

Increasing user engagement in Fedora QA

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING



Submitted to:

Mr. Tarun Kumar

Assistant Professor

Submitted by:

Manisha Kanyal - 19SCSE1010011

SCHOOL OF COMPUTING SCIENCE & ENGINEERING

Galgotias University, Greater Noida

May 2021



**SCHOOL OF COMPUTING SCIENCE AND
ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA**

CANDIDATE’S DECLARATION

I hereby certify that the work which is being presented in the project, entitled “Increasing user engagement in Fedora QA” in partial fulfillment of the requirements for the award of the degree of B.Tech submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the 5 months, August 2021 to December 2021, under the supervision of Mr. Tarun Kumar, Assistant Professor, Department of Computer Science and Engineering, Galgotias University, Greater Noida.

The matter presented in the project has not been submitted by me for the award of any other degree of this or any other places.

Manisha Kanyal
19SCSE1010011

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Tarun Kumar
Assistant Professor

CERTIFICATE

The Final Project Viva-Voce examination of Manisha Kanyal, 19SCSE1010011 has been held on _____ and her work is recommended for the award of Degree of B.Tech.

Signature of Examiner(s)

Signature of Supervisor(s)

Signature of Project Coordinator

Signature of Dean

Date: November, 2013

Place: Greater Noida

TABLE OF CONTENT

1. Abstract	2
2. Table of Content	3
3. List of Tables	4
4. List of Figures	5

ABSTRACT

The Fedora community has tons of helpful documents, tools, and processes, but are in scattered form. Sometimes it is hard to reach/understand without some preliminary knowledge. It is identified as the major obstacle in bootstrapping new members of the community. And the goal of this project is to make the learning curve less steep. The goal is not hoarding/integrating all the information in one place - something in terms of a curated bookmark list, or wiki page would do a better job there. It is believed that some of the reasons for the difficulty in getting engaged are the "wall of text" symptoms. From experience with interns, college and high-school students, as it is known that there is a necessity for quick and easy engagement. In other words, the project wants to be able to answer the "What can I do? How can I participate? What is the best thing to work on right now?" questions. To achieve this goal, the project came up with the concept of "activities" - common tasks that can be done while taking part in Fedora QA processes. The activities will be integrating the "expert knowledge", and presenting the information and steps to take in an interactive, wizard-like fashion.

LIST OF TABLES

1. Introduction.....	9
2. Literature Review.....	12
3. Formulation of the problem.....	17
4. Tools and Technology used.....	18
5. Required Tools	10
5.1 React Library	10
5.2 Express	10
5.3 Git Version Control	10
5.4 CI/CD Tools	10
6. System Design	24
7. Module.....	32
8. Source Code.....	34
9. Results and conclusion.....	42
10. References.....	43

LIST OF FIGURES

1. Activity Diagram	10
2. Use Case Diagram	11
3. Application Diagram	12
4. Deployment Diagram	13

Acronyms

B.Tech.	Bachelor of Technology
M.Tech.	Master of Technology
BCA	Bachelor of Computer Applications
MCA	Master of Computer Applications
B.Sc. (CS)	Bachelor of Science in Computer Science
M.Sc. (CS)	Master of Science in Computer Science
SCSE	School of Computing Science and Engineering

INTRODUCTION

Fedora is a Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat, a subsidiary of IBM, with additional support from other companies. Fedora Linux is a favorite among Linux users who want their operating system to respect open source

The current state, as it can be seen, is that it has tons of helpful documents, tools, and processes, but these are scattered, or sometimes hard to reach/understand without some preliminary knowledge.

The purpose of this is to reduce the complexity of open-source enthusiasts, who contribute to fedora projects. Looking from a newcomer's perspective, the landing page of Fedora should be the most impactful but it's not [2].

It's how they are presented with the contribution process and it should be simplified as much as possible, so contributors can focus more on fixing issues. The landing page should mainly focus on guiding the newcomers through the contribution process. The page should include clear information on what they can do, by giving them options to explore in a step-by-step format.

The aim of this project is to reduce complexities as much as possible and to filter out all the issues based on their programming languages so as to make them easily accessible and understandable. User engagement is important because it can predict profitability, measure the effectiveness of specific marketing campaigns and provide data on how users choose to interact with a site or product. High user engagement can help companies sell ad space, refine marketing strategies and increase sales because it shows that users find value in their product or service.

ABOUT THE COMMUNITY:

Fedora is a Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat, a subsidiary of IBM, with additional support from other companies. Fedora Linux is a favorite among Linux users who want their operating system to respect open source

WHAT KIND OF PEOPLE WORK AT FEDORA?

Fedora has different kinds of people in the fedora community. Fedora has software developers/engineers, quality testers, community managers, designers, content/documentation writers, etc. The Fedora community has lovely people. People who love the community and are willing to contribute and volunteer their time, mentorship, resources, and code to make sure things go well.

WHAT PROBLEM IS THE PROJECT TRYING TO SOLVE?

The project's motive is to give information on the development schedule, meeting times and releases, quality testing, etc, and make it impactful so that newcomers can contribute. So I'm working on making the dashboard easier to use and simplified.

The current state, as it can be seen, is that it has tons of helpful documents, tools, and processes, but these are scattered, or sometimes hard to reach/understand without some preliminary knowledge. It has been identified as the major obstacle in bootstrapping new members of the community. And the goal for this project is making the learning curve less steep. The purpose of

This is to reduce the complexity of open-source enthusiasts, who contribute to fedora projects.

Looking from a newcomer's perspective, the landing page of Fedora should be the most impactful but it's not.

The activities(issues, timeline, schedules) will also be offered contextually, base on the user's skills and interests (e.g. showing easyfix-tagged bugs from projects written in Python to those skilled/interested in using it), the current phase of the development cycle (e.g. prioritizing

blockerbug testing, when the meeting is coming) or other meaningful metrics (e.g. presenting the users with manual test cases, that were not performed in a while).

We also want to offer a timeframe with the activities, so the user is able to filter/select them based on the timespan they are willing to spare (couple hours, a day, couple of days, one day a week regularly...).

The activities will act as a comprehensive guide, mimicking the way it can actually perform them. For example "Check testcase X on the nightly rawhide build" activity could consist of several steps:

- checking whether rawhide works
- finding the nightly iso
- installing in a virtual machine
- performing the testcase's steps
- reporting the result
- possibly filing bug report

Each of these steps can be (seen) as simple on its own, but might require external knowledge (looking into openqa, Adam's nightlies finder, finding the current wiki matrix, ...). The goal is to distill the knowledge into a wizard-like tool that takes you through the process, providing links to resources, and detailed information, while presenting concentrated, easy to understand information.

Reporting bugs, for example, is also deceptively non-trivial. While it might be easy for us (and the bugzilla templates are helping) a few hints - where do you find logs, that running a graphical app from command line can give you interesting data, or that it might be good to try another tool/app for the same workflow to identify a possibility of a bug in an underlying library - coming from a long-term experience are invaluable.

LITERATURE SURVEY

Open-source[3] is a user-driven software, collaborative innovation produced by self-organizing teams of contributors dynamically formed through online interactions. There are various benefits that can be found with the emergence of Open source software. The three key benefits are predominantly important. First, OSS software can dramatically reduce the cost of development for IT companies. Such cost does not only include any development cost but the innovation cost is what matters.

