Christopher Addison (cjteam)
William Cao (pastor13)
David Dupre (djdupre)
Jared Le (jaredtle)
Christine Ta (cta95)

Track Forever

I. Abstract

Track Forever is an open-source issue tracker implementation created to provide better support for offline and locally hosted tracking of issues. The implementation also focuses on providing support for importing issues from current industry standard issue trackers and exporting to a human readable standard format on disk.

The application is modeled after other industry web apps, in order to take advantage of the cross-platform support offered by browser technology. In addition, this model allows Track Forever to support an intuitive user interface that can easily adapt to display the various features used in other industry standard issue trackers.

II. Introduction

Issue tracking software is essential to any team's development process. For instance, trackers help teams record and organize software bugs, and assign tasks to team members. Though there are many tools available for issue tracking today (see Competitors section), there are common limitations that can hinder a team's ability to efficiently track their bugs.

For example, some issue trackers are online only (*e.g.* GitHub issue tracker, JIRA Cloud), and as such require a constant internet connection and are vulnerable to DDOS attacks, such as the attack suffered by GitHub[1] in early 2018. Other issue trackers, such as JIRA and YouTrack, require paid subscriptions to support more than a handful of users. However, free and locally hosted issue trackers are often of lesser quality. Redmine is a useful issue tracker for organization level projects, but it does not allow individual project members to access these issues offline, a feature in high demand for those who are on the move but wish to continue working without paying for mobile internet. One program that does allow for local issue storage is TaskWarrior, but it is not designed for anything more than to-do lists for personal projects. It does not have an intuitive UI, nor does it support team collaboration, nor does it support the Windows operating system.

One feature lacking in popular issue trackers is the ability to import and export between other industry-standard issue trackers. Some issue trackers like GitHub offload the work to their users by supplying an API[2], while others only support specifically structured text formats[3]. Discontinued services, such as Google Code, are especially horrid, as existing issue entries may become difficult to access due to incorrect API documentation and outdated or nonfunctional export scripts.

Unfortunately, it's no surprise that import and export functions are lacking in industry issue trackers. There is no single "standard" format for any individual issue, only common conventions, meaning that developer time has to be spent to convert between any and every pair of issue trackers. For instance, Google Code has an indicator of severity, but GitHub does not. In addition, some fields that are functionally the same (like the summary) have different names or identifiers in issue trackers. An example of this is Google Code's "Summary" field, which may correspond to GitHub's "Title" and "Body" field[4] [5]. Even though it may not require much time for someone to fix this small detail, as the project size and number of issues increases, so does total amount of developer time required to ensure a successful import and export.

The **primary goal** of Track Forever is to serve as a free issue tracker that focuses on the features left to the wayside by other issue trackers, mainly offline access, local hosting, and cross-platform support. In order to ensure that Track Forever is useful and accessible, Track Forever also provides easy-to-use tools to migrate issues from industry standard issue trackers to Track Forever itself, which can then be exported to a single unified file format. In addition, Track Forever enables developers who wish to use Track Forever as a self-standing issue tracker by providing offline access, a user-friendly interface, and a client-server implementation. Moreover, it avoids the pitfalls of each, be it vulnerability to DDOS attacks (such as those suffered by GitHub), the cost of professional offline trackers (like JIRA), or the obtuse usability of other open source projects (like Redmine).

Offline access allows users to access their tracked issues across any number of projects even when disconnected from the Internet. This allows developers to maintain productivity when on the go and improves project resilience in the case that central trackers go down. To further improve productivity, Track Forever has a user-friendly interface for viewing and managing issues, offline or online. This cross-platform web-app streamlines the process of importing issues from other trackers, and provides an intuitive interface for developers who want to update their managed issues. For developers collaborating with others, Track Forever's server implementation provides the option to establish a central repository of issues. The server provides authentication and issue versioning, while still enabling offline access for the clients. Even a single developer will be able to create a central server for their own personal use if they want to take advantage of the extra reliability. Notably, the server is not required for general operation, meaning that users who only wish to take advantage of the import functions will not be required to go through extra steps for a potential one-time operation.

Finally, Track Forever aids developers who need to migrate between issue trackers but are faced with obsolete and fragmented documentation for an API or CSV format. By writing the code to perform a translation ahead of time, Track Forever saves developer time and morale so that it can be spent on more important tasks.

These features will prove instrumental to teams that wish to ensure productivity and resilience for their issue trackers. Project progress will no longer be dependent on online issue tracking services over a stable internet connection, both of which are vulnerable to outside factors.

