T-404-LOKA

Tempo for Bitbucket

Final report

Freyr Guðnason
Jón Atli Jónsson
Sigurvin Frank Gardarsson

Instructor: Birgir Kaldal Kristmannsson
Examiner: Stefán Freyr Stefánsson
1. Introduction
2. Tempo for Bitbucket
   2.1 Introduction
   2.2. Architecture
3. Project Management
   3.1 Development
   3.2 Risk analysis
   3.3 Methodology
   3.4 Progress
   3.5 Obstacles
4. Testing
   4.1 Heuristic Evaluation
   4.2 Usability testing
5. Future works
   5.1 Integration with existing Tempo ecosystem
6. Conclusion
1. Introduction

Tempo\(^1\) creates plug-ins for Atlassian’s JIRA\(^2\) platform that have won numerous awards for their popularity and are well known for their usability. Tempo has over 6,000 customers in over 100 countries, despite being a relatively young company and has experienced rapid growth over a short period of time.

Tempo is thinking about further expansion and sees the Bitbucket\(^3\) platform as a possible candidate, as Atlassian\(^4\) owns both JIRA and Bitbucket which could open up the chance to integrate “Tempo for Bitbucket” with their existing products. Creating an add-on for Bitbucket could open up new possibilities and a chance to broaden their user base. The team was given the task of creating a Bitbucket add-on called “Tempo for Bitbucket” as a proof of concept.

This report covers Bitbucket and the system as a whole, it’s architecture and the technologies used. The report lists in detail the project’s approach, the obstacles experienced along the way and the progress made. The tests performed on the system will be detailed, both the heuristic evaluation and the usability test. Lastly, the report presents how the “Tempo for Bitbucket” project may be further continued, and ends on our conclusions for this project.

---

\(^1\) http://tempo.io/
\(^2\) https://www.atlassian.com/software/jira
\(^3\) https://bitbucket.org/
\(^4\) https://www.atlassian.com/
2. Tempo for Bitbucket

2.1 Introduction

Bitbucket is a distributed version control system which currently offers support for hosting Git and Mercurial projects. Atlassian acquired Bitbucket in 2010\(^5\) and added support for Atlassian Connect, a framework to build add-ons for Atlassian products, in June, of 2015\(^6\). “Tempo for Bitbucket” is built using this framework.

The idea behind “Tempo for Bitbucket” is to create a time tracking add-on for Bitbucket. When a user installs the add-on, he experiences it as a part of Bitbucket even though the add-on handles all actions and stores all the data.

The add-on offers users the ability to log their work and view their worklogs in an easy and convenient way. Admins of repositories are given access to more features, e.g. to view worklogs of all the other members in the repository and estimate time for issues. More information about the features can be accessed in the User Manual.

2.2. Architecture

This section covers the architecture of “Tempo for Bitbucket” in detail, the technologies used and their roles listed as well as what problems they solve and why they were chosen over others.

---

\(^5\) https://en.wikipedia.org/wiki/Bitbucket
\(^6\) https://blog.bitbucket.org/2015/06/10/atlassian-connect-for-bitbucket-a-new-way-to-extend-your-workflow-in-the-cloud/
2.2.1. Front end Architecture

The front end of the system is written using the React\(^7\) framework since Tempo uses it for most of it’s other products. The front end tries to follow the React philosophy, which is to break each view into small reusable components. This speeds up development and allows for great reusability. Gulp\(^8\) is used to build all the React components for a single view into a single file to minimize load on the system. Highcharts\(^9\) is utilized to display a chart containing data from the worklogs.

---

\(^7\) https://facebook.github.io/react/

\(^8\) http://gulpjs.com/

\(^9\) http://www.highcharts.com/
2.2.2. Back end Architecture

The back end follows standard REST API rules and is written in NodeJS\(^\text{10}\), taking advantage of the ExpressJS\(^\text{11}\) framework to accelerate the development. Every view in the add-on is handled by a specific controller and for every controller there is a specified service which handles the business logic, except for the service that handles the authentication logic which is used by all controllers. The add-on is built using the Atlassian Connect Express\(^\text{12}\) library which encapsulates authentication using JWT\(^\text{13}\).

