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

Track Forever

Motivation

Issue tracking software is essential to any team's development process. For instance, they 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 not much better. One open-source project that seeks to store issues locally is TaskWarrior, but it does not have an intuitive UI, nor does it support team collaboration, nor does it even support the Windows operating system.

Perhaps the worst-supported feature of 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are especially horrid, as existing issue entries may become difficult to access due to incorrect API documentation and outdated or nonfunctional export scripts.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

[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/.

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 create 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. Creating an easy-to-use tool to migrate issues from many issue trackers to one unified file format will be instrumental to this goal. Track Forever will provide functionality to enable developers who wish to use Track Forever as a self-standing issue tracker, including 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 TaskWarrior).

Offline access to tracked issues will allow users to access their tracked issues across any number of projects when not connected to the internet. This will allow developers to maintain productivity when on the go and improve project resilience in the case that central trackers go down. In order to further improve productivity, Track Forever will also include a user-friendly interface for viewing and managing issues, offline or online. This cross-platform web-app will streamline the process of importing issues from other trackers, and provide an intuitive interface for developers who want to update their managed issues. For developers who are collaborating with others, Track Forever's server implementation will provide the option to establish a central repository of issues. The server will provide 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 will not be 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 will aid 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, we can save developer time and morale so that it can be spent on more important tasks.

We hope these features will prove instrumental to teams that wish to ensure productivity and resilience for their issue trackers. Companies will no longer need to rely on online issue tracking services which are loosely associated with their code, like those of GitHub or JIRA.

Approach

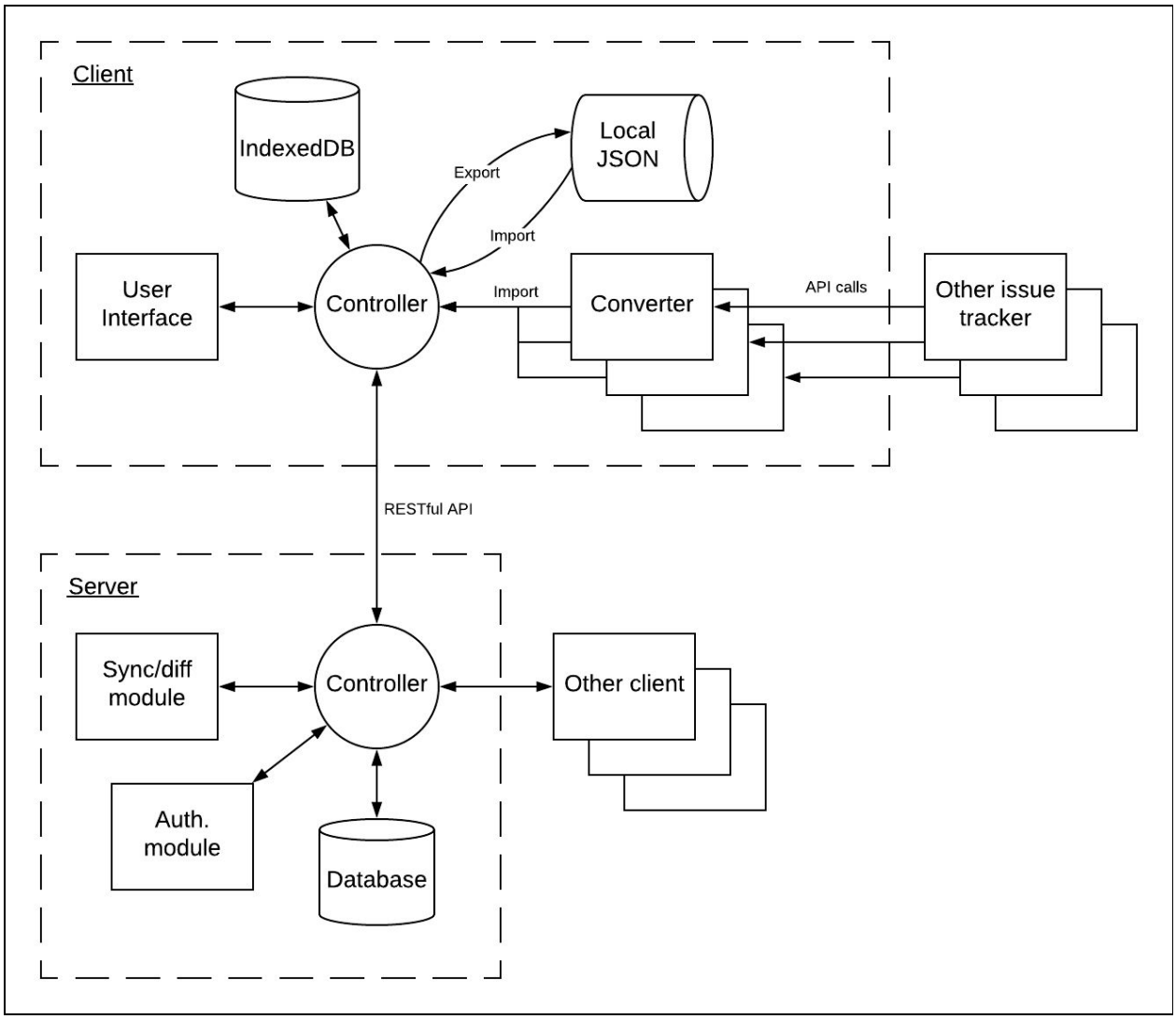
We first created an open-source unified issue tracker storage format that combines the common features across other issue trackers. In order to deal with discrepancies between issue trackers, we will translate fields with the same functionality into a single field in our unified file format. For instance, a “Summary” field from one issue tracker and a “Description” field from another will be combined into a Track Forever field called “description”.

