Czech Technical University in Prague

Faculty of Electrical Engineering and Computer Science

Date: 26.5.2023

Upgrade Of Server Solution

By Afandizada Azad

Supervisor: doc.Ing. Daniel Novak, Ph.D

Department: Electrical Power Engineering

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Afandizada Azad**	Personal ID number:	**498096**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Electrical Power Engineering**		
Study program:	**Electrical Engineering and Computer Science**		

II. Bachelor's thesis details

Bachelor's thesis title in English:

Upgrade of Server Solution

Bachelor's thesis title in Czech:

Aktualizace servového řešení

Guidelines:

The student will upgrade the existing server-side application. The present application runs on Django 1.8/Python 2.7 with several dependencies no longer maintained/available for the current version of python and Django framework. The application must be upgraded to new supported versions of packages where available and use suitable replacement otherwise. The application fulfils two core functionalities:

1. User database with an engine driving each user's run through a smoking cessation programme. The user interacts with the system using a mobile app, which uses the system via an API.
2. Administration interface allowing for design and maintenance of the programme.

Tasks:

1) Study and understand the existing project. Determine which packages can be upgraded and which must be replaced.
2) Perform upgrades and replacements.
3) Verify that all functionality is preserved. Perfrom functional tests.
4) Perform speed and memory tests to ensure the high quality of the system upgrade.

Bibliography / sources:

[1] H. Brendryen, P. Kraft, and H. Schaalma, "Looking Inside the Black Box: Using Intervention Mapping to Describe the Development of the Automated Smoking Cessation Intervention 'Happy Ending'," The Journal of Smoking Cessation, vol. 5, no. 1, pp. 29-56, Jun. 2010.
[2] H. Brendryen, F. Drozd, and P. Kraft, "A digital smoking cessation program delivered through internet and cell phone without nicotine replacement (happy ending): randomized controlled trial.," Journal of medical Internet research, vol. 10, no. 5, p. e51, Jan. 2008.
[3] Kulhánek A., Gabrhelík R., Novák D. &amp; Brendren H. (2018). eHealth intervention for smoking cessation for Czech tobacco smokers: Pilot study of user acceptance. Adiktologie, 18(2).

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Daniel Novák, Ph.D.   Department of Theoretical Computer Science FIT

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **09.02.2023**   Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Ing. Daniel Novák, Ph.D.	doc. Ing. Zdeněk Müller, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Acknowledgment:

First of all, I would like to thank my supervisor doc.Ing Daniel Novák who gave me the opportunity to work on this project and Jindřich Prokop who guided me during my work. Special thanks to my family and friends who supported me during my studies at the university.

Declaration:

I hereby declare that this bachelor’s thesis is the product of my own independent work and that I have clearly stated all information sources used in the thesis according to Methodological Instruction No. 1/2009 – On maintaining ethical principles when working on a university final project, CTU in Prague.

Date: 26.05.2023 Signature

Abstract:

This diploma thesis is focused on applying modern software engineering technologies to improve web application for quitting smoking. We will discuss which technologies we are using in this web application and then how we are going to integrate to the newer version. Mainly we will focus on the backend of the application written on Python web framework called Django [1]. Currently, our main web application is run on Django 1.9 and is called serafin, but there is a newer version which is run on Django 3.2.16 called serafin3 and is still developing. Our goal is to upgrade serafin3 and release it to production. First, we are going to fix a bug that occurred in the newer version of the web application then read release notes for Django 4.0 and investigate changes that were made to update serafin3. Also, transfer new changes that were made by developers in the serafin to serafin3.

Keywords: *Python web server, upgrade of server, Django, web application, Docker*

Contents:

Chapter 1

Introduction

Nowadays, there are a lot of people who wish to quit such a bad habit as smoking cigarettes, but it is not so easy as it seems. Luckily in our modern world there are applications which may help you to quit smoking, and we are going to upgrade one of those applications. My motivation behind writing this bachelor thesis on this topic is that most of my surroundings, family, friends are smokers. This habit negatively affects their health and that is why I want to help people to refuse from cigarettes. First, we need to get acquainted with technologies that our web application uses. The application was developed using python web framework called Django. It helps people to quit smoking by providing resources and support. It has a user-friendly interface and offers features such as tracking progress, individual program for each user and more. Django framework is popular nowadays as it is fast, simple, and secure. However, it is still underqualified and must be developed as currently, there is a bug on the admin side that must be fixed. Additionally, the application needs to be integrated from an old version of Django to a newer version for better performance and additional features.

1.1 Goal achievement

To overcome the issues with our application, the first step is to understand the codebase and learn how the application is built. This will involve studying the existing code and identifying where the bug is located. Once the bug will be found, we will work on fixing it by testing different solutions and making necessary changes to the code. The next step is to integrate the application from the older version of Django to the newer version. To do this, we need to study the differences between the two versions and understand how they impact the application. This will involve researching the new features and changes in the newer version of Django, and how they can be implemented in our application. It is crucial to test the application to ensure that it is functioning correctly after the integration. To aid in this process it is necessary looking for relevant tutorials, documentation, and examples of how to migrate from an older version to a newer version. Once we have a solid understanding of the codebase and the differences between the versions of Django, we can start effectively integrating the application to the newer version. Next step will be transferring changes from the current working web application to our new one serafin3. At the end we will feed our database and test all the changes that were made to make sure that everything is working correctly. This will ensure that our application is running smoothly and providing the best possible experience for users.

Summary of Goals:

- Fixing the bug
- Updating Django version
- Transferring commits from older version

Chapter 2

Structure and Technologies used

First, we need to understand the structure of the web application and which technologies are used for running it. The web application is stored on **GitLab** [2], a web-based Git repository manager that provides source code management and continuous integration deployment capabilities. GitLab allows developers to collaborate on code and manage projects in a centralized location.

Git [3] - is a distributed version control system that allows developers to track changes in their code and collaborate with other developers on a project.

By using GitLab to store our application, it is easy to track changes in the code and manage issues. It will also be possible to set up continuous integration and continuous deployment workflows, which will automatically build, test, and deploy the application when new code is pushed to the repository.

Docker [4] - is a platform that enables developers to easily deploy, run and manage applications in containers.

Containers [5] - are isolated environments that include all the necessary dependencies, libraries, and configurations needed for an application to run.

Docker allows developers to run their applications in a consistent environment, regardless of the host system they are running on. This means that the application will run the same way on a developer's local machine, a test server, or a production server.

In the context of fixing the bug and integrating our application, Docker can be very helpful. By using Docker, we can run the application locally in a containerized environment that mimics the production environment. This will make it easier to spot the bug and try different solutions because the application will be running in the same environment as it would in production.

Django has essential components of web application such as models, views, templates, and URLs [6]. The Figure 1 illustrates how Django web application is working.

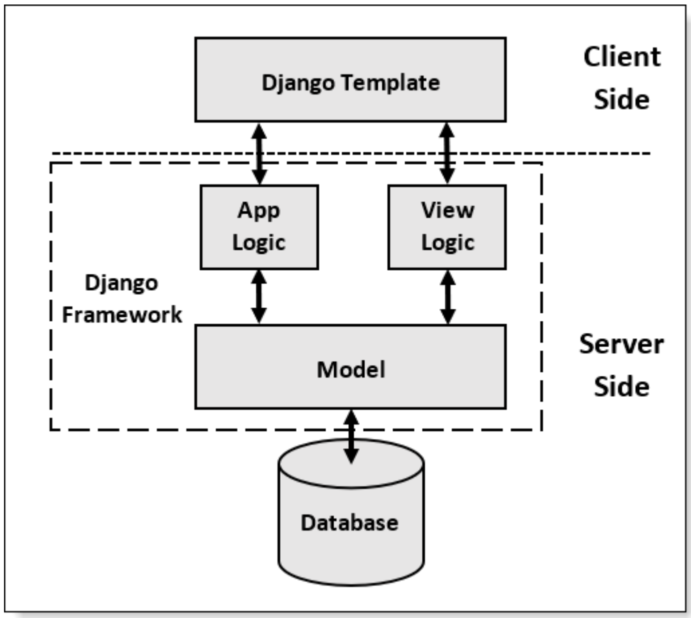


Figure 1: Backend architecture [7]

Models – are classes that represent the database table. They define fields or attributes of the table such as name, age, or email of the user.

