

**Software Engineering**

**Discussion Forum**

**Student:** Filip Denisa-Mariana

**Group:** 30432

**Semigroup:** 1

**Lecturer:** Dr. Eng. Prof. Eneia Todoran

**Faculty of Automation and Computer Science, TUCN**

**2021-2022**

Contents

[1. Abstract 3](#_Toc92044628)

[2. Models 3](#_Toc92044629)

[2.1. Formal Models 3](#_Toc92044630)

[2.1.1. Session Types 3](#_Toc92044631)

[2.2. Semi-formal Models 4](#_Toc92044632)

[2.2.1. User Profile Models 4](#_Toc92044633)

[**2.2.1.1.** **Profile** 4](#_Toc92044634)

[**2.2.1.2.** **Statistics** 5](#_Toc92044635)

[2.2.2. Forum Models 5](#_Toc92044636)

[**2.2.2.1.** **Post** 5](#_Toc92044637)

[**2.2.2.2.** **Reply** 5](#_Toc92044638)

[3. Diagrams 6](#_Toc92044639)

[3.1. Use Case Diagram 6](#_Toc92044640)

[3.2. Class Diagrams 7](#_Toc92044641)

[3.3. Deployment Diagrams 8](#_Toc92044642)

[4. Code 10](#_Toc92044643)

[5. References 10](#_Toc92044644)

[6. Appendix - Miniproject 10](#_Toc92044645)

# Abstract

The Discussion Forum project models a forum, in which users can create posts about different subjects, and other users can reply to the thread, giving their opinion on the matter at hand. The replies (answers) can be rated by the creator of the post. This project was developed using the concept of session types, the main logic behind it being carried through a protocol described with the **Scribble** protocol description language.

# Models

## Formal Models

## Session Types

Throughout the project, **session types** were used to implement the flow of the interaction between the users of the web application, which was achieved by using the **Scribble** protocol description language.

[**Multiparty Session Types**](https://www.doc.ic.ac.uk/~yoshida/multiparty/multiparty.pdf) extend the theory of binary session types to encompass interactions containing more than two participants. The main idea behind multiparty session types is to describe the interactions between participants in a top-down manner as a global type. These global types can then be “projected” into local types, which describe the interaction from the point of view of a particular participant.

The **Scribble** specification for the used protocol can be described as follows:

The protocol is carried out between three participants:

1. the user that creates the post (called Creator – C)
2. the administrator of the whole website (called Administrator – A)
3. the user that replies to the published post (called Replier – R)

The protocol is started when the Creator C wants to publish a post in the forum. He creates the post, gives it a title and some content, and queries the administrator by trying to publish it. The administrator can decide whether to accept or reject the post. The loop can then be continued, with the Creator C trying to publish multiple other posts.

If the Administrator A accepts the post, then it officially appears in the forum, and other users (repliers) can interact with it. A Replier R will reply to the post, and the Creator C will rate the received answer, from a scale of 1 to 10. The loop can then be continued, with other repliers.

If the Administrator A rejects the post, it will never be published on the official forum, therefore other users cannot interact with the post.

module DiscussionForum  
  
type <py> "types.IntType" from "types.py" as Int  
type <py> "types.StringType" from "types.py" as String  
  
global protocol Forum(role Creator as C,  
 role Administrator as A,   
 role Replier as R) {  
 rec LOOP {  
 choice at C {  
 query(post:String) from C to A;  
 continue LOOP;  
 } or {  
 choice at C {  
 ACCEPT() from A to C; # If the admin approves the post from C  
 reply(String) from R to C;  
 rate(Int) from C to R;

continue LOOP;  
 } or {  
 REJECT() from A to C; # If the admin rejects the post from C   
 }   
}

}

}

## Semi-formal Models

The models used for the Discussion Forum project were separated into two big categories, models for the profile of the user, and models for the discussion forum.

## User Profile Models

### **Profile**

The **Profile** model was constructed for holding important information for the users of the platform. It has the following fields:

|  |  |
| --- | --- |
| **Name** | **Type** |
| **User** | OneToOneField |
| **Image** | ImageField |
| **Age** | IntegerField |
| **Birthday** | DateField |
| **City** | CharField |
| **Country** | CharField |

The User field corresponds to the User model imported from the **django.contrib.auth.models** module, containing important fields of a user, such as username, password, email, first name and last name. The rest of the fields are from the Profile model contain additional information about a user, creating their detailed background.