We will also create a conversion tool that will use existing issue tracker APIs to download tracked projects and convert the results into our unified format, which will include a project file and issue files in JSON. The project files include name, owner name, source (e.g. GitHub), description, and a list of project issues. The issue files include fields shared by the most popular issue trackers we plan to support: summary, status, assignees, submitter name, comments (as another short structure), labels, and time created/updated/closed.

Although we are initially building our implementation to import issues from the Google Code Archive and GitHub, we will extend our program to include compatibility with popular issue trackers that have similar API support. For example, we could start with some of the several popular issue trackers like JIRA and YouTrack that offer a RESTful API. This approach will allow code reuse between the support of various issue trackers. We will also focus on building modularity so that other contributors can easily implement their own conversion plugins, allowing for quick conversion to our unified file format. This will simplify the process of migrating from issue trackers that risk being discontinued.

We will create a webapp similar to GitHub’s issue tracker to view issues in our unified format offline. Secondary to viewing imported issues offline, Track Forever will include a server to allow the service to be used as a fully functional issue tracker, meaning that users will be able to share issues online and collaboratively update them.

Architecture



(Fig 1.) Architecture diagram

Track Forever will have 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 our format using a different conversion tool for each tracker. The work of importing issues will be done client-side so that Track Forever is usable with or without a server. Once converted, the issues will be stored in a client-side database and will become accessible to the user through the web app's user interface. Issues can also be imported and exported to the same JSON format used in the conversion process.

While online, the web app will interface 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.

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.

