

UNIVERSITY OF TROMSØ

INF-2900

SOFTWARE ENGINEERING

Turi

Group:

1

SPRING 2015



UiT / NORGES ARKTISKE
UNIVERSITET

Contents

1	Introduction	2
1.1	The product name: turi	2
1.2	Summary of goals (planned functionalities)	2
2	Design of the site	3
3	Gems	3
3.1	Devise	3
3.2	Pundit	3
3.3	Leaflet	3
3.4	rails-asserts	3
3.5	Geocoder	3
3.6	Puma	3
4	Trip Features	4
4.1	Discuss	4
4.2	Event management	4
4.3	Equipment planning	4
4.4	Gallery	4
4.5	Blog	4
4.6	Share your trip (public setting)	5
4.7	Explore trips	5
4.8	Search	5
4.9	Route Planner	6
5	Other features	7
5.1	Friend requests	7
6	Tests	7
7	Git	7
7.0.1	Travis CI	7
7.0.2	Heroku	7
7.0.3	CodeClimate	7
7.0.4	Hakiri	10
8	Group collaboration	10
9	The development process	10
9.1	Git Workflow	10
9.1.1	Developing of a new feature	10
9.1.2	Starting a merge request	10
9.1.3	Validating the merge request	11
9.1.4	Deploying to production	11
9.2	Conclusion of the workflow	11
10	Something about the different sprints?	11

1 Introduction

Imagine you want to go on a trip - to Norway for example. You ask your friends to join you but they reject because they can not afford the money or they have no spare time. What now? You can either go alone or try to find some people who can join you. The first place we look today is on the world wide web. We already can find some portals where you can find partners for your trip. But what about planning your actually trip? We couldn't find a tool which allows you to plan your trip in detail. Planning your trip gets especially difficult if you find your trip mates online and you can not meet because you live too far away from each other. You can now either use the phone or create e.g. some Facebook group. Still those ways lack of several features we want to implement in our trip planner which makes it unique. In fact we came up with the idea to create a trip planner which offers a combination between socialising and planning for your trips.

1.1 The product name: turi

The name derived from the Norwegian word "tur" (a trip/a walk). turi was short and pregnant enough and sounds best to the ear. It is also a Norwegian name, but this will be no problem when we think about topics like a trademark etc.

1.2 Summary of goals (planned functionalities)

The following functionalities should be supported (bold means that we they were implemented during the course sprints):

- **Discuss with participants**
- **Event management (for appointments top plan)**
- **Route planning**
- **Equipment planning**
- **Gallery to visualize your trip experiences**
- **Blog about your experience**
- **Share your trip (public/private)**
- Copy a previous trip from others (e.g. routes)
- Rating functionality and other common social functions (commenting, likes..)
 - **Friend requests**
- **Explore trips based on their location**
- Badges and rewards for participating and using Turi
- Search and find trips according your interests

Finally we were not able to implement all of our goals (the goals which are not in bold) and we had tons of more ideas. But for the first release of turi (1.0.0) we decided to require all these features above.

2 Design of the site

3 Gems

We use several gems in our project, this means that we not need find up the wheel for already created features, and we can focus on our own feature. All the gems used can be found in the gemfile in the project source, but the most important ones are:

3.1 Devise

Devise is a popular authentication solution for Rails based on Warden [3]. In the first iterations we created our own authentication system, but we found out that we could use *Devise*, after the lecture about authentication. This made it possible to focus on other task in the project, and not focus on making a secure and safe authentication system.

3.2 Pundit

Pundit is a authorization system[4], which we use in almost all features in projects, our main usage of the gem is used for who can and can't do things to the trip or the trip features. For example a editor and the owner of a trip is able to edit the trip title, description and so on, but a viewer is not able to do this.

3.3 Leaflet

Leaflet[5] is a JavaScript library that let us implement the interactive map for the routes. Before discovering *Leaflet*, we looked at a similar gem from Google, but it was too heavy for our requirements. As an example, Google Maps seemed to snap waypoints to the nearest road, which is counter-productive for us since we want the users to be able to plan any kind of trip, including hikes.