Views – are functions or classes that take a request from the user to the application and return a response after performing some operations such as showing a content on the page or redirecting to another page. Views are defined in the file “views.py”.

Templates – define the structure and layout of the HTML pages that are shown to the user. They can be written in HTML, CSS or using Django’s own templating language.

URL’s – web addresses that users enter to the browser, and they redirect them to the specific page. They are mapped to the views in the file called “urls.py” using URL configuration.

As we discussed the technologies that are used for running the application, we can get to the concept. Our application allows us the creation of addiction treatment plan which is spread out over several days. The idea is based on the Decision Tree algorithm which helps to make predictions and decisions. Each user gets so called sessions which helps him to quit smoking.

Session – refers to the part of our decision tree which is defined to be our current program. In each session there is a subtree which belongs to the specific day and is made up of a group of actions. Each subtree consists of nodes and edges.

Node - represent different actions which admin can take such as sending email.

Edges – connects each node and executes action in the node depending on the condition. Otherwise skips this node.

In our application, each session includes a questionary. The questionary is designed to gather information about the user's status and their progress in quitting smoking. Based on the answers provided by the user, the session is generated specifically for them. This allows the application to provide a personalized and tailored experience for each user, ensuring that the resources and support provided are most relevant and effective for them in their quit journey. Addition to that in the backend there is a graphical user interface which allows to see and edit sessions for each user.

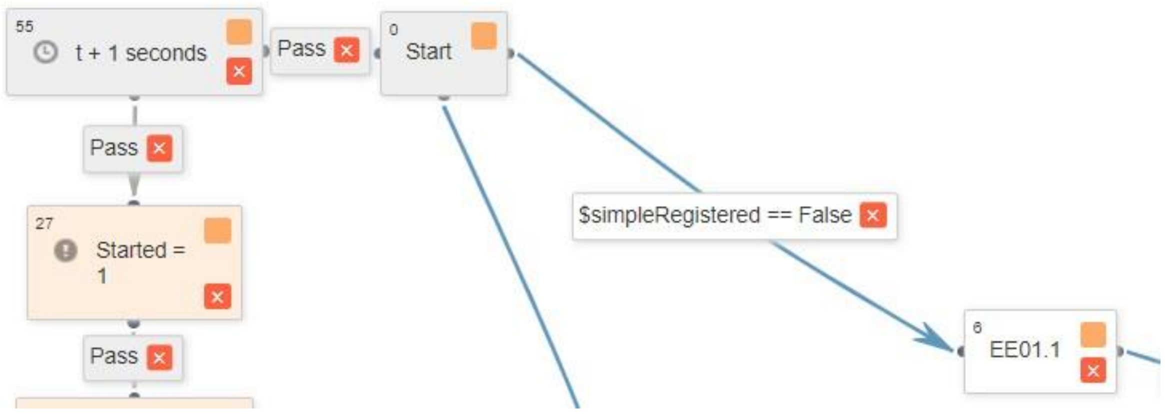
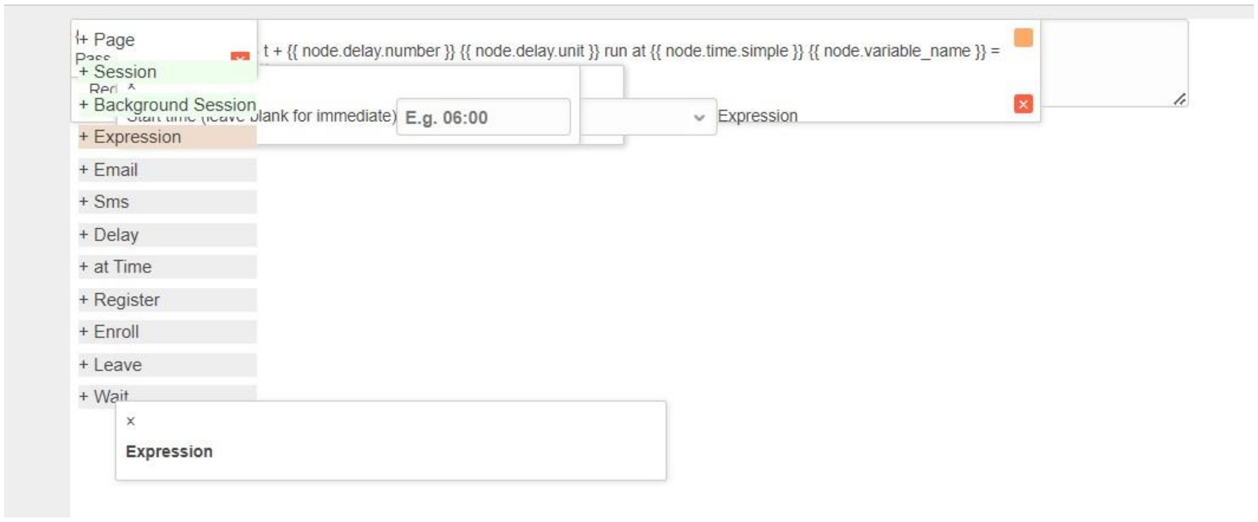


Figure 2: User interface in the backend.

As we can see this is our decision tree with nodes and edges. From the Figure 2 we observe that node with expression ‘pass’ directly go to the next node skipping the previous one. Nodes with orange color are used for assigning a value, while grey colored for sending email or push notification.

Now we got the information about our web application and technologies that it is using, so let us examine the bug itself.



t + {{ node.delay.number }} {{ node.delay.unit }} run at {{ node.time.simple }} {{ node.variable_name }} =

Start time (leave blank for immediate)

Figure 3: Illustrates the issue.

From Figure 3, it is evident that the sessions in the user interface are disorganized and not connected by edges. This is an issue that is present in the web application that is running on Django 3.2.16 This is a huge problem as admin cannot take any actions and sessions are not generated correctly. That is what we are going to fix by studying the code and trying different approaches for the problem. From the first look it can be assumed that the problem is in the template or static files which contain JavaScript and CSS. Another guess that as our project is using docker for running itself and installing all the necessary components such as libraries it might not be loading or installing them correctly. We will discuss each of these approaches and solutions in the next chapter.

Chapter 3

Upgrading the server solution

3.1 Setting up the project:

Before working on the project, it is necessary to clone repository from a GitLab. This can be done firstly by installing git on our machine and then running **git clone <url of our repository>** command in the command line. After cloning the repository, we can see that all necessary files are on our machine.

In our project there are files called **docker-compose.yml** and **docker-compose.override.yml**.

docker-compose.yml - is a YAML file that defines how Docker containers should be built and run. It includes details such as container names, images to be used, environment variables and ports to be exposed.

docker-compose.override.yml - is a separate YAML file that is used to override or extend the configuration specified in docker-compose.yml. It allows for additional configuration settings or modifications to be made without having to modify the original docker-compose.yml file.

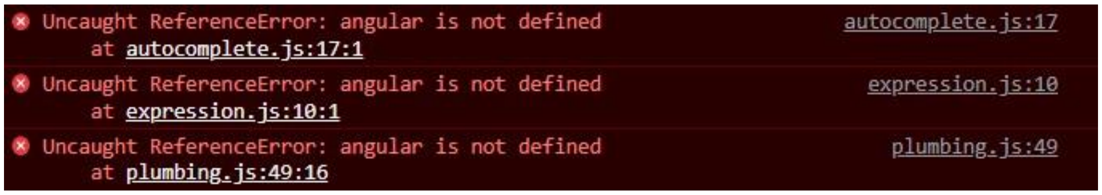
Before running our container, we need to build it. This can be done with the following command in the command line while being in the project directory **docker-compose build**. After building it we can finally run our container with the next command **docker-compose -f docker-compose.override.yml up**.