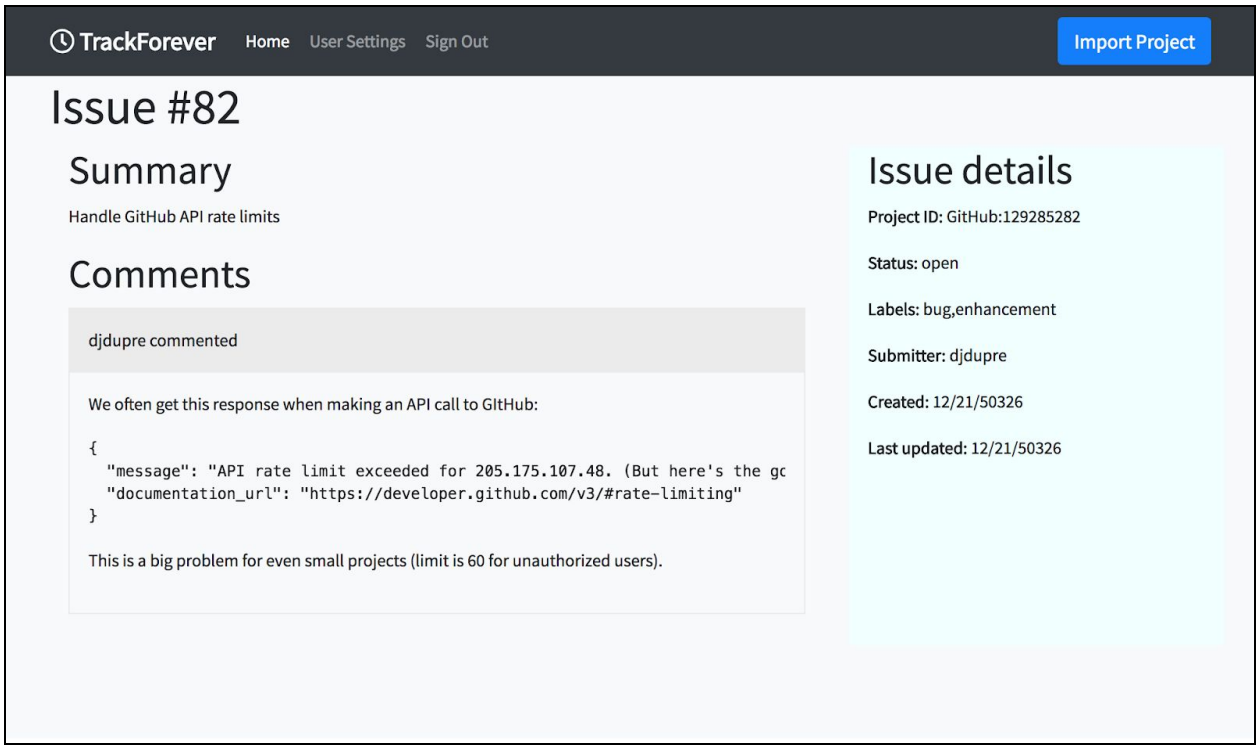
As the issue trackers we plan to support are varied, the technologies we use to consume their API and convert to our format may be varied as well. We chose JavaScript (specifically TypeScript) for these conversions so that our web app will be able to easily call them, without requiring extra setup client-side.

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.

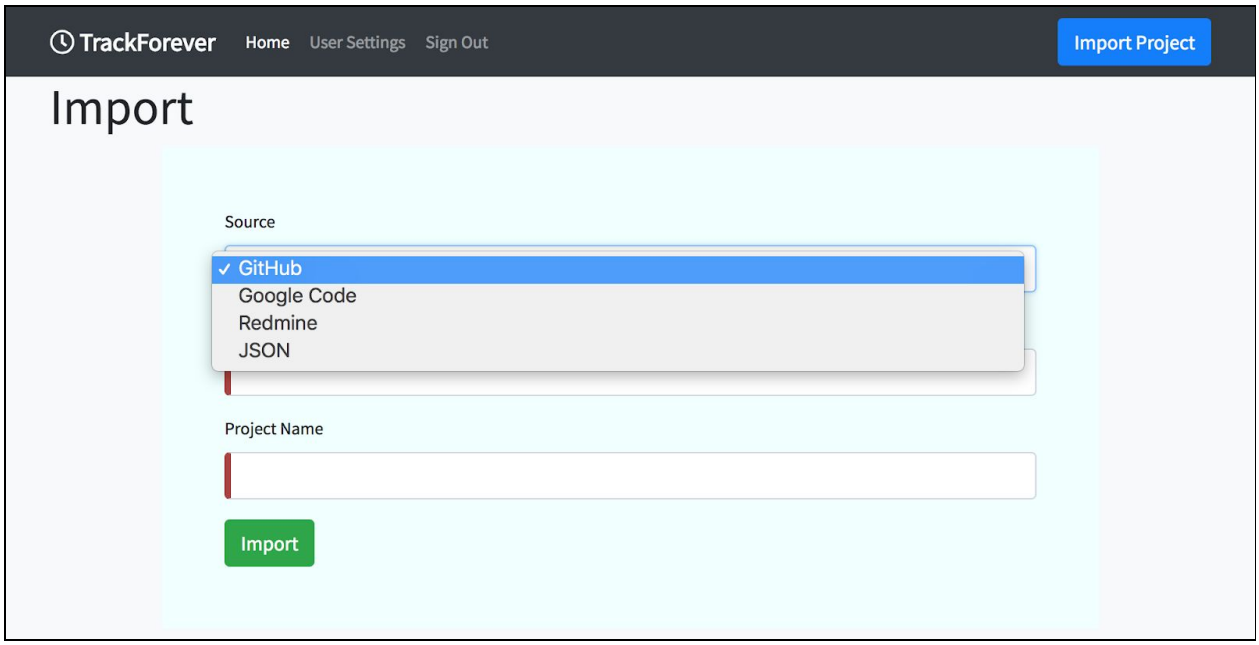
The front-end of Track Forever will use Angular 5[6], a web application framework developed by Google. Angular was chosen over other popular web frameworks (or no web framework) because it is easy to set up with many features including Karma[7] for unit testing and tslint[8] for code style integration with the IDE. Angular also uses TypeScript[9] by default, which will help with code readability and static verification. The web app will also use bootstrap[10] CSS to speed up development, maintain a consistent style, and create a responsive website. We will also be using Firebase authentication[11] to maintain our users. We will allow users to sign in through email, GitHub, Facebook, and possibly other social networks.