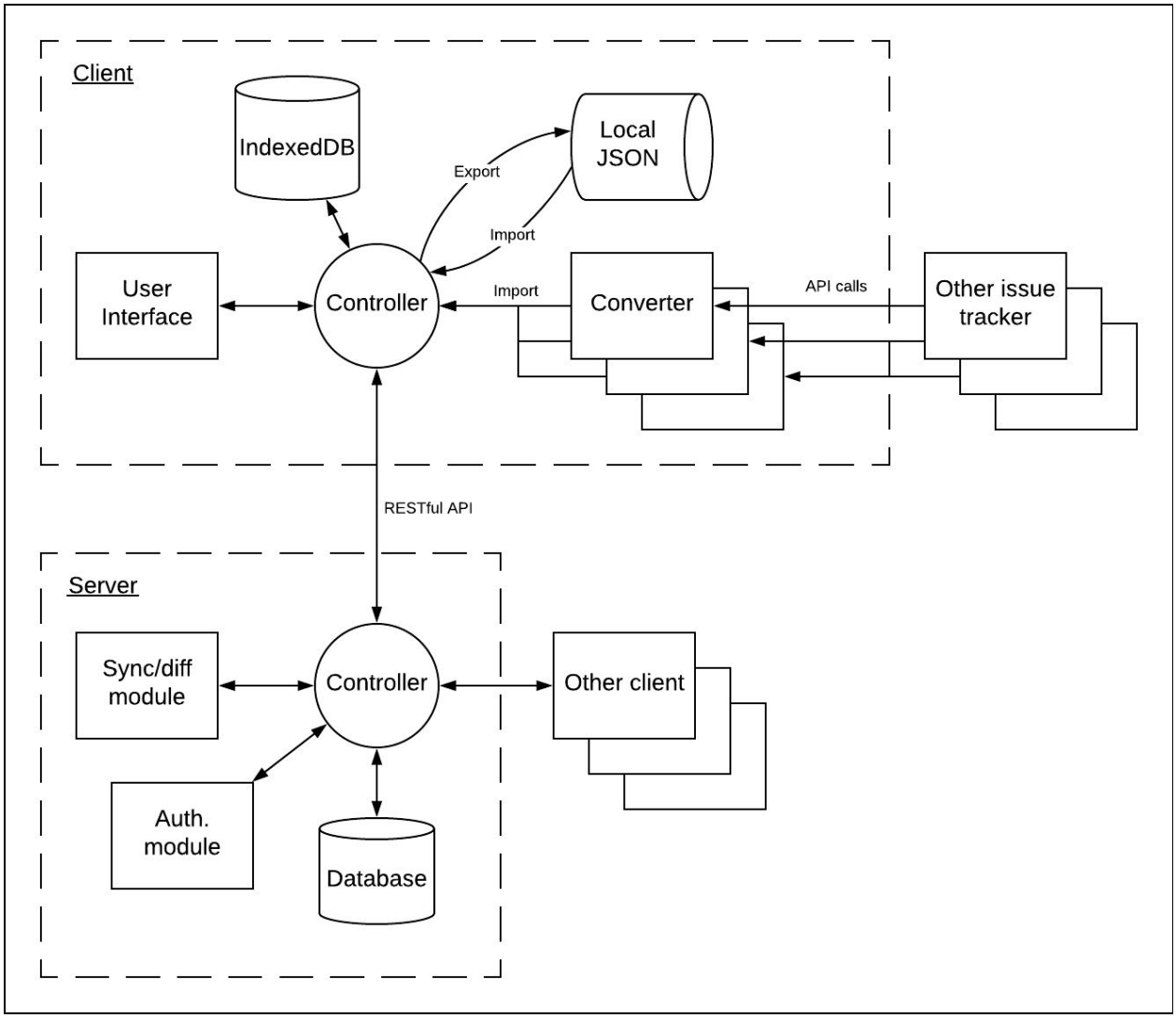
III. Approach

The first step toward creating an industry-compatible issue tracker was to create a unified issue tracker storage format which combined the common features across industry issue trackers. In order to deal with the discrepancies between issue trackers, we translated fields with the same functionality into a single field in our unified file format. For instance, the first comment from one a Google Code project and the “Description” field from GitHub serve the same purpose, and are combined into a Track Forever field called “description”.

In order to perform these conversions to the unified format, we created conversion tools for GitHub, Redmine, and the Google Code. These tools use existing issue tracker APIs to download tracked issues for a specified project. Once downloaded, Track Forever then converts the results to the unified format. The final converted project includes all the information necessary to view the issues, as well as the meta details about the project itself (such as the original source, and users involved). The issue files include fields shared by the issue trackers Track Forever supports: summary, status, assignees, submitter name, comments (each of which is another self-contained structure), labels, and time created/updated/closed.

Track Forever’s main interface to the user is provided via a web application similar to the issue trackers provided by other industry standards, such as Github. Screenshots of the web application may be found below in section III-C.

A. Architecture



(Fig 1.) Architecture diagram

Track Forever has the following architecture, as shown in Fig. 1: Data is taken either from file upload or fetched from another issue tracker API and converted to the Track Forever format using a different conversion tool for each tracker. The work of importing issues is done client-side so that Track Forever is usable with or without a server. Once converted, the issues are stored in a client-side database and are accessible to the user through the web app. Issues can also be imported and exported to the same JSON file format used in the conversion process.

While online, the web app interfaces with a server to create, share, and update issues with other clients. The server handles synchronizing changes between users by keeping its own instance of the issues database with additional data for user authentication. The current revision of the issues

and projects are tracked allowing a user to merge changes when they go back online and synchronize with the server.