The OPTIONS method is used several times in the controllers instead of GET methods to pass an array with the payload when fetching data. The size of the array is unknown and the OPTIONS method was not used for anything else, therefore it was decided to reserve it to handle these occurrences.

<table>
<thead>
<tr>
<th>logWorkController</th>
<th>adminController</th>
<th>userController</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET IssuesByRepo()</td>
<td>OPTIONS loggedWorkOnRepo()</td>
<td>PUT logworkByUser()</td>
</tr>
<tr>
<td>GET IssuesByRepoAndAssignee()</td>
<td>OPTIONS loggedWorkOnRepoDaily()</td>
<td>GET loggedOnPeriodByUserOnRepo()</td>
</tr>
<tr>
<td>POST logwork()</td>
<td>OPTIONS loggedOnPeriod()</td>
<td>GET loggedOnPeriodByUser()</td>
</tr>
<tr>
<td>POST estimateTimeOnIssue()</td>
<td>GET membersInRepo()</td>
<td>DELETE logworkByUser()</td>
</tr>
<tr>
<td></td>
<td>OPTIONS membersTab()</td>
<td>OPTIONS loggedworkByUser()</td>
</tr>
<tr>
<td></td>
<td>OPTIONS issuesTab()</td>
<td>GET loggedworkByUserOnRepo()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GET totalLoggedWorkByUserOnRepo()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GET totalLoggedWorkByUserOnRepoDaily()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>log WORK Service</th>
<th>adminService</th>
<th>userService</th>
</tr>
</thead>
<tbody>
<tr>
<td>getIssueListByRepoSlug()</td>
<td>getTotalLoggedOnPeriod()</td>
<td>deleteLoggedWorkByUser()</td>
</tr>
<tr>
<td>getIssueListByRepoSlugAndAssignee()</td>
<td>getTotalDailyLoggedWorkOnRepo()</td>
<td>updateLoggedWorkByUser()</td>
</tr>
<tr>
<td>getLoggedWorkByIssuelRepoSlug()</td>
<td>getEstimatedTimeWorkByRepoSlug()</td>
<td>getLoggedWorkByUserNameOnRepo()</td>
</tr>
<tr>
<td>addLoggedWork()</td>
<td>getLoggedWorkByRepoSlug()</td>
<td>getLoggedWorkByUserName()</td>
</tr>
<tr>
<td>setEstimatedTimeOnIssue()</td>
<td>getMembersInRepo()</td>
<td>getTotalDailyLoggedWorkByUser()</td>
</tr>
<tr>
<td>getEstimatedTimeByIssueId()</td>
<td>getMembersindividualWorkInRepo()</td>
<td>getTotalDailyLoggedWorkByUserOnRepo()</td>
</tr>
<tr>
<td>getEstimatesOnAssigneeIssues()</td>
<td>getEstimatesAndLoggedWorkOnIssues()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>jwtService</th>
</tr>
</thead>
<tbody>
<tr>
<td>getUsernameByJwt()</td>
</tr>
</tbody>
</table>

\(^{10}\) https://nodejs.org/
\(^{11}\) http://expressjs.com/
\(^{12}\) https://bitbucket.org/atlassian/atlassian-connect-express
\(^{13}\) http://jwt.io/

Figure 2.2: Overview of the controllers and services
2.2.3. Database Architecture

The database used is PostgreSQL\textsuperscript{14} and the system uses the ORM framework Sequelize\textsuperscript{15} to handle all connections to the database. Sequelize creates or updates the tables in the database when the system is started, depending on if the tables exist.

The database has two tables. The first stores all logged work by the users of the system. The other one stores all estimates made by admins.

Sequelize also offers support to MySQL, MariaDB, SQLite and MSSQL. The solution doesn’t support these types of databases, but it's possible by installing the appropriate packages for the desired database type.