The -f option is used to specify the file to use as the Docker Compose configuration file. In this case, docker-compose.override.yml is the file being used. The up command is used to create and start containers for all the services defined in the Docker Compose configuration.

Then a superuser needs to be created to be able log in and see admin panel in our Django application. To do this it is necessary to get the id of the container and run the command inside of it. While docker container is running we open another command line and run **docker container ls** command to see all the containers that are running. After finding container's id we run **docker exec -it container_id python manage.py createsuperuser**. The exec command will allow to run necessary commands inside the container while it keeps running and Django command **manage.py createsuperuser**. After this it asks to set up the email and password for login. Finally, we are ready to go and start working with our project and exploring it.

3.2 Testing procedure and bug fixing

Now as we set up the project, we can work on the code itself. From the Figure 3 as I mentioned before the problem might be in the templates or static files most probably in JavaScript. We can investigate this by using console in our web browser.



```
Uncaught ReferenceError: angular is not defined                    autocomplete.js:17
    at autocomplete.js:17:1
Uncaught ReferenceError: angular is not defined                      expression.js:10
    at expression.js:10:1
Uncaught ReferenceError: angular is not defined                        plumbing.js:49
    at plumbing.js:49:16
```

Figure 4: Logs in the console

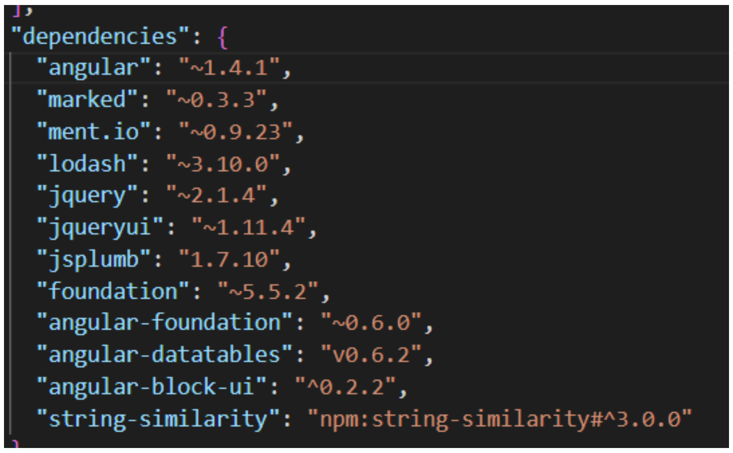
On Figure 4 we can see the following errors which tell us that there is a common problem in multiple files where “angular” is not defined.

Angular [8] – is JavaScript framework which is written in TypeScript.

Now we need to check those files and see what the problem is. Investigation of those files could not find anything what would cause the error. It is stated that the angular is not defined and the assumption is that the library angular is not installed properly and that is the reason why JavaScript files cannot see it. It was crucial to find the file where all the dependencies were written and such file was bower.json.

Bower [9] – is a file that contain libraries and dependencies for staticfiles.

From the Figure 5 below we see that the angular library is in the bower file, however our JavaScript files cannot see it.



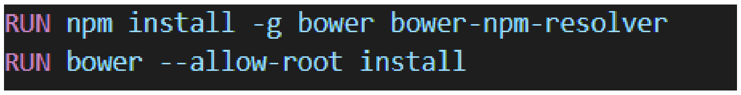
```
"dependencies": {
  "angular": "~1.4.1",
  "marked": "~0.3.3",
  "ment.io": "~0.9.23",
  "lodash": "~3.10.0",
  "jquery": "~2.1.4",
  "jqueryui": "~1.11.4",
  "jsplumb": "1.7.10",
  "foundation": "~5.5.2",
  "angular-foundation": "~0.6.0",
  "angular-datatables": "v0.6.2",
  "angular-block-ui": "^0.2.2",
  "string-similarity": "npm:string-similarity#^3.0.0"
```

Figure 5: Bower file

This issue might occur due to the Dockerfile too as it is not installing dependencies properly.

Dockerfile [10] – is a script of the user's commands which should be run in the command line to assemble an image.

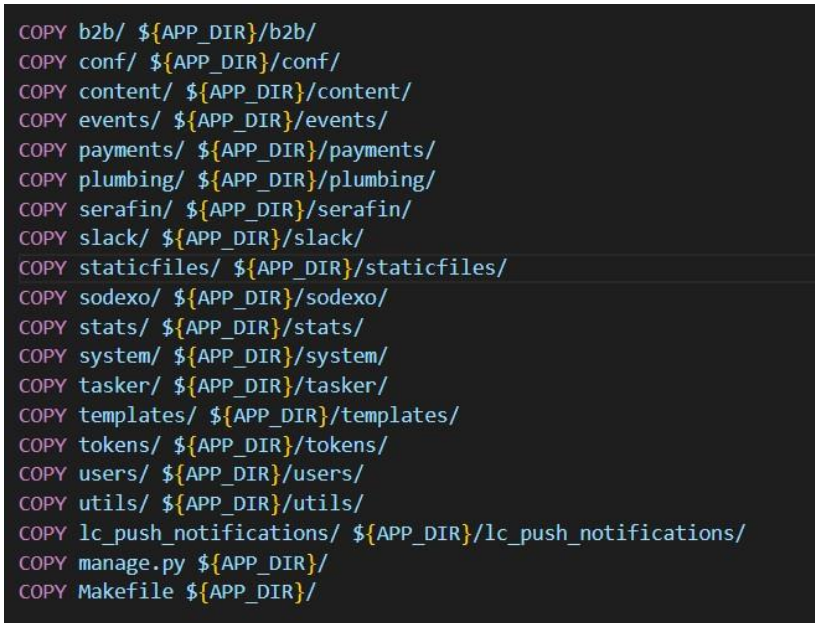
Returning to the web browser's console, it can be seen that there are some missing files that are required such as lib folder where libraries should be located. This means that the dependencies in bower are not installed. Now we should investigate the Dockerfile to see what causes this issue. First thought is that there is simply no command to install the bower, so let us see if this is true. Going to the Dockerfile we can see that there is a command which installs the bower file.



```
RUN npm install -g bower bower-npm-resolver
RUN bower --allow-root install
```

Figure 6: Commands for installing bower.json

Hence, the problem might be that something blocks the installation of our bower. What gets into eyes are lines where copying the folders are done and there is our staticfiles folder.



```
COPY b2b/ ${APP_DIR}/b2b/
COPY conf/ ${APP_DIR}/conf/
COPY content/ ${APP_DIR}/content/
COPY events/ ${APP_DIR}/events/
COPY payments/ ${APP_DIR}/payments/
COPY plumbing/ ${APP_DIR}/plumbing/
COPY serafin/ ${APP_DIR}/serafin/
COPY slack/ ${APP_DIR}/slack/
COPY staticfiles/ ${APP_DIR}/staticfiles/
COPY sodexo/ ${APP_DIR}/sodexo/
COPY stats/ ${APP_DIR}/stats/
COPY system/ ${APP_DIR}/system/
COPY tasker/ ${APP_DIR}/tasker/
COPY templates/ ${APP_DIR}/templates/
COPY tokens/ ${APP_DIR}/tokens/
COPY users/ ${APP_DIR}/users/
COPY utils/ ${APP_DIR}/utils/
COPY lc_push_notifications/ ${APP_DIR}/lc_push_notifications/
COPY manage.py ${APP_DIR}/
COPY Makefile ${APP_DIR}/
```

Figure 7: Commands for moving sources.

After analyzing the Dockerfile a concern was raised regarding the order of installing Bower and copying the required files. The existing approach of installing Bower before copying the files looked unconventional and could potentially lead to issues during the build process. Further investigation revealed that this approach prevented the Bower command from executing correctly, as it requires the presence of the Bower file in the current working directory during the build process. As the files were not present in the container before copying, Bower was unable to install the required packages. To fix this issue we can change the order of the installation process, so that the files were copied into the container before installing Bower. This will ensure that the required files are present during the build process, allowing Bower to install the required packages. After implementation of this change, the build process was successful, and the issue was resolved. With the fix of the bug, our next goal becomes integrating web application to the newer version of Django.