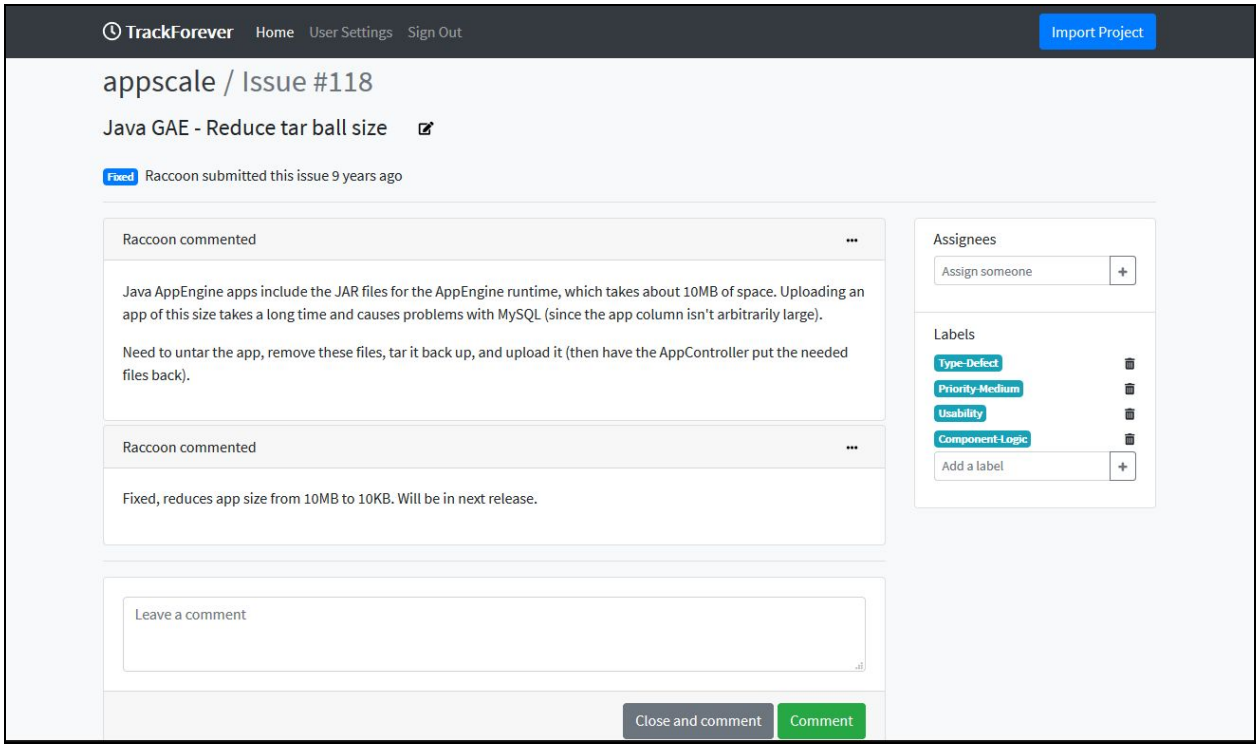
B. Technologies

We chose JSON as our file format for issue tracker data because it is human-readable and has broad support across different languages. These are both qualities that will ease the development of plugins and clients from contributors outside our team. Track Forever uses JavaScript (specifically TypeScript) for these conversions partially because all the supported issue tracks and many others happen to use a JSON format in their API.

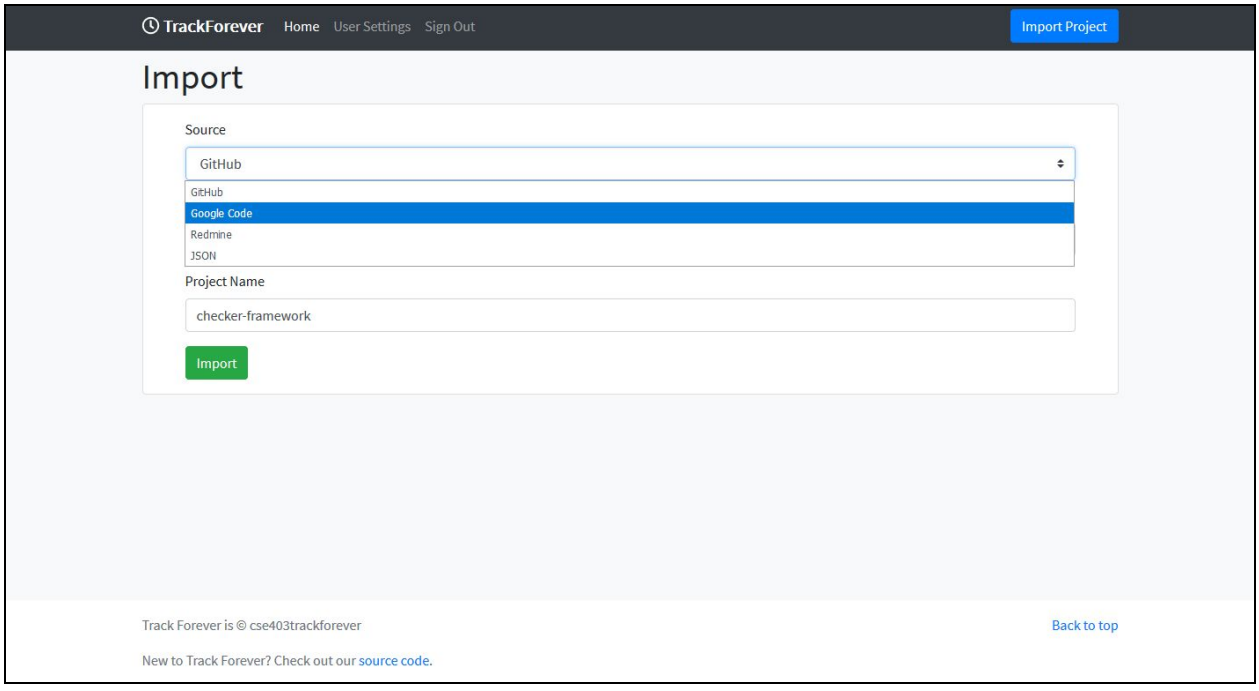
To allow for online and offline support and synchronisation, the current revision is tracked by a hash code, in a manner similar to git. The hash is calculated from the hash of the previous revision and the changes in this revision. For a change to be accepted on the server the calculated hash must match between server and client. If it doesn't, we know that the user has not received the latest changes and will be asked to merge their changes with the ones on the server before resynchronising the updated issue or project. We had momentarily considered using a git-like system. However, deltas in issues are not well represented using line-based diff system, and the overhead for implementing and maintaining such a system is above the scope of this project.

