Testing Plan of ERP-team12

Tyler Znoj 4005987
Killian Kelly 40014508
Abdul Sirawan 40074202
Adrien Kamran 40095393
Qandeel Arshad 40041524
Cedrik Dubois 40064475
Anthony Chraim 40091014
Matthew Giancola 40019131

A report submitted in partial fulfilment of the requirements of SOEN 390 Concordia University

Date 3/1/2021

TABLE OF CONTENTS

Testing Procedure

1. General Approach

The main approach which will be adopted for this project is Unit testing of modules. The application is a web-based application and it is developed using Django, which is a high-level Python web framework. Django has a test-execution framework and assorted utilities which allows us to simulate requests, insert test data, inspect the output of the application and generally verify if the code is doing what it should be doing. The preferred way to write tests in Django is using the **unittest** module built-in to the Python standard library.

For the purpose of this project, we will rely on the use of unittest throughout the entire development phase. Also, according to the requirements, the system shall have at least 50% test coverage in Controllers classes. The tool which will assess the test coverage of our application is called **django-nose**. It is a django coverage testing tool that displays the stats of our testing coverage.

1.1 Bugs Handling

I. **Usage of SDK**

Our first approach in handling bugs is using SDK (Software Development Kit) in the development process. Pycharm and Visual Studio Code are used for this project. They facilitate the creation of applications by having a compiler, and debugger. Their debugging system is extremely useful in identifying errors and bugs existing in the system. They also display a message indicating the source and the cause of the error.

II. **Communicating the bugs**

Bugs consume an incredible amount of time from the development team in efforts of resolving them. However, this wasted time can be put into a better usage if the bug is killed quicker. One of the approaches our team is adopting is communicating bugs with the team members first. There is a high chance that another team member has encountered the same bug and knows the solution, because we are eventually working on the same project and using similar development tools. Also, working collaboratively in resolving a bug is much faster than researching and reading documentation individually. Consequently, bugs are communicated with the rest of the team members upon their emergence. And whoever is experiencing the same bug, they partner up together and work towards resolving the bug.

III. Classifying the bugs

When a bug emerges, the team will conduct a meeting to discuss its severity and classify it whether one of the following:

a) **High-severity**

The quality assurance team shall evaluate the bug to high-severity if it seems that it hinders the functionality of the program. The program must work in compliance with the functional requirements. Therefore, when a bug occurs during the development of a feature from the functional requirements, it should be mitigated promptly and perform unit testing to verify the functionality of the program.

b) **Medium-severity**

The quality assurance team shall evaluate the bug to medium-severity if it impacts the performance and security of the program. After implementing a functional requirement feature, the QA team will analyze the security and performance of the program and ensure that it is in accordance with the requirements. Thus, any bug relevant to those two non-functional requirements.