3.3 Integration to the newer version

To update our web application, it is crucial to have a look to the Django documentation [11] and see what the changes are.

The good question we may ask ourselves is why do we need to upgrade to the latest Django version? There are several reasons for this:

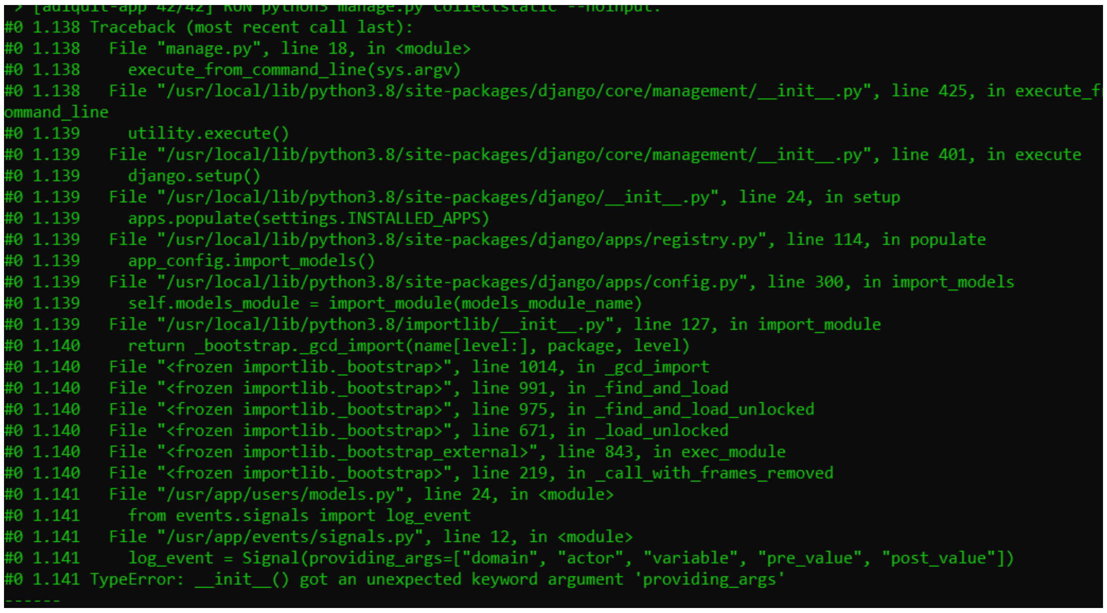
- Some bugs are fixed
- New features and improvements
- Older versions of Django are no longer supported

What do we need to do before upgrading it:

- Read updating notes for each version from our current one
- Check compatibility with Python version and each dependencies with Django 4.0
- Look at the deprecation timeline for the relevant versions.

Typically, dependencies for a Python project are listed in a file called requirements.txt. However, in our case, we are using Pipenv to manage our dependencies, so the information is in the Pipfile. At the beginning of reading documentation for Django 4.0, it appears that this version supports only Python 3.8 or higher, so we need to be sure that our Python version is compatible.

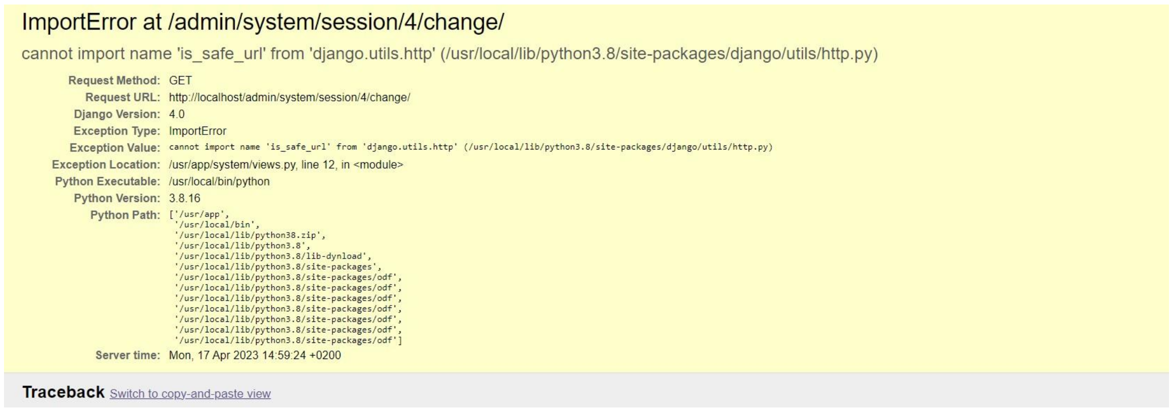
Fortunately, in our Pipfile, it is already specified that our Python version should be 3.8, so we should be good to go. Now, as after the change of Django version in Pipfile it is important to run pipenv and lock the changes to update the Pipfile.lock. Now, let us run the project and we will continue based on our errors. The Figure 8 is the first message we get as we run the project.



```
> [adiquit-app 42/42] RUN python3 manage.py collectstatic --noinput:
#0 1.138 Traceback (most recent call last):
#0 1.138   File "manage.py", line 18, in <module>
#0 1.138     execute_from_command_line(sys.argv)
#0 1.138   File "/usr/local/lib/python3.8/site-packages/django/core/management/__init__.py", line 425, in execute_f
ommand_line
#0 1.139     utility.execute()
#0 1.139   File "/usr/local/lib/python3.8/site-packages/django/core/management/__init__.py", line 401, in execute
#0 1.139     django.setup()
#0 1.139   File "/usr/local/lib/python3.8/site-packages/django/__init__.py", line 24, in setup
#0 1.139     apps.populate(settings.INSTALLED_APPS)
#0 1.139   File "/usr/local/lib/python3.8/site-packages/django/apps/registry.py", line 114, in populate
#0 1.139     app_config.import_models()
#0 1.139   File "/usr/local/lib/python3.8/site-packages/django/apps/config.py", line 300, in import_models
#0 1.139     self.models_module = import_module(models_module_name)
#0 1.139   File "/usr/local/lib/python3.8/importlib/__init__.py", line 127, in import_module
#0 1.140     return _bootstrap._gcd_import(name[level:], package, level)
#0 1.140   File "<frozen importlib._bootstrap>", line 1014, in _gcd_import
#0 1.140   File "<frozen importlib._bootstrap>", line 991, in _find_and_load
#0 1.140   File "<frozen importlib._bootstrap>", line 975, in _find_and_load_unlocked
#0 1.140   File "<frozen importlib._bootstrap>", line 671, in _load_unlocked
#0 1.140   File "<frozen importlib._bootstrap_external>", line 843, in exec_module
#0 1.140   File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
#0 1.141   File "/usr/app/users/models.py", line 24, in <module>
#0 1.141     from events.signals import log_event
#0 1.141   File "/usr/app/events/signals.py", line 12, in <module>
#0 1.141     log_event = Signal(providing_args=["domain", "actor", "variable", "pre_value", "post_value"])
#0 1.141 TypeError: __init__() got an unexpected keyword argument 'providing_args'
------
```

Figure 8: Signal unexpected keyword provided.