The front-end of Track Forever uses Angular 6[6], a web application framework developed by Google. Angular was chosen over other popular web frameworks (or no web framework) because it is easy to set up with many features including Karma[7] for unit testing and tslint[8] for code style integration with the IDE. Angular also uses TypeScript[9] by default, which helps with code readability and static verification. The web app also uses bootstrap[10] CSS to speed up development, maintain a consistent style, and deliver a responsive website. Track Forever also uses Firebase authentication[11] to enable users to sign in through email, GitHub, and Facebook.

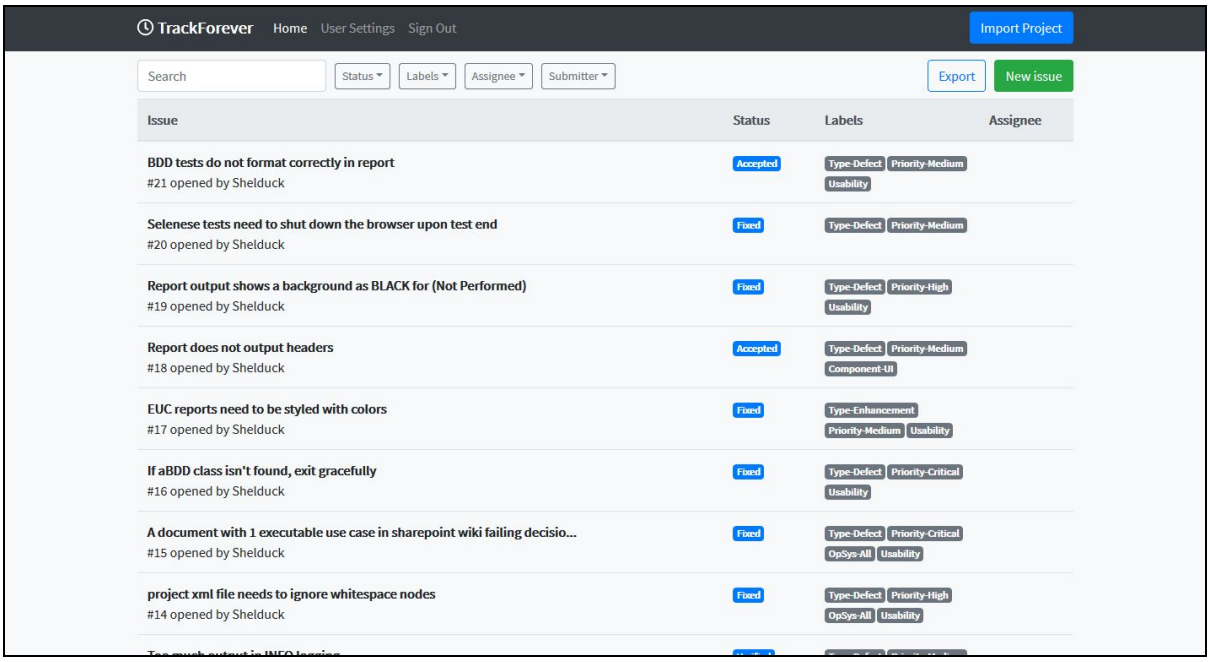
C. User Interface



(Fig. 2) Issue page



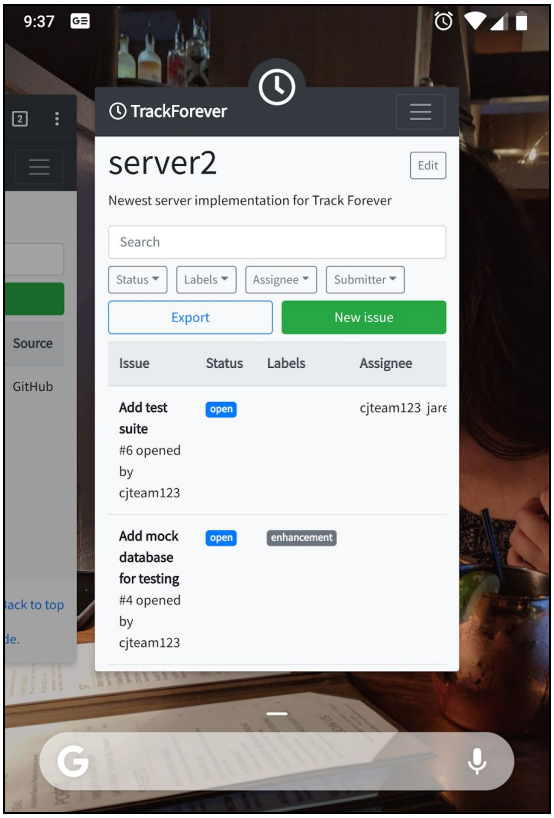
(Fig. 3) Import page



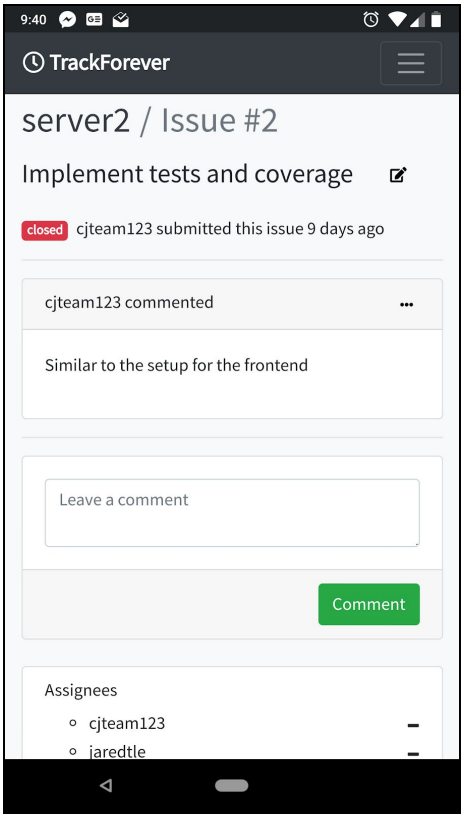
(Fig. 4) Issues list in the project page

New users are greeted with a landing page and a sign in button. After signing in, the user is directed to the My Projects page containing a list of all their projects. The navigation bar lets signed-in users visit their projects page by clicking “Home”, view their settings, sign out, or import a project from another issue tracker at any time (see Fig. 3). Clicking a project will navigate to a project details page, which contains the full project description and a paginated list of issues (Fig. 4). The issue list can be searched and filtered by label, assignee, and submitter. Clicking an issue in the list navigates to that issue’s page which lists comments (see Fig. 2).

Track Forever also supports mobile interfaces. The interface provides the exact same functionality as the main application, as seen in the screenshots below.



(Fig. 5) Issues list as viewed in widget mode



(Fig. 6) An individual issue listing

VI. Competitors

There are plenty of issue trackers in use today, some of which are detailed in Fig. 7. In modern iterations of this type of software, the backbone is typically supported by a popular database program. Although the front-end implementation of these issue trackers vary greatly, many use webapps. What these trackers lack, however, is a comprehensive or standardized method to import issues from other trackers. Moreover, many of these trackers are not accessible offline. This can be an inconvenience for developers that currently do not have a network connection as they will be unable to look up current issues to work on. Our approach is to fill in the usage gaps left by these other programs, allowing for offline issue viewing and better conversion between other issue trackers.

	GitHub	JIRA	Redmine	TaskWarrior	Track Forever
Offline access	No	Via outdated plugin	No	Yes	Yes
Import Function, Number compatible	No. Provides API	Only from specifically structured .csv	Only from specifically structured .csv	Yes, supports 17[12]	Yes, 3 + JSON
Export Function	No, API only	Yes, but does not support large exports[13]	Yes	Yes[14]	Yes