3.4 rails-asserts

3.5 Geocoder

Geocoder is a so called "complete geocoding solution for Ruby"[?], it provides a location based on IP, location and so on, in the project we use it for location based search, for example in the trip start and end location.

3.6 Puma

Puma is a "simple, fast, threaded, and highly concurrent HTTP 1.1 server for Ruby/Rack applications."[1]. Heroku recommend *Puma* over the stock rails

server (WEBrick) by saying that: "While WEBrick should be fine for development, it was not designed to handle a high concurrent workload that a Ruby app must serve in production. A production web server should be used instead." [2]. Even tho Heroku only uses one core (for the free program), this secures that the project is ready for a deployment onto a multi-core server in the future.

4 Trip Features

The trip and it's features are the main focus of our project.

4.1 Discuss

4.2 Event management

4.3 Equipment planning

A Equipment planning is essential for a trip, knowing what you need to bring and the ability to delegate things to other participants in the trip. In our implementation of each trip can have multiply list, which can contain multiply items. Each item in return contain a a number (number of items) and a price, these items can be assigned to other participants. Charts provides a overview of all lists and for each individual list, these charts provides a summary of the assignments (items and price) for each participant in a pie chart.

This feature had two different user stories attaches to it (*#154 and #155*), and combined they had a point value of 13. This estimation was reasonable, and the implementation of the features spanned the whole sprint 2. We changed the look and feel of the features multiply time in the sprint, before ending up at the current implementation, which we felt was the most intuitive. There wasn't any participlal problem we faced when implementation this feature, but there was a lot of work to get everything working on a single view page, instead of having views for each of the controllers. A particulate tricky part of the feature was the form for the item assignment, since we used the same form for creating, editing and deleting, this caused the create method of the equipment-assignment controller to become quite large and complex.

4.4 Gallery

4.5 Blog

The blog platform allows a trip editor to create a blog associated with the trip. The user can edit the blog through a simple editor interface which also allows for urls and images. A blog can be marked as public, which means that non-participants can read it. The platform has a blog index list which lists the blogs in order of creation and shows an excerpt from each entry.

The text input is handled through a plugin named CKEditor[6]. This plugin integrates with a text area and allows the user to work with text in a similar fashion as a regular text editor The plugin automatically adds the appropriate html tags necessary to render the text properly in a browser. As default the editor has an overwhelming amount of tools available, but for this implementation

the number of tools has been greatly reduced to remove unneeded functions and improve ease-of-use.

Generally implementing the blog was straight-forward to implement and the only issues that presented themselves were designing the tests and getting the interface to look presentable. The blog was assigned a single user story estimated at 5 points, which in retrospect seems fairly appropriate.

4.6 Share your trip (public setting)

A trip should be private to the participants of the trip, unless they decide it should be shared with the public. The public should only be able to view a small subset of the information about the trip, which the participant agrees to share. Therefore each trip needs a private setting and its own public view, this was the one of the main things we wanted to implement in the 3rd sprint for the project.

4.7 Explore trips

4.8 Search

The search system is implemented through address tags which are read out of a form and appended on the address bar. These tags are read as arguments by the controller methods running search and used to do search on the database. The search allows the user to do a search on title, start and end location and date, and tags. The same engine is also used to search for other users.

Making a search engine for trips made it necessary to consider what the user would want to search for. A trip has several data fields but not every field makes for a useful search option. A search on location is useful in that it finds trips related to a certain location, while a search on description leaves few guarantees for what the result will be. The final implementation allows for search on title, start and end locations and dates, and tags. A possible missing option is the option to search for trips created by a specific user, although this is not necessarily needed since the engine also allows for searching for specific users.

The solution used here is not the only possible solution. A second option would be to use a separate model and controller for search. This allows for storing searches and makes the logic behind the implementation cleaner, depending on the number of fields used as search terms. The downside is extra logic is needed to clean the database of old searches regularly. Since each search is its own entry, it could end up growing much faster than any of the other databases. The solution used is simpler, but faster to implement and the difference in user experience is minimal.