The error message indicates an issue with the log_event signal, defined in the events.signals module, caused by an unexpected keyword argument "providing_args" being passed to the Signal class. It appears that this keyword argument was removed in newer versions of Django [12]. To address this, the log_event signal can be modified by removing the providing_args argument and using only Signal(). Another issue is the replacement of the ugettext_lazy function with the gettext_lazy function. This function translates the string into Unicode when it is accessed [13]. To resolve this issue, it is necessary to search through the project for the instances where the deprecated function was used and replace it with the newer version. Fortunately, the code editor enables to find and replace all instances of ugettext_lazy quickly. After completing these modifications, the Dockerfile has to be run and verified that our project started without any errors in the command line. Command line does not output any error however, during testing a website itself, a new issue appeared.



```
ImportError at /admin/system/session/4/change/
cannot import name 'is_safe_url' from 'django.utils.http' (/usr/local/lib/python3.8/site-packages/django/utils/http.py)

   Request Method: GET
      Request URL: http://localhost/admin/system/session/4/change/
   Django Version: 4.0
   Exception Type: ImportError
  Exception Value: cannot import name 'is_safe_url' from 'django.utils.http' (/usr/local/lib/python3.8/site-packages/django/utils/http.py)
Exception Location: /usr/app/system/views.py, line 12, in <module>
Python Executable: /usr/local/bin/python
   Python Version: 3.8.16
      Python Path: ['/usr/app',
                    '/usr/local/bin',
                    '/usr/local/lib/python38.zip',
                    '/usr/local/lib/python3.8',
                    '/usr/local/lib/python3.8/lib-dynload',
                    '/usr/local/lib/python3.8/site-packages',
                    '/usr/local/lib/python3.8/site-packages/odf',
                    '/usr/local/lib/python3.8/site-packages/odf',
                    '/usr/local/lib/python3.8/site-packages/odf',
                    '/usr/local/lib/python3.8/site-packages/odf',
                    '/usr/local/lib/python3.8/site-packages/odf',
                    '/usr/local/lib/python3.8/site-packages/odf',
                    '/usr/local/lib/python3.8/site-packages/odf']
      Server time: Mon, 17 Apr 2023 14:59:24 +0200

Traceback Switch to copy-and-paste view
```

Figure 9: Debug error page

In Figure 9, the standard debug error page provided by Django indicates that the is_safe_url method from the Django module could not be imported. This debug page is useful as it provides the information about what is wrong in our project. As it suggests it cannot import is_safe_url method from Django module. This method checks whether the given URL is safe or not [14]. This package allows other projects to use that same function without having to include the entire Django framework. After investigation of Django documentation, it was found out that the name of this method was changed to url_has_allowed_host_and_scheme. Consequently, the solution involves replacing the old method name with the updated one. After resolving the previously mentioned problem another error occurs which is displayed in Figure 10.

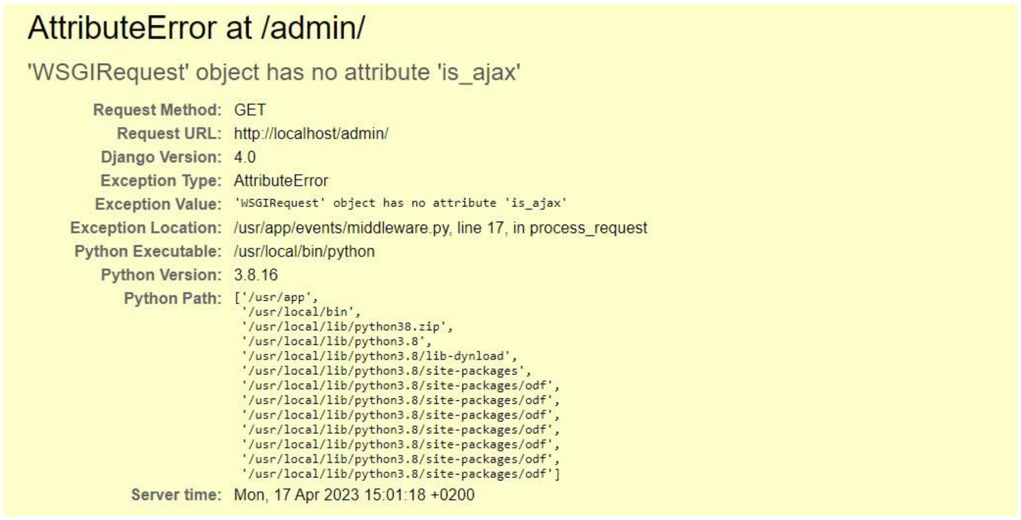

```
AttributeError at /admin/
'WSGIRequest' object has no attribute 'is_ajax'

    Request Method:  GET
       Request URL:  http://localhost/admin/
    Django Version:  4.0
    Exception Type:  AttributeError
   Exception Value:  'WSGIRequest' object has no attribute 'is_ajax'
Exception Location:  /usr/app/events/middleware.py, line 17, in process_request
 Python Executable:  /usr/local/bin/python
    Python Version:  3.8.16
       Python Path:  ['/usr/app',
                      '/usr/local/bin',
                      '/usr/local/lib/python38.zip',
                      '/usr/local/lib/python3.8',
                      '/usr/local/lib/python3.8/lib-dynload',
                      '/usr/local/lib/python3.8/site-packages',
                      '/usr/local/lib/python3.8/site-packages/odf',
                      '/usr/local/lib/python3.8/site-packages/odf',
                      '/usr/local/lib/python3.8/site-packages/odf',
                      '/usr/local/lib/python3.8/site-packages/odf',
                      '/usr/local/lib/python3.8/site-packages/odf',
                      '/usr/local/lib/python3.8/site-packages/odf',
                      '/usr/local/lib/python3.8/site-packages/odf']
       Server time:  Mon, 17 Apr 2023 15:01:18 +0200
```

Figure 10: Debug error page

In previous versions of Django method is_ajax was used to check whether the request is ajax.

AJAX [15] – is a technique used to update a content of a web page without refreshing the entire page.

After reviewing the code of our application, it can be observed that the is_ajax method is being utilized in a separate function that displays the session content on the backend. This function initially verifies if the user is authenticated or not. If the user is not authenticated, an unauthorized error is displayed on the page. Next, it checks if the request is an Ajax request, and if it is, another function is called to update the content on the page.

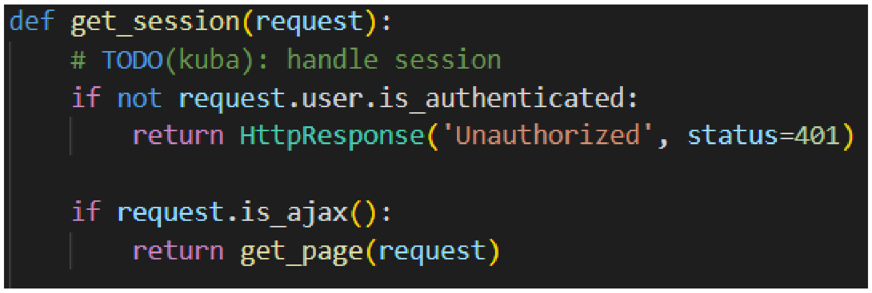
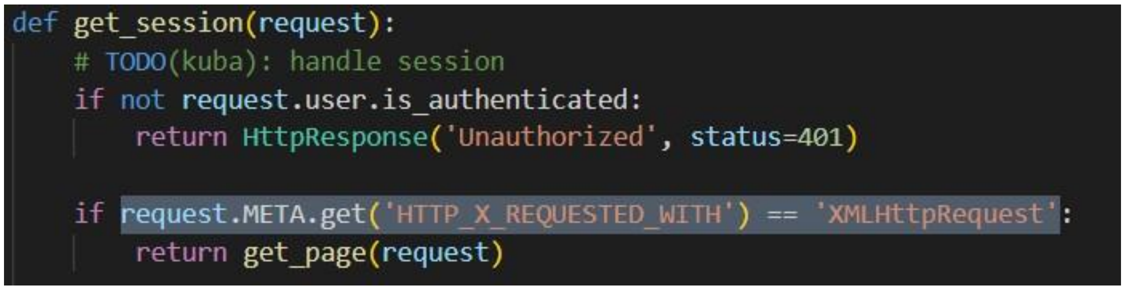

```python
def get_session(request):
    # TODO(kuba): handle session
    if not request.user.is_authenticated:
        return HttpResponse('Unauthorized', status=401)

    if request.is_ajax():
        return get_page(request)
```

Figure 11: Function for session generation

This method was removed in the later version of Django, so we must look up for the alternatives. After studying the release notes an alternative method was found which is shown in Figure 12.



```
def get_session(request):
    # TODO(kuba): handle session
    if not request.user.is_authenticated:
        return HttpResponse('Unauthorized', status=401)

    if request.META.get('HTTP_X_REQUESTED_WITH') == 'XMLHttpRequest':
        return get_page(request)
```

Figure 12: Updated method taken from release notes [16]

In our updated version our header X-Requested-With which provides information about the request is set to XMLHttpRequest by the browser to determine if the request was made via AJAX. This is because when the browser sends an AJAX request, it sets the X-Requested-With header to XMLHttpRequest, and Django's request.META dictionary stores HTTP headers in a case-insensitive manner.