Cross-platform	Yes	Yes, web client and hosted server	Yes, web client and hosted server	No	Yes, web app and hosted server
Supports collaboration (comments, assignees)	Yes	Yes	Yes	No	Yes
Free	Yes	No	Yes	Yes	Yes

(Fig. 7) Issue Tracker Comparison

Some other approaches to this problem have good integration with other issue trackers but are not powerful enough. For example, BugWarrior[15] is an open-source plugin for TaskWarrior[16] that supports imports from 17 different issue trackers, but much of the information is lost in the import process. Unlike Track Forever's format, BugWarrior does not keep track of issue assignees or comments. Track Forever will also have the benefit of a unified format while BugWarrior uses a different format for each supported issue tracker. Lastly, Track Forever runs in the browser, but TaskWarrior lacks support for even the Windows operating system[17].

Track Forever could import from TaskWarrior to gain broad but limited support for the same issue trackers that BugWarrior imports from. This would mean piping issues from the original source

through BugWarrior, then TaskWarrior's JSON export, then into another plugin of Track Forever. Though BugWarrior lacks a unified format, the formats are similar enough that support could be added for more issue trackers with relative ease. However, this plugin has not been implemented in consideration of still significant time and resources necessary to evaluate and test.

There are also proprietary, enterprise approaches (like JIRA[18] and the JIRA-like YouTrack[19]) to unified issue trackers, but they do not have as broad integration with other issue trackers. Even though some issue trackers can export issues into various file formats (.csv, .xml, and .json), there is no universal standard that these issue trackers agree on making it difficult to migrate from one issue tracker to another. By creating a file format that is open-source, we expect that integration will come more easily as more contributors join the project.

VII. Challenges and Risks

There is always the risk that an attempt at a "unified" standard becomes just another competing standard. For this tool to stand out above others, it must support importing from as many issue trackers as possible. Though Track Forever only supports importing from three issue trackers now (Redmine, GitHub, and Google Code), the procedure for integrating additional issue trackers is documented in a contributing wiki[20]. This procedure can and should be made simpler. For example, the implementer should only be required to add the module to at most one list, and the form could be generated dynamically with less boilerplate.

There was also the risk that the coupling between Track Forever's issue format and implementation would make it impossible to implement an alternative issue viewer. This risk was mitigated by being conservative about adding new fields, perhaps with the exception of the hash and prevHash fields.

