IN SUPREMÆ DIGNITATIS • 1343 •

UNIVERSITÀ DI PISA

Master's Degree in Cybersecurity and Computer Science

Message in a bottle

Advanced Software Engineering

Federico Bernacca 536683
Paola Petri 561151
Nicolò Pierami 544707
Manfredo Facchini 547348
Francesco Kotopulos De Angelis 549045

GitHub repo: Message in a bottle

Contents

1 Introduction

The approach that will be used in the following report consists in accurately describing the most sensitive implementation choices, with attached several screenshots extracted from the client application that show the correct functioning of the developed program.

2 Implementation

In the following section the focus is on the implementation choices at the backend level.

2.1 Database

Since the project is a messaging application between users registered to the service, it was considered appropriate to provide a persistent storage mechanism for users registered with the service. To implement this mechanism, the *User* class initially provided has been extended, adding some fields, including the message lists associated with each user:

- *to_be_sent*: list of scheduled messages;
- *sent*: list of sent messages;
- *received*: list of received messages.

Each one of this lists has *Messages* as type and each of its element has *Message* as type: in order to save a non-primitive data structure in the database, the choice was to serialize it in the *json* object, resulting in a string that can be stored in the database. When one of these *json* lists is extracted from the database, it is deserialized to *Messages*.

2.2 Celery

Celery software was used for the delivery of messages, notifications, and the lottery. In particular, there are the following periodic celery tasks:

2.2.1 Message delivery

The initial idea was to associate a task for each message programmed by a user but the idea was discarded as there would have been a very long task queue, leading to performance degradation; so the final approach Use a Celery periodic task to asynchronously check the database (1 minute) for messages that need to be sent. In this case, there is a single task that queries the database at regular intervals and delivers the messages when the delivery time is expired.

2.2.2 Notification

A similar approach to the previous one was used for notifications: at regular intervals, one task checks if messages have been received and another checks if they have been read from their recipients; in particular:

- When a message arrives, the recipient receives a notification on the mailbox
- When reading a message, the sender receives a notification on the mailbox.

In the next section, this functionality will be illustrated via screenshots.

2.2.3 Lottery

A similar approach to the previous one was used for the lottery: at regular intervals, a task draws a lucky number, and if a user has played this number, he wins 100 points. Once the threshold of 150 points is reached, a user can delete a scheduled message.

2.2.4 Delete a user

When a user wants to unsubscribe from the service, if it has scheduled messages, the system permits its unsubscribing and this user disappear from the program, except from his scheduled messages, that have to reach its recipients. This celery task check if in the database, there are some remaining information to delete when a user has unsubscribed and there are no more messages to delivery.

2.3 GUI

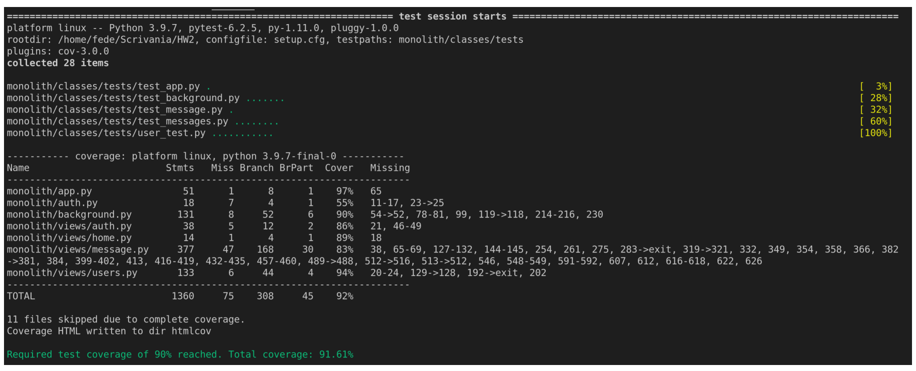
It was also chosen to create a GUI to make the project more realistic, using *HTML*, *CSS* and *JavaScript* for some interactions.

3 Screenshots

In this section section are shown the screenshots containing the results of the tests e some screenshoots of the GUI.

3.1 Tests

The following picture shows the outputs of the command **pytest**.

