NLPeace: Github Repository Release1 Demo: here !
Release2 Demo: here!

Deployed App: here!

Grade for Release 1

NLPeace 490 Grade for Release 2

Team members

Name and Student id	GitHub id	Email address	Link to issues	Number of story points that member was an **author** on.	Release 2 Videos
Fatima El Fouladi 40108832	seaiam	fatimaelfouladi@gmail.com	here	71	DemoRel…
Anum Siddiqui 40129811	AnumSidd	anum_siddiqui2000@hotmail.com	here	92	Anum-Re…
Jeff Wilgus 29206345	jeffrey-w	jeff.wilgus@gmail.com	here	59	release-2…
David Lemme 40157270	davrine	Daviddantelemme@gmail.com	here	64	David Le… (Correction: I didn't create chatRoom model)
Mira Aji 40041473	miraaji	mira_aji@outlook.com	here	61	mira_aji.…
Adam Qamar 40175980	aqa02	adamqam2@gmail.com	here	19	Adam_Re…
Shabia Saeed 40154081	shabiasaeed	shabia-ab@hotmail.com	here	71	Shabia_S…

Raya Maria Lahoud 40129965	rayalahoud	rayamarialahoud@gmail.com	here	59	raya-dem…
Nelly Bozorgzad 40289770	nellyb4	nellybozorgzad@gmail.com	here	30	Nelly Boz…
Joshua-James Nantel-Ouimet 40131733	NanoProd	joshuajamesnano@gmail.com	here	61	https://youtu.be/BSNQoheE4Q8
Andrew Chan 40133396	AofSpade	andantjer@gmail.com	here	23	AndrewC…

We count the story points by counting the story points of each issue the team member had committed to. The assignees on the issues give the specific issues members are getting credit for.

Project summary

NLPeace is a social networking app available both on desktop and mobile. We aim to connect people and foster a safe environment free of hate and offensive content. We are leveraging natural language processing to build a strong language model that will allow for our content moderation to be automatic. Hateful content is thus nipped at the bud. As our network grows, we aim to amass more data to create a stronger language model and ultimately, a safer, more peaceful experience.

Risk

- **Inaccurate Moderation:**
  Harmless content might be inaccurately flagged as hate speech or we might fail to detect actual harmful speech which could lead to user frustration.
  Risk Level: High
  Mitigation Strategy: Regularly update and test the system using diverse examples, including false positives and negatives. Additionally, provide users with the ability to appeal content decisions. We will also add admin users that can perform QA on the moderation.

- **Avoidance Techniques:**
  Users might try to bypass moderation by using coded language or through media to convey harmful speech.
  Risk Level: Medium
  Mitigation Strategy: Update the moderation rules regularly to adapt to new avoidance techniques. Additionally, implement keyword filtering to detect evasive content.

- **User Backlash and Public Relations Issues:**
  Users may perceive the moderation efforts as either too strict or not strict enough, leading to public backlash and negative publicity for the app. Moreover, controversial content moderation decisions could spark outrage on social media.
  Risk Level: Medium
  Mitigation Strategy: Clearly communicate the app's moderation policies and guidelines to users and the public to manage expectations and minimize misunderstandings. Develop a crisis communication plan to manage potential PR issues and controversies effectively.

- **Potential technological obsolescence**
  The AI algorithms and models are quickly evolving; newer, more advanced

models and techniques emerge rapidly. This can result in the app's AI moderation system becoming less effective over time.
Risk Level: High
Mitigation Strategy: Adopt an agile development approach to allow quick iterations and updates. This flexibility will let us integrate newer AI models and techniques as they become available. Our AI models will continuously learn as our user-base grows and as more content is posted, it will be added to our NLP pipeline to train our models.

- **Security Vulnerabilities:**
  Bad coding practices can lead to security vulnerabilities in the code of the application. When this happens attackers or even regular users may be able to gain access to privileged data or functions.
  Risk Level: High
  Mitigation Strategy: All developers should brush up on good coding practices and all code should be peer reviewed.

Legal and Ethical issues

- There might be an ethical and legal issue arising from our collection of user posts to build and train our NLP model. We aim to mitigate this by having clear terms of service that detail how we will use user data to make the platform safer and train our model. We will be transparent in our data usage.

- Handling user information such as emails and posts (if their account is private) is exposing us to some legal risks. This risk is explored above.