The search system was assigned a single user story estimated at 8 points. The scoring of the story seemed appropriate as implementing search required some time to study the different options available and implement it. Not every field could be read through the same query which meant extra work for comparison and merging of search results before getting the final result. This led to some

trial and error to get the query results correct. Unfortunately the user story in question only refers to searching for trips and not users and no additional user story were made for the user search. This means that there is no tracking available for the work done on the user search and no estimation for work load.

4.9 Route Planner

When planning a trip, it could be very useful to illustrate the route, and points of interests on a map. This map should be visible to all the participants, so that the route, or proposed route could be discussed.

Bjørnar and Kristoffer was given the responsibility of implementing the feature, and when planning the second sprint a single user story (CITE) was created for the entire feature. The user story was given the maximum points for complexity by the scum poker session, thus it was expected that the entire feature was to be implemented during the second sprint.

In retrospect, the user story could have been classified as an “epic”, with multiple fine-grained user stories. The implementation could have been divided into three main parts; the map control and GUI, the back-end model and controller, and the view. And given how the workload have been on this feature, we would have given the map control 13 points, the back-end, including tests would get 8 points, and the view would get about 3 points.

Implementing the feature was mainly done by peer-programming. This worked really well for the map, which was implemented in JavaScript, using the Leaflet API, where one was writing code, and the other was helping, and could very quickly look up manuals and documentation on the fly.

The model and controller was a bit more complicated for this feature, than for the other features. A route is a collection of waypoints, and at first, we thought about having these as two separate models, but after some help from Runar, we decided on a structure where the route model uses nested-attributes of the waypoint model. This way a route is created all at once, and not by appending waypoints to a route.

We did not put much effort into the view during the implementation of the feature, and at the end, we did not feel this was a bad choice, since the group did not decide on a uniform design until at the end of the third sprint. And since Florian had put a lot of work into the layouts, applying a design to the views was trivial.

(this part could be somewhere else??) The feature was implemented on a separate branch on Github, and was only merged into the development branch at the end of the third sprint, when the feature was done. This was not a deliberate choice, but the view was always a work in progress, and we did not feel the feature was polished enough to be introduced in the master branch. There was also the continuous development functionality, that required that all

tests needed to pass before a pull request could be accepted, and we did not start writing tests until at the end of the third sprint.

5 Other features

5.1 Friend requests

6 Tests

7 Git

We use github.com instead of the git repository supplied to us from the school, this gave us the opportunity to work outside the school network. In addition it gave us the ability to use 3rd party applications, this is explained the following subsections. The github repository address is: <https://github.com/turi-inc/turi>.

7.0.1 Travis CI

Travis CI is a hosted continuous integration service. It is integrated with GitHub and provides testing for our project. So when we created a pull request on the development branch of the Github repository, Travis would test our new code automatically and give us a clear indication if the test was passing or not. When the pull request is merged with the development branch it would be tested once again by Travis to be sure that everything was working correctly before the pull request is automatically merged with the master branch of the repository. The log from Travis is public and can be seen here: <https://travis-ci.org/turi-inc/turi/builds>.

7.0.2 Heroku

Heroku is a cloud platform which host our project for free. When a pull request is merged with the master branch it's automatically pushed to Heroku. This gives us and other people to see a preview of the project. We had some minor problem getting this to work properly since we use Sqlite3 when we develop, but Heroku does not support this and so we had to switch over to postgres in the production environment.

7.0.3 CodeClimate

Is also a 3rd party application which checks our code for test coverage, complexity and duplications. The check is done on the master branch of the repository, every time a new pull request is merged with the branch. It gives us an indication about the code health and grades our code, on the basis of test coverage, complexity and duplications. The summary of the code climate of our project can be found here: <https://codeclimate.com/github/turi-inc/turi>. CodeClimate also have some cool features like Trends over the "health" of our code over time, and the location "hotspots" in the code.