[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/.

User Interface



(Fig. 2) Issues details page



(Fig. 3) Import page

The webapp will use bootstrap to create a simple, responsive UI, similar to GitHub's. After successfully signing in, the page will be directed to the My Projects page. This page contains a list of all projects imported into our issue tracker. Clicking a project will navigate to a project details page, which contains a list of issues. Issues can be expanded to view issue details (see Fig. 2). There is an import button that will allow the user to import a new project (see Fig. 3). This is only an initial website design, so we expect UI elements and features (navigation bar, filtering and sorting, buttons, icons, colors) to change as we continue implementing our tool.

Competitors

There are plenty of issue trackers in use today, some of which are detailed in Fig. 4. 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, but only to limited formats	Yes, but no unified format [14]	No
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. 4) Issue Tracker Comparison

[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.

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, in consideration of still significant time and resources necessary to evaluate and test this plugin, our team will not be pursuing this option.

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.

Eventually, as Track Forever progresses, we can begin to optimize and address problems that affect issue trackers. For instance, research papers from within the last five years suggest that a number of issues submitted into modern issue trackers were actually duplicates[20]. We could improve the webapp with a reminder that appears before a user submits an issue to check or search for possible duplicates or related issues that have been previously submitted, as this is shown to decrease the number of duplicate issues within a project[21].

[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] Gu, H., L. Zhao, and C. Shu. *Analysis of Duplicate Issue Reports for Issue Tracking System*, 2011*, SCOPUS*, www.scopus.com.
[21] Petra Heck and Andy Zaidman. 2013. An analysis of requirements evolution in open source projects: recommendations for issue trackers. In *Proceedings of the 2013 International Workshop on Principles of Software Evolution* (IWPSE 2013). ACM, New York, NY, USA, 43-52. DOI=http://dx.doi.org/10.1145/2501543.2501550

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, we need to support as many issue trackers as possible. We plan to overcome this risk by prioritizing tools for adding support for other issue trackers. This will enable ourselves and other contributors to easily integrate other issue trackers, increasing the number of use cases (and therefore our project’s appeal). By building a simple format for tracked issues that is strictly independent of our program, we will also allow others to build their own implementations of the issue viewer, encouraging participation and growth in the project.

Another challenge will be adapting issue trackers to fit into one format. Although most issue trackers share fields such as title, number, and description, many have custom fields and text features that are difficult to reimplement, let alone maintain across a conversion. In some cases, mapping between each custom field and our own implementation can be made for each conversion module. However, in some cases we cannot make the translation, and some information may be lost.

Metrics

Conversion quality

An important measure to 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 will be 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. Other categories we plan to measure include retention of information, user identification (if that feature exists in the original issue tracker), and readability after the import. To measure Track Forever's compatibility with another issue tracker, we can, for instance, import projects from GitHub and use the rubric below to grade each imported issue. Afterwards, we can take the average result and use that as a measure of compatibility and quality with our issue tracker.

