Emergent Architecture Design
Octopeer for Bitbucket

Out of Context

Version 1.0

Date:
17-06-2016

Group members:
Arthur Guijt
4377338
a.guijt@student.tudelft.nl

Lars Stegman
4365801
l.s.stegman@student.tudelft.nl

Laurens Kroesen
4350286
l.kroesen@student.tudelft.nl

Cas Buijs
4345185
s.j.m.buijs@student.tudelft.nl

Thomas Overklift
4080890
t.a.r.overkliftvaupelklein@student.tudelft.nl

Responsible Teacher:
Alberto Bacchelli
a.bacchelli@tudelft.nl

Teaching Assistants:
Bastiaan Reijm
A.B.Reijm@student.tudelft.nl
Aaron Ang
A.W.Z.Ang@student.tudelft.nl

Page Index

1. Introduction

This document provides an architectural overview of the tool ‘Octopeer for Bitbucket’. The document is split up in several sections. First some design goals are discussed. Design goals are guidelines or practices that should be followed or incorporated when building the tool. Satisfying all design goals builds confidence in the fact that the final product can pass quality checks.
In the rest of the document we discuss components that form the system and clarify some design choices.

1.1 Design Goals

The main design goal of the project is satisfying all product quality properties included in the ISO/IEC 25010:2011 standard. This means the system should be:

- Functionally suitable
  - This means the tool should be complete and correct in the sense that the user can rely on the fact that the data that the tool provides is correct and useful (gives insight in pull request statistics).
  - Our tool does not directly provide insights in pull request statistics, but makes use of the Octopeer visualiser which can be found at https://github.com/mboom/TI2806. The visualizer is built into our extension.
- Perform efficiently
  - This means that a user shouldn’t experience any performance loss in the browser (e.g. lagging) when the tool is running, while the tool should at the same time be able to collect enough data to be functionally suitable.
  - Our frame-rate measurements using the chrome performance profiler for the page without and with extension are approximately equal.
- Compatible
  - This means that the tool should be able to work with other software. The tool only needs to work in Chrome (per the non-functional requirements), but it should be able to run at the same time as other Chrome extensions.
  - Extension has been confirmed to work alongside other extensions and on Windows and macOS.
- Usable
  - This means that the tool should have an understandable and easy to use user interface, so that any user can start using the tool effectively without problems.
  - The interface is currently mostly restricted to the settings UI, which is similar to the settings of Android. In the settings page there is a link to the visualiser tool, but that is out of the scope of our extension.
- Reliable
  - This means that the tool should be robust, and crash as little as possible. The tool should also be able to restart, without crashing the operating system, if it does crash.
  - Crashes haven’t occurred so far. Causing a crash with javascript requires severe errors and if such a crash occurs - very unlikely - chrome handles it and is capable of reporting errors to a collector.
- Secure
  - This means that user data should be stored safely, and anonymously if the user so desires.

    - We are not yet storing the user data anonymously as this costs extra development time without the largest extra benefit for the current product.
- Maintainable
    - This means that the tool should be well written, and properly formatted. Because of this, it can be easily maintained or extended.
    - The code is relatively simple and should technically be easy to understand.
- Portable
    - This means that the tool should be able to work on other systems (with little effort). The tool can be ported directly to any machine with the Google Chrome browser. Apart from that it needs to be adaptable in such a way that it can be used in a different browser, without having to rewrite the entire tool.
    - As seen under Compatible, we have confirmed it to work on other operation systems, apart from that, about chrome specific apis, the messaging part is modular and can easily be replaced, the settings part is still hardcoded. Similar apis and extension structure should be available in the target browser for easy porting.

Apart from meeting the design goals that can be extracted from this ISO standard, we also want to focus on code quality and code review in particular. Because the tool itself is meant to analyze pull requests, we aim to use the tool ourselves in order to improve our own code review practices, as well as to improve the tool in turn with the experience gained from using it.

