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, when services are discontinued (e.g. Google Code), a team's existing issue entries may become difficult to access. When this happens, the developers must now spend their valuable time porting issues from a discontinued issue tracker to their new one. Not only does this waste time, developers may also have issues porting from one issue tracker to another. For instance, Google Code has an indicator of severity, but GitHub does not, which may cause a loss of information (unless they put that information in another field, which has its own limitation, such as sorting). Another limitation of migrating from one issue tracker to another is that some fields that are functionally the same (like a 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[1] [2]. 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.

Although one may think that it is not common for issue trackers to shut down, in 2016 and 2017, both Google Code and Microsoft CodePlex have been shut down. Google Code was discontinued on January 25, 2016 and their tools for exporting no longer work[3] [4]. Because of this, developers that want to grab data from Google Code will have to spend more time reading through Google's API to retrieve projects/issues or rely on unofficial and outdated third party scripts. This is a similar case for Microsoft CodePlex, which ended support on December 15, 2017[5]. Their official migration tool is now considered "historical and no longer works" just 4 months after Microsoft ended support[6]. The tools for these issue trackers are already dysfunctional, making it difficult for developers to port their projects and issues to other issue trackers.

The primary goal of Track Forever is to future-proof a great number of issue trackers so that when an issue tracking service is discontinued, the existing issues remain accessible. Creating an

[1] "Google Code Archive - Long-Term Storage for Google Code Project Hosting." *Google*, Google, code.google.com/archive/schema.
[2] "GitHub REST APIv3." *Issues | GitHub Developer Guide*, GitHub, developer.github.com/v3/issues/.
[3] "Bidding Farewell to Google Code." *Google Open Source Blog*, opensource.googleblog.com/2015/03/farewell-to-google-code.html.
[4] "Export to GitHub – Google Code." *Google*, Google, code.google.com/export-to-github/.
[5] Bright, Peter. "Microsoft Closing down CodePlex, Tells Devs to Move to GitHub." *Ars Technica*, Ars Technica, 31 Mar. 2017
[6] Microsoft. "Microsoft CodePlex WorkItemMigrator." *GitHub*, GitHub, github.com/Microsoft/CodePlexWorkItemMigrator.

easy-to-use tool to migrate issues from many issue trackers to one unified file format will be instrumental to this goal. Not only is this unified file format significant, but speed and efficiency will also play a big role in this project. In addition, Track Forever will also offer offline access to tracked issues across any number of projects and services through the use of a portable user interface. This will allow developers who are on the move without a network connection to still access issues, which increases productivity. Because Track Forever runs locally and is a free and open source software, a company can use this tool on their own servers and extend it as they please. This may prove to be instrumental to productivity and resilience as companies no longer need to rely on a service like GitHub, which has been recently targeted by distributed denial of service attacks[7]. Companies that rely on an online issue tracker like GitHub will have their productivity hampered during such incidents. Another goal for Track Forever is collaboration support. Since team collaboration is a major part of many popular issue trackers like Jira and YouTrack, this project aims to do the same.