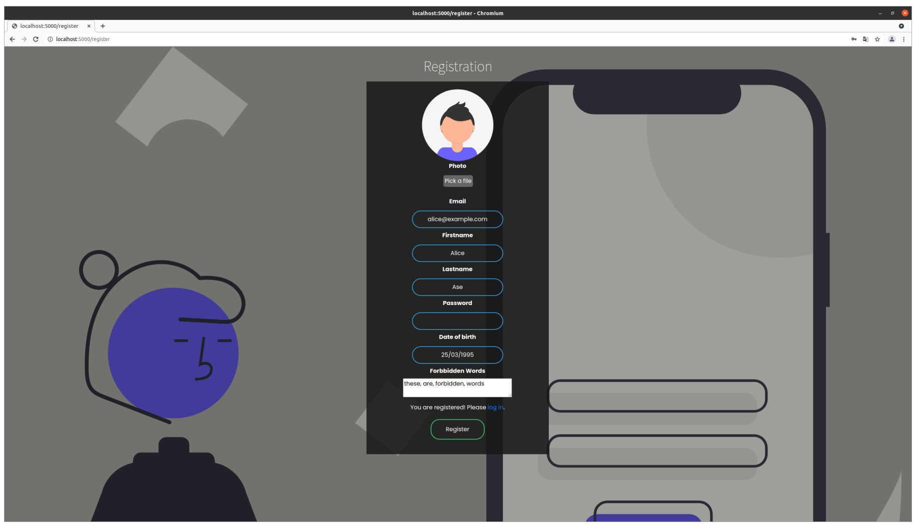


```
============================================================= test session starts ==============================================================
platform linux -- Python 3.9.7, pytest-6.2.5, py-1.11.0, pluggy-1.0.0
rootdir: /home/fede/Scrivania/HW2, configfile: setup.cfg, testpaths: monolith/classes/tests
plugins: cov-3.0.0
collected 28 items

monolith/classes/tests/test_app.py .                                                                                                     [  3%]
monolith/classes/tests/test_background.py .......                                                                                        [ 28%]
monolith/classes/tests/test_message.py .                                                                                                 [ 32%]
monolith/classes/tests/test_messages.py ........                                                                                         [ 60%]
monolith/classes/tests/user_test.py ...........                                                                                          [100%]

----------- coverage: platform linux, python 3.9.7-final-0 -----------
Name                          Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------------------------
monolith/app.py                  51      1      8      1    97%   65
monolith/auth.py                 18      7      4      1    55%   11-17, 23->25
monolith/background.py          131      8     52      6    90%   54->52, 78-81, 99, 119->118, 214-216, 230
monolith/views/auth.py           38      5     12      2    86%   21, 46-49
monolith/views/home.py           14      1      4      1    89%   18
monolith/views/message.py       377     47    168     30    83%   38, 65-69, 127-132, 144-145, 254, 261, 275, 283->exit, 319->321, 332, 349, 354, 358, 366, 382
->381, 384, 399-402, 413, 416-419, 432-435, 457-460, 489->488, 512->516, 513->512, 546, 548-549, 591-592, 607, 612, 616-618, 622, 626
monolith/views/users.py         133      6     44      4    94%   20-24, 129->128, 192->exit, 202
-------------------------------------------------------------------------
TOTAL                          1360     75    308     45    92%

11 files skipped due to complete coverage.
Coverage HTML written to dir htmlcov

Required test coverage of 90% reached. Total coverage: 91.61%
```

Figure 1: Pytest

3.2 Register

The following picture shows the form that users can use to sign up for the service:



localhost:5000/register - Chromium

Registration

Photo

Email: alice@example.com

Firstname: Alice

Lastname: Ase

Password:

Date of birth: 25/03/1995

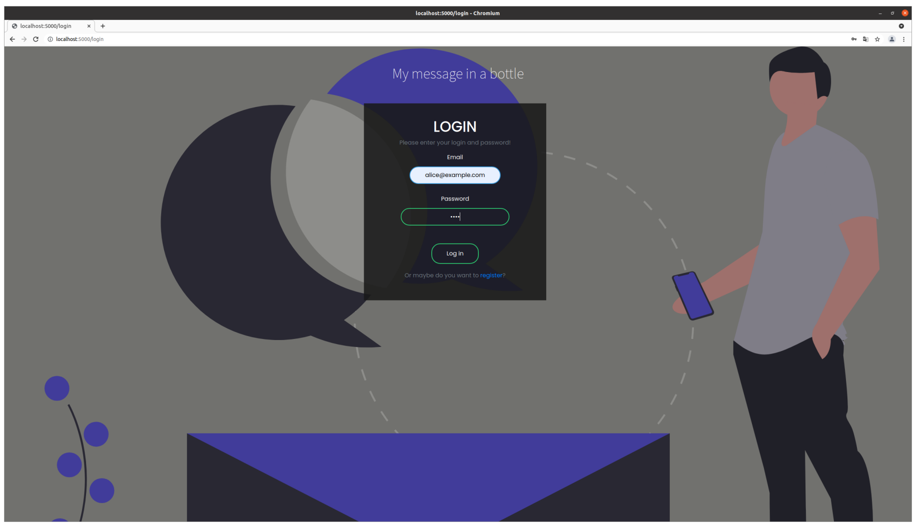
Forbbidden Words: these, are, forbidden, words

You are registered! Please log in.

Figure 2: Register form

3.3 Login

The following picture shows the form that users can use to login into the service:



localhost:5000/login - Chromium

My message in a bottle

LOGIN

Please enter your login and password!

Email: alice@example.com

Password: ••••

Or maybe do you want to register?

Figure 3: Login form

3.4 Dashboard

The following picture shows the dashboard of the service. Here the user can delete his personal information, send a message, see his mailbox, play the lottery, log out and unsubscribe.

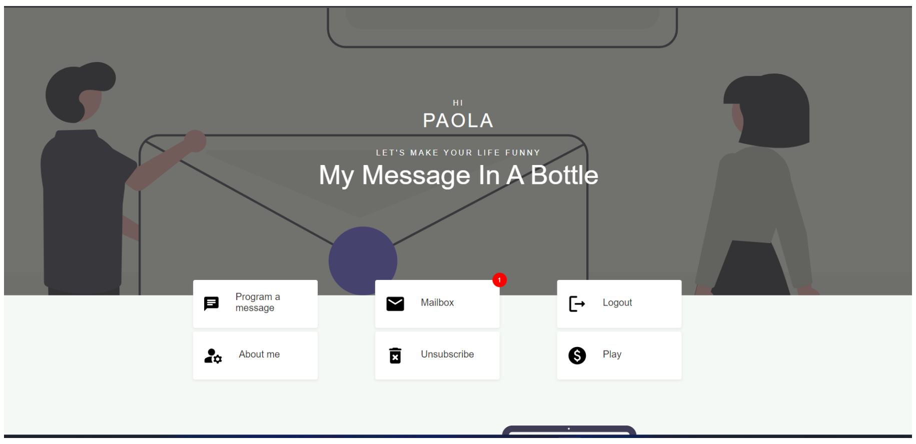
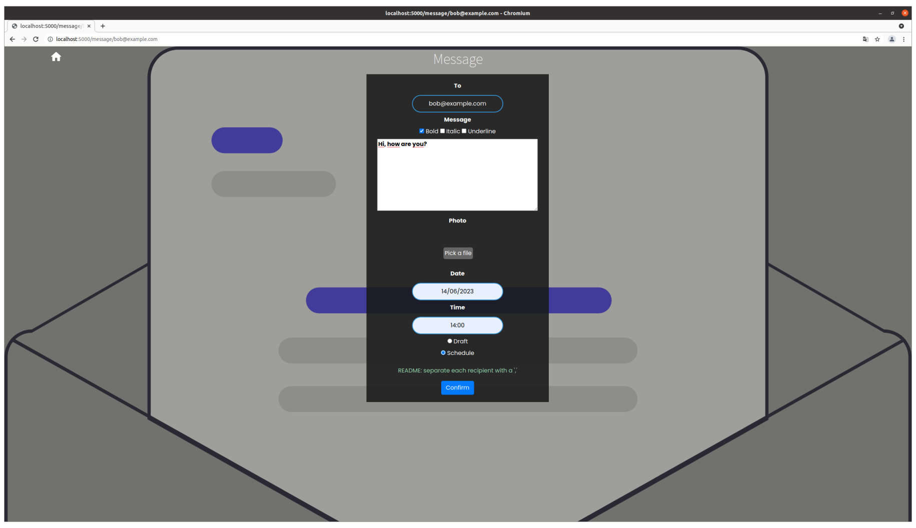