2. Software Architecture Views

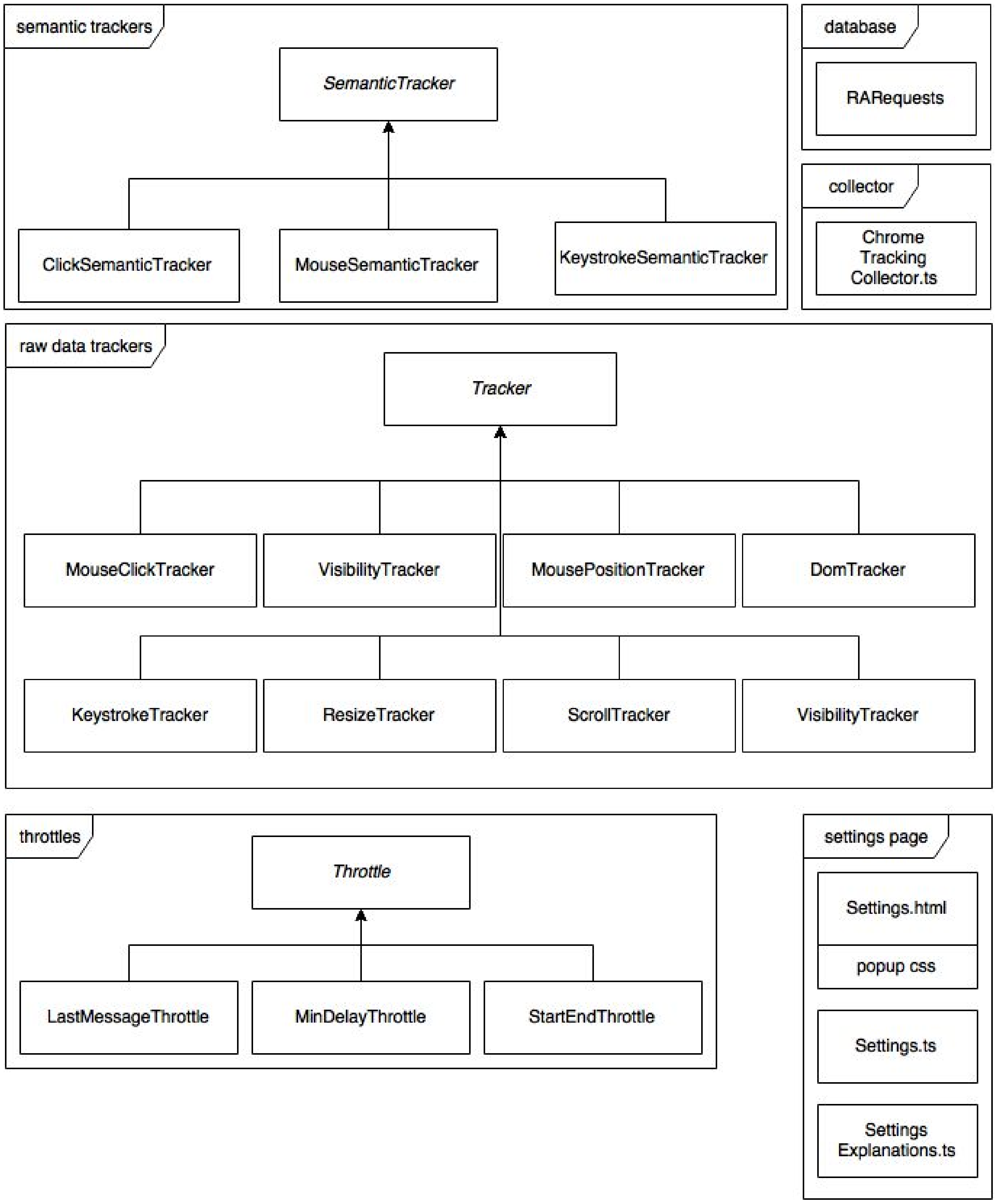

Figure 1, System Composition Diagram

2.1 Subsystem Decomposition

We expect the application to consist of a few components:

- **Trackers** keep track of the events we want data on, for example mouse movement, window resize, keystrokes but also current user. Should rely on having a connector.
- **Collectors** provide a central place for trackers to send their data to. Every page has a collector. The collectors send the collected data to the centralised connector.
- **Connectors** provide a layer between the tracker and the AJAX calls towards the (database) server.
- The **user interface** is the component that the user interacts with, depends on having trackers to enable & disable.
- A possible extra subsystem is the **analytics component** which would provide a service to the user interface (and possibly other components) by providing information about - for example - a certain pull request and simple statistics. Requires a connector to fetch information from a central server or a connector to a local data source.