Approach

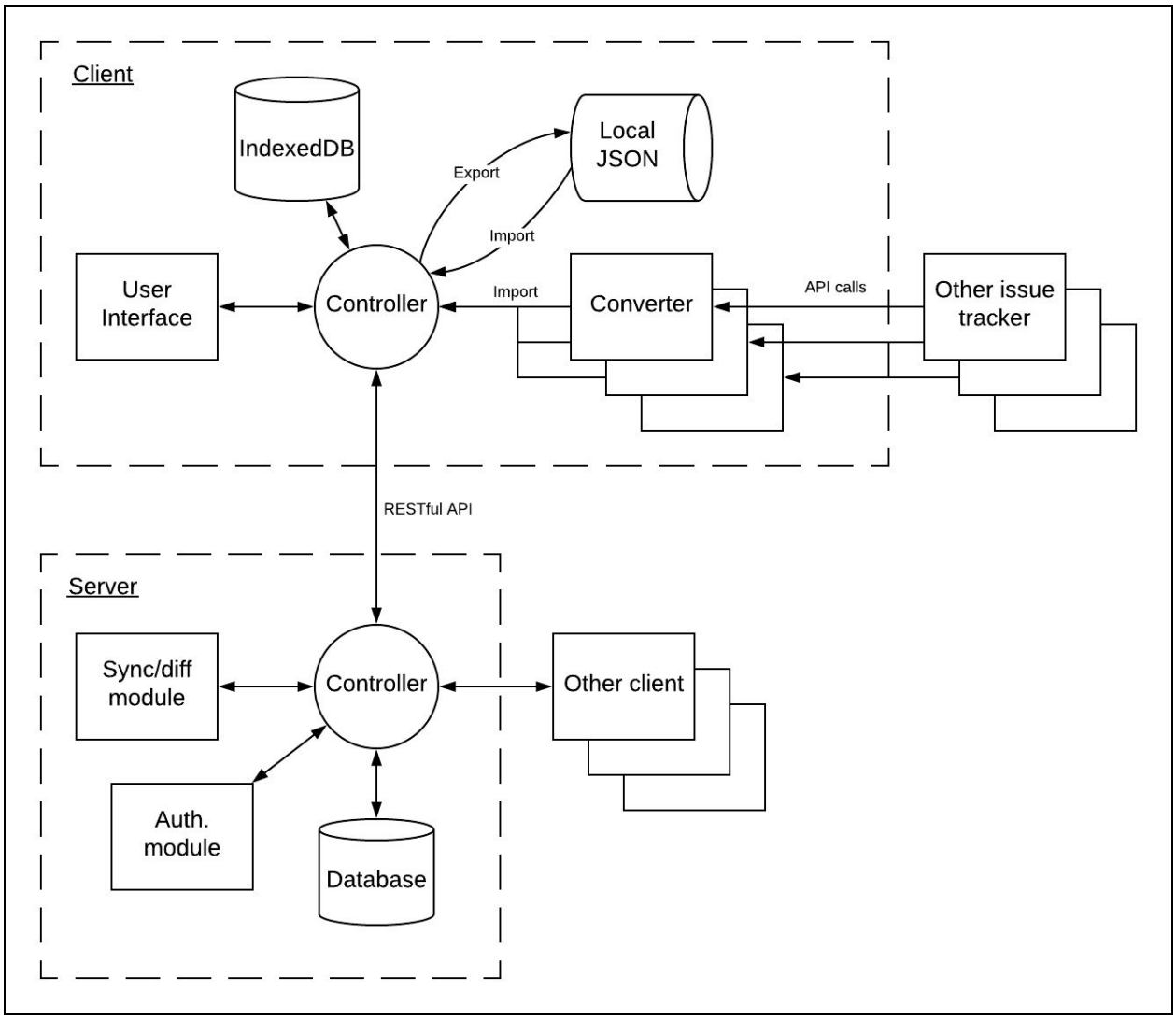
We first created an open-source unified issue tracker storage format that combines the common features across other issue trackers. We will also create a conversion tool that will use existing issue tracker APIs to download tracked projects and convert the results into our unified format, which will include a project file and issue files in JSON. The project files include name, owner name, source (e.g. GitHub), and description. 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 (see fig. 2 below). 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.

[7] "February 28th DDoS Incident Report." *GitHub Engineering*, GitHub, 1 Mar. 2018, githubengineering.com/ddos-incident-report/.

Architecture



(fig 1.) Architecture diagram

Track Forever will have the following architecture. Data is taken either from file upload or fetched from another issue tracker's API and converted to our format using a different conversion tool for each tracker. This critical task 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.