	master Merge pull request #65 from runarf/develop Florian Sauter committed	# 203 errored b0f2de0	🕒 2 min 42 sec 📅 about a month ago
	develop Merge pull request #65 from runarf/develop Florian Sauter committed	# 202 passed b0f2de0	🕒 2 min 35 sec 📅 about a month ago
	triproutes recommit, because there were errors with the previous commit Bjornar Prytz committed	# 196 failed 11f55b5	🕒 1 min 31 sec 📅 2 months ago
	master Merge pull request #66 from fsauter/feature/appui-update-2-3 Martin committed	# 195 errored cfb4198	🕒 12 min 58 sec 📅 2 months ago
	develop Merge pull request #66 from fsauter/feature/appui-update-2-3 Martin committed	# 194 passed cfb4198	🕒 2 min 34 sec 📅 2 months ago
	triproutes We rebased and manually merged the triproute branches of Runa Bjornar Prytz committed	# 190 failed 663d57f	🕒 1 min 39 sec 📅 2 months ago
	master Merge pull request #64 from omtan/master Martin committed	# 189 passed 748e912	🕒 4 min 📅 2 months ago
	develop Merge pull request #64 from omtan/master Martin committed	# 188 passed 748e912	🕒 2 min 16 sec 📅 2 months ago
	master Merge pull request #62 from ritualz/feature/equipment Florian Sauter committed	# 184 passed 9a81517	🕒 5 min 26 sec 📅 2 months ago

Figure 1: A sample of the build logs from Travis CI

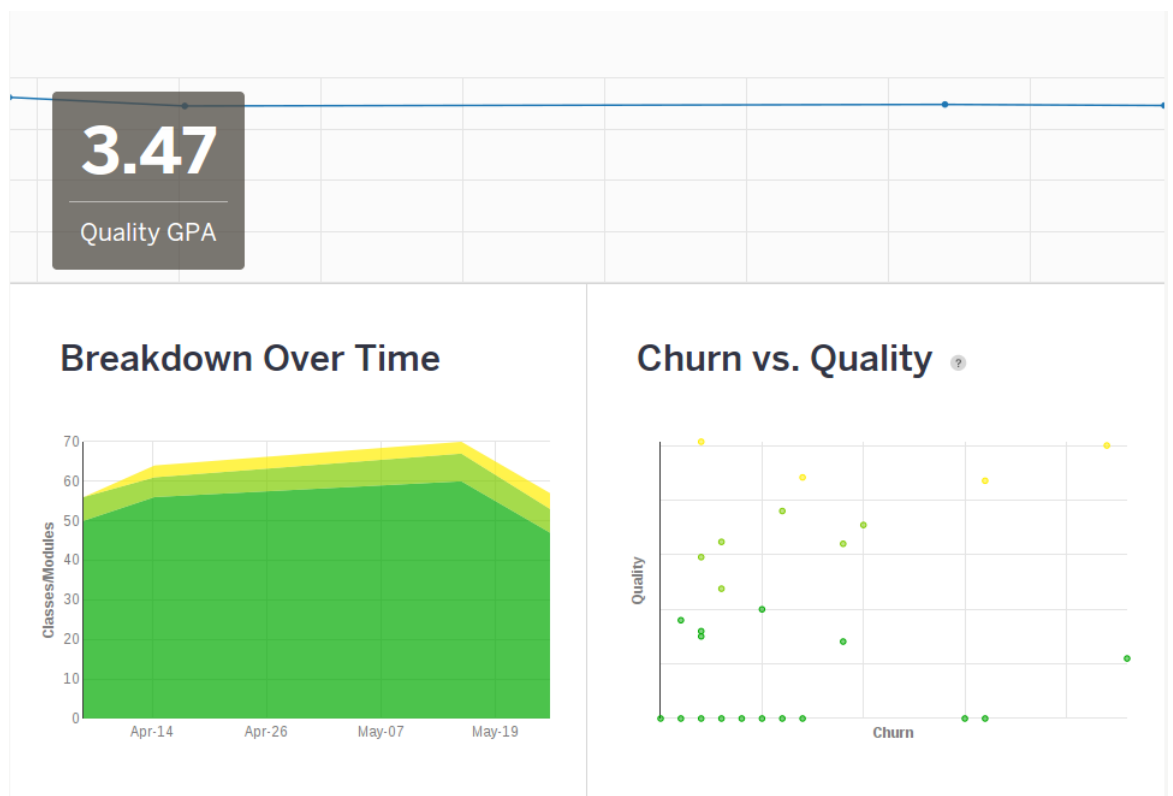


Figure 2: CodeClimate trends

7.0.4 Hakiri

This 3rd party application is used to show if there is any gem which is not up to date, and if there is any known security flaws in any of the gems we use in our project.