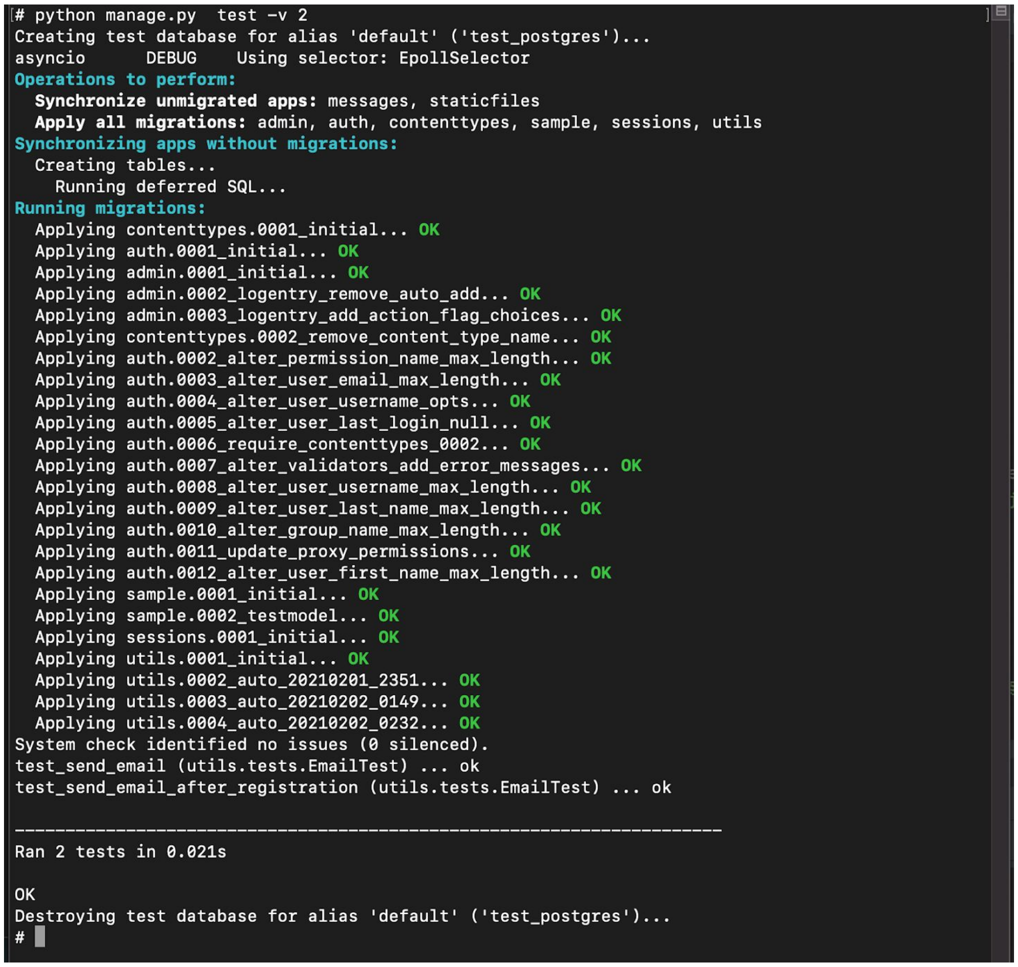
c) **Low-severity**

The quality assurance team shall evaluate the bug to low-severity if it is related to the UI of the application. Any bugs relevant to the design of the application will be fixed before submitting each sprint. Therefore, it is left until the end to be resolved after fixing bugs related to the functional and non-functional requirement.

2. Testing Approach

2.1 Unit Testing

The application shall be composed of many unit tests that test most of the functions in many modules, especially the controllers modules which should have at least 50% test coverage. The plan is that every developer has to write unit tests for the functionalities that he/she implemented, before pushing the changes to the repository. Later on, those unit tests will be evaluated by 2 selected members representing our quality assurance team. The test platform used will be the Django included test framework using the unittest module built-in the Python standard library. Below is an example of a test run using the tool:

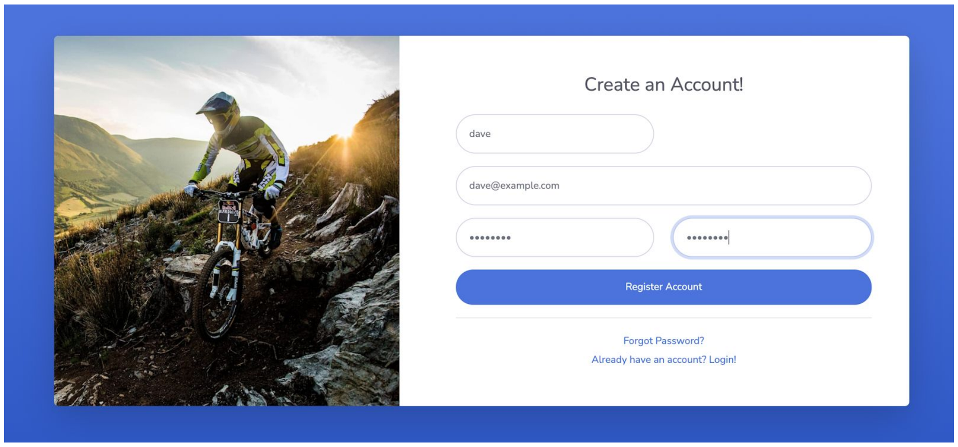

```
# python manage.py  test -v 2
Creating test database for alias 'default' ('test_postgres')...
asyncio      DEBUG    Using selector: EpollSelector
Operations to perform:
  Synchronize unmigrated apps: messages, staticfiles
  Apply all migrations: admin, auth, contenttypes, sample, sessions, utils
Synchronizing apps without migrations:
  Creating tables...
    Running deferred SQL...
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sample.0001_initial... OK
  Applying sample.0002_testmodel... OK
  Applying sessions.0001_initial... OK
  Applying utils.0001_initial... OK
  Applying utils.0002_auto_20210201_2351... OK
  Applying utils.0003_auto_20210202_0149... OK
  Applying utils.0004_auto_20210202_0232... OK
System check identified no issues (0 silenced).
test_send_email (utils.tests.EmailTest) ... ok
test_send_email_after_registration (utils.tests.EmailTest) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.021s

OK
Destroying test database for alias 'default' ('test_postgres')...
# 
```

2.2 Continuous Integration Test

To ensure that new features introduced do not affect the current state of the application, continuous integration tests will be performed on each pull request before it is reviewed. This will be done using CircleCi, a free integration testing platform. This will be connected to the github repository and will set up the correct environment necessary and perform all unit tests of the application.

2.3 Email Test

For functionality concerning emails, Mailtrap will be used. It provides a service for the safe testing of emails sent from the development and staging environments. Mailtrap catches the email in a virtual inbox to analyse the proper functionalities concerning emails. It provides a SMTP server and configuration necessary for this. It will also be used to render any HTML email to check if the proper rendering and check the email template for responsiveness. Mailtrap can also be used in the automatic continuous integration tests with their API. An example is shown below:


Create an Account!