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>id serial</td>
<td>id serial</td>
</tr>
<tr>
<td>date timestamp with time</td>
<td>date timestamp with time</td>
</tr>
<tr>
<td>zone</td>
<td>zone</td>
</tr>
<tr>
<td>issueid integer</td>
<td>issueid integer</td>
</tr>
<tr>
<td>estimatedonissue integer</td>
<td>workedhours integer</td>
</tr>
<tr>
<td>accountname character</td>
<td>comment character</td>
</tr>
<tr>
<td>varying(255)</td>
<td>varying(255)</td>
</tr>
<tr>
<td>reposlug character</td>
<td>username character</td>
</tr>
<tr>
<td>varying(255)</td>
<td>varying(255)</td>
</tr>
<tr>
<td>CONSTRAINT estimates_pkey</td>
<td>CONSTRAINT works_pkey</td>
</tr>
<tr>
<td>PRIMARY KEY (id)</td>
<td>PRIMARY KEY (id)</td>
</tr>
</tbody>
</table>

\textit{Figure 2.3: Database tables}

\textsuperscript{14} http://www.postgresql.org/
\textsuperscript{15} http://docs.sequelizejs.com/
3. Project Management

3.1 Development

3.1.1 Version control

Since Bitbucket is itself a version control site, it was the only one considered to store the “Tempo for Bitbucket” codebase. The ability to use Bitbucket while developing an add-on for it allowed the team to familiarize itself with it so that the “Tempo for Bitbucket” add-on could mimic the look and feel of Bitbucket as closely as possible, which is one of Tempo’s main goals for every product.

3.1.2 Unit testing

Jasmine\textsuperscript{16} was chosen as the framework for unit testing the front end. Gulp was used to run the tests and display the results of the tests. Mocha\textsuperscript{17} was used to unit test the back end.

3.1.3 Documentation

All reports for this project were made using Google Drive\textsuperscript{18}. At the start of the project a shared folder was set up so that the team could collaborate on the same files at the same time, greatly improving work output.

3.2 Risk analysis

At the start of the project 16 possible risks that could occur were discovered. Each of them was assessed by giving them a severity rating and the likelihood of them happening was rated. The purpose was to find ways to minimize the effect each one could have, as well as the best possible solution should that occur.

\textsuperscript{16} \url{http://jasmine.github.io/}
\textsuperscript{17} \url{https://mochajs.org/}
\textsuperscript{18} \url{https://www.google.com/drive/}
Over the project’s lifetime, seven instances included in the risk analysis occurred, the major ones being that the project was well behind schedule, big projects in other courses and the injection point not arriving in time, as can be seen in the Risk Analysis report.

3.3 Methodology

Every aspect of the project was planned meticulously from the start. Guidelines were set regarding team member’s workload throughout the project. The Scrum methodology was adapted to the project and each sprint spanned a two week period excluding “sprint zero” and sprint 5 which were 3 weeks. In total there were 8 sprints including "sprint zero".

![Figure 3.1: Sprint Overview](https://www.mountaingoatsoftware.com/uploads/blog/ScrumMediumLabelled.png)

The reasons for choosing the Scrum methodology is twofold. It was chosen mainly because teams within Tempo use it when developing their products and the team had some prior experience with it. Team members agreed that it would suit this project well since the project would undergo changes as it evolved, making Scrum ideal because of its flexibility.

---

19 Image from https://www.mountaingoatsoftware.com/uploads/blog/ScrumMediumLabelled.png obtained 13/12/2015
In the beginning the requirements for this system were split into stories and divided into groups (A, B or C) to indicate their priority. Each sprint started with the team planning the stories for that particular sprint and every morning the team would discuss yesterday's progress and what they would work on that day.

Originally the plan was for each team member to commit at least 27 hours each week to the project for the first 12 weeks while team members were still enrolled in other courses. The plan was revised since 27 hours per week turned out to be a bit too ambitious and was lowered to 20 hours each week. After the exams each member was to commit around 8 hours per day, including weekends if deemed necessary.