An example rubric:

Test	Weight	Pass/Fail
Was important information preserved? (Not including user information)	2	
Are there any visible errors in the final converted text?	1	
Can the submitter, commenter, and assignees all be identified (if they exist)?	1	
Is the issue readable?	1	

Breadth of support across issue trackers

Not only is the conversion quality important to Track Forever, but also the breadth of support provided by it. To measure such a thing, we can count the number of issue trackers that are supported by our issue tracker. Another measure of support is the percent market share that is supported. For example, if we support JIRA and YouTrack, we can say that we support around 55% of the combined market share of issue trackers[22]. Obviously, the higher the combined market share that we support, the potential success of Track Forever will be greater since it will available for more users.

[22] "Issue Tracking Tools Report 2016 - YouTrack." *JetBrains*, www.jetbrains.com/youtrack/promo/tracking-tools-report-2016/.

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 will measure the performance of each conversion plugin in terms of issues converted per second. Measuring this is will be fairly straightforward. We will grab several large projects with at least 100 issues 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 will be repeated for each existing plugin and their corresponding issue tracker.

Browser support

The users of our program will spend most, if not all their time interacting with this program through our web app. Because of this, compatibility and usability among popular browsers including but not limited to Google Chrome, Mozilla Firefox, Microsoft Edge, Internet Explorer, and Safari are essential to Track Forever. By supporting a greater number of web browsers, we can reach out to users of different operating systems - mobile and desktop. There is a possibility that each browser will have a tier of usability. For instance, a web browser that can import to our issue tracker successfully but cannot view the issues offline will be put into a lower tier than a browser that can both import issues and view those issues offline.

Example Rubric for a browser:

Test	**Weight**	**Pass/Fail**
Can this browser import issues to our issue tracker? If so, how many?	2	(Number of issues trackers that can be imported successfully) / (total number of issue trackers)
Can this browser be used to view issues offline?	1	

Example score for this: (4 / 5) * 2 + 1 = 2.6 out of the total 3

Justification

Even though this rubric may skew the percentages towards a higher score, a higher score such as 80% may not be representative of a good conversion. For instance, our first rubric for conversion quality may get an average of around 80% for all issues even though all issues imported may not be readable. The rubrics are still a work in progress, but perhaps another metric that can be measured alongside the conversion quality rubric is a percentage for each category for a specific issue tracker. This supporting rubric will provide greater accuracy as to which aspects Track Forever performed well or poorly in.

Example Rubric for an issue tracker:

Test	Percentage of Total Issues
Did the issue maintain all content?	80%
Is the issue absent of errors?	76%
Can the submitter, commenter, and assignees all be identified (if they exist)?	90%
Is the issue readable?	97%

Initial (Week 7) 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.

During Week 7, we conducted an initial test of the automated report generation (the full report can be found on GitHub[23], and a summary table can be found below). Results were mixed but promising, and varied by the specific issue tracker tested. For example, we imported issues from 2 projects on GitHub (Microsoft Dot-Net[24] and Minecraft Forge[25]), and almost all of the imported information was preserved. However, it was during this testing that we discovered that not only is GitHub rate limited, but also that our code was only importing open issues. This caused many of the issues to not import, which led to our random issue selection often choosing issues that simply didn't import. Since we decided issues that didn't import at all would be given a score of 0, Track Forever ended up receiving low final scores. Among the issues that were actually imported however, the only real issues involved Markdown interpretation and inline image rendering, both of which should be fixable.

Due to a lack of available projects with which to test, Redmine was not tested thoroughly and dummy data was used to populate the report. Furthermore, we are still in the process of debugging the Redmine import. It is unclear whether there is a rate limit for Redmine (since we are making thousands of requests) or authentication is required for us to retrieve issues from their server[26]. However, several Google Code projects were successfully tested. The Open-Book project [27], a project that was primarily in Chinese and contained over 1000 issues, tested extremely well among issues that were actually imported. Small problems found in this process includes incorrect Markdown interpretations by Track Forever and headings that caused adjacent characters to appear incorrectly. Appscale[28] had some rendering issues with the final text, but most other information was preserved. Although there were some issues with retrieving all issues, imported issues had few to no text issues.