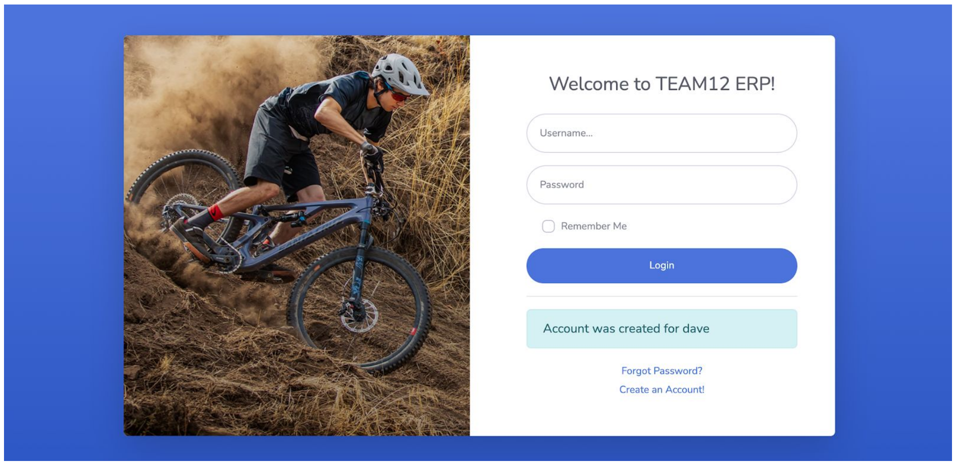
dave

dave@example.com

••••••••

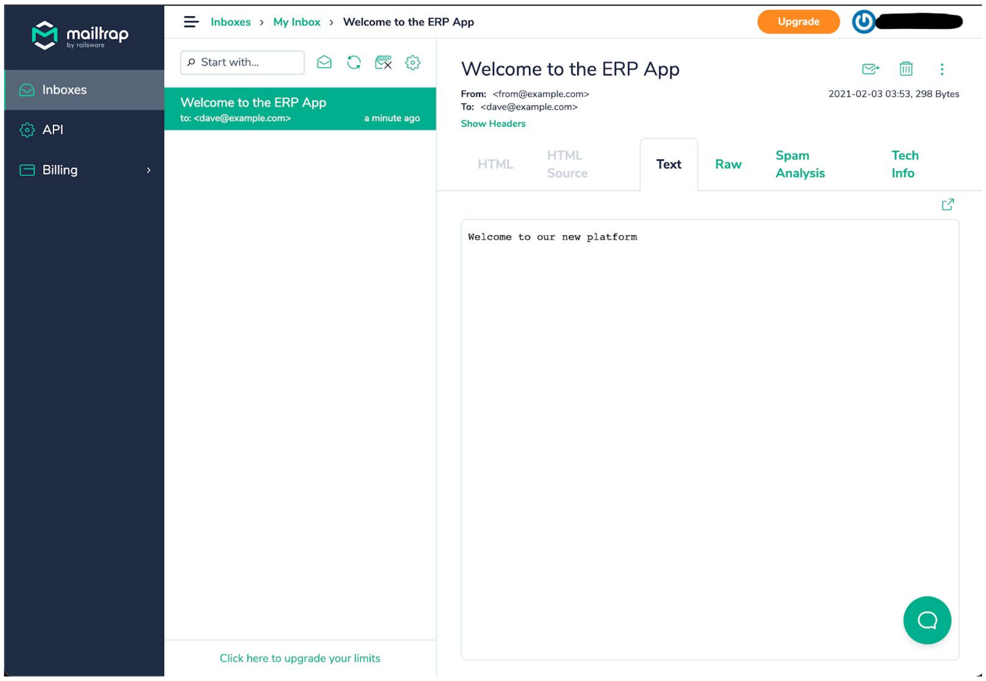
••••••••

A user named dave registers for an account.


Welcome to TEAM12 ERP!

Account was created for dave

dave successfully creates a new account.



[redacted]

Welcome to the ERP App
to: <dave@example.com> a minute ago

Welcome to the ERP App

From: <from@example.com>
To: <dave@example.com>

2021-02-03 03:53, 298 Bytes

```
Welcome to our new platform
```

An email is sent to his email to welcome him.

3. Communication Approaches

3.1 Weekly meeting

The QA assurance team which is composed of 2 selected members will perform a weekly meeting between each other to discuss the code and the tests submitted by the development team. The QA assurance team will start with the controllers classes as they pose the greatest risk because they control the functionality of the program. The QA team shall start the process by performing coverage tests for all controllers modules and evaluate the states. If a controller module has lower than 50% coverage, it is investigated further to decide on different tests to test the functionality of the module. The missing unit test is added and ensured that major functionalities of the controller modules are tested exhaustively.

Moreover, the QA team will assess the tests written for the rest of modules and evaluate if they are efficient and necessary. In the next scheduled meeting of the development team, the QA team will be assigned a slot of time in the meeting to present their findings to the other team members.