)
78      Accord_suppleant = models.CharField(max_length=200, choices=
        CHOICES_ACCORD_SUPPLEANT, default='S lectionnez une option')
79      remarque = models.TextField(null=True, blank=True)
80      num_poste = models.CharField(max_length=200, null=True, blank=True)
81      salaire = models.DecimalField(default=0, decimal_places=3,
        max_digits=100, null=True, blank=True)
82      Accord_CF = models.CharField(max_length=200, choices=OPTION_CHOICES
        , default='S lectionnez une option')
83      date_soumission = models.DateField(default=timezone.now)
84      statut_demande = models.CharField(max_length=200, choices=
        STATUT_CHOICES, default='En cours de traitement')
85
86      def __str__(self):
87          """
88          Return a string representation of the request
89          """
90          return str(self.cours)
91
92      class Meta:
93          """
94          Meta class for Demandes
95          """
96          verbose_name_plural = "Demandes en cours"
```

4.2 Views

Views are defined in the views.py file. Views are functions or classes that handle HTTP requests from the user. Views return an HTTP response that is displayed to the user.For this application, we have privileged the implementation of classes for the views because easier to handle Example of view :

```
1  class FormProfPageView(View):
2      """
3      A class handling the form submission request for professors
4      ...
5      Attributes
6      ----------
7      template_name : str
8          the HTML template file for the form submission view
9
10     Methods
11     -------
12     get(self, request)
13         Handle GET requests to display the form for submission
14
15     post(self, request)
16         Handle POST requests to process form submission data
17     """
18
19     template_name = 'formprof.html'
20
```

```
21      def get(self, request):
22
23          try:
24              # Assertion to check if the user is authenticated
25              assert request.user.is_authenticated, "vous n' tes  pas
    authentifi . Vous devez vous connecter avant de pouvoir acc der
     cette page."
26
27              form = DemandesForm()
28              context = {'form': form}
29              return render(request, self.template_name, context)
30
31          except AssertionError as e:
32              context = {'error_message': e.args[0]}
33              return render(request, 'generic_error.html', context)
34
35
36      def post(self, request):
37
38          try:
39              # Assertion to check if the user is authenticated
40              assert request.user.is_authenticated, "Vous n' tes  pas
    authentifi . Vous devez vous connecter avant de pouvoir acc der
     cette page."
41
42              form = DemandesForm(request.POST)
43              context = {'form': form}
44
45              # Assertion to check if the form is valid
46              assert form.is_valid(), "Le formulaire est invalide.
    Veuillez  v rifier les champs."
47
48              # Retrieve information about the course and submission date
     from the request
49              cours = form.cleaned_data.get('cours')
50              year = form.cleaned_data.get('date_soumission').year
51              prof = request.user
52
53              # Check if there is already a request for the same course
    and year
54              existing_demande = Demandes.objects.filter(cours=cours,
    date_soumission__year=year, professeur=prof).first()
55
56              # Assertion to check if no request exists for the same
    course and year
57              assert not existing_demande, f"Une demande portant votre
    nom comme professeur titulaire pour ce cours existe d j  cette
    ann e {year}. Si elle n'a pas  t  soumise par vous, veuillez
    contacter l'administration."
58
59              # If no request exists, save the form data
60              demande = form.save(commit=False)
61              demande.professeur = request.user
62              demande.save()
63              return redirect('list-demande')
64
```

```
65          except AssertionError as e:
66              context = {'error_message': e.args[0]}
67              return render(request, 'generic_error.html', context)
```

4.3 Controller

The controller in this application is defined by the methods that handle the logic of the application. These methods include operations like creating a new request, adding and deleting request, among other functionalities.

4.4 Overview