The octopeer extension collects data about its users when they are on a pull request page. To track and record this data we obviously have trackers on every pull request page. In the figure below we have laid out the architecture of the trackers and the components needed to send the data to the database.

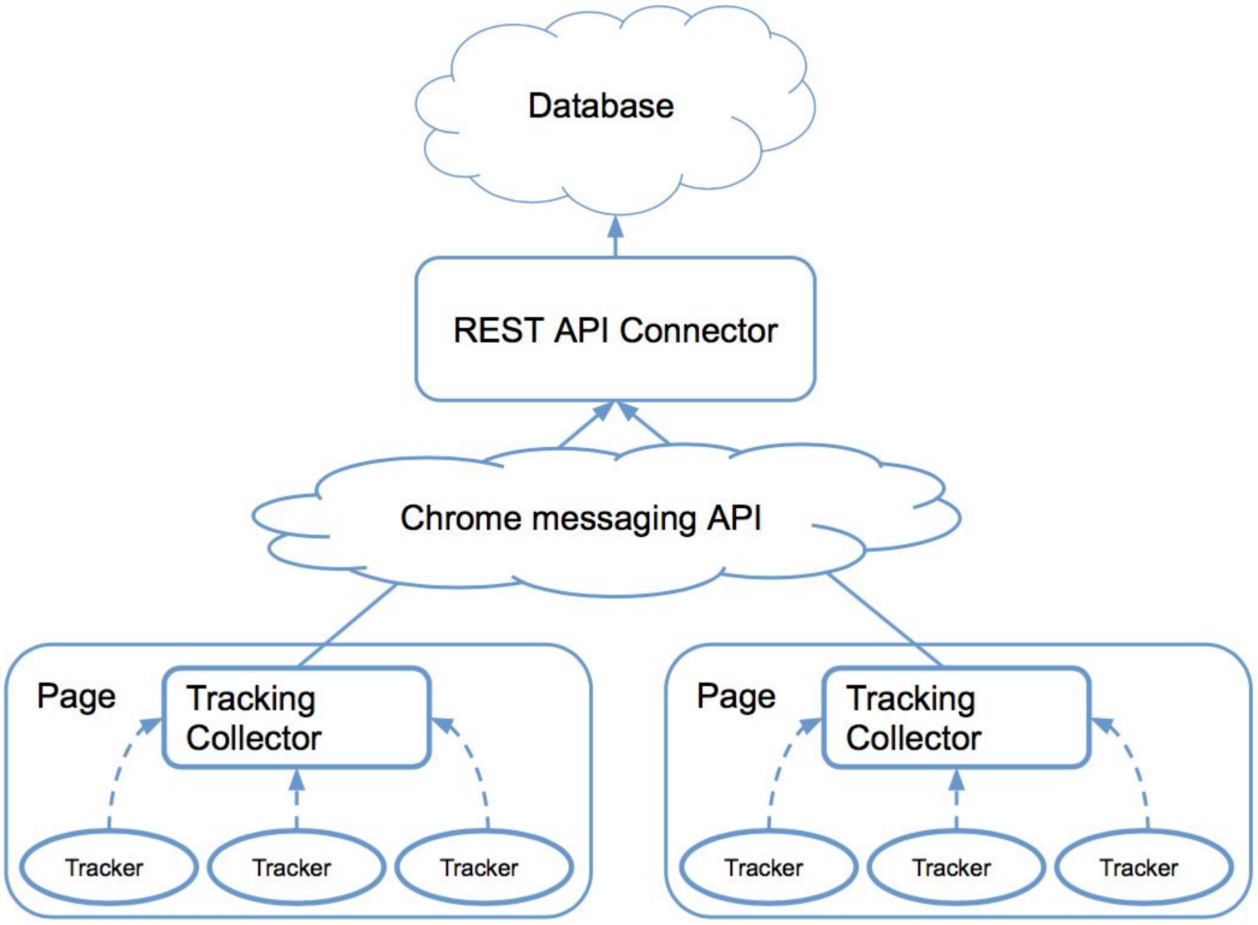

Figure 2, Communication Hierarchy

Every Bitbucket page has dedicated trackers and one Tracking Collector. Every tracker has a reference to the collector it needs to send its data to. When a tracker records a piece of data it is sent to the collector. This is done by the tracker calling a method on the collector with the recorded data as an argument. The collector adds the session data, which is the same for every tracker on the page, to the received data and passes the data to the REST API Connector using the Chrome Messaging API.