Velocity

***Project Total*:** 92 stories, 338 points over 24 weeks

Iteration 1 (13 stories, 46 points)

In this iteration, we worked on mainly technical tasks, like setting up the base of our project. This involved setting up the Django application and the developer databases. We added a CI pipeline as well as containerization of the project. Most importantly, we researched hate speech datasets and implemented a NLP model which reached 93% accuracy in detecting hate speech and offensive language.

Iteration 2 (9 stories, 25 points)

In this iteration, we worked on updating the containerization of the project and setting up the databases within the container. This task was arduous and took longer than expected. We also set up the CI pipeline to run the tests inside the container. We took care of account creation, log-in, log-out and the update of profile picture as well as profile banner.

Iteration 3 (14 stories, 25 points)

In this iteration, we worked on a few issues we pushed forward from iteration 2. Users can now update their bio, username and password. Users can also reset their passwords if they forget it. We implemented posts with text and images as well as replies to posts. We also fixed some frontend bugs.

Iteration 4 - Release 1 (18 stories, 52 points)

This iteration was focused on deployment, bug fixes and posting. We implemented the central features for posting such as, commenting, reposting , reporting , liking and disliking. We also fixed a few bugs both on the frontend and backend. Finally, our app is now deployed to Heroku and ready to use !

Iteration 5 (15 stories, 43 points)

In this iteration, we implemented the NLP monitoring of posts. We also fixed a few bugs and implemented features such as deleting posts, blocking and reporting users, and admin monitoring of reported users. We also implemented features for the user's profile, allowing them to now keep track of their media posts, reposted posts, and followers and following.

Iteration 6 (8 stories, 23 points)

In this iteration, we refactored our code to and introduced a business logic layer to make the code in the views easier to read and understand. We also implemented real-time direct-messaging from user to user. Users can now send text chats as well as files. We also fixed a few bugs on the frontend and some features that our TA mentioned as missing.

Iteration 7 (12 stories, 46 points)

This iteration was focused on completing dm features, such as sharing gifs and files. We also incorporated our model into dms so that it can moderate hate or violent speech. We also introduced video posts, pinned posts, and editing. Users can now unblock eachother. A few technical tasks were also completed, such as a refactoring of our code base into services to aid views, and real-time notifications.

Iteration 8 - Release 2 (15 stories, 30 points)

As this iteration was the last for release 2, it was mostly focused on fixing different bugs in the frontend and backend of the app. DMs are now fully implemented and users can scroll up and load more messages, as well as see a history of conversations they have partook in. We also implemented advertisement integration into the app, which is currently filled with placeholder values. Finally, the user profile has now all tabs available.

Iteration 9 (13 stories, 36 points)

This iteration, implemented the main functionalities related to communities, mainly creating one, joining, editing privacy configuration, and posting. Deployment was finalized and the syncing between the main branch and the deployment. Users can now add hashtags to their posts and delete DMs they've sent.

Iteration 10 (14 stories, 46 points)

In this iteration, we implemented the option for users to turn on and off the NLP monitoring of their content. Trending posts as well as interactions in communities were fully implemented. Several bugs related to DMs and posting were also resolved. Finally, a few of our team members worked on their 491 projects.

Iteration 11 (13 stories, 48 points)

TBD

Iteration 12 (10 stories, 43 points)