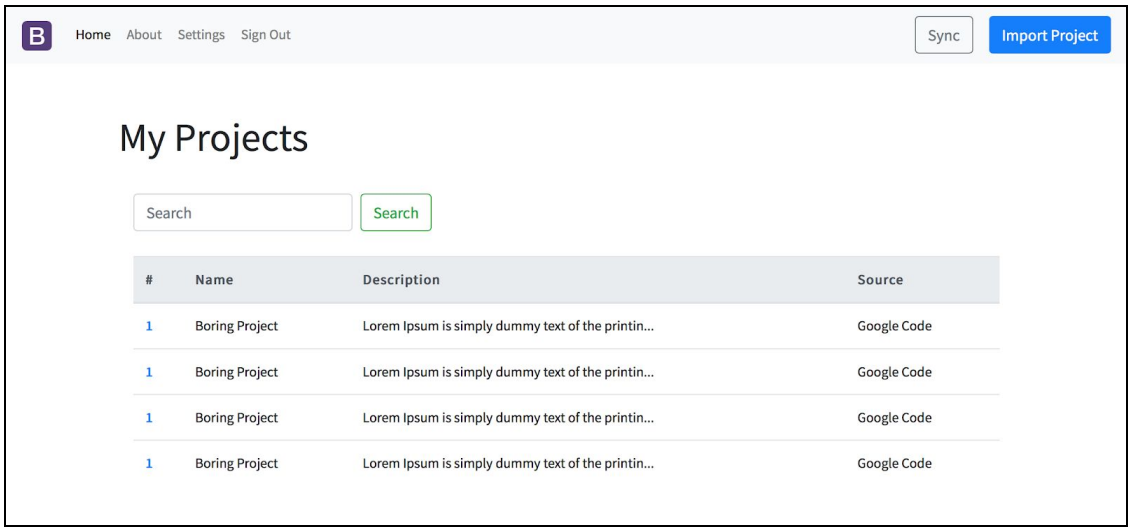
Technologies

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.

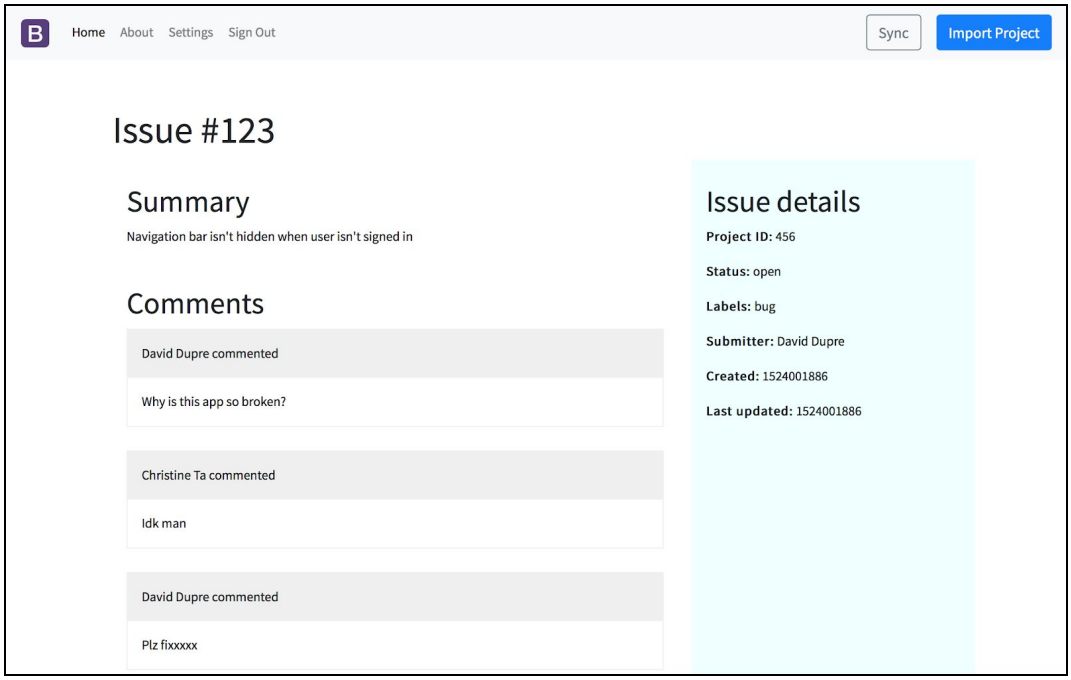
As the issue trackers we plan to support are varied, the technologies we use to consume their API and convert to our format may be varied as well. We chose JavaScript (specifically TypeScript) for these conversions so that our web app will be able to easily call them, without requiring extra setup client-side.