Another challenge was adapting issue trackers to fit into one format. One problem is that some issue tracker's issues have both a title and descriptions, while other have only an issue title. Track Forever mitigates this by converting any issue descriptions into the first comment's body. Another problem is differences in markdown support between trackers. GitHub uses its own "GitHub flavored markdown", while Google Code uses a reduced version. Track Forever works around this by exposing the issue source to the component displaying the issue so it can apply markdown rules conditionally.

Finally, there are several limitation to Track Forever's approach. Although offline storage is incredibly convenient to offline developers, collaboration functions and user communication can be far more difficult within Track Forever compared to other online issue trackers. In addition, because Track Forever seeks to support as many issue trackers as possible, some of the more nuanced features of supported issue trackers are often lost. For example, GitHub's "reactions" and Redmine's custom fields are not supported in Track Forever's format, though both of these fields are advertised as positive features for their respective trackers.

VIII. Metrics

Conversion quality

An important measure of the compatibility of our issue tracker with another is its ability to convert issues from that issue tracker to ours. For instance, a greater weight is placed on the absence of errors after the import as one of the big goals in Track Forever is to create a program that a developer can trust to port issues without manual editing. We also measure retention of information, user identification, and readability after the import. To measure Track Forever's compatibility with another issue tracker, we import projects from each supported issue tracker and use the rubric below to grade a random sample of imported issues. Sample size is chosen to give 95% confidence of all issues successfully importing with a confidence interval of 15%. We take the average score as a measure of Track Forever's compatibility with other issue trackers.

Test	**Points**	**Grading**
Did the issue successfully import?	-	Pass/fail based on random sampling. If this fails, all other categories receive automatic zeroes. CSV script accepts 0 for missing, 1 for present.
Was all information preserved? ● Time submitted/updated ● Labels, status ● Comment/Summary field contents ● Hyperlinks, embedded objects	4	Partial credit available based on fraction of information preserved. See items listed to the left. One error in any category loses the whole point.
Was user information preserved? ● Submitter/commenter IDs ● Assignee IDs	2	Partial credit, with 1 point for each bullet. One missing ID loses the whole point.
Are there any rendering errors on the webpage? ● No markdown/formatting errors ● No missing character symbols ● All elements resize properly	3	Partial credit, with 1 point for each bullet. One error in any category loses the whole point.
Are there any other pieces of information not preserved? ● Platform specific features (*e.g.* GitHub reactions) ● Other tracker-specific information	2	Remove 1 point for every type of information not preserved, up to 2 points.

As an example, here is a completed evaluation of a Google Code Issue:
Openbookproject Issue #96[21]

Test	**Points**	**Grading**
Did the issue successfully import?	-	Yes
Was all information preserved? ● Time submitted/updated ● Labels, status ● Comment/Summary field contents ● Hyperlinks, embedded objects	4	4
Was user information preserved? ● Submitter/commenter IDs ● Assignee IDs	2	2
Are there any rendering errors on the webpage? ● No markdown/formatting errors ● No missing character symbols ● All elements resize properly	3	3
Are there any other pieces of information not preserved? ● Platform specific features (*e.g.* GitHub emoji) ● Other minor information	2	1 Missing correct generated animal name from Google Code.

Final score: 10/11

Performance

Another aspect of our project that is crucial is the performance. We do not want developers to have to wait a half an hour to convert their issues. However, conversion is usually a one-time task, so we can expect that users will likely tolerate a small amount of wait time (*e.g.* a coffee break). We measure the performance of each conversion plugin in terms of issues converted per second. Measuring this is fairly straightforward. We import projects from several issue trackers from scratch and measure the time (in seconds) it takes to convert all the issues. Then, we use these numbers to calculate the number of issues converted per second for that plugin. This test is repeated for each existing plugin and their corresponding issue tracker.

Web Application Quality

We are using Google's Lighthouse to score the quality, performance, usability, and accessibility of our web application. The test suite is built into Google Chrome developer tools and thoroughly tests the given webpage, following any links and triggering any javascript it can find. Its output is a set of scores for performance, progressive web app, accessibility, best practices, and search engine optimisation. These scores are broken down individually into passing, failing, needs

improvement, and not applicable. For failing and needs improvement results, suggestions and documentation is provided to help fix the issue. While this is not a perfect metric for usability, it does give good insights into areas which we do well at and other that we do not. Additionally, it provides a score for each section to give us a numerical figure for how well Track Forever is doing.

IX. Testing Results

Due to the nature of this project, generating a report requires manual evaluation of performance. However, creation of the final formatted PDF, and all associated figures, is automated as much as possible.

In order to generate results, projects must first be imported into Track Forever. The imported issues must then be compared against the original issues, to look for any missing information, improper rendering, or other issues that may arise. For each compared issue, a rubric is filled out and then added to a comma-separated-value file (CSV). Finally, the automated script tabulates the results and generates a report (the current report can be found on GitHub[22], and a summary table can be found below).

Chosen Projects

In order to test Track Forever's performance, we chose 6 projects, 2 from each issue tracker that Track Forever supports for importing issues. These projects were chosen for their size (being manageably large without years or decades of history), their activity (active projects are preferred, Google Code excepted), and for their potential rigour (heavy use of custom features, formatting, or alternate languages). The projects are:

- Google Code
  - Open-Book Project[23]
  - Appscale[24]
- GitHub
  - Microsoft Dotnet[25]
  - Defects4J[26]
- Redmine
  - Redmine[27]
  - Ruby 1.8[28]

Results

The test results for **Google Code** are the strongest of the supported issue trackers. For both of the projects tested, Open-Book Project and Appscale, every single issue was imported into Track Forever. All information was preserved with the exception of comment submission dates, resulting in a score of 10 and 9.9 out of 11 for Appscale and Open-Book respectively. Readability for Google Code actually improved upon the original Google Code website in some cases, but the markdown renderer was overly zealous in applying hyperlinks in others.