3.4 Progress

![Project burndown](image)

Figure 3.2: Project burndown

Figure 3.2 shows the overall burndown over this project's timeframe. As the figure shows the team didn't manage to complete all the stories in our product backlog. All A and B stories have been implemented as well as some C stories, though some are left to be implemented later.

Table 3.1 shows what the team estimated after the project was restructured and how many hours each member spent on the project. The project was restructured because it fell behind schedule and the team agreed to commit extra hours into the project. The team went well over the given timeframe for this project according to the
estimation of the course, the estimation made by Tempo from the start was 1400-1600 worked hours.

<table>
<thead>
<tr>
<th>Name</th>
<th>Estimated</th>
<th>Actual Work</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freyr</td>
<td>360+</td>
<td>463</td>
<td>129%</td>
</tr>
<tr>
<td>Jón</td>
<td>360+</td>
<td>431</td>
<td>120%</td>
</tr>
<tr>
<td>Sigurvin</td>
<td>360+</td>
<td>477</td>
<td>133%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>960+</strong></td>
<td><strong>1371</strong></td>
<td><strong>128%</strong></td>
</tr>
</tbody>
</table>

*Table 3.1: Time spent on the project*

In the following sections this report will go into more detail about the two most notable sprints, as they shaped how the end product is today, further details for all sprints can be found in the *Scrum Report*.

3.4.1 Sprint 3 (Sauron)

This sprint turned out to be a defining sprint for this project as everything came to a complete stop because of issues involving Spring\(^{20}\) and the team decided to switch to NodeJS, as described in section 3.5.3.

Prior to that change there had been no decrease in story points, but in the last four days of the sprint the team managed to finish 48% of the originally planned story points.

3.4.2 Sprint 6 (Gandalf the White)

This sprint was the most successful one in finished stories, as 149 story points were completed. At this point in time other courses were over, so the focus was entirely on the “Tempo for Bitbucket” project.

\(^{20}\) https://spring.io/
The team set out from the start to design the system in small reusable parts which enabled the use of previously developed components which sped up the progress and contributed to the number of story points finished in this sprint.

3.4.3. Retrospectives

When looking back, the team is quite satisfied with the end result even though the progress in the earlier sprints was not as good as anticipated. Of course there were certain obstacles faced along the way, as mentioned in section 3.5 but the team believes it has found ideal solutions for most of them. The structure of each sprint has also evolved with every passing sprint as detailed discussions were always held to improve the process.
3.5 Obstacles

3.5.1 Injection point

The initial vision for the project was for users to be able to log work and for admins to be able to estimate workload on each issue, inside the built-in issue tracker on the Bitbucket site. Bitbucket does however not provide an injection point to allow this behavior. The product owner therefore contacted the developers of Bitbucket to request the injection point and at the early stages of the project the team was hopeful of getting it. It soon became apparent that the injection point would not arrive in time.

The team decided, in collaboration with Tempo, to build a separate view inside Bitbucket to house that functionality. Since this new site was not meant to be a permanent site, the design layout was kept to a minimum, while still being aesthetically pleasing and the focus was shifted to get the functionality right and more effort was poured into the worklogs overview page.

The team also created an issue on the main Bitbucket site to address the lack of an injection point, but it was clear in the later stages that any results would not benefit the “Tempo for Bitbucket” project.

3.5.2 Webhook

One of the intended features in the “Tempo for Bitbucket” system was that a user could log his work through the command line interface. For that to work, a webhook would be created when a user installed the add-on, the webhook would then receive data when the user committed or pushed code to Bitbucket.

Each Bitbucket add-on has a context parameter set either to “personal” or “account”. This system has the context set to “personal” because the user has to be able to reach the add-on in all repositories he has access to.

When a user installs the add-on a webhook was supposed to be created. This was not the case so the team contacted Atlassian and they suspected that webhooks
were intentionally disabled for add-ons with context set to “personal”, therefore this task was set on hold.