The front-end of Track Forever will use Angular 5, 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 for unit testing and tslint for code style integration with the IDE. Angular also uses TypeScript by default, which will help with code readability and static verification. The web app will also use bootstrap css to speed up development, maintain a consistent style, and create a responsive website. We will also be using Firebase authentication to maintain our users. We will allow users to sign in through email, GitHub, and possibly other social networks.

Interface



(fig. 1) Webapp home page UI, second iteration



(fig. 2) Issue details page UI

The webapp will be similar to GitHub's. After signing in, the home page will be a list of all projects imported into our issue tracker (fig. 1). Clicking a project will navigate to a project details page, which contains a list of the issues. Issues can be expanded to view issue details (fig. 2). There is an import button that will allow the user to import a new project. This is only an initial website design, so UI elements such as the nav bar, the sync button, icons, colors, will change as we continue implementing our tool.

Competitors

There are plenty of issue trackers in use today. In modern iterations of this type of software, the backbone is typically supported by a popular database program. The front-end implementation of these issue trackers vary greatly, although 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 them. 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.

Some other approaches to this problem have good integration with other issue trackers but are not powerful enough. For example, BugWarrior[8] is an open-source plugin for TaskWarrior that supports imports from 17 different issue trackers, but it does not support collaboration and nor

[8] "Bugwarrior." *PyPI*, pypi.org/project/bugwarrior/.

does it run on Windows[9]. We plan to make this project into something that supports teams of developers, greater accessibility regardless of operating system, and a native intuitive interface that users will enjoy. There are also proprietary, enterprise approaches (like Jira[10] and YouTrack[11]) 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 our project 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[12]. 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[13].

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

[9] "Welcome to TaskWarrior." *TaskWarrior*, taskwarrior.org/.

[10] Atlassian. "Jira | Issue & Project Tracking Software." *Atlassian*, www.atlassian.com/software/jira.

[11] "YouTrack: Issue Tracking and Project Management Tool for Developers." *JetBrains*, www.jetbrains.com/youtrack/.

[12] Gu, H., L. Zhao, and C. Shu. *Analysis of Duplicate Issue Reports for Issue Tracking System*, 2011*, SCOPUS*, www.scopus.com.

[13] 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

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 this project 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
Did the issue maintain all content?	1	
Is the issue absent of errors?	2	
Can the submitter, commenter, and assignees all be identified (if they exist)?	1	
Is the issue readable?	1	

Hypothesis for this rubric: average result of 4 / 5.

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[14]. Obviously, the higher the combined market share that we support, the potential success of this project will be greater since it will available for more users.

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. We will measure the performance of each conversion plugin in terms of issues converted per second. Measuring this is will be fairly

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

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 TrackForever 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%

Schedule and Milestones

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 two bug trackers 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 (current) - Implementation continues

✓ Translate the two existing conversion tools (Github and Google Code) to Typescript
- Finish a conversion tool for one more bug tracker
- Refine web app UI
- Implement authentication
- Implement the REST API for interfacing between the webapp and back end
- 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

- Configure web app to call API and display bug entries
- Implement offline support in web app such that the page remains even after refresh

- Refine web app UI, add additional features such as settings and about page
- Revise user manual to accommodate for any potential changes
- Revise project proposal

Week 7 - Implementation continues, testing and QA

- Run tests and document initial project results

Week 8 - Testing and QA

- Continue testing, polishing, and bug fixing.
- If possible, consider working on optimizations.
- Prepare for project presentation

Week 9 - Testing and report writing

- Rerun test suite, identify remaining bugs, add any additional tests
- Begin drafting of final report, also begin measuring performance statistics
- Complete instructions in repository that allow for reproduction of experimental results

Week 10 - Testing and wrap-up

- Automate instructions for reproducing experimental results
- Final Code Review (alongside the ones we've been doing)
- Finish final presentation slides

Week 11 - Final presentation and report