### **Statistics**

The **Statistics** model is tightly related to the Profile model, saving and storing the data from the current user session. The fields of this model are:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Information** |
| **User** | ForeignKey | User from the current request session |
| **visit\_count** | IntegerField | The total number of website visits of the user |
| **post\_count** | IntegerField | The total number of posts created by the user |
| **reply\_count** | IntegerField | The total number of replies given by the user |

## Forum Models

### **Post**

The **Post** model represents the post a user can create in the discussion forum. A post starts a conversation in which multiple users can participate and discuss upon.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Information** |
| **User** | ForeignKey | User from the current request session, author of the post. |
| **post\_id** | AutoField | (Primary Key) The id of the current post. |
| **title** | CharField | The title of the created post. |
| **post\_content** | CharField | The content of the created post. |
| **date\_added** | DateTimeField | Creation date and time of the post. |
| **image** | ImageField | The chosen image for the post. |
| **admin\_approved** | BooleanField | Boolean field stating if the administrator approved the post. |

For a post to be shown in the home page of the web application, it must be approved beforehand by the administrator of the website. If it is approved, then it is published on the main forum, and other users can interact with the post.

### **Reply**

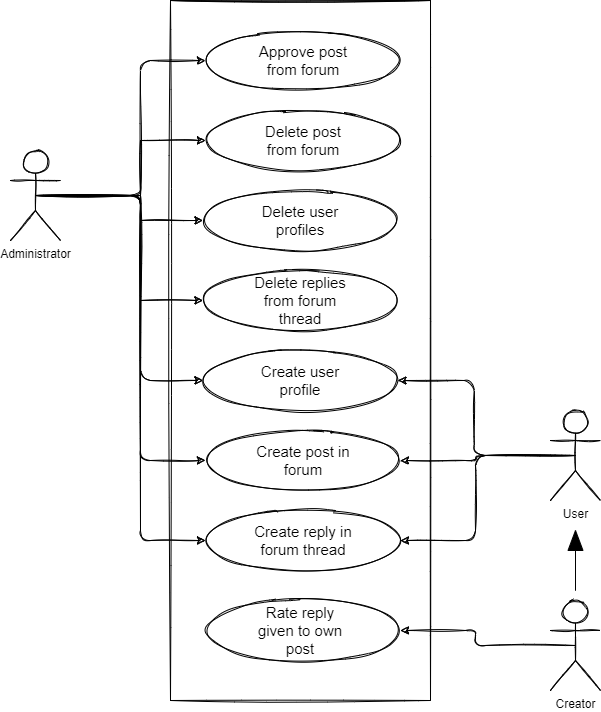
The **Reply** model represents the interaction of a user with a created post. It is the input of a user regarding a created conversation, and it allows people to interact between them.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Information** |
| **User** | ForeignKey | User from the current request session, author of the reply. |
| **reply\_id** | AutoField | (Primary Key) The id of the current reply. |
| **post** | ForeignKey | Post of which the reply is created. |
| **post\_content** | CharField | The content of the created reply. |
| **date\_added** | DateTimeField | Creation date and time of the reply. |
| **image** | ImageField | The chosen image for the reply. |
| **rate** | FloatField | The rate of the reply, given by the author of the post. |

The user that created the original post can rate the received replies, informing other users of the most useful ones.

# Diagrams

## Use Case Diagram



## Class Diagram

The project was divided into three main parts:

1. **user\_profile** – the application responsible for the configuration of the profile of a user
2. **discussion** – the application responsible for the configuration of the discussion forum
3. **forum** – the main part of the project, responsible for binding the two applications together

Each part of the project has its own class diagram. My own classes, such as the models described in the 2.2. section of this documentation, are inherited from the base classes of Django.

**Discussion application class diagram**

A screenshot of a video game

Description automatically generated

**User Profile application class diagram**

A screenshot of a video game

Description automatically generated

## Deployment Diagram

The project was designed with the programming language **Python 3.9.9**, the eight-maintenance release of Python 3.9. Python is an interpreted high-level general-purpose programming language that lets you work quickly and integrate systems more effectively. Its design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability) with its use of [significant indentation](https://en.wikipedia.org/wiki/Off-side_rule). Its [language constructs](https://en.wikipedia.org/wiki/Language_construct) as well as its [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) approach aim to help [programmers](https://en.wikipedia.org/wiki/Programmers) write clear, logical code for small and large-scale projects.

To keep the Python external requirements of the project (such as packages and modules, that do not come in the standard library) in a separate environment, a virtual environment was used. **virtualenv** is a tool to create isolated Python environments.

**Django 4.0** is a [Python](https://en.wikipedia.org/wiki/Python_(programming_language))-based [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) [web framework](https://en.wikipedia.org/wiki/Web_framework) that follows the model–template–views (MTV) [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern_(computer_science)). Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes [reusability](https://en.wikipedia.org/wiki/Reusability) and "pluggability" of components, less code, low coupling, rapid development, and the principle of [don't repeat yourself](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself). Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative [create, read, update and delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) interface that is generated dynamically through [introspection](https://en.wikipedia.org/wiki/Type_introspection) and configured via admin models.