There is another possibility to get this functionality, but then the user has to create the webhook himself inside each repository, therefore it depends on the user which is impractical. Webhooks created manually do not send authentication tokens which creates a security risk for the API and therefore this was not implemented.

3.5.3 Switch from Spring to NodeJS

The team initially chose the Spring framework for the back end of the system, since the team had experience using it, but ran into many obstacles setting it up correctly with the Atlassian Connect framework. Despite following the work schedule, progress was slow and each feature the team tried implementing was met with a number of hurdles. When seeking help with the back end an idea surfaced to switch from Spring to NodeJS and use Atlassian’s own library to speed up development. After consulting with both the product owner and the instructor, the switch was decided. Similar success was very quickly achieved with NodeJS, as had been achieved in weeks using Spring.
4. Testing

Þorgeir Gísli Skúlason, an intern designer at Tempo, laid the groundwork for the design of the “Worklogs” view in collaboration with the team and did some preliminary testing for the layout of that site. Although a lot would change from the early design, the team deemed it not necessary to perform a separate usability test until the core functionality of the site was completed. In collaboration with the designer, the team also did a Heuristic Evaluation, described in more detail in the following section, and a usability test with all functionality in place, as described in section 4.2.

4.1 Heuristic Evaluation

Þorgeir constructed, with feedback and help from the team, a heuristic evaluation of the system. A heuristic evaluation is much like a usability test, but instead of potential end users testing the system, user experience experts evaluate the system, in this case the user experience team at Tempo.

The evaluation was performed on December 1st by Þorgeir and the team. Three individuals tested the system, they were all briefly introduced to the project and given nine tasks to finish. Since user experience experts were evaluating the system, they tried to find ways to break the functionality of the system in ways they themselves would not necessarily do, but they could envision other users doing. This exposed some small bugs that might otherwise not have been found, for example ways to break the validation of user’s input.

A lot of small fixes were made given the outcome of this evaluation, for example the add-ons pagination was changed to match the Bitbucket pagination, the ability for the chart to maintain the selected period when selecting members, as well as editing and deleting worklogs and other minor tweaks.

When tasked with estimating workload and logging work on issues, most of the user experience experts tried to use the built in issue tracker and all of them agreed that
it was the most logical place for that specific functionality. That further demonstrated the need for an injection point into the issue tracker, as mentioned in section 3.5.1.

Another important request that the user experience team made, was to be able to pick the period using a datepicker and not an input field. Given the setup of the project, this would require a rather extensive overhaul of the system and since the project was drawing to a close, the team decided in the interest of time to rather focus on improving other aspects of the add-on.

4.2 Usability testing

Since the project’s development started off slowly the usability tests were pushed back as the team deemed that time was better spent developing the core features. The team performed the usability test on December 10th and processed the results even though no sweeping changes to the system could be made, even if the outcome of the test was poor. This would give Tempo a better feel for the usability of the product and what changes would be the most urgent ones if development continued.

4.2.1 Methodology

The usability test was twofold, whereby users held either the role of an administrator or that of a user of the system. For the admins the test consisted of twelve tasks which they were assigned but the users were tasked with seven tasks, both listed below. Both scenarios were designed in such a way that each task covered a basic functionality of the system for each user group.

Two groups of users performed the tests. The first group consisted of computer science students at University of Reykjavík and the latter of employees from Tempo. The two groups were chosen as they represent the end user well, as both groups have good knowledge of computers, like the end user of a version control site is expected to have. See the Usability Test report for more information on how the test was conducted.
4.2.2 Admin results

Admins were asked to solve the following tasks:

1. *Estimate 7 hours on the following issue: “Css style standardization”*
2. Log 4 and 30 minutes on the following issue: “New design for buttons” on the current date with the comment: “Usability test”
3. Find the overview site for the logged hours of the team
4. How many hours have been logged on this repository in total?
5. How many hours have been logged on this repository in the last two weeks?
6. How many hours has each team member logged in the last two weeks?
7. How many hours did ‘jonatli91’ log on November 10th?
8. The user ‘jonatli91’ has logged the wrong time on November 10th on issue nr. 6, change it to 5 hours
9. Delete your worklog from step 2
10. How many hours have been logged on issue nr. 3 in total?
11. You are looking at worklogs from all members of this repository, find the overview for your worklogs only for this repository
12. You have logged time on other repositories, look at the overview for all your worklogs for all repositories