TBD

Overall Arch and Class diagram

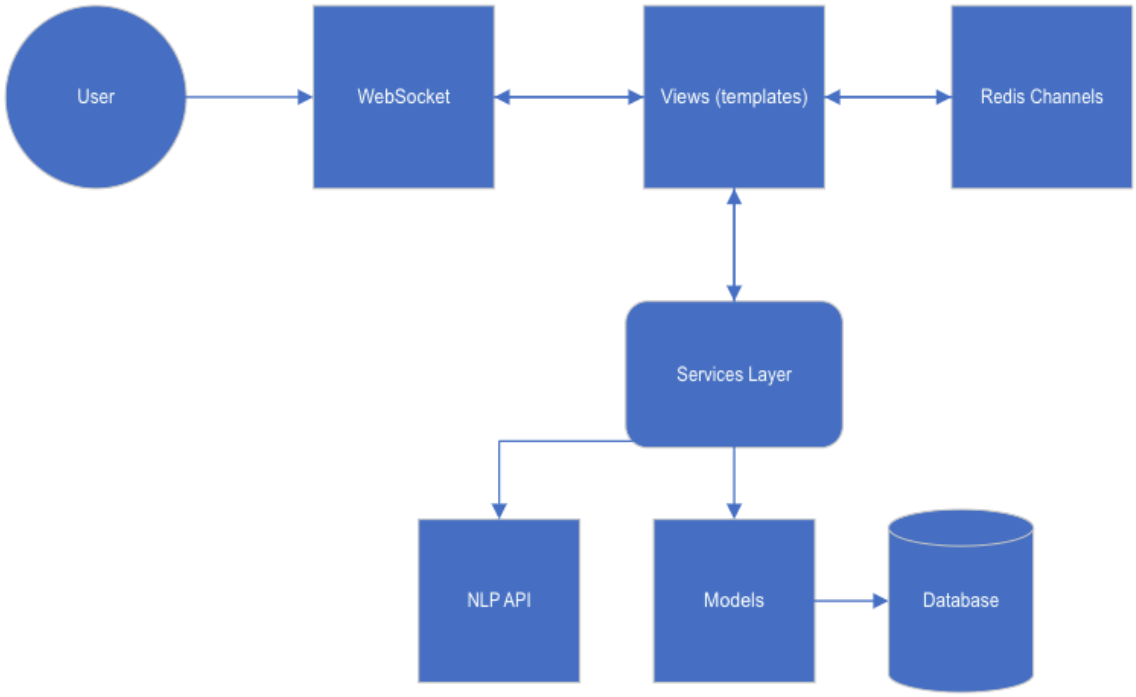


Figure 1: Architecture Diagram

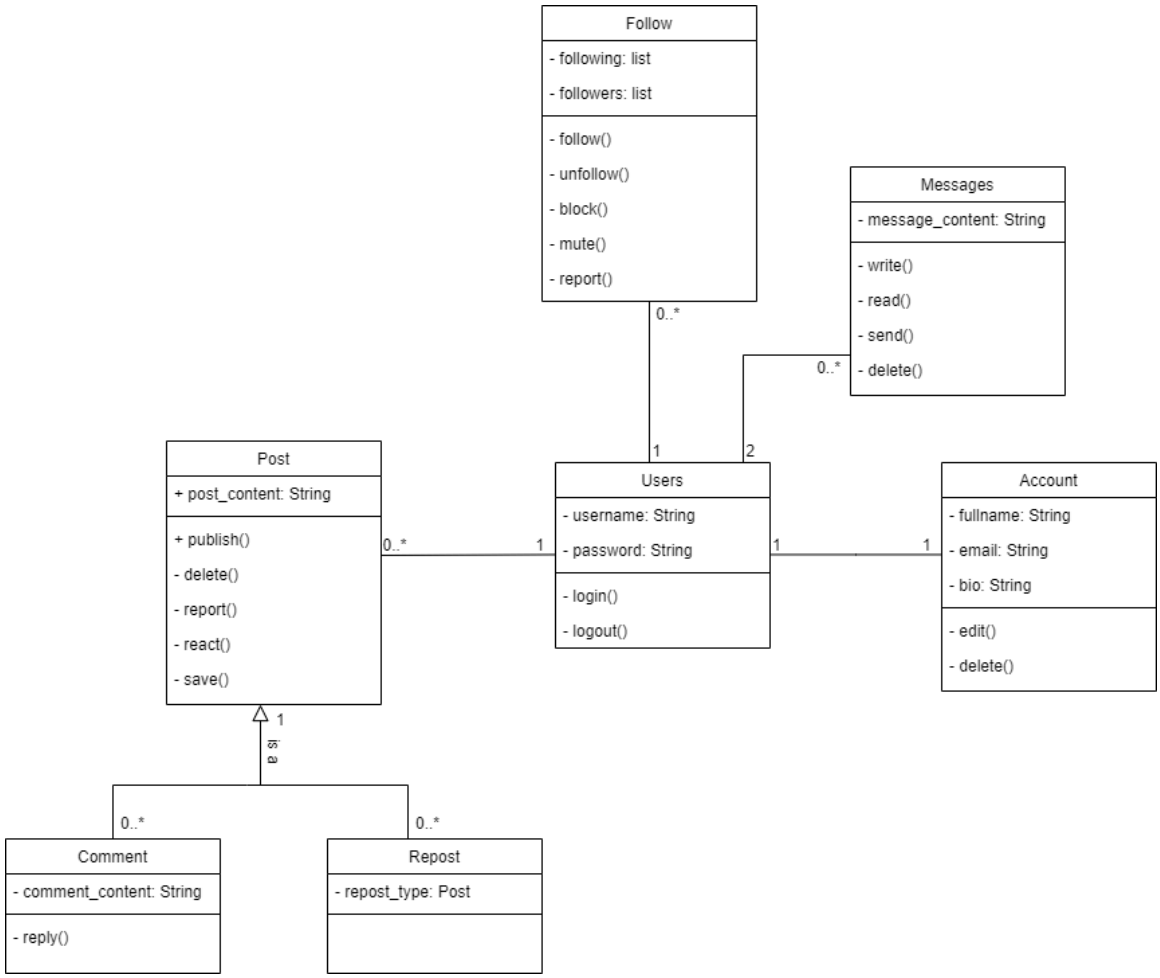


Figure 2: Class Diagram

Infrastructure

Django

Django is a Python web framework that streamlines web app development, handling tasks like database management and user authentication. In our app, Django will manage user accounts, posts, and interactions, providing a secure foundation for building dynamic web applications efficiently. This allows developers to focus on creating unique features while leveraging Django's powerful built-in functionalities.

Redis

Redis is an in-memory data structure store, used as a database, cache, and message broker. In our Django application, Redis is primarily utilized for managing real-time functionalities like WebSockets through Redis Channels. This enhances the application's capability to handle asynchronous communication and real-time features, such as live notifications or chat functions. Redis' fast in-memory operations ensure high performance and quick response times for these real-time features, making the user experience smoother and more interactive.

Pipenv

Pipenv is a Python packaging tool that combines dependency management and environment management. It automates the setup of virtual environments and uses a Pipfile to track dependencies. Deterministic builds are ensured through the Pipfile.lock, promoting consistent project reproduction across systems.

Pylint