The **Web Server Gateway Interface** (WSGI) is a simple [calling convention](https://en.wikipedia.org/wiki/Calling_convention) for [web servers](https://en.wikipedia.org/wiki/Web_server) to forward requests to [web applications](https://en.wikipedia.org/wiki/Web_application) or [frameworks](https://en.wikipedia.org/wiki/Web_framework) written in the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)).

**SQLite** is a [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS) contained in a [C](https://en.wikipedia.org/wiki/C_(programming_language)) [library](https://en.wikipedia.org/wiki/Library_(computer_science)). In contrast to many other [database management systems](https://en.wikipedia.org/wiki/Database), SQLite is not a [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server) database engine. Rather, it is embedded into the end program.

The application is run on the localhost (IP address: 127.0.0.1), on the port 8000. It is executed from the command line, using the following command:

venv\Scripts\python manage.py runserver

Shape

Description automatically generated with medium confidencewhere manage.py is Django’s command-line utility for administrative tasks.

# Code

# References

Simon Fowler**,** ”*Session Types in Programming Languages*”- <http://www.simonjf.com/2016/05/28/session-type-implementations.html>

Raymond Hu, Nobuko Yoshida and Kohei Honda ”*Session-Based Distributed Programming in Java*”, Imperial College London

Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng ”*The Scribble Protocol Language*”, Imperial College London

The Scribble Team, ”*Scribbling Protocols Overview, Specification and verification of distributed applications using multiparty session types”* - <https://groups.inf.ed.ac.uk/abcd/meeting-january2014/raymond-hu.pdf>

scribble.org, *”Scribble: Describing Multi Party Protocols”* **-** <http://www.scribble.org>

docs.djangoproject.com,”*Sessions”* **-** <https://docs.djangoproject.com/en/4.0/topics/http/sessions/>

wikipedia.com, ”*SQLite”* - <https://en.wikipedia.org/wiki/SQLite>

wikipedia.com, ”*Python (programming language)*” - <https://en.wikipedia.org/wiki/Python_(programming_language)>

wikipedia.com**,** ”*Web Server Gateway Interface”*- <https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface>

wikipedia.com**,** ”*Django (Web framework)”* **-** <https://en.wikipedia.org/wiki/Django_(web_framework)>

guru99.com, ”*Deployment Diagram: UML Tutorial with Example”* - <https://www.guru99.com/deployment-diagram-uml-example.html>

# Appendix - Miniproject

templates/base.html

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="utf-8">  
 <title>{% block title %}Project{% endblock %}</title>  
 </head>  
 <body>  
 <header>  
 <h1>Software Engineering Project</h1>  
 {% if user.is\_authenticated %}  
 <a href="{% url 'logout' %}">Log Out</a>  
 {% else %}  
 <a href="{% url 'login' %}">Log In</a> / <a href="{% url 'signup' %}">Sign Up</a>  
 {% endif %}  
 <hr>  
 </header>  
 <main>  
 {% block content %}  
 {% endblock %}  
 </main>  
 </body>  
</html>

templates/hello\_world.html

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Hello World</title>  
</head>  
<body>  
  
</body>  
</html>

minprojects/urls.py

from django.conf.urls import url, include  
from django.contrib import admin  
from django.urls import path  
  
  
urlpatterns = [  
 path("admin/", admin.site.urls),  
 path("", include("core.urls")),  
 path('accounts/', include('django.contrib.auth.urls')),  
]

hello\_world/urls.py

from django.urls import path  
from .views import home\_page\_view  
  
urlpatterns = [  
 path("", home\_page\_view, name="hello\_world"),  
]

hello\_world/views.py

from django.http import HttpResponse  
  
  
def home\_page\_view(request):  
 return HttpResponse("Hello, World!")

hello\_world/models.py

from \_\_future\_\_ import unicode\_literals  
  
# Create your models here.  
from django.db import models  
from django.contrib.auth.models import User  
from django.db.models.signals import post\_save  
from django.dispatch import receiver  
  
  
class Profile(models.Model):  
 user = models.OneToOneField(User, on\_delete=models.CASCADE)  
 birth\_date = models.DateField(blank=True, help\_text="Format: YYYY-MM-DD")  
 age = models.IntegerField(blank=True, help\_text="Your current age.")  
 country = models.CharField(max\_length=30, blank=True, help\_text="Your country of origin.")  
 city = models.CharField(max\_length=30, blank=True, help\_text="Your city of origin.")  
  
 def \_\_str\_\_(self):  
 return self.user.username