Upon initialisation of the REST API Connector it registers itself with the Chrome Message API by adding a listener to it. The listener only responds when a specified id is used to send the message. When the listener is called by the Message API, the connector sends the received data to database using a HTTP request. The database then responds with a message telling whether the procedure has finished correctly.

2.2 Flow Diagrams

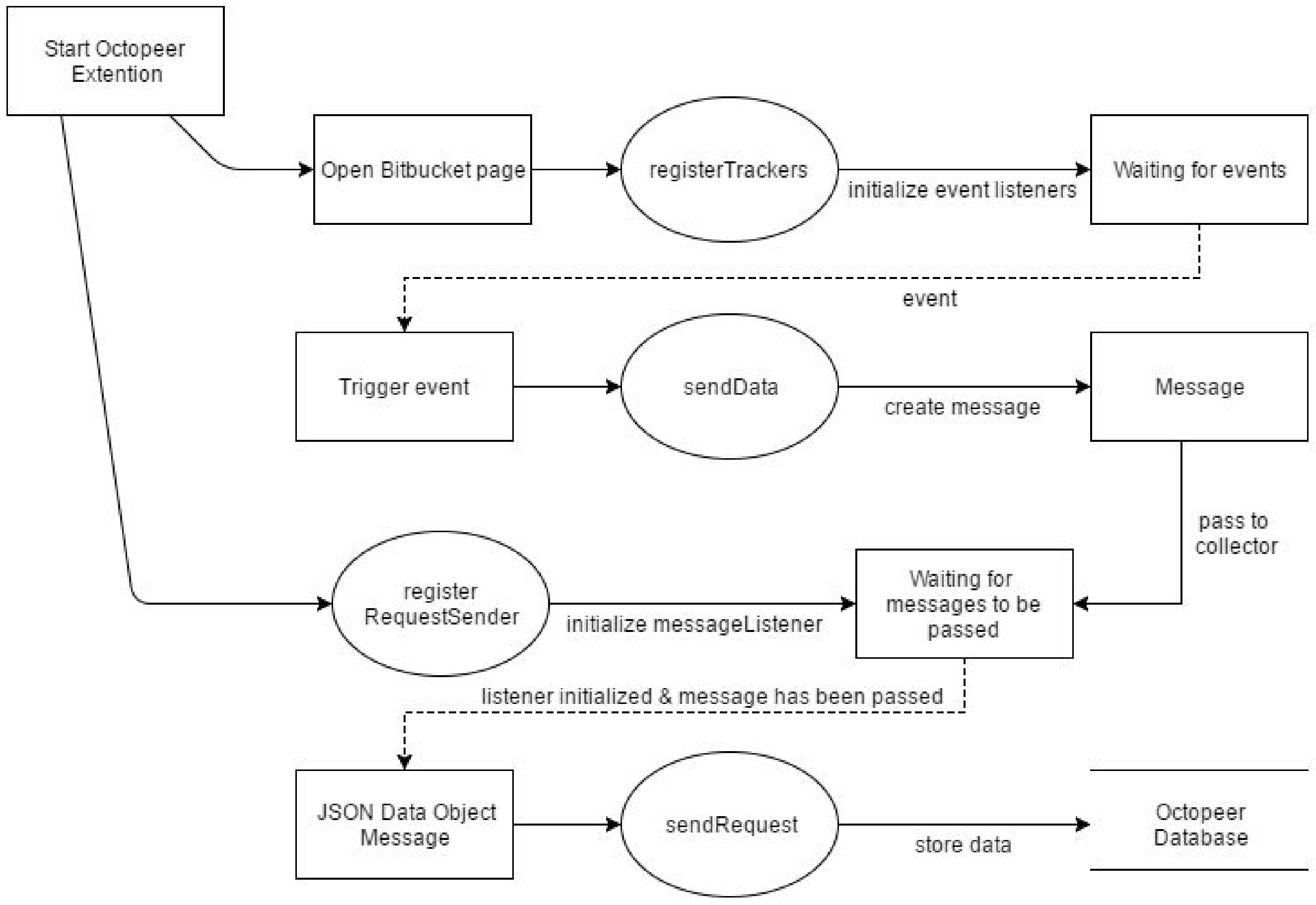

Figure 3, Flow diagram: Pull Request Tracking

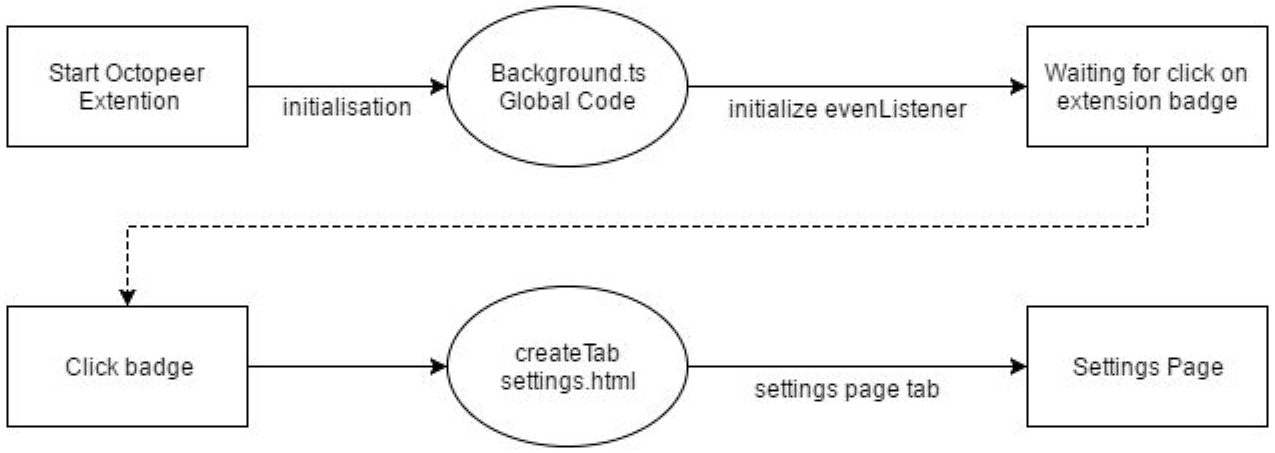

Figure 4, Flow diagram: Octopeer Icon