The software received effort and contribution from talented developments around the globe. Secondly, the OSS can facilitate the organizations to implement the IT systems into their business process, especially for small-medium enterprises or public institutes. There is no doubt that IT can make the business a better world. However, due to the cost of IT implementations, public institutes may not implement the IT system due to their limited budget. Introducing the OSS systems can diminish such dilemmas. Several metrics have been employed to assess the OSS success, and such assessments were developed in consideration of the research proposal and the audiences, such as number of subscribers, number of active developers, number of downloads, frequency of CVS commits, and the extent of code reuse, etc

Last but not least, the information and knowledge can be diffused in terms of participating in OSS projects like GSOC, Outreachy, Hacktoberfest, GirlsScript, and various OSS projects, which is beneficial for those people who want to have a good command of programming language since they can learn via projects.

Prior OSS Studies

In this section, a comprehensive extent of literature reviews is conducted. Based on the reviews, three subsections are listed by the research topics of prior OSS studies. In particular, the first stream includes the literature studying the individual motivations to contribute or participate in the OSS projects. The second stream summarized the literature discussing how the network

characteristics influenced the OSS performances. In the third stream, it is reviewed how the legalization, especially the OSS licenses, influences the OSS performances.

Individual Motivations: Thousands of individuals participate in OSS projects for diverse purposes. Their contributed product will be released to the public for free usage. Why are they willing to contribute to such OSS projects since their no economic returns from their contributed projects? What are their motivations? After surveying prior literature, it has been found two key motivations, which can be concluded as intrinsic motivations and extrinsic motivations.

Intrinsic Motivation: The intrinsic motivations have been studied since the 1970s. It is driven by an interest or enjoyment in the task itself and exists within the individual instead of depending on external pressure or desire for incentive reward [8]. The intrinsic motivation was widely regarded as a key motive for individuals to participate in the OSS projects. Previous literature found more than half developers indicated that the enjoyment in programming and the sense of satisfaction originated from the participation in OSS software constantly motivating them to sustainably contribute to OSS projects. Besides the sense of enjoyment and satisfaction, Lakhani and Wolf [1] found that several individuals could show their creativity and new ideas in terms of implementing them into the OSS projects, which conferred a great sense of accomplishment for them. Such a sense was believed as another key motivation for constant contribution. Although the intrinsic motivations were found as the original motives for breeding the OSS campaign, Roberts et al. [5] thought the intrinsic motivations had several defects, such as short effectiveness and strong self-direction, which might challenge the sustainability of OSS projects in future. Thus, it is imperative to unveil the extrinsic motivations for OSS participants.

Extrinsic Motivation: Extrinsic motivation denoting performing an activity is built upon the desired outcome like momentary incentives, reputations, and profits etc., which is the opposite of intrinsic motivation. In other words, with the disappearance of external incentives, then the extrinsic motivations will decrease or even disappear. Von Krogh and his collaborators investigated a large OSS project and interviewed several participants and summarized that the

participation was built upon the pursuit of communal resources, which includes reputations, control of technology, and learning opportunities. The first dimension of extrinsic motivation is the acquisition of reputation.

The reputation can be obtained in terms of (1) actively participating in the OSS projects, (2) providing solutions to the existing bugs or problems, (3) providing innovative revision or modification to the current OSS. With the increase of reputation, the participants will be conferred with higher authority, which will eventually enable him or her to dominate the entire project, such as the recruitment of developers or the decision power in the project management. To this end, the reputation in the OSS community is widely regarded as an extremely important extrinsic motivation for encouraging OSS developers to constantly contribute to OSS projects.

The second acquired communal resource is the “control over technology”. Such extrinsic motivation is formalized from the demand of self-usage. It is found that the key reason to participate in an OSS project is to customize such a project for its own usage [3]. Although novel and creative ideas can be realized through OSS projects, the overall quality of OSS projects cannot reach the industrial standard due to the relaxed management system and software testing procedures.

Thus, those experienced users may compile their revised OSS and incidentally update it. In terms of long-term investment, those users may have a good command of this technology, which would benefit them in the future. The third communal resource is the learning opportunities. The participants of OSS projects do not only include those experienced developers but those users or fresh programmers. By joining the OSS project, these people were provided a very good opportunity to learn (1) programming by collaborating with talented programmers in the world, and (2) knowledge about project management in software engineering as well. Besides these three key extrinsic motivations, prior literature also identified several other extrinsic factors motivating individuals to contribute to the OSS project such as a sense of reciprocity and job opportunities.

Sense of reciprocity is prevailing in the OSS user support community, in which those end-users proactively respond or answer other's questions in order to obtain the help from others when they need it in the future. In addition, prior literature found some IT companies might recruit some talented programmers from the OSS community, which encourage some participants to diligently work on their contributed OSS projects.

Besides the explanation from motivation theory, several studies found institutional management or leadership also affected individual motivations. For instance, by interviewing the developers from the Debian project, O'Mahony and Ferraro [1] found the democratic management style outperformed the bureaucratic one though the latter was found to be efficient. In addition, Li and her collaborators found individuals preferred to join those OSS projects which employed the transformational leadership style. In other words, such leadership style signals that everyone could be conferred as a project leader in the future.

In the second stream, prior studies investigated how network structures affected the OSS performances. The development of OSS projects cannot be done without collective actions. In this regard, prior literatures employed the social network analysis to articulate such collective actions, i.e. collaborative behaviors, which include inter-project individual-individual collaborative networks [7], project-affiliated networks [8], and intra-project communication or collaborative networks [10] etc. Thus, the social capital theory was mostly adopted as a theoretical underpinning to explicate such social relationships.

Due to the open nature of the OSS projects, the participants can freely contribute to multiple projects. In terms of such shared participants or concurrently contributed projects, two types of affiliated networks can be initialized. The first one is called a project-affiliated network, in which the projects with shared participants are interconnected. The second one is called participant-affiliated network, in which the developers/project administrators who contributed to the same projects are interconnected. Prior studies found the characteristics of participant-affiliated networks, such as network distances or network density, had a significant

impact on the OSS performance. After five-year observation on more than 2000 OSS projects hosted in Sourceforge.net, Singh and his collaborators found internal collaborative cohesion had significantly positive impact on the OSS performances, manifested by the extent of CVS commits, but the external cohesion presented an inverted-U impact on the OSS performances. Hahn et al. [8] found the OSS developers preferred to join in those projects initialized by those who had collaborated before in terms of investigating the collaborative ties. Grewal et al. [7] argued the OSS performance was significantly influenced by the extent of network embeddedness in terms of studying the developer-affiliated network.

Besides the inter-project network, prior studies also found that the intra-project network also played a role in OSS performances. Differing from the works studying OSS performances, the intra-project studies mainly discussed the individual collective behaviors. For instance, Conaldi et al. established an analytical model for collaborative networks and empirically verified it by using the debugging network from a large OSS project categorized the participants from Debian (a leading Linux OS distributor) into three key roles, which included knowledge seekers, knowledge contributors, and knowledge brokers in terms of analyzing the emailing communication network, and argued that the knowledge brokers could facilitate the information flow and distribution. By studying the internal communication networks of two leading OSS projects, Singh and Tan found the stability and efficiency cannot be concurrently reached, and advocated the OSS project administrators had to adjust the balance based on their key goals.