<table>
<thead>
<tr>
<th>Task</th>
<th>HR Students</th>
<th>Tempo</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:01</td>
<td>1:18</td>
<td>1:10</td>
</tr>
<tr>
<td>2</td>
<td>0:45</td>
<td>0:24</td>
<td>0:35</td>
</tr>
<tr>
<td>3</td>
<td>0:50</td>
<td>0:07</td>
<td>0:29</td>
</tr>
<tr>
<td>4</td>
<td>0:04</td>
<td>0:09</td>
<td>0:7</td>
</tr>
<tr>
<td>5</td>
<td>1:43</td>
<td>0:44</td>
<td>1:14</td>
</tr>
<tr>
<td>6</td>
<td>0:26</td>
<td>0:07</td>
<td>0:17</td>
</tr>
<tr>
<td>7</td>
<td>0:26</td>
<td>1:43</td>
<td>1:05</td>
</tr>
<tr>
<td>8</td>
<td>0:40</td>
<td>0:32</td>
<td>0:36</td>
</tr>
<tr>
<td>9</td>
<td>1:00</td>
<td>0:24</td>
<td>0:42</td>
</tr>
<tr>
<td>10</td>
<td>1:05</td>
<td>1:04</td>
<td>1:05</td>
</tr>
<tr>
<td>11</td>
<td>0:18</td>
<td>0:05</td>
<td>0:12</td>
</tr>
<tr>
<td>12</td>
<td>0:28</td>
<td>0:07</td>
<td>0:18</td>
</tr>
<tr>
<td>Average</td>
<td>0:44</td>
<td>0:33</td>
<td>0:39</td>
</tr>
<tr>
<td>Rating</td>
<td>4.5</td>
<td>4</td>
<td>4.3</td>
</tr>
</tbody>
</table>

*Table 4.1: Timing for admin tasks*
4.2.3 Users results

Users were asked to solve the following tasks:

1. Log 4 hours and 30 minutes on issue “New design for buttons” on today and with the comment “notepadrófun”
2. Find the overview to view all your time logs in this repository
3. How many hours have you logged for the last two weeks?
4. How many hours did you work on the 10th November?
5. You logged the wrong amount of time on 10th November on issue nr. 6, change it to 5 hours
6. Delete your worklog from step 1
7. You have logged time on other repositories, look at the overview for all your worklogs for all repositories

<table>
<thead>
<tr>
<th></th>
<th>HR Students</th>
<th>Tempo</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:12</td>
<td>1:31</td>
<td>1:22</td>
</tr>
<tr>
<td>2</td>
<td>0:37</td>
<td>0:11</td>
<td>0:24</td>
</tr>
<tr>
<td>3</td>
<td>0:07</td>
<td>0:19</td>
<td>0:13</td>
</tr>
<tr>
<td>4</td>
<td>0:23</td>
<td>0:35</td>
<td>0:29</td>
</tr>
<tr>
<td>5</td>
<td>0:19</td>
<td>1:24</td>
<td>0:52</td>
</tr>
<tr>
<td>6</td>
<td>0:56</td>
<td>0:13</td>
<td>0:35</td>
</tr>
<tr>
<td>7</td>
<td>0:08</td>
<td>0:28</td>
<td>0:18</td>
</tr>
<tr>
<td>Average</td>
<td>0:31</td>
<td>0:40</td>
<td>0:36</td>
</tr>
</tbody>
</table>

| Rating | 4.3 | 3.3 | 3.8 |

*Table 4.2: Timing for user tasks*

4.2.4 Discussion

The average time for each task was 38 seconds, there are six tasks that seem to be more time consuming than others, admin tasks number 1, 5, 7 and 10 and user tasks number 1 and 5.