Pylint is a static code analysis tool for Python that checks source code against a coding standard, looks for programming errors, and offers suggestions for code improvement. It can also help enforce a coding standard, detect code smells, and identify unused code. Pylint is highly customizable and integrates well with most

development environments, making it a valuable tool for maintaining code quality in Python projects.

Name Conventions

Our preferred language is Python. The widely accepted style guide for that language is found here. We may choose to include our linter as part of our CI/CD pipeline to enforce the conventions described by the linked document but will defer that decision until higher-priority matters have been addressed.

Code

*Key files: top **5** most important files (full path). We will also be randomly checking the code quality of files. Please let us know if there are parts of the system that are stubs or a prototype so we grade these accordingly.*

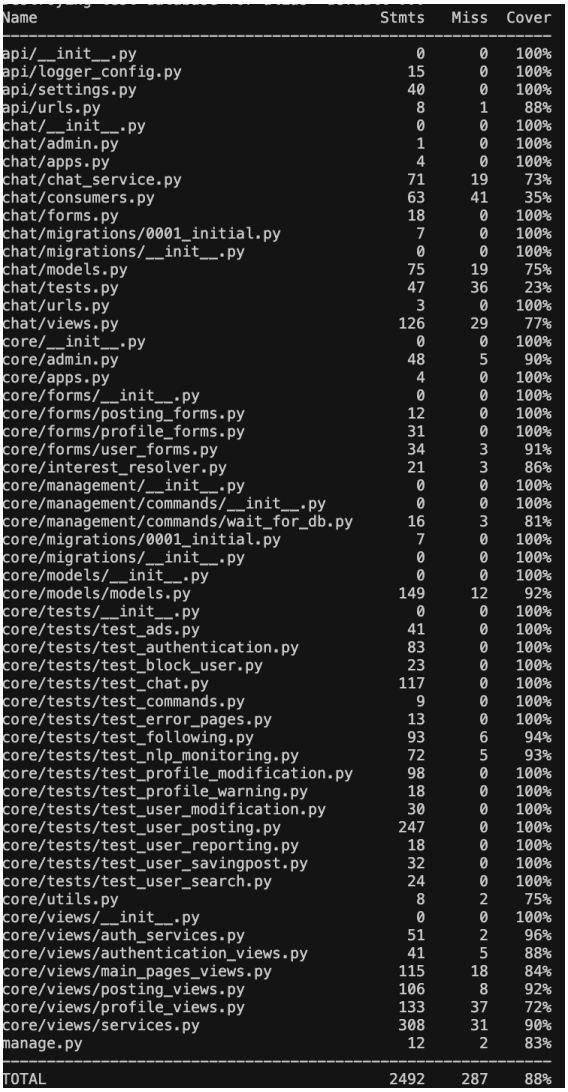
File path with clickable GitHub link	Purpose (1 line description)
backend/api/chat/consumers.py	This is the page containing the logic for the real-time chat
backend/api/core/views/services.py	This file holds the business logic for our views
backend/api/core/models/models.py	This file holds our core models
backend/api/core/forms/user_forms.py	This file holds user forms such as register, log-in and some profile modification.
backend/api/core/views/main_pages_views.py	This file hold the views for the main pages of our project.