In sum, it can be found in the literature on inter-project networks mainly discussing how OSS projects could be outperformed in terms of network ties or vertex positions. The intra-project network studies mainly concentrated on particular behaviors or actions in OSS project development, such as debugging, communications, or collaborations. A brief summarization is given in Table [1].

PROBLEM FORMULATION

Newcomers who are open-source enthusiasts face difficulty while contributing to Fedora's issues.

The best way to present a contribution guide is explained, using infographics and clear instructions.

Iconography improvements are also very important in this case. For example, the link icon is not visible enough, the user doesn't realize that the headers like "Current development schedule" are clickable. The gray question mark icons on the timeline are not clickable although they give the impression that they are. The button, at the end of the page, would be more readable with a shorter description.

The meetings and development schedules are "secondary" information in this case so they don't need to be as visible.

Fedora page should be careful not to use technical jargon and acronyms when possible. They can easily scare away newcomers. For example "Fedora 31 blockers and FEs" might not be a familiar term to everyone. Assume users do not know technical concepts so have them briefly explained whenever possible.

Admin should be deciding what is presented to the user, in the best way possible, without them needing to enable/disable the content they see.

While the idea of users controlling the content is good, in this case, it might add another layer of confusion.

REQUIRED TOOLS

Increasing user engagement in Fedora QA is a web tool. Hence, the application is divided broadly into 2 parts.

Front End where user/admin will interact.

1. React Library - It's a JavaScript library for DOM manipulation that handles navigation through the HTML5 push state. To put it simply, it's a view library that uses components to change content on the page without refreshing, the core principle behind single-page applications. It was developed by Facebook in 2013 and can be used for mobile, web, and VR applications.

There are a few reasons behind choosing reactjs of which below are mentioned:

- a. Virtual DOM in React makes the user experience better and developer's work faster.

DOM (document object model) is a logical structure of documents in HTML, XHTML, or XML formats. Describing it in layman's terms, it is a viewing agreement on data inputs and outputs, which has a tree form. Web browsers are using layout engines to transform or parse the representation HTML-syntax into a document object model, which we can see in browsers.

The main concern about traditional DOM construct is the way it processes changes, i.e., user inputs, queries, and so on. A server constantly checks the difference caused by these changes to give the necessary response. To respond properly, it also needs to update the DOM trees of the whole document, which is not ergonomically valid because DOM trees are fairly large today, containing thousands of elements.

The team behind React managed to increase the speed of updates by using virtual DOM. Unlike other frameworks that work with the Real DOM, ReactJS uses its

abstract copy – the Virtual DOM. It updates even minimalistic changes applied by the user but doesn't affect other parts of the interface. That is also possible thanks to React components isolation, which it'll get to in a minute, and a special data structure in the library.

- b. Permission to reuse React components significantly saves time. Another advantage that Facebook introduced with React is the ability to reuse code components of a different level anytime, another meaningful time-saving effect. Think of designers. They constantly reuse the same assets. If they didn't, they'd have to draw corporate logos, for instance, over and over again. It's pretty obvious: Reusing is a design efficiency.

In programming, this process is a bit more difficult. System upgrades often turn into a headache as every change can affect the work of other components in the system. Managing updates is easy for developers because all React components are isolated and change in one doesn't affect others. This allows for reusing components that do not produce changes in and of themselves to make programming more precise, ergonomic, and comfortable for developers.

- c. One-direction data flow in ReactJS provides a stable code React allows for direct work with components and uses downward data binding to ensure that changes in child structures don't affect their parents. That makes code stable. Most complex view-model systems of JS-representation have a significant but understandable disadvantage – the structure of data flow.

In the view-model system, child elements may affect the parent if changed. Facebook removed these issues in React, making it just the view system. Instead of using explicit data binding, ReactJS uses one direction – downward – data flow. In such a structure, child elements cannot affect parent data. To change an object, all a developer needs to do is modify its state and apply updates.

Correspondingly, only allowed components will be upgraded.

- d. Redux: convenient state container Before writing an angry comment that Redux is framework-agnostic and you can happily use it with Angular or Vue and that it isn't exclusive to React. However, it's worth mentioning Redux here simply because the tool is considered to be every-React-engineer's must-learn instrument applied in 50 to 60 percent of React apps.

Yes, you can use Redux with Angular, but the probability of a React developer knowing Redux is much higher than knowing Angular. And you'll find more community support for tackling the React-Redux learning curve. So, why is it good? Redux simplifies storing and managing component states in large applications with many dynamic elements where it becomes increasingly difficult.

Redux stores application state in a single object and allows every component to access application state without dealing with child components or using callbacks. For instance, when you have two components that share the same state (like detailed and general views on the image below) and stand apart in the tree, without Redux, data has to be passed through multiple intermediary components with all the problems that go with it.

- 2. Express - It's a web application framework for Node.js, designed for building web applications and APIs.

There are a few reasons why it is always preferred expressjs:

- a. Scale the application quickly The first benefit of using Express.JS for backend development is that you would be able to scale your application quickly.

As you know that there is a support of Node.JS, so with the help of addition in nodes and adding extra resources to it, you can quickly scale your application in any manner.

- b. JavaScript is simple to learn. As mentioned above, JavaScript is very famous plus easy to learn. You would be able to use it for the development purpose on Express.JS.
 - c. Same language can be used to code Frontend Another benefit of using Express.JS is that you would be able to do the code of both frontend and backend with the help of using JavaScript.
Although there are several such platforms, they offer different language support for frontend and backend, which makes it quite challenging to work with. But you will never face any such problem with Express.JS.
 - d. Less developer cost to maintain the app Not everybody knows that Express.JS is a full-stack JavaScript because of which you would not have to hire different developers for managing the frontend and backend of a web application.
 - e. Supported by Google v8 engine Express.JS is supported with the Google V8 engine with the help of which you would be able to get higher performance without any lag or error in the processing.
 - f. Caching Express.js supports the caching feature, and the advantage of the catch is that you would not have to re-execute the codes again and again. Moreover, it will help web pages to load faster than ever.
3. Git and Github - Git helps in maintaining version control for the application while Github is a cloud-based hosting service that lets you manage Git repositories.

The following were the reasons behind choosing git and github for version control and collaboration:

- a. It makes it easy to contribute to your open source projects. To be honest, nearly every open-source project uses GitHub to manage their project. Using GitHub is free if your project is open source and includes a wiki and issue tracker that makes it easy to include more in-depth documentation and get feedback about

your project. If you want to contribute, you just fork a project, make your changes and then send them a pull request using GitHub web interface

- b. Documentation By using GitHub, you make it easier to get excellent documentation. Their help section and guides have articles for nearly any topic related to git that you can think of.
 - c. Showcase your work Are you a developer and wish to attract recruiters? GitHub is the best tool you can rely on for this. Today, when searching for new recruits for their project, most companies look into the GitHub profiles. If your profile is available, you will have a higher chance of being recruited even if you are not from a great university or college.
 - d. GitHub is a repository This was already mentioned before, but it's important to note, GitHub is a repository. What this means is that it allows your work to get out there in front of the public. Moreover, GitHub is one of the largest coding communities around right now, so it's wide exposure for your project.
 - e. Track changes in your code across versions When multiple people collaborate on a project, it's hard to keep track revisions—who changed what, when, and where those files are stored. GitHub takes care of this problem by keeping track of all the changes that have been pushed to the repository. Much like using Microsoft Word or Google Drive, you can have a version history of your code so that previous versions are not lost with every iteration.
 - f. Integration options GitHub can integrate with common platforms such as Amazon and Google Cloud, services such as Code Climate to track your feedback, and can highlight syntax in over 200 different programming languages.
4. Travis CI/CD - Travis CI enables us to build and deploy the application in an automated fashion. Travis CI allows independent testing of pull requests & branches. For easy monitoring, the test results are displayed on the GitHub UI.