```
ProjectSuppleezM/
|-- SuppleezApp/
|   |-- migrations/
|   |   |   |-- 0001_initial.py
|   |   |   |-- 0002_demandes_cours_optionnel_and_more.py
|   |   |   |-- 0003_alter_demandes_cours_optionnel_and_more.py
|   |   |   |-- 0004_remove_cours_cours_optionnel_and_more.py
|   |   |   |-- 0005_alter_demandes_date_soumission_and_more.py
|   |   |   |-- 0006_alter_archivedemandesupprime_cours_optionnel_and_more.py
|   |   |   |-- 0007_alter_cours_options_alter_demandes_options_and_more.py
|   |   |   |-- 0008_alter_archivedemandesupprime_date_soumission_and_more.py
|   |   |   |-- 0009_alter_archivedemandesupprime_options_and_more.py
|   |-- templates/
|   |   |   |-- 404.html
|   |   |   |-- 500.html
|   |   |   |-- archiveprofesseur.html
|   |   |   |-- encodercours.html
|   |   |   |-- formprof.html
|   |   |   |-- generic_error.html
|   |   |   |-- home.html
|   |   |   |-- listeDemande.html
|   |   |   |-- login.html
|   |   |   |-- signup.html
|   |   |   |-- template_erreur_csrf.html
|   |-- admin.py
|   |-- apps.py
|   |-- forms.py
|   |-- apps.py
|   |-- models.py
|   |-- signals.py
|   |-- tests.py
|   |-- views.py
|-- SuppleezM/
|   |-- asgi.py
|   |-- settings.py
|   |-- urls.py
|   |-- wsgi.py
|-- static/
|   |-- all static file
|-- db.sqlite3
|-- manage.py
|-- requirements.txt
```

4.5 Details of the Overview

The most important files in our Django application include:

- `settings.py`: Contains the configuration parameters of the application such as database information, secret keys, security parameters, etc.
- `urls.py`: Contains the URLs of the application.
- `asgi.py` and `wsgi.py`: Necessary for the deployment of the application.
- `models.py`: Contains the models of classes which define the structure of the database.
- `views.py`: Contains the view functions that process HTTP requests and return the appropriate HTTP responses.
- `test.py`: A file containing the tests of the application.
- `templates/`: A directory containing the HTML files used to render the web pages.

5 Test File

The test suite for models and views is divided into several classes. Each class contains specific tests, inheriting from the `TestCase` or `SimpleTestCase` classes provided by Django. Here's an outline of the various tests and their objectives:

5.1 Model Tests

UserAppTest: Verifies the correct creation of a user.

test_user_creation: Creates a user with a predefined set of parameters and verifies its existence in the database.

CoursModelTest: Verifies the Cours model.

test_cours_representation: Ensures that the string representation of the `Cours` model is correct.

ImputationSalaireModelTest: Verifies the `ImputationSalaire` model.

test_Imputation_representation: Checks that the string representation of `ImputationSalaire` is accurate.

DemandesModelTest: Verifies the `Demandes` model.

test_demandes_representation: Creates a set of requests with proper relationships and verifies that the string representation is accurate.

ArchiveDemandeTraiteModelTest: Verifies the archive model for processed requests.

test_ArchiveTraite_representation: Confirms that the string representation of archived processed requests is correct.

ArchiveDemandeSupprimeModelTest: Verifies the archive model for deleted requests.

test_ArchiveSupprime_representation: Confirms that the string representation of archived deleted requests is correct.

5.2 View Tests

HomeProfViewTest: Verifies the features of the homepage.

test_home_page_loads_properly: Checks that the homepage loads correctly.

test_template_name_correct: Ensures that the appropriate template is used to render the homepage.

test_url_available_by_name: Confirms that the homepage loads correctly using its URL name.

test_url_exists_at_correct_location: Confirms that the homepage URL is correct.

ListDemandesViewTest: Ensures correct display of the request list.

test_template_used: Verifies that the correct template is used.

test_view_returns_200_when_authenticated: Confirms that the view returns a status code of 200 when the user is authenticated.

test_user_requests_are_displayed: Confirms that the user's requests are displayed correctly in the template.

ListDemandesArchiveViewTest: Checks the display of archived requests.

test_template_used: Ensures that the appropriate template is used to display the archives.

test_view_returns_200_when_authenticated: Confirms that the view returns a status code of 200 when the user is authenticated.

test_archived_requests_are_displayed: Verifies that archived requests are displayed accurately in the template.

FormProfPageViewTest: Verifies the professor form page.

test_get_view_authenticated_user: Confirms that the form page is accessible to an authenticated user.

test_get_view_non_authenticated_user: Confirms that the form page is not accessible to an unauthenticated user.

CustomLoginViewTest: Verifies the custom login view.

test_get_login_view: Confirms that the login page is accessible via GET.

test_post_valid_login_form: Verifies that a user can log in with a valid form.

test_post_invalid_login_form: Checks that the login form shows an appropriate error message when it is invalid.

SignUpViewTest: Verifies the signup page.

test_get_signup_view: Ensures that the signup page is accessible via GET.

test_post_valid_signup_form: Confirms that a user can sign up with a valid form.

test_post_invalid_signup_form: Ensures that the signup form shows the correct error message when it is invalid.

CoursViewTest: Verifies the course form view.

test_get_cours_view_authenticated: Confirms that the course page is accessible to an authenticated user.

test_get_view_non_authenticated_user: Confirms that the course page is not accessible to an unauthenticated user.

test_post_valid_cours_form: Verifies that a valid form redirects correctly and creates a new `Cours` object.

test_post_invalid_cours_form: Checks that an invalid form shows an appropriate error message.

LogoutViewTest: Verifies the logout functionality.

test_logout_authenticated_user: Confirms that logout redirects the user to the homepage.

test_get_view_non_authenticated_user: Confirms that logout is not possible for an unauthenticated user.

6 Frontend and Backend Technologies

Our application employs HTML, JavaScript, CSS, and Bootstrap, alongside additional scripts to enhance the user experience. We will focus specifically on jQuery in this discussion

jQuery

Description: jQuery is a JavaScript library that simplifies DOM manipulation and interactions with browser events.

Usage: We integrate jQuery to facilitate JavaScript operations and manipulate the DOM more efficiently.

By incorporating these scripts, we enhance our web application with advanced features, providing a smoother and more interactive user experience.

7 Contact

For any question or suggestion, feel free to send me an email to :ulrich.toujinana@student.unamur.be or to ulrich.nana90@gmail.com .