With regard to **GitHub**, an API rate limit prevents Track Forever from importing all issues for a larger project (usually around 1000 issues) resulting in a failure to import the project. However, the Microsoft Dot-Net project fits within the 5000 authenticated requests per hour rate limit and was imported with some success. The most substantial missing information from GitHub imports is references to and from other issues and pull requests. Also missing are comment submission dates, close and open dates (for when an issue is reopened), comment reactions, emojis, title changes, and the time when labels are added to the issue. Comment and issue content was otherwise preserved and well-formatted. Much of these issues also appeared when importing the Defects4J[29] project.

Unfortunately, as for the **Redmine** repositories, both feature at least 2500 issues, and as such are running into the similar issues to GitHub, as the sheer volume of requests tends to cause issues, both in Track Forever and on the remote issue tracker. In addition, it seems that even some of the additional services used to deal with a Cross-Origins issue[30] also has a rate limit of 600 requests per hour[31]. Experimentation with more public repositories[32] [33] hosted on a different domain resulted in an HTTP 500 internal server error on their end after only a couple of requests.

Results Summary Tables

Project Tested	Successful Import	Information Preservation	User Identification	Rendering Errors	Other Unpreserved Information
GitHub: Microsoft Dotnet	100%	94.44%	100%	70.37%	25%
GitHub: Defects4J	100%	100%	100%	81.41%	23.08%
Google Code: Appscale	100%	100%	100%	100%	50%
Google Code: Openbook Project	100%	100%	100%	97.75%	50%

Redmine: Redmine	-	-	-	-	-
Redmine: Ruby 1.8	-	-	-	-	-

Project	Number of Issues	Time (Seconds)	Rate (Issues/Sec)
GitHub: Microsoft Dotnet	217	33 seconds	6.58
GitHub: MineCraft Forge	113	5 seconds	22.6
Google Code: Openbook Project	1201	31 seconds	38.74
Google Code: Appscale	220	23 seconds	9.56
Redmine: Redmine	-	-	-
Redmine: Ruby 1.8	-	-	-