The preferred way of testing that was opted was Travis for it's CI and CD features.

CI and CD stand for continuous integration and continuous delivery/continuous

deployment. In very simple terms, CI is a modern software development practice in which incremental code changes are made frequently and reliably. Automated build-and-test steps triggered by CI ensure that code changes being merged into the repository are reliable. The code is then delivered quickly and seamlessly as a part of the CD process.

In the software world, the CI/CD pipeline refers to the automation that enables incremental code changes from developers' desktops to be delivered quickly and reliably to production.

CI/CD allows organizations to ship software quickly and efficiently. CI/CD facilitates an effective process for getting products to market faster than ever before, continuously delivering code into production, and ensuring an ongoing flow of new features and bug fixes via the most efficient delivery method.

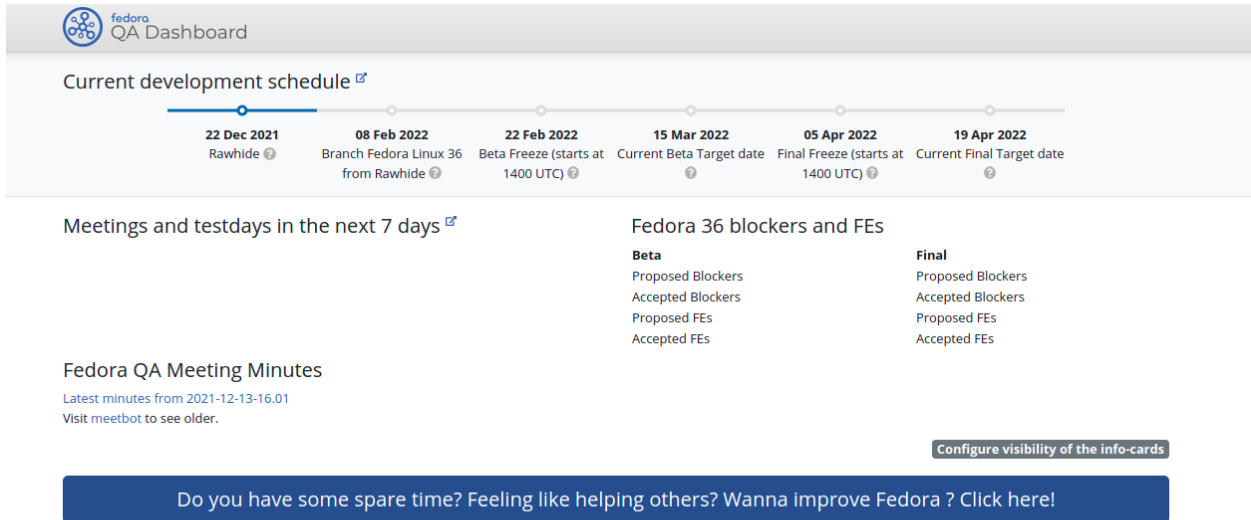
Continuous integration (CI) is practice that involves developers making small changes and checks to their code. Due to the scale of requirements and the number of steps involved, this process is automated to ensure that teams can build, test, and package their applications in a reliable and repeatable way. CI helps streamline code changes, thereby increasing time for developers to make changes and contribute to improved software.

Continuous delivery (CD) is the automated delivery of completed code to environments like testing and development. CD provides an automated and consistent way for code to be delivered to these environments. Continuous deployment is the next step of continuous delivery. Every change that passes the automated tests is automatically placed in production, resulting in many production deployments. Continuous deployment should be the goal of most companies that are not constrained by regulatory or other requirements. In short, CI is a set of practices performed as developers are writing code, and CD is a set of practices performed after the code is completed.

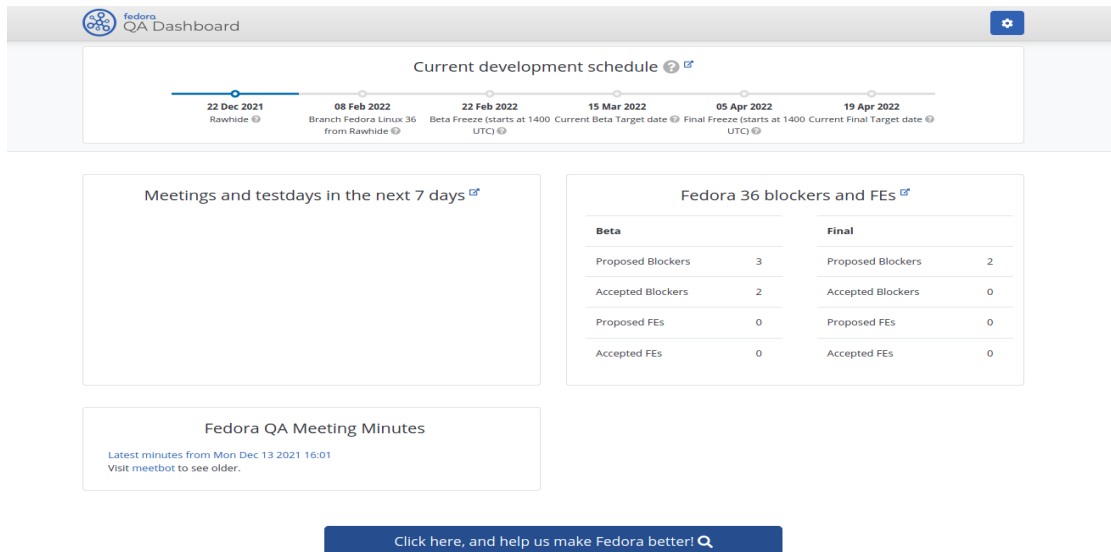
SYSTEM DESIGN

The page intended to be made as simple as possible in order the newcomers can easily understand the issues and work on them.