Testing and Continuous Integration

Each story needs tests before it is complete. If some class/methods are missing unit tests, please describe why and how you are checking their quality. Please describe any unusual aspects of your testing approach.

List the **5** most important test with links below.

Test File path with clickable GitHub link	**What is it testing (1 line description)**
backend/api/core/tests/test_authentication.py	This tests users authentication
backend/api/core/tests/test_authentication.pyy	test_correct_login() tests that users can log-in
backend/api/core/tests/test_authentication.py	test_incorrect_login() tests that , given an incorrect password, a user cannot login
backend/api/core/tests/test_profile_modification.py	test_update_profile_pic_view_unauthenticated ensures only logged in users can edit profile pictures
backend/api/core/tests/test_backend.py	test_privacy_settings_authenticated() ensures authenticated users can update their privacy settings



Name	Stmts	Miss	Cover
api/__init__.py	0	0	100%
api/logger_config.py	15	0	100%
api/settings.py	40	0	100%
api/urls.py	8	1	88%
chat/__init__.py	0	0	100%
chat/admin.py	1	0	100%
chat/apps.py	4	0	100%
chat/chat_service.py	71	19	73%
chat/consumers.py	63	41	35%
chat/forms.py	18	0	100%
chat/migrations/0001_initial.py	7	0	100%
chat/migrations/__init__.py	0	0	100%
chat/models.py	75	19	75%
chat/tests.py	47	36	23%
chat/urls.py	3	0	100%
chat/views.py	126	29	77%
core/__init__.py	0	0	100%
core/admin.py	48	5	90%
core/apps.py	4	0	100%
core/forms/__init__.py	0	0	100%
core/forms/posting_forms.py	12	0	100%
core/forms/profile_forms.py	31	0	100%
core/forms/user_forms.py	34	3	91%
core/interest_resolver.py	21	3	86%
core/management/__init__.py	0	0	100%
core/management/commands/__init__.py	0	0	100%
core/management/commands/wait_for_db.py	16	3	81%
core/migrations/0001_initial.py	7	0	100%
core/migrations/__init__.py	0	0	100%
core/models/__init__.py	0	0	100%
core/models/models.py	149	12	92%
core/tests/__init__.py	0	0	100%
core/tests/test_ads.py	41	0	100%
core/tests/test_authentication.py	83	0	100%
core/tests/test_block_user.py	23	0	100%
core/tests/test_chat.py	117	0	100%
core/tests/test_commands.py	9	0	100%
core/tests/test_error_pages.py	13	0	100%
core/tests/test_following.py	93	6	94%
core/tests/test_nlp_monitoring.py	72	5	93%
core/tests/test_profile_modification.py	98	0	100%
core/tests/test_profile_warning.py	18	0	100%
core/tests/test_user_modification.py	30	0	100%
core/tests/test_user_posting.py	247	0	100%
core/tests/test_user_reporting.py	18	0	100%
core/tests/test_user_savingpost.py	32	0	100%
core/tests/test_user_search.py	24	0	100%
core/utils.py	8	2	75%
core/views/__init__.py	0	0	100%
core/views/auth_services.py	51	2	96%
core/views/authentication_views.py	41	5	88%
core/views/main_pages_views.py	115	18	84%
core/views/posting_views.py	106	8	92%
core/views/profile_views.py	133	37	72%
core/views/services.py	308	31	90%
manage.py	12	2	83%
TOTAL	2492	287	88%

Figure 3: Code Coverage of 88%

Continuous integration enables code changes from multiple developers to be merged automatically, in which automated tools test and validate the code before it's integrated. Continuous integration for this Django application was set up using GitHub Actions which is a CI platform provided by GitHub. It works by running a test whenever a pull request is made, and informing the developer whether the tests have passed or not.

Link to CI: [django.yml]