After this change the log in page could be seen.

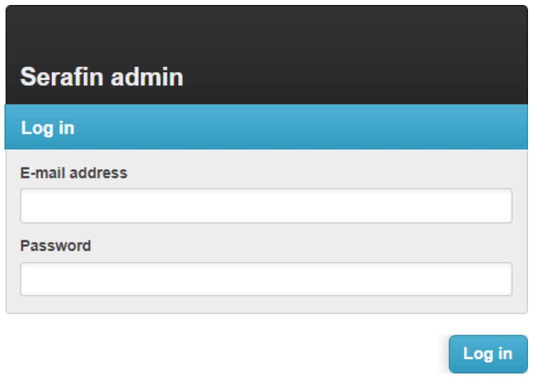
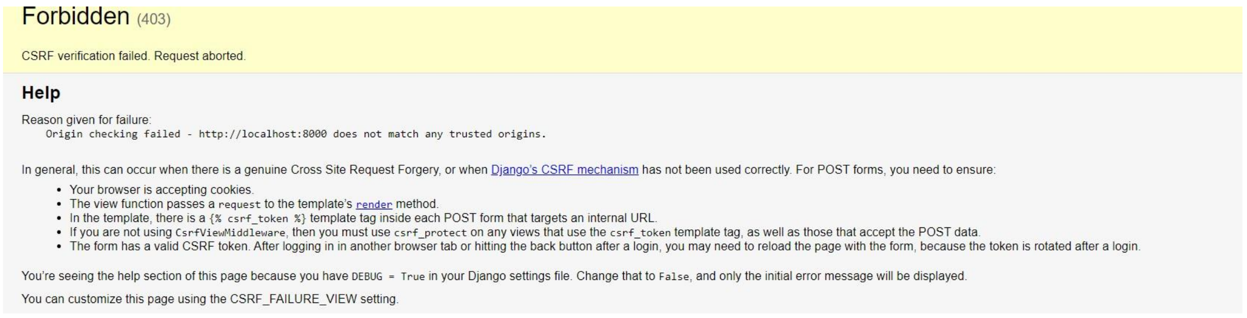


Figure 13: Login page to our web application

Unfortunately, after entering the credentials the following error occurs.



Forbidden (403)

CSRF verification failed. Request aborted.

Help

Reason given for failure:

```
    Origin checking failed - http://localhost:8000 does not match any trusted origins.
```

In general, this can occur when there is a genuine Cross Site Request Forgery, or when Django's CSRF mechanism has not been used correctly. For POST forms, you need to ensure:

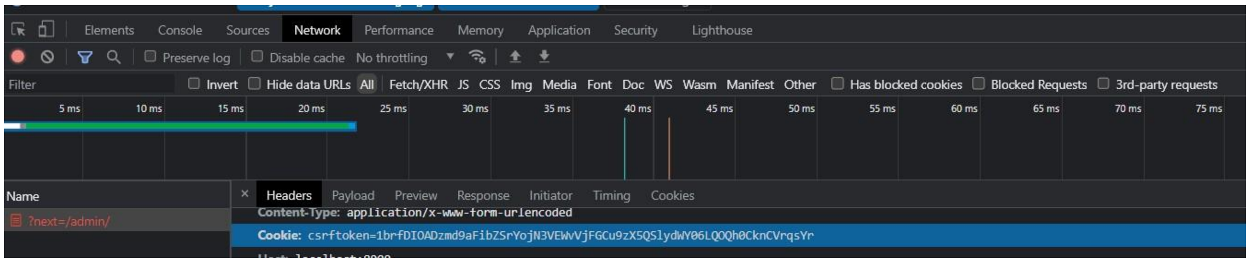
- Your browser is accepting cookies.
- The view function passes a `request` to the template's `render` method.
- In the template, there is a `{% csrf_token %}` template tag inside each POST form that targets an internal URL.
- If you are not using `CsrfViewMiddleware`, then you must use `csrf_protect` on any views that use the `csrf_token` template tag, as well as those that accept the POST data.
- The form has a valid CSRF token. After logging in in another browser tab or hitting the back button after a login, you may need to reload the page with the form, because the token is rotated after a login.

You're seeing the help section of this page because you have `DEBUG = True` in your Django settings file. Change that to `False`, and only the initial error message will be displayed.

You can customize this page using the CSRF_FAILURE_VIEW setting.

Figure 14: Debug error page

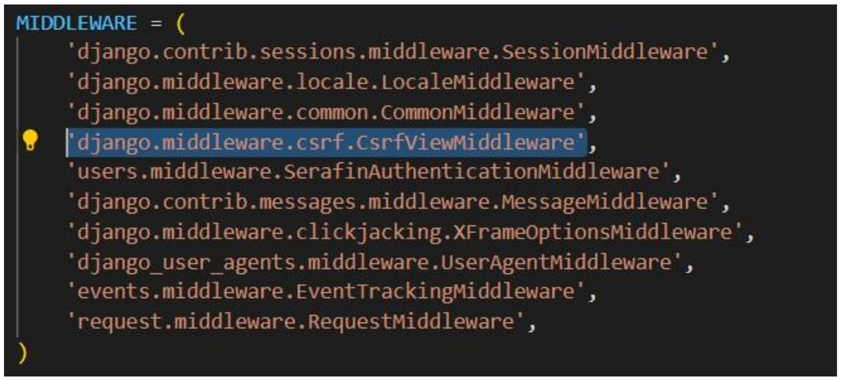
It looks like that there is a problem with csrf token and request is aborted. Initial thought was that the csrf token was not delivered. To check this, we need to open console in our web browser, go the network tab and refresh the page.



```
?next=/admin/
Content-Type: application/x-www-form-urlencoded
Cookie: csrftoken=1brfDIOADzmd9aFibZSrYojN3VEWvVjFGCu9zX5QSlydWY06LQOQh0CknCVrqsYr
```

Figure 15: Web browser console

Upon inspecting the headers, it is evident that the CSRF token is present in the cookie, indicating that the issue is not related to the token. To resolve this problem, we need to consult the Django documentation on Cross-Site Request Forgery (CSRF) protection [17] and conduct a thorough investigation. Firstly, we will need to verify whether CsrfViewMiddleware is included in the settings.py file of our project. Middleware [18] in Django is a mechanism used to extend the request/response processing flow by adding additional functionality. It is situated between the web server and the view and is responsible for intercepting the incoming request before it reaches the view and processing the response before it is returned to the web server. CsrfViewMiddleware is responsible for providing protection to forms.



```
MIDDLEWARE = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'users.middleware.SerafinAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django_user_agents.middleware.UserAgentMiddleware',
    'events.middleware.EventTrackingMiddleware',
    'request.middleware.RequestMiddleware',
)
```

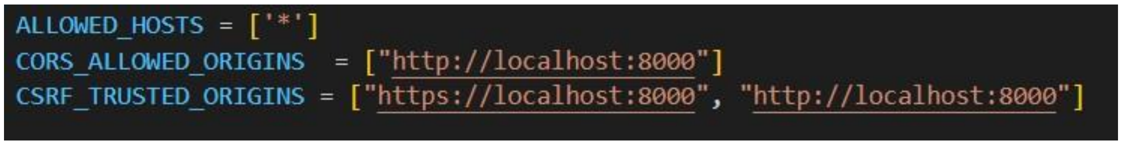
Figure 16: Setting.py file

As we can observe from the Figure 16, CsrfViewMiddleware is already included in our project. However, in newer versions of Django, we need to add certain configurations such as ALLOWED_HOSTS, CORS_ALLOWED_ORIGINS, and CSRF_TRUSTED_ORIGINS [19] to protect against Cross-Site Request Forgery (CSRF) attacks.