[23] https://github.com/cse403trackforever/trackforever/blob/master/reports/week7/
[24] https://github.com/Microsoft/dotnet
[25] https://github.com/MinecraftForge/MinecraftForge
[26] http://www.redmine.org/projects/redmine/wiki/Rest_api
[27] https://code.google.com/archive/p/openbookproject/
[28] https://code.google.com/archive/p/appscale/

Results Summary Table

Project Tested	Info Preserved?	Display Errors?	User ID Preserved?	Successful Import?
GitHub : Microsoft Dotnet	50%	50%	50%	47%
GitHub : MineCraft Forge (only imported issues)	82.76%	79.31%	100%	100%
Google Code : Appscale	27.78%	52.78%	-	38.19%
Google Code: Openbook Project	5.62%	5.62%	-	5.62%
Google Code: Openbook Project (only imported issues)	98.5%	95%	-	100%

NOTE: The Google Code API does not provide any useful user identification across projects[29].

[29] https://code.google.com/archive/schema

Schedule and Milestones

Changes

Since our initial schedule, we've continually updated our schedule as we progress each week. There are a few main differences between our initial schedule and what it is now. Initially, we planned a more aggressive schedule with many implementation milestones each week. We left the later weeks in the schedule with some more flexibility in case we had underestimated the time needed and had to carry things over. This ended up happening, but planning ahead for it definitely helped. In addition, since it was harder to set milestones for the later weeks, we left those weeks to do tests and bug fixing. Instead, we wrote our tests alongside our implementation. Otherwise, we didn't see too many changes other than breaking down tasks or minor tweaks.

Week 1 - Project proposal

✓ Project pitch and team formation

Week 2 - Project proposal

✓ Set up team tools and github repository
✓ Refine project proposal by reviewing initial ideas and approaches, conduct further research on market and competitors, challenges, and goals
✓ Create a weekly project schedule and a scrum-like board for task management

Week 3 - Design and planning

✓ Get and understand all file formats of bug entries from *at least* two issue trackers (considerations include Google Code, JIRA, and YouTrack)
✓ Design a unified file format (fields, representation on disk, etc.) for our issue tracker
✓ Design the backend system architecture
✓ Design a mockup of the UI for the web app

Week 4 - Implementation begins

✓ Design API contracts for interfacing between web app and backend
✓ Finish conversion tools for Google Code and GitHub to our own format
✓ Implement the UI of the web app
✓ Choose a tool to allow offline support for our web app (i.e. a service worker)
✓ Initial user manual draft completed, submitted (Project 4a)

Week 5 - Implementation continues

✓ Translate the two existing conversion tools (Github and Google Code) to Typescript
✓ Finish a conversion tool for Redmine

✓ Refine web app UI
✓ Continue implementing authentication feature
✓ Update tests and set up CI process
✓ Ensure that automated test suite is operational and integrated into build system and CI process
✓ Revise project proposal

Week 6 - Implementation continues

✓ Implement offline support in web app such that the page remains even after refresh
✓ Add Github and email support for authentication feature
✓ Write tests to cover import functions
✓ Revise user manual to match current UI functions
✓ Revise project proposal to address given feedback

Week 7 - Implementation continues, testing and QA

✓ Implement export issues to JSON
✓ Implement user page
✓ Perform and record manual tests for issue conversion quality
✓ Automate tests for other metrics (speed, browser support, etc.)
✓ Create instructions in repository that allow for reproduction of experimental results
✓ Revise project proposal (proj7)
✓ Compile tests into an initial results report (proj7)
✓ Prepare presentation slides

Week 8 (current) - Implementation continues, testing and QA

✓ Finalize presentation slides
- Revise project proposal (proj8)
- Make issues and projects mutable
- Implement the server and REST API, including syncing feature
- Continue testing, polishing, and bug fixing

Week 9 - Testing and report writing

- Continue testing, polishing, bug fixing
- Rerun test suite, identify remaining bugs, add any additional tests
- Draft final report (proj9)
- Begin measuring final performance statistics

Week 10 - Testing and wrap-up

- Conduct a final code review (alongside the ones we've been doing)
- Finish final report (proj10b)
- Finish final presentation slides

Week 11 - Final presentation and report

- Prepare final presentation slides (proj11a)
- Conduct final presentation
- Prepare final report resubmission (proj11b)