8 Group collaboration

When we got this assignment, we formed a Google Group to share our ideas and organize ourselves. It was also used to schedule weekly two-hour scrum meetings. These meetings started with a stand-up session, where everyone talked about what they were working on, any issues they had, and what they planned to work on that week.

In the sprint planning meetings, we consistently used scrum poker after we had made sure all the user stories were clear to everyone. We did not do any retrospectives apart from the ones planned in the course, with Weihai Yu.

Day to day we sat together in the lab, what ever each of us were working on. This meant that if one of us had an issue, they always had someone they could ask for help without too much effort. Everyone in the group was very helpful, so this arrangement worked out perfectly. Many of the programming sessions were done in pairs, which is a great way to avoid getting stuck for too long. During the last iteration, the weekly meetings were getting shorter and more informal because we prioritized programming, and we communicated naturally with an on-demand basis.

9 The development process

Using tools like GitHub and Travis CI we came up with the following workflow:

9.1 Git Workflow

Since we use Git together with GitHub, we are able to make use of the continuous integration tool Travis CI. Therefore we decided to go with the following development workflow. Please note that the term "origin" represents for the main turi repository on GitHub.

9.1.1 Developing of a new feature

The developer creates a local feature branch with a telling name. A feature always relates to a user story in Agilefant.

9.1.2 Starting a merge request

If the developer finished with the development of his local feature, he pushes the feature branch to his own remote repository (which is a fork of the origin repository). Before he pushes his changes, he has to do a rebase on the current develop branch of the origin to make sure all sources are up to date and we don't mess up the git history we thousand of branches. After making sure that

everything is up to date, he can create a merge request on GitHub from his feature branch to the origin develop branch.

9.1.3 Validating the merge request

After the merge request is submitted, it's open for discussion. For additional validation, Travis builds every merge request to ensure that all tests are running. If the Travis CI build is passing and the merge request can be fast forwarded (so the request was rebased) another developer can accept the merge. The person who accepts the merge should be never be the owner of the merge request.

9.1.4 Deploying to production

As soon as a merge request is accepted, Travis CI will run again against the latest sources of the origin develop branch. If the build is successful Travis will push the develop branch to the master branch. Therefore we will always have a stable version of turi on the master branch. A developer should never push changes directly to the master branch.

After a push to the master Travis will push the code to Heroku and run the database migrations. Therefore we always have a stable snapshot version on heroku.

9.2 Conclusion of the workflow

10 Something about the different sprints?

- What were our plans for the first iteration?
 - Basic functionality of trip
 - Authentication system
 - Integrate AppUI
- What did we learn from the first iteration?
 - Focus on model and control tests (codereview)
 - Check for gems before trying to implement something from scratch
 - ...
- Retrospect 1
- Plans for iteration 2
 - Implement a lot of features for the trip
 - * Map
 - * EquipmentList
 - * blog
 - * discussion
 - * FriendRequests
 - * Participants? (or did we do this in sprint 1?)

* Events

- What did we learn from iteration 2?
- Retrospect 2
- Plans for iteration 3
 - Clean up
 - Finish the last features
 - Make the design uniform (aka try to make things look the same)
 - Bug fixes
- what did we learn from the last iteration?

References

- [1] *Puma webserver - GitHub*
<https://github.com/puma/puma>
- [2] *Heroku Ruby webserver advise*
<https://devcenter.heroku.com/articles/ruby-default-web-server>
- [3] *Devise - GitHub*
<https://github.com/plataformatec/devise>
- [4] *Pundit - GitHub*
<https://github.com/elabs/pundit>
- [5] *Leaflet JS*
<http://leafletjs.com/>
- [6] *CKEditor*
<https://en.wikipedia.org/wiki/CKEditor>