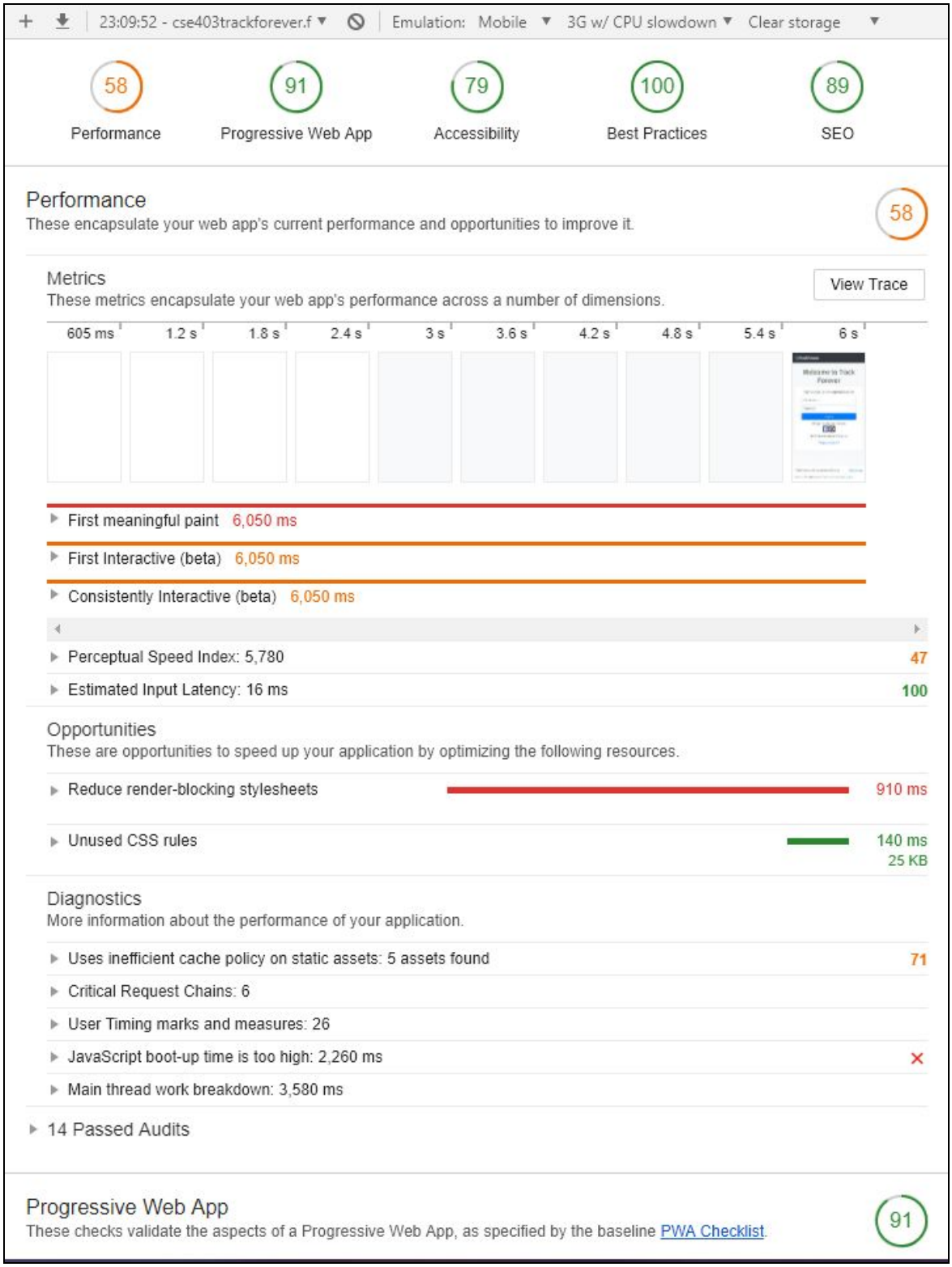
Validity

As with any project, there are several factors that could have influenced the scores shown above. First and foremost, all of the above testing was performed by humans. Comparing the formatting, information preservation, and general success of a conversion is something that only humans can do, given the potential changes in layout and markup that the issues might have undergone. Because these evaluations were all done by seperate humans (working in parallel, but coordinating nonetheless), there is some room for individual bias, group inconsistency, and simple human error. In an attempt to mitigate these possible biases, we came up with subcategories for each of the tests that we will deduct points from if a subcategory was violated. We are very strict and unforgiving when evaluating an issue. For instance, one instance of a violation (e.g. a heading appears incorrectly) results in the complete loss of a point in that subcategory.

Also, because the evaluators are not familiar with the projects they are evaluating, they may assign too much or too little value to information that was lost in the conversion. For example, if one team makes heavy use of a custom Redmine field for their project, then their experience with Track Forever would be far worse than a team that does not use the custom fields.

Lighthouse Report Summary

Google's lighthouse report rates Track Forever fairly well for a progressive web app. Despite some slowdown, presumably due to the sheer amount of data that the app must deal with, the app still manages to respond fairly quickly to user input, while minimizing unnecessary work for the client. The full summary is seen in Fig. 8 below.



(Fig. 8) Lighthouse Report

X. Future Work

At the moment, Track Forever supports importing directly from three issue trackers: GitHub, Google Code, and Redmine. This selection does not include other issue trackers popular in industry, such as JIRA, YouTrack, and Bugzilla. The Track Forever team did not have the time to implement and properly test such converters, although it is possible to implement these conversions at a later date by following the contribution wiki[34].

Endnotes

1. “February 28th DDoS Incident Report.” *GitHub Engineering*, GitHub, 1 Mar. 2018, githubengineering.com/ddos-incident-report/.
2. “GitHub REST APIv3.” *Issues | GitHub Developer Guide*, GitHub, developer.github.com/v3/issues/.
3. “HowTo import issues” *Redmine Wiki*, Redmine, www.redmine.org/projects/redmine/wiki.
4. “Google Code Archive - Long-Term Storage for Google Code Project Hosting.” *Google*, Google, code.google.com/archive/schema.
5. “GitHub REST APIv3.” *Issues | GitHub Developer Guide*, GitHub, developer.github.com/v3/issues/.
6. “Angular Docs.” *Angular Docs*, angular.io/.
7. “Karma.” *Karma*, http://karma-runner.github.io/.
8. “TSLint.” *Palantir*, https://palantir.github.io/tslint/.
9. “Typescript - JavaScript that scales.” Microsoft, https://www.typescriptlang.org/.
10. “Bootstrap.” Bootstrap, https://getbootstrap.com/.
11. “Firebase Authentication.” Google, https://firebase.google.com/docs/auth/.
12. “Bugwarrior.” *PyPI*, pypi.org/project/bugwarrior/.
13. “Exporting Search Results to Microsoft Excel”, *Jira Software Support*, confluence.atlassian.com/.
14. *Taskwarrior - Taskwarrior JSON Format*, taskwarrior.org/docs/design/task.html.
15. “Bugwarrior.” *PyPI*, pypi.org/project/bugwarrior/.
16. “Welcome to TaskWarrior.” *TaskWarrior*, taskwarrior.org/.
17. “TaskWarrior Setup.” *Taskwarrior - What's Next?*, taskwarrior.org/download/.
18. Atlassian. “Jira | Issue & Project Tracking Software.” *Atlassian*, www.atlassian.com/software/jira.
19. “YouTrack: Issue Tracking and Project Management Tool for Developers.” *JetBrains*, www.jetbrains.com/youtrack/.
20. https://github.com/cse403trackforever/trackforever/wiki/Extending-the-import-module
21. https://code.google.com/archive/p/openbookproject/issues/96
22. https://github.com/cse403trackforever/trackforever/blob/master/reports/
23. https://code.google.com/archive/p/openbookproject/
24. https://code.google.com/archive/p/appscale/
25. https://github.com/Microsoft/dotnet
26. https://github.com/rjust/defects4j
27. https://www.redmine.org/projects/redmine
28. https://bugs.ruby-lang.org/projects/ruby-18
29. https://github.com/rjust/defects4j
30. https://github.com/cse403trackforever/webapp/issues/35
31. https://github.com/cse403trackforever/webapp/issues/161
32. http://www.hostedredmine.com/projects/test-credit-system
33. http://www.hostedredmine.com/projects/operation-authority
34. https://github.com/cse403trackforever/trackforever/wiki/Extending-the-import-module