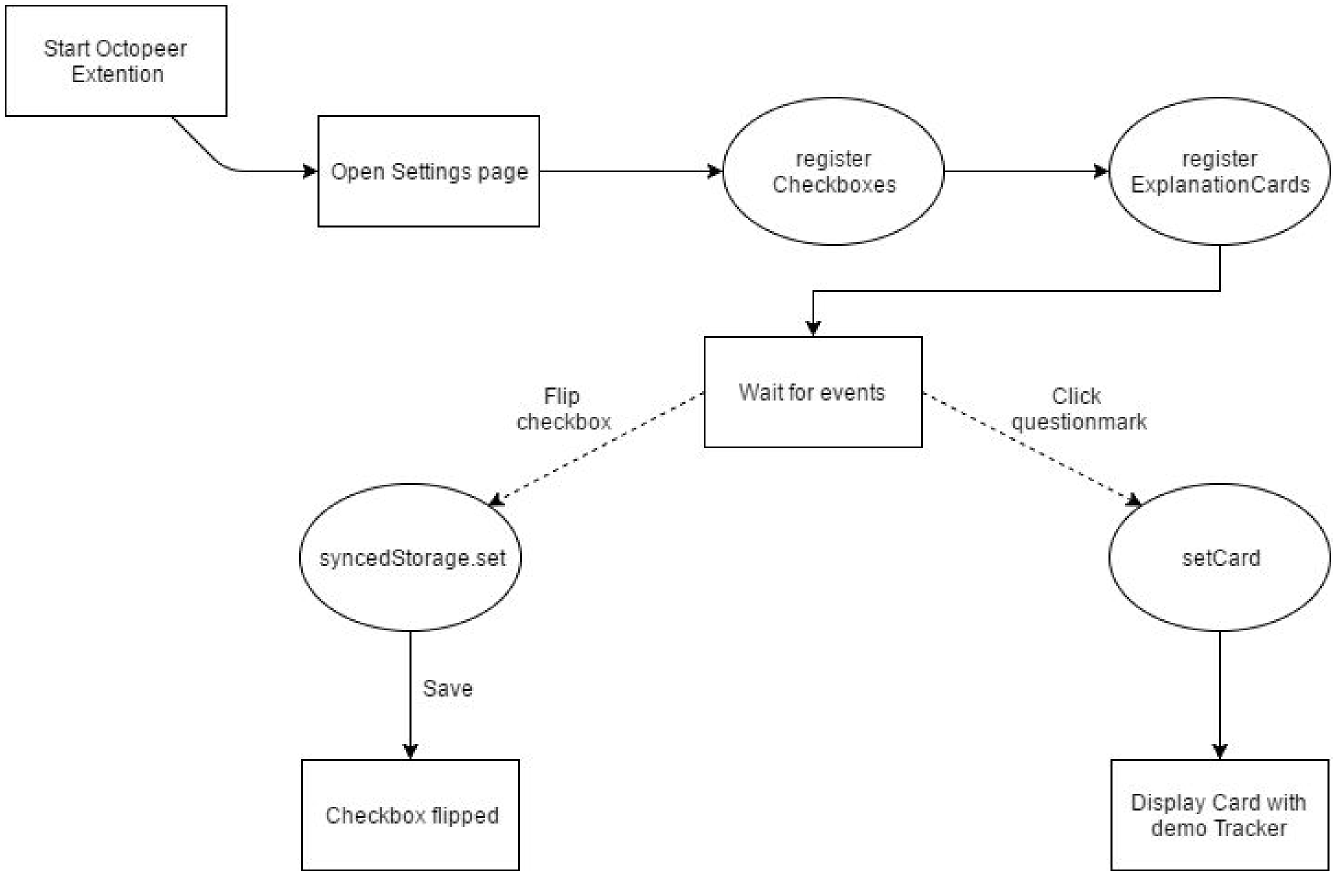

Figure 5, Flow diagram: Settings Page

Figure 3 is a flow diagram that describes the start of the octopeer extension and the basic usage on a pull request page. *Figure 4* is a flow diagram that describes what happens when the octopeer extension is started and the badge in the top right corner of the browser is clicked. This opens the settings page. *Figure 5* is a flow diagram that describes what can happen once the user has opened the settings page.

2.3 Hardware & Software Mapping

This product does not need specific hardware. If the user has Google Chrome as a browser and a host to provide the database hosting, it can start using the extension. An Internet connection to visit Bitbucket is also quite helpful, since the extension is used to look at pull request behaviour on Bitbucket.

The computers only need a connection to the Internet to communicate with the database and Bitbucket. If the database is hosted inside an organization, the computers do not need to connect to an external database. An Internet connection is not optional however, since a connection to Bitbucket is still needed.

