Project 1 (Group 16)- vogueX—Fashion-Recommender

Sravanth Reddy Bommana*
sbomman@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Akash Sarda
aksarda@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Arvind Srinivas Subramanian
asubram9@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Vinita Ramnani
vjramnan@ncsu.edu
North Carolina State University
Raleigh, NC, USA

Sidhant Arora
sarora22@ncsu.edu
North Carolina State University
Raleigh, NC, USA

ABSTRACT

This document explains how we as a team followed the Linx Kernel best practices in our project vogueX—Fashion-Recommender. It also provides evidence and will explain how various attributes of rubric match the Linux Kernel best practices.

KEYWORDS

recommender system, weather api, search api

ACM Reference Format:
Sravanth Reddy Bommana, Akash Sarda, Arvind Srinivas Subramanian, Vinita Ramnani, and Sidhant Arora. 2022. Project 1 (Group 16)- vogueX—Fashion-Recommender. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

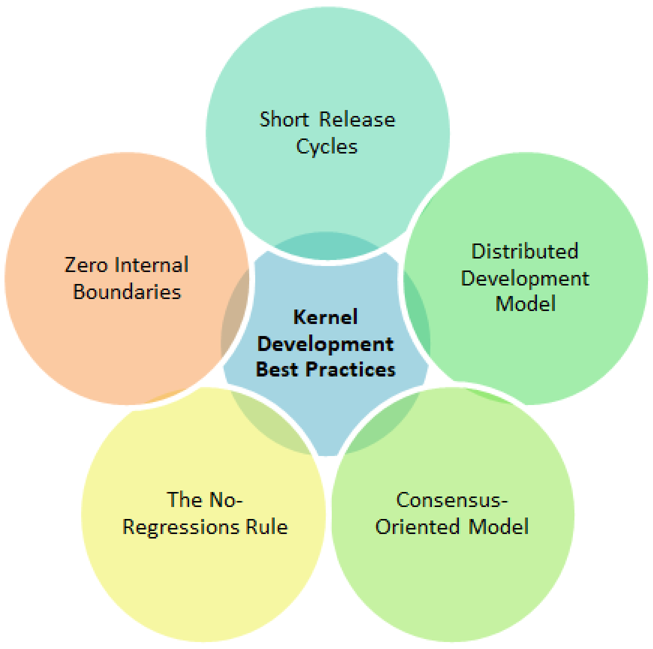
vogueX—Fashion-Recommender, is here to save your day. This isn't any regular fashion recommender but a recommender that will look out for you not only in terms of style but in terms of comfort. Now you may think how would one do that? We do this by providing you choices based on:

- Weather of the day, to let you know if you should avoid certain apparel or carry some extra accessories
- Season to illustrate different patterns
- Occasion to keep you in check with the highest rated choices

And if this doesn't seem enough, one can extend this in a thousand different ways, some of which are:

Integrating the health app with it to take an extra step and use the predicted menstrual cycle to enhance the outfit recommendation as to make it more comfortable. Introducing a feedback mechanism to keep track of the user's preferences in order to give better suggestions Integrating the calendar app to take care of the important days and send a recommendation accordingly.

Figure 1: Linux Kernel Best Practices



2 LINUX KERNEL BEST PRACTICES

Linux is one of the best examples of power of open source. Over the years linux has evolved a lot in terms of functionality and it's community is growing everyday. All the credit for project's longevity and success go to the best development practices followed by the contributors. We have followed the following set of principles from linux kernel best practices for our development.

2.1 Distributed development model

A distributed model is the best way of taking the development ahead, assigning different portions of the kernel (such as networking, wireless, device drivers, etc.) to different individuals, based on their familiarity with the area. This enabled seamless code review and integration across the thousands of areas in the kernel without any compromise in kernel stability.[reference]

In our project we have also followed the modularized approach. We divided the project into various modules like user interface, DB design, Weather API integration, Search API integration. TO achieve this division of modules we held meetings to discuss in which we followed round-robin way of speaking. We also leveraged voting based decision making for various aspects in the system.Work load was evenly distributed among team members and each of them contributed in form of commits. We maintained seperate dev and main branches. Development happens on dev branch and releases happen on main branch. We followed the approach of raising pull requests for every commit made to dev branch to maintain stable releases.

2.2 Consensus Oriented Model

The Linux kernel community strictly adheres to the consensus-oriented model, which states that a proposed change cannot be integrated into the code base as long as a respected developer is opposed to it. Although this might frustrate individual developers, such a practice ensures that the integrity of the kernel is not tampered with; no single group can make changes to the code base at the expense of the other groups. Consequently, the kernel's code base remains as flexible and scalable as always.[reference]

We incorporated consensus oriented model in our fashion recommender with the following practice: All discussion's were held by the consensus of all team members and the issues were discussed and solved before they were terminated. There exists a discord channel which is active and was used heavily to discuss various essential things about the project. There exists files like CONTRIBUTING.md and CODEOFCONDUCT.md which discusses the ideology behind how our code was written and how it can be extended and what future work can be done in order to extend it.

2.3 Zero Internal Boundaries

Developers generally work on specific parts of the kernel; however, this does not prevent them from making changes to any other part as long as the changes are justifiable. This practice ensures that problems are fixed where they originate rather than making way for multiple workarounds, which are always a bad news for kernel stability. Moreover, it also gives the developers a wider view of the kernel as a whole.[reference] The whole team used the same tools like Visual Studio Code, Flask, python, sqlite, mysql. All the code was maintained remotely on git and stable code was being merged to main branch. Dev branch was updated on basis of pull requests raised and reviewed by other teammates. Required software was added to requirements.txt so that every contributor is updated.

2.4 No Regression Rules

The kernel developer community continually strives to upgrade the kernel code base, but not at the cost of quality. This is why they follow the 1 no-regressions rule, which states that if a given kernel works in a specific setting, all the subsequent kernels must work there too. However, if the system does end up affected by regression, kernel developers waste no time in addressing the issue and getting the system back to its original state.[reference]

To ensure that every functionality added doesn't break other functionality we used pytest and added testcases and integrated it with circle to ensure the status of the buils is always safe. This way we can ensure that the newly added functionality doesn't break old code.

2.5 Short Release Cycles

In the early days of kernel development, major releases would come once every few years. This, however, led to several glitches and problems in the development cycle. For instance, long release cycles meant that vast chunks of code had to be integrated at once, which proved to be rather inefficient. It also translated into a lot of pressure for the developers to integrate features in the upcoming release even if they were not completely stable or ready. Moreover, delay in releasing new features was frustrating for users and distributors alike.[reference]

To ensure we follow this every functionality and every small bug is added as an issue to the issue board and is assigned to a teammate. We also made regular releases from the main branch ensuring that the code is stable. With this short release cycles everyone was aware of all the new code/functionality being added and also made reduced the pressure on developers.

3 ONLINE RESOURCES

https://medium.com/@Packt$pub/linux - kernel - development - best - practices - 11c1474704d6$