MailerOwl - All in One Email Marketing Solution

Dev Mehta
dmehta3@ncsu.edu
North Carolina State University

Divyang Doshi
ddoshi2@ncsu.edu
North Carolina State University

Manogna Potluri
mcpotlur@ncsu.edu
North Carolina State University

Priyam Garg
pgarg6@ncsu.edu
North Carolina State University

Yash Bhansali
ygbhansa@ncsu.edu
North Carolina State University

ABSTRACT

The main goal of this document is to describe and summarize MailerOwl's general functionalities. Users can send one-time, recurring, or scheduled emails using MailerOwl. From the admin panel, users may also monitor the status of their emails. A user can use our application to schedule recurring emails to be delivered over a predetermined period of time. Marketing teams can use this solution to regularly promote their goods and special offers to frequent customers. In the sections that follow, the technologies and features that are used to create are described in greater detail.

ACM Reference Format:
Dev Mehta, Divyang Doshi, Manogna Potluri, Priyam Garg, and Yash Bhansali. 2022. MailerOwl - All in One Email Marketing Solution. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

It can become irritating to frequently send customers follow-up emails about your product. If your firm has a large client list and a large advertising budget, it may be challenging to track the list of customers on a daily basis. One customer's forgetfulness by the seller or the marketing team could result in a huge loss for the company. For sending repeat emails as well as scheduling emails for clients or different users, our software **MailerOwl** is a great choice. Along with offering the fundamental email features, our solution **MailerOwl** also gives users a productive approach to plan and send emails.

Users will benefit from **MailerOwl's** ability to schedule and send emails more quickly. We used technologies like Python, Docker, Django, PostgreSQL, RabbitMQ, Celery, and MailJet to develop this. We transfer emails using mailjet as the server. Therefore, our application will enable customers to schedule and send emails at their convenience over a specific time period or an immediate email.

2 FUTURE SCOPE

New advanced features can be added in the product such as:

- Good user interfaces can be developed to improve user experience.
- Content Optimizers can be added to the email which can help users to improve their email content and analytics can be added to show the attributes which need improvement.
- Several Machine Learning Models can be integrated to implement segmentation to target likely audience and also to detect spam emails.
- To remove the dependency on MailJet, we can use other generic API's like Gmail.
- Add more user roles to control how many emails a random user can send to try out the application.

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 LINUX BEST PRACTICES

The entire project was created using all of the guidelines and procedures used in software development. The best Linux practices and their use in the current project are as follows, with examples:

3.1 Zero Internal Boundaries

We made sure that everyone on our team had access to and the best knowledge of all the resources being used to create the project. With everyone on the team having a solid understanding of Python and Django, the product's back end was created. The database was created using PostgreSQL. All of the team members felt at ease utilizing Celery as the worker and RabbitMQ as the asynchronous task queue. VCL was used for project deployment while MailJet was used for sending emails through server. For containerizing the entire application, we used Docker. Each team member was successful in finishing the duties they were given.

3.2 Short Release Cycle

Since the MVP version of the product was the only thing on the agenda, it took the whole release cycle to complete it. As a result, there were no quick releases throughout the four-week interval. The majority of the significant code was released within a week, and a few bug patches were pushed following the testing cycle.

3.3 Distributed Development Model

Given the interdependence of the jobs and the schedules of each team member, the overall workload was fairly balanced among the members of the team. Two of the team members worked on constructing services to send and schedule the email as well as developing Controller classes after we established the project's concept. Two team members worked on the Docker files and the login and registration modules. One team member concentrated on developing the application's necessary models as well as the documentation. However, everyone was informed of the state and development of each project division.

3.4 Consensus Oriented Model

The project idea, as well as the results and objectives we aimed to attain, were the subject of numerous debates and meetings that led to consensus. Additionally, the general design of the product was chosen after careful consideration and team alignment. All the problems, difficulties, and technical specifics were addressed and polished during the development process, and a single strategy or solution was chosen.

3.5 No Regression Rule

Regression testing is not currently necessary for any of the project's scope's components. There will be a number of newly specified modules that will need to be tested for regression after the introduction of the future scope.