2.4 Persistent Data Management, Database Design

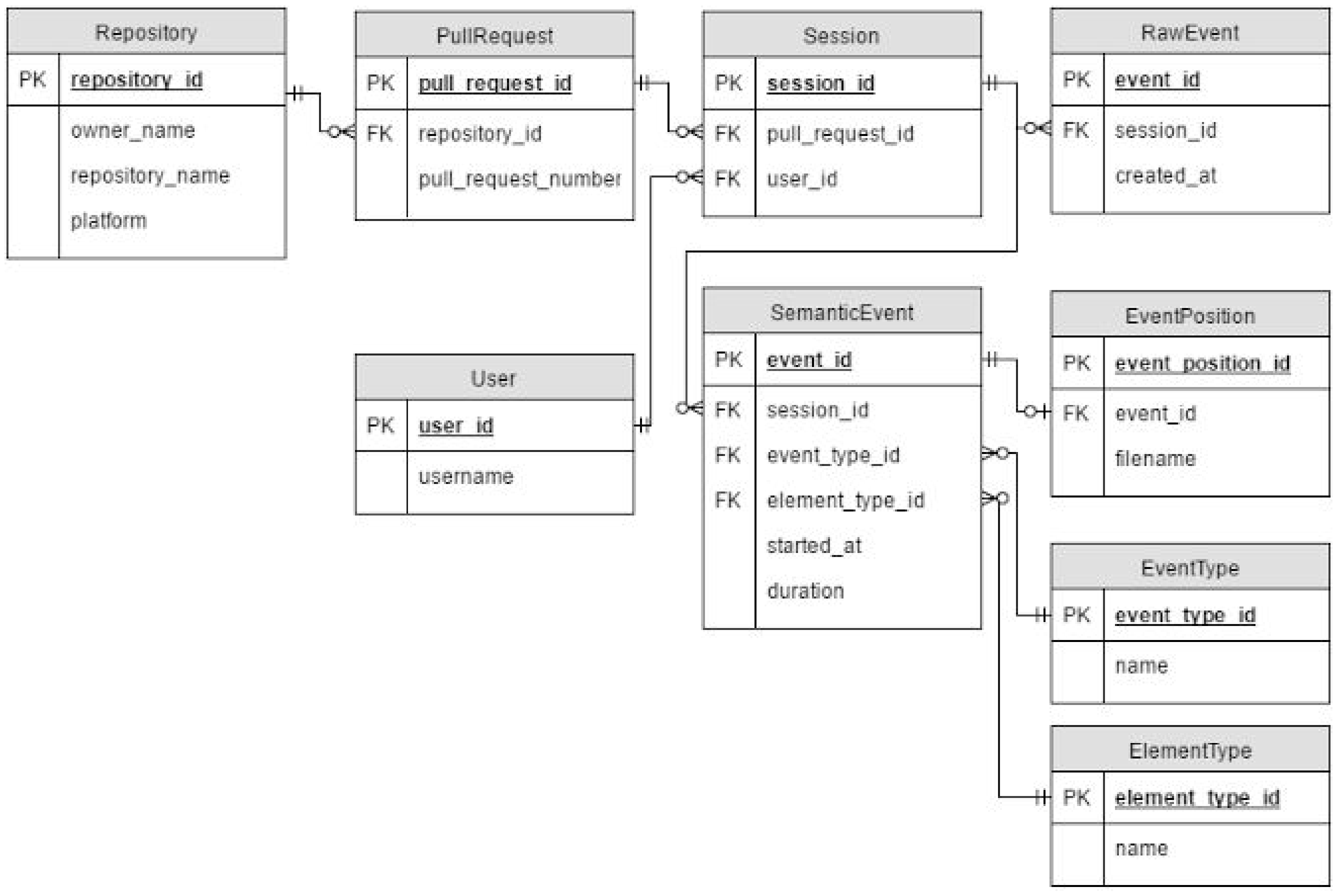

Figure 6, Database Composition

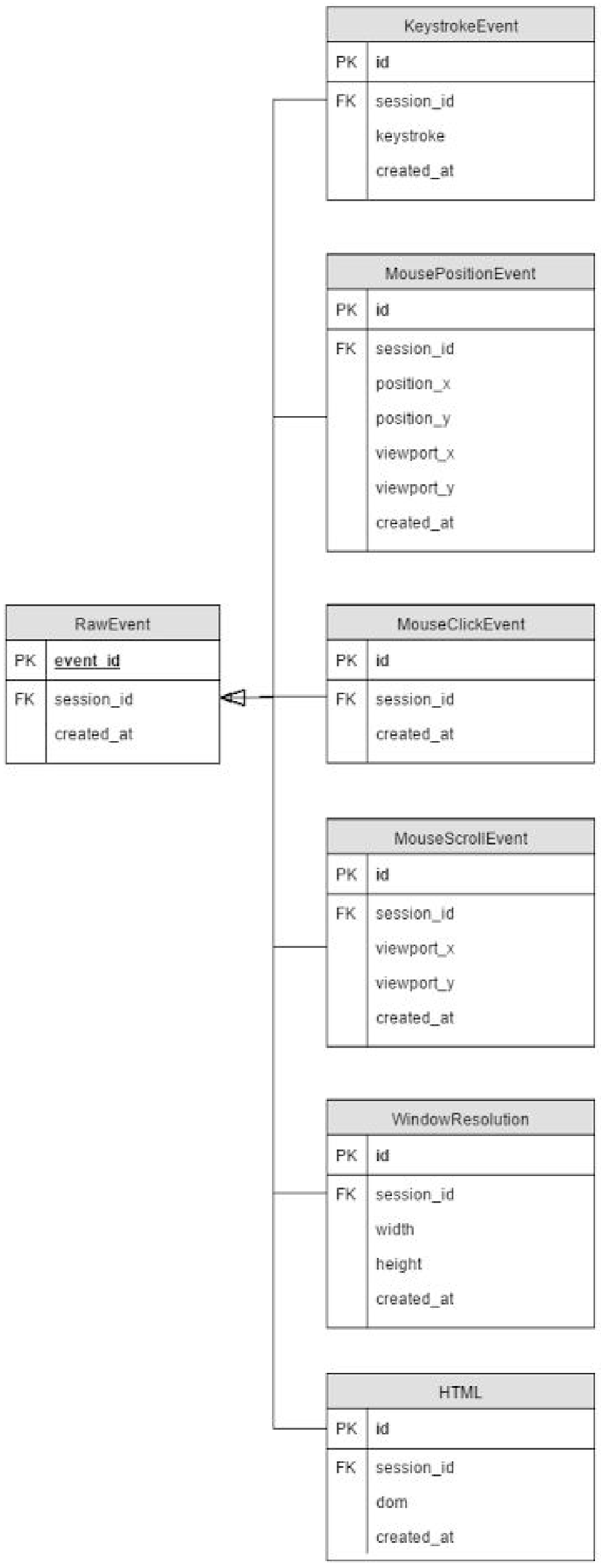

Figure 7, Database Composition: Raw Data Event Types

In *figures 6 & 7* you can see the composition of the database where the data that the tool tracks is stored. A session is the central element in the database to which all other data is. Each session has a unique ID and is tied to a single pull request and user. Multiple events are added to a session when they occur. Events are subdivided in raw data and semantic events, which are then again subdivided in specific events such as mouse clicks, or in the case of semantic events, event properties.

2.5 Concurrency

As we sent XmlHttpRequests, we get into the domain of concurrency problems.
We only sent requests to store data at the moment, and with this we are not running into concurrency problems. The Chrome browser accepts a maximum of 10 connections to one domain and will queue any connection that goes over this limit. When one of the 10 connections is finished, the browser will take a connection from the queue until it's empty.
To make sure that not too many connections are opened at the same time, we try to avoid sending duplicate data, which will keep the queue of the Chrome browser as empty as possible.
We also need to make sure that previous information is removed from the session when the browser is closed.

One of the raw data trackers is also a bit more complicated concerning concurrency. The DOM Tracker tracks entire html pages and sends copies of it to the database if it changes. Concurrency comes into play when we add special information to the DOM concerning the rendered positions of elements. The positions need to be updated after which the DOM is sent to the database. If multiple mutations occur, the DOM is traversed multiple times concurrently which can cause problems.

To stop this from happening, we stop observing the DOM once we've started mutating the DOM. This might cause some data loss if many mutations occur after each other, but this is not very likely. Once the dom has been mutated and a copy has been created to sent to the database, we start observing it again.

Concurrency information regarding data collection and sending is discussed in the section about 'System Decomposition'.

3. Glossary

- Bitbucket
    - Hosting provider for Git based code repositories.
- FK
    - Foreign Key: A field in a relational table that uniquely identifies a row in another relational table.
- PK
    - Primary key: This uniquely identifies each record in a relational table.
- Git
    - A version control system for source code that models version data structure as directed acyclic graphs.
- GitHub
    - Hosting provider for Git based code repositories.
- Pull Request (PR)
    - A request from a developer to incorporate his code changes and /or additions into the existing code base. The new code is typically reviewed by other developers before being merged.