When trying to solve admin tasks 1 and 10 and user task 1, most people tried to use the built in issue tracker. That is an issue that cannot be fixed without the injection point, as covered in section 3.5.1.
The input fields for selecting a period in the chart were problematic for most users when trying to solve admin task 5, most commented that they wanted datepickers instead. As discussed in the heuristic evaluation section, changing this would be too drastic given the short time remaining, but would certainly be one of the first things to consider should the work on “Tempo for Bitbucket” continue.

Participants also wanted the ability to select a single day. The input fields in the chart control the period selected, and do not provide the ability to select a period smaller than 3 days. The team believes this to be the main reason admin task 7 and user task 5 were so time consuming which further supports the previous conclusion that datepickers should be added if the project continues.

Aside from the abovementioned issues, the users were generally very pleased with the system as a whole. Most of the people found it a good idea to log hours in a version control site like Bitbucket, which could eliminate visiting a separate site for that purpose alone. The participants noted that the look and feel of the add-on matched that of Bitbucket, which is exactly what was set out with from the start. The users also appreciated the feedback the system gave them on their actions, be it logging work, estimating workload or editing and deleting existing worklogs, once completed successfully.
5. Future works

Should the “Tempo for Bitbucket” project continue the first thing to do is fix the remaining bugs which were exposed during the usability test and the heuristic evaluation. Most of the bugs from the heuristic evaluation have been fixed already, but only a few of the ones unveiled by the usability testing could be addressed, so the system is mostly as it was when tested.

The team has a vision for additional features that can be added to the project. The current worklogs view is great to get a quick overview of your and your team’s worklogs, but adding a calendar view as well would add a lot of value to the product. In the usability test, users wanted to be able to view a single day which could be presented nicely in a calendar.

Should the injection point into the Bitbucket issue tracker arrive, as discussed in section 3.5.1, implementing a log work feature as well as a quick overview of logged hours and estimated hours into it would greatly improve the usability of the product, given the data from both the usability tests and the heuristic evaluation.

The team had not used React before this project and was not aware of the Flux\(^2\) architecture and the benefits it could bring. The team believes that taking the time to integrate it into “Tempo for Bitbucket” would vastly improve the pace and quality of the front end development.

5.1 Integration with existing Tempo ecosystem

“Tempo for Bitbucket” allows Tempo to have a new product, built for another platform which could therefore possibly expand their user base. The new users could be small teams and individual developers who are looking for a lightweight and easy way to keep track of their work.

All Bitbucket accounts will be migrating to Atlassian accounts in January, 2016. Since Tempo already keeps track of Atlassian accounts, they could store the new users

\(^2\) [https://facebook.github.io/flux/](https://facebook.github.io/flux/)
alongside existing ones and if users use both platforms they could log work through Bitbucket and have it integrate seamlessly with JIRA and the existing Tempo product line.

There are some fundamental differences between JIRA and Bitbucket that would need to be addressed in order to integrate the two platforms, for example they both have unique issue trackers. When using both platforms, issues would therefore need to have a unique identifier to distinguish between platforms.
6. Conclusion

Overall the team is very pleased with the end product and believes that a time tracking add-on for Bitbucket has great potential. The vision for the project has evolved and changed since the beginning of the project.

Throughout the project’s lifespan the team has learned a great deal, as prior to the project’s commencement the team’s experience using React and NodeJS was non-existent. While developing the add-on, our understanding of React and NodeJS has substantially progressed along with the project, which has enhanced our abilities as programmers and given us invaluable experience.

We would like to thank all of the people at Tempo for all the help they provided, both by taking part in the heuristic evaluation and the usability testing, as well as for all the other assistance during the project’s development.

Special thanks to Þorgeir Gísli Skúlason for working with us on design and Eric Nilsen for help with development and deployment. Finally we want to thank the product owner Viðar Svansson, our instructor Birgir Kaldal Kristmannsson and our examiner Stefán Freyr Stefánsson for all of their advice and support during the project.