ALLOWED_HOSTS – is a list of valid hostnames or IP addresses that can be used to send requests to the Django server. Used as a security measure to prevent HTTP Host header attacks.

CORS_ALLOWED_ORIGINS – is a list of valid origins for Cross-Origin Resource Sharing requests. Used to allow or restrict access to resources on the server from other domains.

CSRF_TRUSTED_ORIGINS – is a list of valid origins that are allowed to bypass CSRF protection.



```
ALLOWED_HOSTS = ['*']
CORS_ALLOWED_ORIGINS  = ["http://localhost:8000"]
CSRF_TRUSTED_ORIGINS = ["https://localhost:8000", "http://localhost:8000"]
```

Figure 17: Allowed hosts and origins added

After adding allowed hosts and origins we can finally login to the web site.

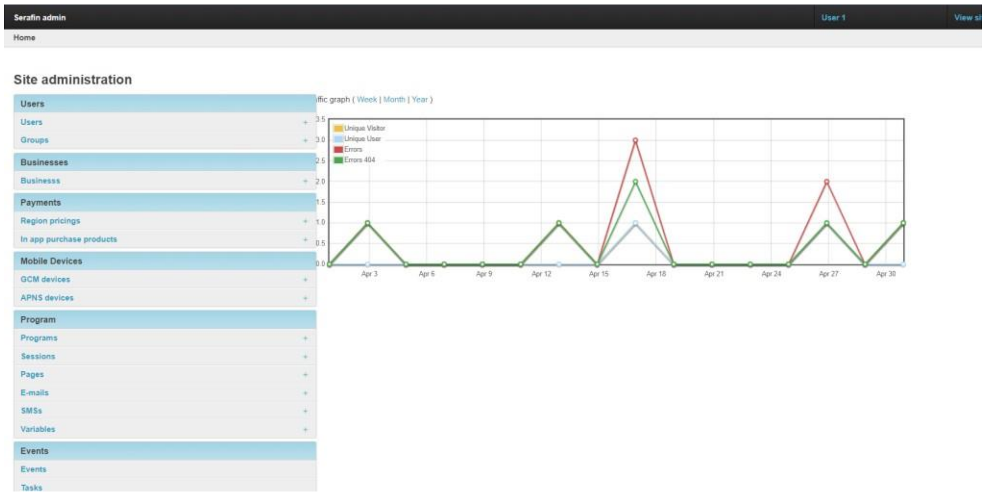
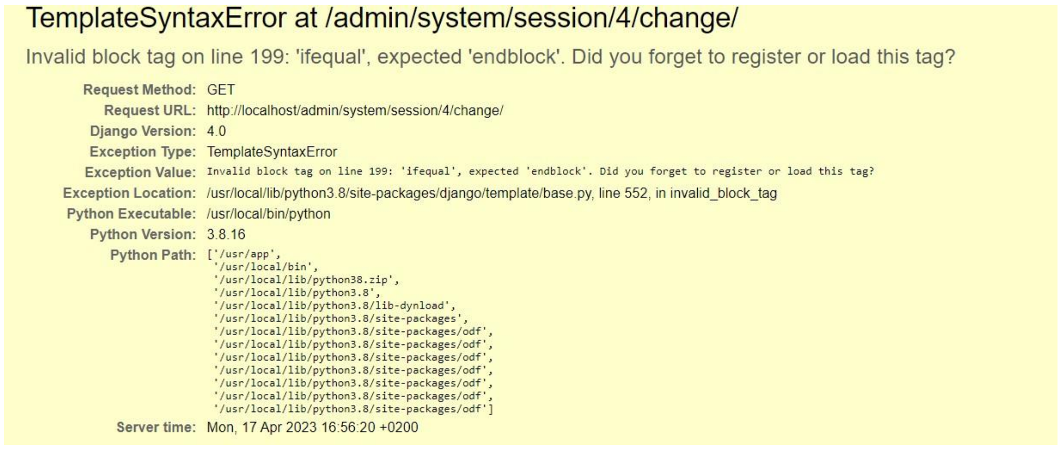


Figure 18: Our web application

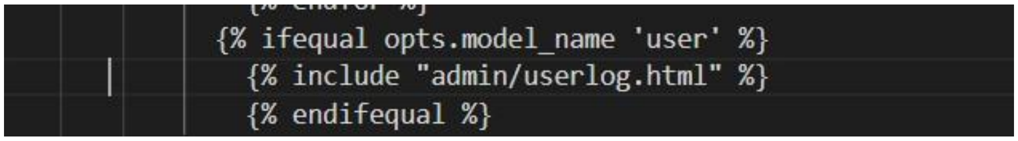
With our current progress, the web application seems to be functioning as expected, and our next goal is to test it. If we try to get to the sessions, we can see another error which is in the template file and located on the line 199 which is ifequal tag.



```
TemplateSyntaxError at /admin/system/session/4/change/
Invalid block tag on line 199: 'ifequal', expected 'endblock'. Did you forget to register or load this tag?

    Request Method: GET
       Request URL: http://localhost/admin/system/session/4/change/
    Django Version: 4.0
    Exception Type: TemplateSyntaxError
   Exception Value: Invalid block tag on line 199: 'ifequal', expected 'endblock'. Did you forget to register or load this tag?
Exception Location: /usr/local/lib/python3.8/site-packages/django/template/base.py, line 552, in invalid_block_tag
 Python Executable: /usr/local/bin/python
    Python Version: 3.8.16
       Python Path: ['/usr/app',
                     '/usr/local/bin',
                     '/usr/local/lib/python38.zip',
                     '/usr/local/lib/python3.8',
                     '/usr/local/lib/python3.8/lib-dynload',
                     '/usr/local/lib/python3.8/site-packages',
                     '/usr/local/lib/python3.8/site-packages/odf',
                     '/usr/local/lib/python3.8/site-packages/odf',
                     '/usr/local/lib/python3.8/site-packages/odf',
                     '/usr/local/lib/python3.8/site-packages/odf',
                     '/usr/local/lib/python3.8/site-packages/odf',
                     '/usr/local/lib/python3.8/site-packages/odf',
                     '/usr/local/lib/python3.8/site-packages/odf']
       Server time: Mon, 17 Apr 2023 16:56:20 +0200
```

Figure 19: Template error

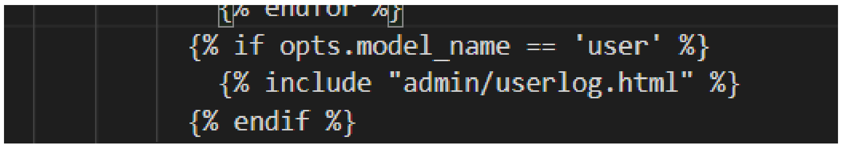
It should be noted that the ifequal tag has been deprecated and is no longer supported [20]. Therefore, we must replace it with a suitable alternative.



```
{% ifequal opts.model_name 'user' %}
  {% include "admin/userlog.html" %}
  {% endifequal %}
```

Figure 20: Code in the template

This method checks whether opts.model_name is user, so we can easily replace it with the following code.



```
  {% endfor %}
{% if opts.model_name == 'user' %}
  {% include "admin/userlog.html" %}
{% endif %}
```

Figure 21: Replaced code in the template

After this minor change and tests from my side it looks like that everything is working perfectly. Consequently, we can proceed to the subsequent task of migrating changes from the previous version of our web application to the current one.