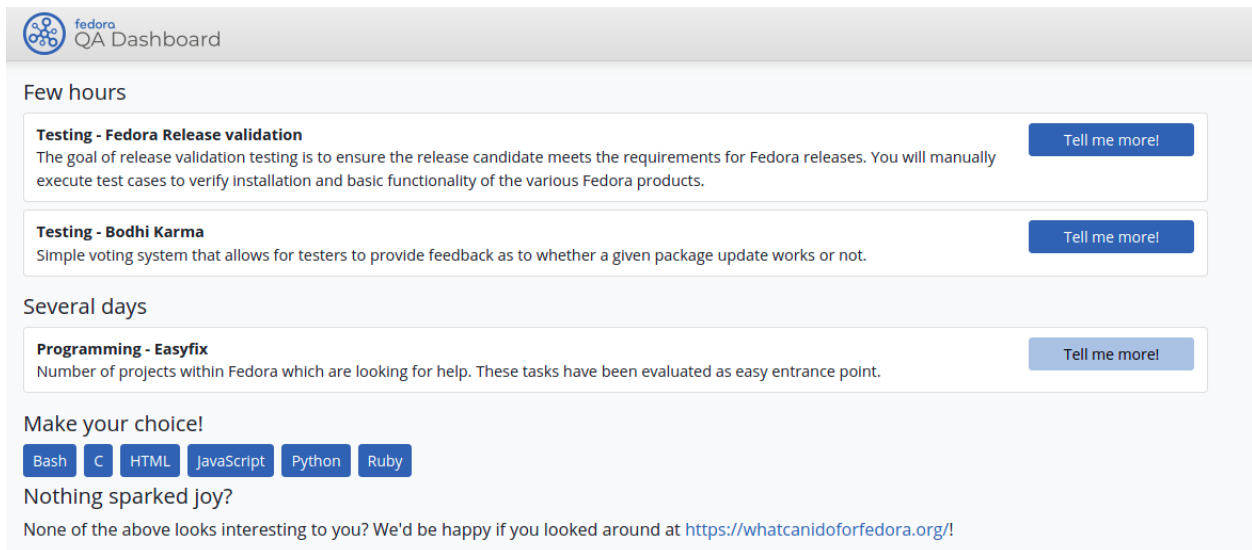
The UI(user interface) of the page used to looked like that:



The page that is intended to made:



The filtration of issues based on its language, the UI(user interface) of filtration of issues on the basis of its language:

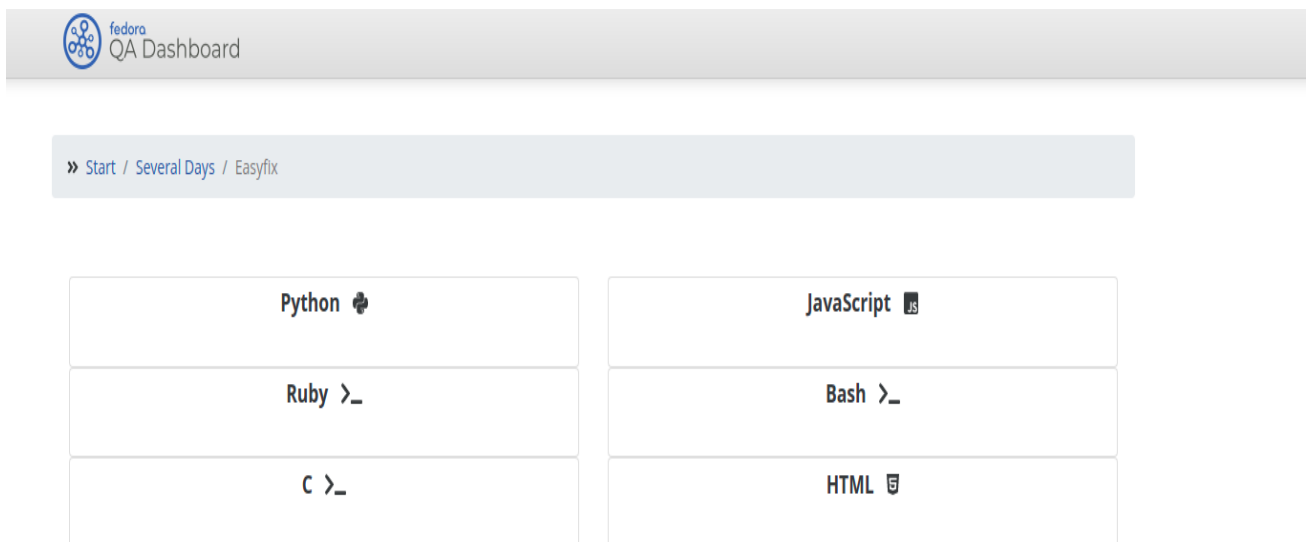


The screenshot shows the Fedora QA Dashboard with the following content:

- fedora QA Dashboard** (header)
- Few hours**
 - Testing - Fedora Release validation**: The goal of release validation testing is to ensure the release candidate meets the requirements for Fedora releases. You will manually execute test cases to verify installation and basic functionality of the various Fedora products. [Tell me more!](#)
 - Testing - Bodhi Karma**: Simple voting system that allows for testers to provide feedback as to whether a given package update works or not. [Tell me more!](#)
- Several days**
 - Programming - Easyfix**: Number of projects within Fedora which are looking for help. These tasks have been evaluated as easy entrance point. [Tell me more!](#)
- Make your choice!**
 - Buttons for: [Bash](#), [C](#), [HTML](#), [JavaScript](#), [Python](#), [Ruby](#)
- Nothing sparked joy?**

None of the above looks interesting to you? We'd be happy if you looked around at [https://whatcanidoforfedora.org/!](https://whatcanidoforfedora.org/)

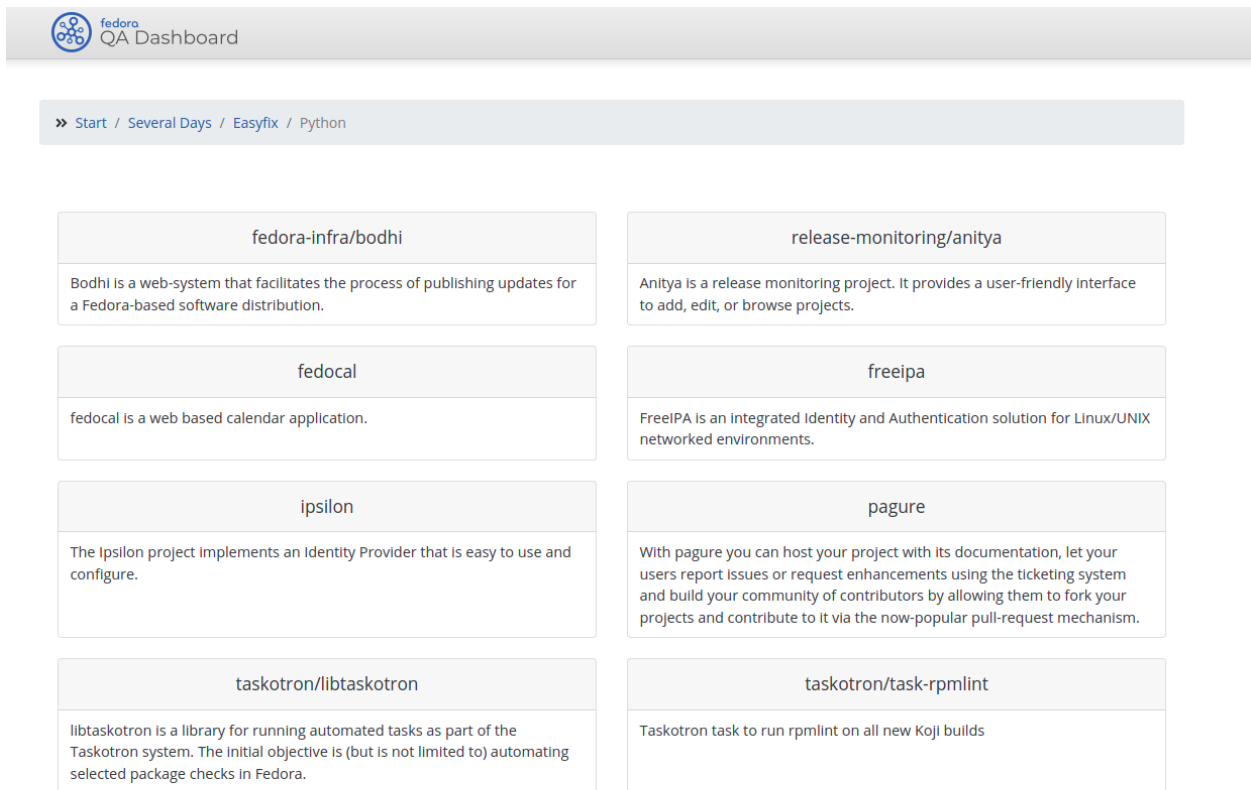
The page that is intended to made:



The screenshot shows a filtered view of the Fedora QA Dashboard:

- fedora QA Dashboard** (header)
- » Start / Several Days / Easyfix** (breadcrumb)
- Two columns of language filters, each with three buttons:
 - Column 1:** [Python](#) (with a plus icon), [Ruby](#) (with a minus icon), [C](#) (with a minus icon)
 - Column 2:** [JavaScript](#) (with a plus icon), [Bash](#) (with a minus icon), [HTML](#) (with a minus icon)

When python is clicked the issues will filter out this way:



The screenshot shows the Fedora QA Dashboard interface. At the top left is the logo and text "fedora QA Dashboard". Below it is a breadcrumb trail: "» Start / Several Days / Easyfix / Python". The main content area displays eight issue cards in a two-column grid. Each card has a header with the issue path and a description of the project.

fedora-infra/bodhi Bodhi is a web-system that facilitates the process of publishing updates for a Fedora-based software distribution.	release-monitoring/anitya Anitya is a release monitoring project. It provides a user-friendly interface to add, edit, or browse projects.
fedocal fedocal is a web based calendar application.	freeipa FreeIPA is an Integrated Identity and Authentication solution for Linux/UNIX networked environments.
ipsilon The Ipsilon project implements an Identity Provider that is easy to use and configure.	pagure With pagure you can host your project with its documentation, let your users report issues or request enhancements using the ticketing system and build your community of contributors by allowing them to fork your projects and contribute to it via the now-popular pull-request mechanism.
taskotron/libtaskotron libtaskotron is a library for running automated tasks as part of the Taskotron system. The initial objective is (but is not limited to) automating selected package checks in Fedora.	taskotron/task-rpmlint Taskotron task to run rpmlint on all new Koji builds

ARCHITECTURE DIAGRAMS

An architectural diagram is a diagram of a system that is used to abstract the overall outline of the software system and the relationships, constraints, and boundaries between components. This section of the report includes various diagrams to understand the architecture of the Fedora application.

Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions. In figure a, there is a single entry point for users when they visit the application. Based on user choices, the diagram clearly mentions all the possible paths a user can choose.

Activity diagrams should be used in alignment with other modeling techniques like interaction diagrams and State diagrams. The main reason behind using these diagrams is to model the work flow behind the system being designed. These Diagrams are also useful for analyzing a use case by describing what actions need to take place and when they should occur, describing a complicated sequential algorithm and modeling applications with parallel processes. Some advantages of using activity diagrams are listed below.

1. UML modeling language included that these diagrams are normally easily comprehensible for both analysts and stakeholders.
2. In UML for the IT Business Analyst, “The activity diagram is the one most useful to the IT BA for depicting work flow [because] it is simple to understand-both for BAs and end-users.”
3. Since they are among the most user-friendly diagrams available, they are generally regarded as an essential tool in an analyst’s repertoire.
4. Additionally, as stated above, activity diagrams allow an analyst to display multiple conditions and actors within a workflow through the use of swimlanes. Swimlanes,

however, are optional as a single condition or actor is normally displayed without them.

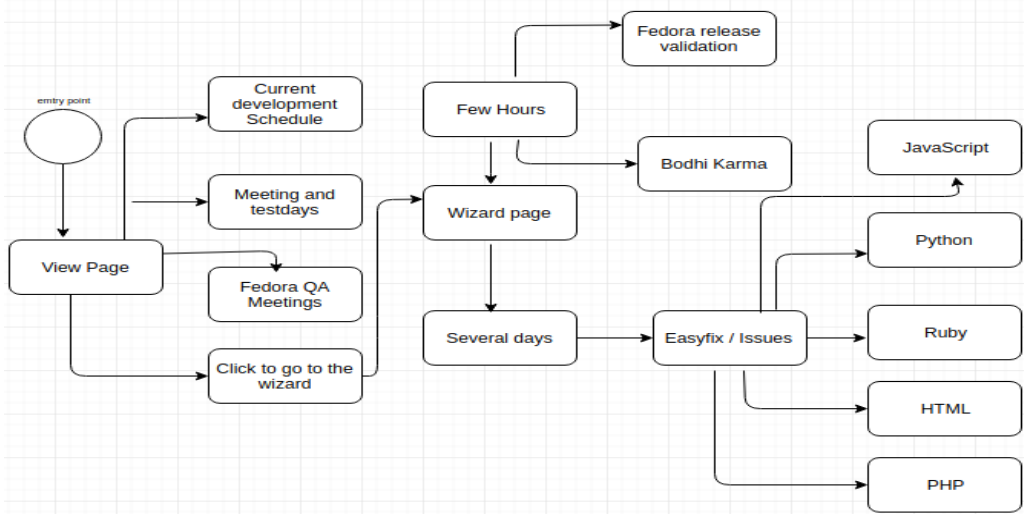


Figure A - Activity Architecture

USE CASE DIAGRAM

Use case diagrams are used to demonstrate the different ways that a user might interact with a system. For OneLink, there are 3 actors (users) - Registered User, Unregister User, and Admin User. These three actors interact with various components of the application as represented in the figure.

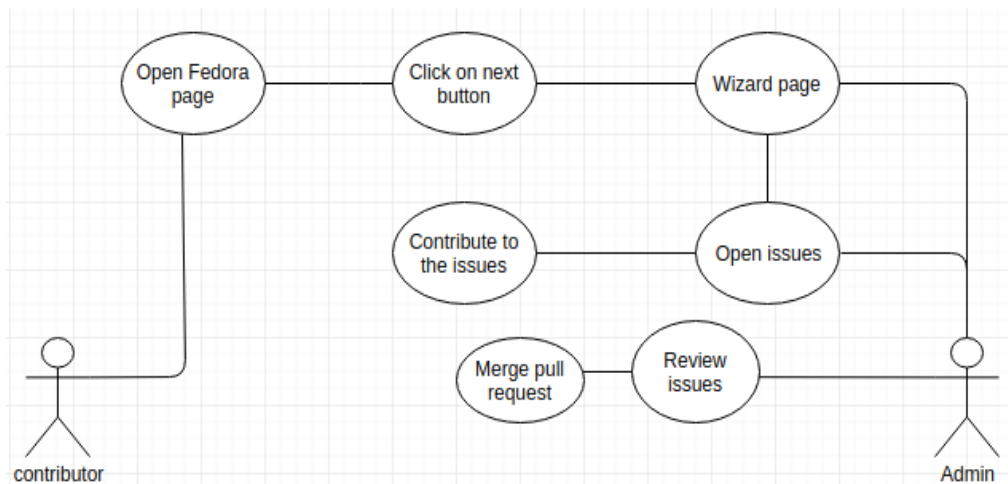


Figure B-Use Case Diagram

APPLICATION ARCHITECTURE DIAGRAM

An application architecture describes the patterns and techniques used to design and build an application. The MERN[4] application can be divided into a 3-tier system having Client Tier, Business Logic Tier, and Database Tier. Client Tier is the front end of the application. Business Logic is the backend of the application