Figure 4: Dashboard

3.5 Message Form

The following image shows the form where the user can write a message. He can choose the style of the text and if he wants to schedule the message or save it as a draft.



To: bob@example.com

Message: [✓ Bold] Italic Underline

Hi, how are you?

Photo

Date: 14/06/2023

Time: 14:00

Draft [✓ Schedule]

README: separate each recipient with a ','

Figure 5: Message form

3.6 Mailbox

The following image shows the mailbox of the user. Here he can see his message divided into four categories: Inbox, Draft, Scheduled, and Sent. The user can also search messages by the search function.

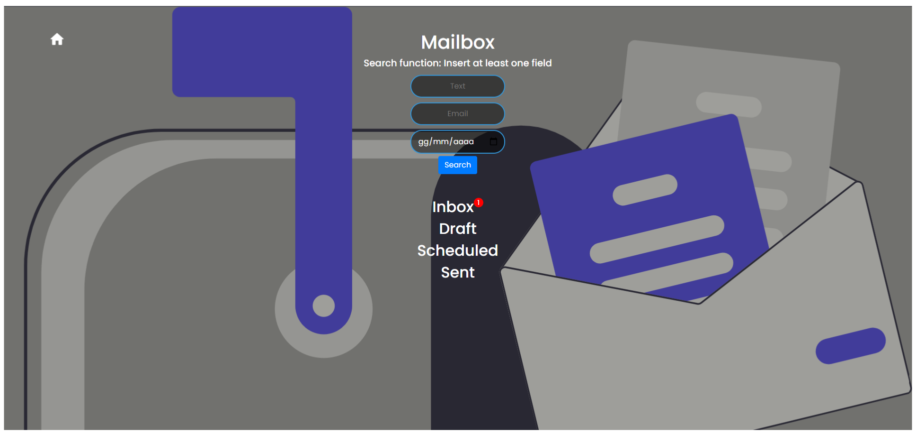
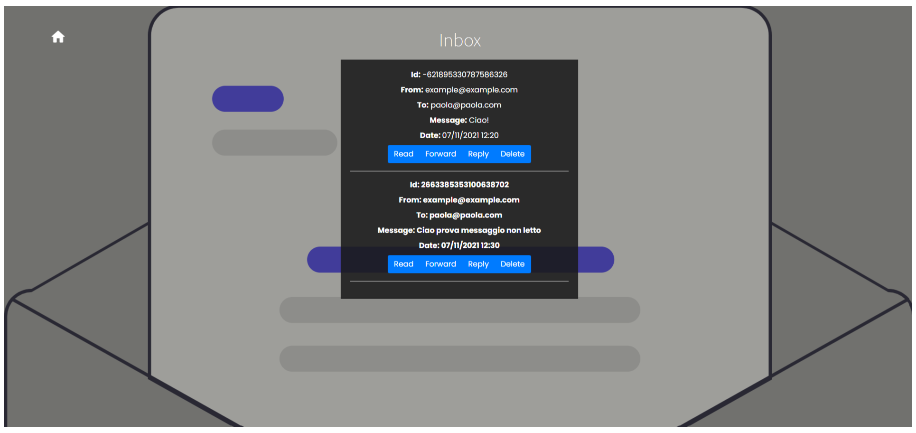


Figure 6: Mailbox

3.6.1 Inbox

The following image shows the inbox section of the user in which the messages to read are written in bold.