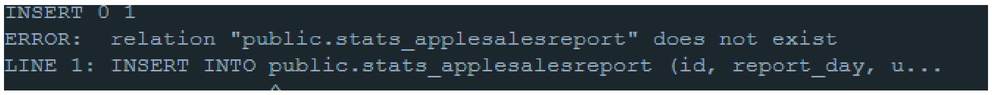
3.4 Transfer commits and feeding database

To proceed with our testing, we first need to transfer the changes from the older version of the web application to the newer one. GitLab provides us with a history of all the commits made by other developers that we need to transport to the newer version. We must be careful during the transfer process and ensure that everything works properly, as some changes in the older version may not be compatible with the newest version of Django. Fortunately, we found that all the changes were compatible, and we were able to proceed with feeding our database.

To feed our database, we need to open our container and follow a set of instructions. Firstly, we need to clear the entire database and remove all data using the following command: **python manage.py flush --noinput**. The **--noinput** flag is used to bypass the confirmation prompt that appears before flushing the database. To manage our database, we can use PostgreSQL, which is one of the most popular apps for this task.

PostgreSQL [21] - is an open-source database management

To install PostgreSQL inside the Docker container, it is necessary to run the following commands. First, to update the package list from the repository using the command "apt update". This command fetches the latest package information available and is important to run before installing or updating packages. Next, install PostgreSQL using the command "apt install postgresql-client". After successful installation, we can generate data for our database using a script provided by our web application. Then run the script inside our Docker container. To feed our database, the following command is used "psql -v ON_ERROR_STOP=1 -h db -U serafin_dev serafin_dev < <file with data>". After running this command, Docker will ask for the password for accessing the database. After entering the correct password, Docker will start feeding the database with the data. Unfortunately, the error is occurring during this process.



```
INSERT 0 1
ERROR:  relation "public.stats_applesalesreport" does not exist
LINE 1: INSERT INTO public.stats_applesalesreport (id, report_day, u...
                    ^
```

Figure 22: Relation error

This error indicates that the database table "stats_applesalesreport" does not exist in the public schema of the database. It seems that the table either hasn't been created yet, or it was dropped from the database. One possible solution for this problem can be running migrations inside our container with the following command **python manage.py makemigrations**. Unfortunately, it seems that the problem is in something else. During the investigation process by comparing older version of application with the current one it appears that there are some missing files and modules in stats folder as the error indicates. The current task is to transfer those files to the current web application. During this process some issues appeared as well. Such problems as old syntax were used as the older version is running on Django 1.9. New libraries had to be added to the Pipfile to ensure that the application and new APIs are working fine. After rebuilding the container and running it migrations were applied and new tables were added. The process of feeding our database was finished successfully and our project is ready. It is now possible to test it again and see if everything is working fine. Following the successful test, it is now possible to create a separate branch on the Gitlab and upload our project and merge it with the master branch.

Chapter 4

Conclusion

4.1 Summary

It is crucial for the application to receive updates and try to not use outdated technologies as they are developing every day. Developers must upgrade their application to provide users with new features, fixed bugs, and secured application. The main goal of this thesis was to upgrade the existing web application that had used outdated Django version. The use of the latest version provided better security, faster performance, and easier maintenance. It was necessary not only for users as a new version of Django fixes bugs, guaranties secured code and official support, but also it was crucial for developers too. As updated version provides more features and simplifies work it is now easier for developers to create a new content for users. The current web application which is run on Django 1.9 receives updates as developers create new features and fix existing bugs. Another task was to transfer these changes that were made in the current web application to the newer one which is going to be deployed.

4.2 My contribution

It was challenging for me as I was not familiar with Django, Docker, and other technologies so well. I had to read documentations, watch some guides, analyze where the error might be and about optimized solution. I would say that after this work I really improved my knowledge and skills as a web developer. With my contribution to this web application other developers can now expansion its functionality and provide users with more features which can help them to quit smoking.

4.3 Future work

There is great potential of this web application in the long term as Django receives updates often. Every new version gets new features, fixes bugs, makes work for developers easier and improves security for users. There is a great possibility for developers to optimize this web application in the future with newest versions of Django. There are more examples on how this web application can be improved. One of them is to improve the security by implementing additional measures such as two-factor authentication, session management, and data encryption. Another is by adding new APIs, libraries which can optimize code by making it more readable and understandable. Also, user testing can be conducted to gather feedback on the application. This can help identify areas for improvement and further development or spot bugs which were missed.

To conclude this thesis, I would like to say that it gave me a lot of experience in work with Django, Docker, and other areas. I improved a lot my skills and got closer to the goal of becoming a qualified software engineer.

Bibliography

[1] Django Project. Available at: https://www.djangoproject.com/. (Accessed date: 21.12.2022)

[2] The DEVSECOPS platform GitLab. Available at: https://about.gitlab.com/. (Accessed date: 21.12.2022)

[3] Atlassian *What is Git: Atlassian Git Tutorial*, *Atlassian*. Available at: https://www.atlassian.com/git/tutorials/what-is-git. (Accessed date: 21.12.2022)

[4] Accelerated, containerized application development (2023) *Docker*. Available at: https://www.docker.com/. (Accessed date: 24.12.2022)

[5] What is a container? (2022) *Docker*. Available at: https://www.docker.com/resources/what-container/. (Accessed date: 24.12.2022)

[6] Bansal, Shivam. "Python-Django(Views, Templates & Models)." *Medium*, DataDrivenInvestor, https://medium.datadriveninvestor.com/python-django-views-templates-models-f0844a00db70. (Accessed date: 12.01.2023)

[7] Nige, Big. "Structure." *Mastering Django*, masteringdjango.com/django-tutorials/mastering-django-structure/. (Accessed date: 12.01.2023)

[8] Deshpande, Chinmayee. "What Is Angular?: Architecture, Features, and Advantages [2022 Edition]." *Simplilearn.com*, Simplilearn, https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular#:~:text=Angular%20is%20an%20open%2Dsource,for%20developers%20to%20work%20with. (Accessed date: 18.01.2023)

[9] "Bower: Webstorm." *WebStorm Help*, https://www.jetbrains.com/help/webstorm/using-bower-package-manager.html. (Accessed date: 04.02.2023)

[10] "Dockerfile Reference." *Docker Documentation*, https://docs.docker.com/engine/reference/builder/#:~:text=A%20Dockerfile%20is%20a%20text,line%20to%20assemble%20an%20image. (Accessed date: 05.02.2023)

[11] "Django." *Django Project*, https://docs.djangoproject.com/en/4.1/howto/upgrade-version/. (Accessed date: 14.02.2023)

[12] "Django." *Django Project*, docs.djangoproject.com/en/4.0/releases/3.1/#id2. (Accessed date: 15.02.2023)

[13] Bessas, Apostolis. "Using Gettext in Django." *Using Gettext in Django · That's Weird...*, blog.bessas.me/posts/using-gettext-in-django/. (Accessed date: 15.02.2023)

[14] “Is_safe_url.” *Django 1.9.13 Release Notes - Django 4.2.1 Documentation*, django.readthedocs.io/en/stable/releases/1.9.13.html. (Accessed date: 15.02.2023)

[15] Getting started - developer guides: MDN. Developer guides | MDN. (n.d.). From https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started (Accesseddate: 06.03.2023)

[16] Django. Django Project. (n.d, from https://docs.djangoproject.com/en/3.1/releases/3.1/#id2 (Accessed date: 04.04.2023)

[17] Django. Django Project. (n.d.). from https://docs.djangoproject.com/en/4.2/howto/csrf/ (Accessed date: 20.04.2023)

[18] Django. Django Project. (n.d.), from https://docs.djangoproject.com/en/4.2/topics/http/middleware/#:~:text=Middleware%20is %20a%20framework%20of,for%20doing%20some%20specific%20function. (Accesseddate: 01.05.2023)

[19] “ALLOWED_HOSTS AND ORIGINS.” *Django Project*, docs.djangoproject.com/en/dev/ref/settings/#allowed-hosts. (Accessed date: 01.05.2023)

[20] “Ifequal Deprecated.” *RSS*, code.djangoproject.com/ticket/31532 (Accessed date: 03.05.2023)

[21] About. PostgreSQL. (n.d.). From https://www.postgresql.org/about/ (Accessed date: 04.05.2023)