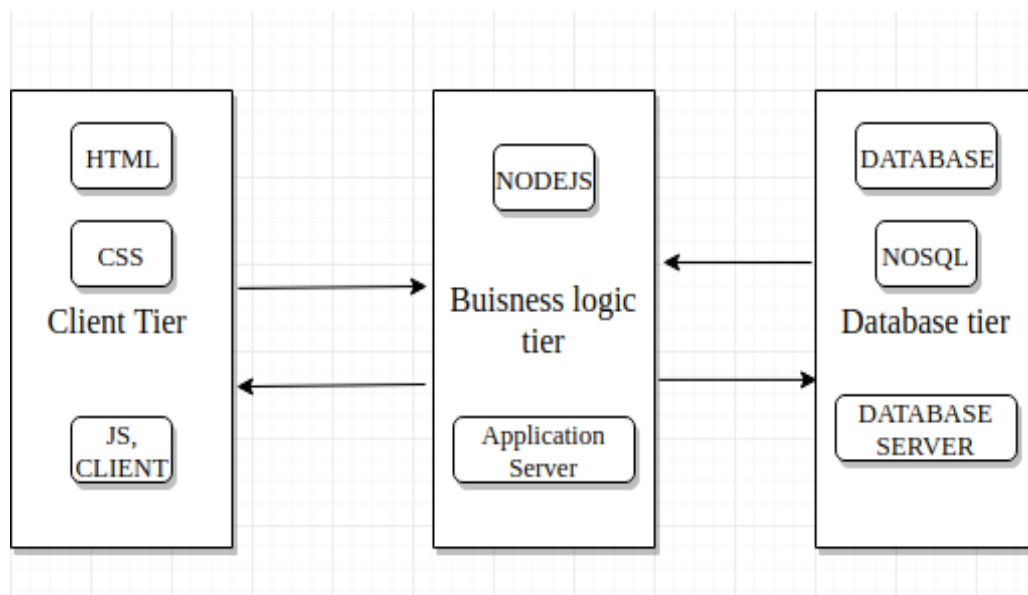


Figure C- Application Architecture

DEPLOYMENT ARCHITECTURE DIAGRAM

A Deployment[5] diagram shows how and where the system is to be deployed; that is, its execution architecture. Whenever the source code of the application will be updated on GitHub, a trigger will run on TravisCI to compile the application code. Upon compilation, the code will be tested and if passed then promoted to deployment on the hosting server.

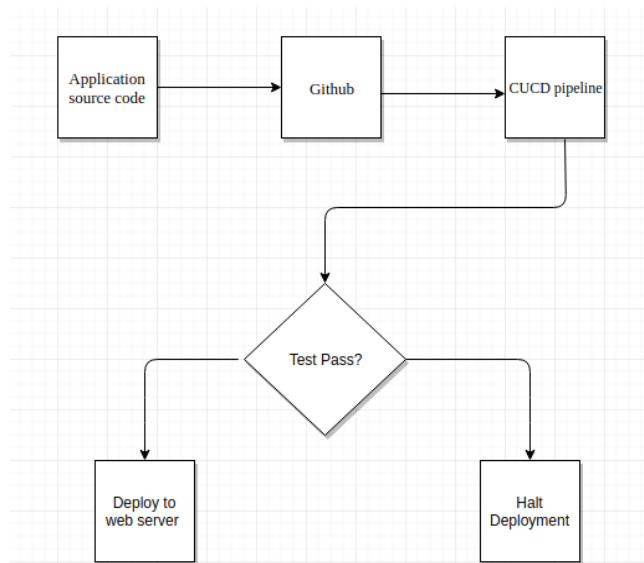


Figure D- Application Architecture

MODULE

An Application Module is a logical container for coordinated objects related to a particular task, with optional programming logic. For Event Base, application is divided into following modules:

Analytics Module: It is the main feature module of the application. This module deals with services like recording profile visits, maintaining counters, and recording user behavior on the application. Data from Analytics modules is persistent. It will also show analytics in the form of charts.

Dashboard Module: It includes features of user profile management. It helps to maintain user data. It acts as an interface for users and admin to manage subscriptions and view account related information from a single control panel. For users with admin privileges, user management is also an added feature handled by Dashboard Module.

User Module: It implements features of authentication and authorization. This module is responsible for user login management, user registration and lost password retrieval.

Server Module: It includes an interface for storing information in the database with help of restful APIs. Every information displayed on the frontend is fetched from the Server Module of application and every update is done by the server module only

Application Modules - An Application Module is a rational vessel for correlated objects related to a particular task, with voluntary programming logic. Application is divided into following modules:

SOURCE CODE

reduxActions.js component

```
import ActionTypes from '../constants';
export const loadDataResp = payload => ({
  type: ActionTypes.LOAD_DATA_RESP,
  payload: payload
})
export const loadData = payload => dispatch => {
  dispatch({
    type: ActionTypes.LOAD_DATA,
    payload: payload
  });
  fetch(window.env.ORACULUM_API_URL_v1 + "landing_page").then(blob =>
  blob.json()).then(data => {
    dispatch(loadDataResp(data))
  })
  .catch((error) => {
    console.error('Error:', error);
  });
}
export const loadWizardDataResp = payload => ({
  type: ActionTypes.LOAD_WIZARD_DATA_RESP,
  payload: payload
})
export const loadWizardData = payload => dispatch => {
  dispatch({
    type: ActionTypes.LOAD_WIZARD_DATA,
    payload: payload
  });
  fetch(window.env.ORACULUM_API_URL_v1 + "actions/all")
  .then(blob => blob.json())
  .then(data => {
    dispatch(loadWizardDataResp(data))
  })
  .catch((error) => {
    console.error('Error:', error);
  });
}
```

```

    });
}
export const changeLanguage = payload => dispatch => {
  dispatch({
    type: ActionTypes.CHANGE_LANGUAGE,
    payload: payload
  })
}

```

Index.js components

```

import ActionTypes from "../constants"
import i18n from 'i18n-js'
import en from '../translation/en'
import fr from '../translation/fr'

const defaultState = {
  landing_page: {
    blockerbugs: {},
    devel: 0,
    meetings: [],
    schedule: [],
    last_qa_meeting: {},
    stable: 0,
    config_mode: false,
    enabled_components: ["events", "blockers", "minutes"],
  },
  wizard: {
    actions: [],
    providers: [],
    all_actions: [],
  },
  locale: {
    i18n: i18n
  }
}

```

```

export default (state = defaultState, action) => {
  switch (action.type) {
    case ActionTypes.LOAD_DATA_RESP:
      return {
        ...state,
        landing_page: action.payload,
      }
    case ActionTypes.LOAD_WIZARD_DATA_RESP:
      return {
        ...state,
        wizard: {
          ...state.wizard,
          providers: action.payload.providers,
          all_actions: action.payload.actions,
        },
      }
    case ActionTypes.CHANGE_LANGUAGE:
      let locale = action.payload
      localStorage.setItem('locale', locale)
      i18n.locale = locale
      i18n.fallbacks = true
      i18n.translations = { en, fr }
      let newI18n = { ...i18n }
      return {
        ...state,
        locale: {
          ...state.locale,
          i18n: newI18n
        }
      }
    default:
      return {
        ...state,
      }
  }
}