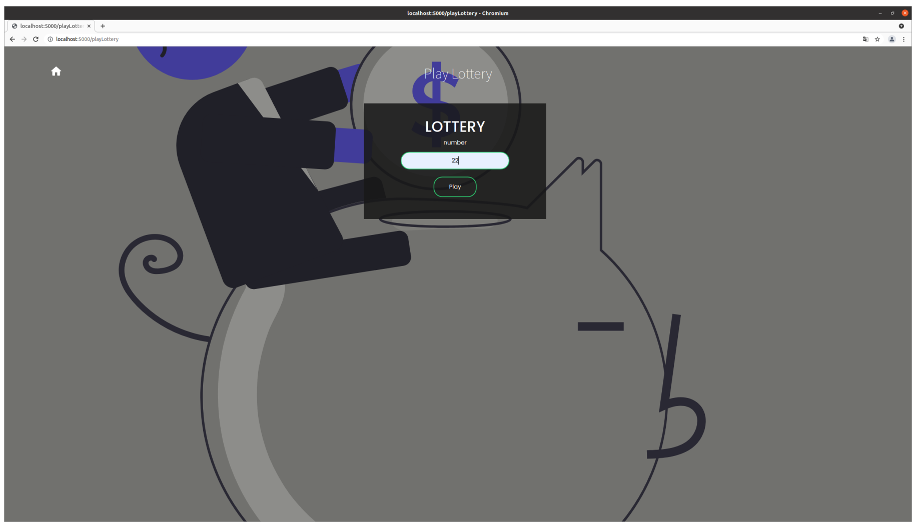
Id: -621895330787586326
From: example@example.com
To: paola@paola.com
Message: Ciao!
Date: 07/11/2021 12:20

Id: 2663385353100638702
From: example@example.com
To: paola@paola.com
Message: Ciao prova messaggio non letto
Date: 07/11/2021 12:30

Figure 7: Inbox

3.7 Lottery

The following image shows the lottery. Here the user can earn points by choosing a number. if the number is drawn within one month, the user will earn 100 points. With 150 points a user can delete a scheduled message. The user can change the number until the extraction.

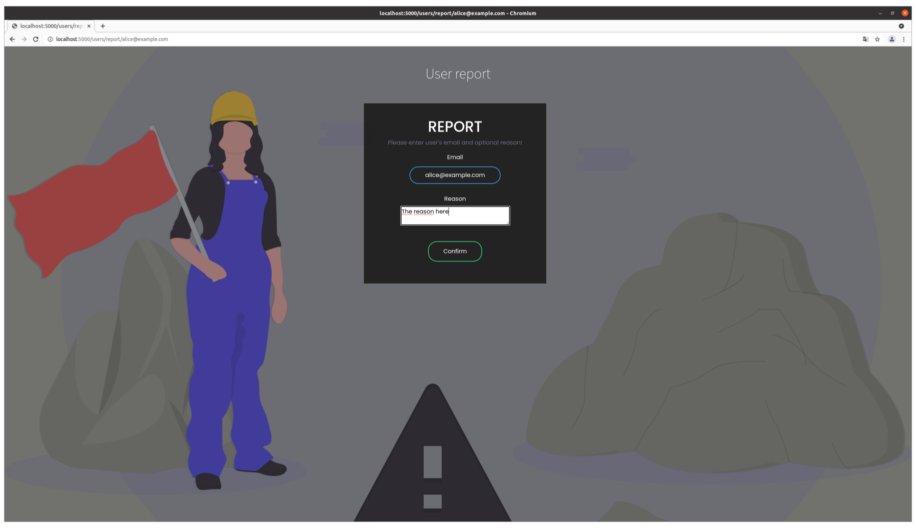

localhost:5000/playLottery - Chromium
localhost:5000/playLotter
localhost:5000/playLottery
Play Lottery
LOTTERY

number: 22

Figure 8: Lottery

3.8 Report

The following image shows the report form. The user can input the reason and report a user. With three reports the user is banned.


localhost:5000/users/report/alice@example.com - Chromium
localhost:5000/users/rep
localhost:5000/users/report/alice@example.com
User report
REPORT

Please enter user's email and optional reason!

Email: alice@example.com

Reason: The reason here

Figure 9: Report