```

LandingPage.js

```
import React, { Component, useState } from "react"
import { Container, Row
,DropdownToggle,DropdownItem,DropdownMenu,UncontrolledDropdown} from
"reactstrap"
import { Link } from "react-router-dom"
import Layout from "../layout/Layout"
import Timeline from "./Timeline"
import Events from "./Events"
import Blockers from "./Blockers"
import Minutes from "./Minutes"
import Hideable from "./Hideable"
import _ from "lodash"
import { connect } from "react-redux"
import { loadData, changeLanguage } from "../actions/reduxActions"
import '../index.css'
import Cookies from "universal-cookie"
class LandingPage extends Component {
  constructor(props) {
    super(props)
    this.cookies = new Cookies()
    this.available_components = ["events", "blockers", "minutes"]
    let enabled_components =
this.cookies.get("landingpage_enabled_components")
    if (enabled_components === undefined) {
      enabled_components = this.available_components.slice()
      this.cookies.set("landingpage_enabled_components",
enabled_components, { path: "/" })
    }
    this.state = {
      blockerbugs: {},
      devel: 0,
      meetings: [],
      schedule: [],
      last_qa_meeting: {},
      stable: 0,
```

```

    config_mode: false,
    enabled_components: enabled_components,
  }
}
componentDidMount() {
  this.props.dispatch(loadData())
  this.props.dispatch(changeLanguage('en'))
}
toggle_config_mode(e) {
  let config_mode = !this.state.config_mode
  this.setState({ config_mode: config_mode })
  e.preventDefault()
}
toggle_component_visibility(e, name) {
  if (name === undefined) {
    e.preventDefault()
    return
  }
  let enabled_components = this.state.enabled_components
  if (enabled_components.includes(name)) {
    enabled_components.splice(enabled_components.indexOf(name), 1)
  } else {
    enabled_components.push(name)
    enabled_components = this.available_components.filter((c) =>
enabled_components.includes(c))
  }
  this.cookies.set("landingpage_enabled_components", enabled_components,
{ path: "/" })
  this.setState({ enabled_components: enabled_components })
  e.preventDefault()
}
render() {
  const available_components = {
    events: <Events data={this.props.meetings} />,
    blockers: <Blockers data={this.props.blockerbugs}
release={this.props.devel} />,
    minutes: <Minutes data={this.props.last_qa_meeting} />,

```

```

}
let components = _.chunk(
  this.state.config_mode ? this.available_components :
this.state.enabled_components,
  2
)
components = components.map((row) => (
  <Row>
    {row.map((item) => (
      <div className="col-md-6">
        <Hideable
          config_mode={this.state.config_mode}
          toggle_handler={this.toggle_component_visibility.bind(this)}
          component_name={item}
          is_enabled={this.state.enabled_components.includes(item)}>
          {available_components[item]}
        </Hideable>
      </div>
    ))}
  </Row>
))
const { i18n } = this.props
return (
  <Layout loading={this.props.stable === 0}>
    <div className="drop-down-container">
      <UncontrolledDropdown setActiveFromChild>
        <DropdownToggle tag="a" className="nav-link" caret>
          {i18n.t('phrases.change_language')}
        </DropdownToggle>
        <DropdownMenu>
          <DropdownItem tag="a" onClick={() =>
this.props.dispatch(changeLanguage('en')) } >English</DropdownItem>
          <DropdownItem tag="a" onClick={() =>
this.props.dispatch(changeLanguage('fr')) } >French</DropdownItem>
        </DropdownMenu>
      </UncontrolledDropdown>
    </div>

```

```

<div className="bluebox">
  <Container>
    <Row>
      <div className="col-md-12">
        <Timeline data={this.props.schedule} />
      </div>
    </Row>
  </Container>
</div>
<Container>
  {components}
  <Row className="padded">
    <div className="col-md-12 text-right">
      <a
        href="#"
        onClick={(e) => this.toggle_config_mode(e)}
        className="badge badge-secondary events"
        role="button">
        {this.state.config_mode
          ? `${i18n.t('phrases.save_changes')}`
          : `${i18n.t('phrases.configure_visibility')}`}
      </a>
    </div>
  </Row>
  <Row className="padded">
    <div className="col-md-12 padded">
      <Link
        className="btn btn-primary btn-block btn-wrap-text btn-lg
active"
        to="/wizard"
        role="button">
        {i18n.t('phrases.do_you_have_some_spare_time')}
      </Link>
    </div>
  </Row>
</Container>
</Layout>

```

```
)  
}  
}  
const mapStateToProps = (state) => {  
  return {  
    ...state.landing_page,  
    ...state.locale  
  }  
}  
export default connect(mapStateToProps)(LandingPage)
```


RESULTS AND DISCUSSIONS

In the end, the page should be a very streamlined, interactive, and even semi-automated tool, the initial implementation will (probably) mostly just be an interactive guide, instructing the user about the necessary steps to take, commands to run, etc. Taking the manual testing as an example, it could for sure report the results for you (e.g. using wikicms), later on, but first the project wants to work on defining, describing and delivering the activities.

The main aim is not necessarily only on new-comers, though. The landing page could show useful information even to veterans. Be it accumulation all your action items from all the different meetings, so you can mark these as done (and meetbot could, for example, sum-up the finished/missing action items from the last meeting in the beginning), or streamlining more complicated processes like determining "What were the changes in rawhide, that made it broken". It is well aware of the fact, that the project is very ambitious, and that many of the scenarios/activities will require additional work to provide metadata - like marking the test cases as "short" or "long", "easy" or "advanced", or tagging the projects in easyfix with the programming language. But it is believed that these changes are beneficial even on their own.

The landing page project also scales nicely in opinion - it can start small, with some hard-coded assumptions/hack, and grow at a steady pace, adding more interesting/helpful features in the future, replacing some of the hacks with proper solutions, and so on. The motto here is "delivery before perfection" (aka "release early, release often") - will accept some (well defined and documented) shortcuts, when it means spending less time on "backend work" and enables us to give actual value fast.

REFERENCES

- [1] J. Yli-Huumo, A. Maglyas, and K. Smolander, “How do software development teams manage technical debt? – An empirical study,” *Journal of Systems and Software*, vol. 120, pp. 195–218, 2016.
- [2] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management,” *Journal of Systems and Software*, vol. 101, pp. 193–220, mar 2015.
- [3] Appelbe B. The future of Open Source Software. *J Res Pract Inf Tech* 2003; 35: 227–236.
- [4] Sandra P. Flexible and extensible object and repository architecture (Fedora). *Lect Notes Comput Sc* 1998; 1513: 41–59.
- [5] Janamanchi B, Katsamakos E, Raghupathi W and Gao W. The state and profile of Open Source Software projects in health and medical informatics. *Int J Med Inform* 2009; 78: 457–472.
- [6] Zhang Z, Katz DS, Merzky A, Turilli M, Jha S, Nand Y. 2016. Application skeleton: generating synthetic applications for infrastructure research. *The Journal of Open Source Software* 1(1):Article 17 [DOI 10.21105/joss.00017](https://doi.org/10.21105/joss.00017).