

SANE Dashboard

Ye Song

July 30, 2016

1 Motivation

SANE(Server Access Network Entity) is a set of crowdsourcing proxies, which plays a important role and act as intermediaries between between crowdsourcing clients and crowdfunding servers. The crowdsourcing clients are not allowed to communicate with crowdsourcing servers directly. Every request must be forwarded by SANE. The Architecture of SANE is based on DHT(Distributed Hash Table). This means, each node of SANE is responsible for a part of data storage and routing. The whole architecture is completely decentralized.

Kademlia is a typical implementation of DHT designed by Petar Maymounkov and David Mazières in 2002. Each node within the DHT is identified by a unique node which is generated without taking real geographical position into account. Within SANE the logical distance between nodes is represented by exclusive or (XOR) of the two node IDs. Due to the geographical independence of traditional Kademlia protocol, the clients of crowdsourcing may be far away from the SANE service node geographically. This will always lead to intolerable network latency. Given the above fact, the author of SANE has extended the traditional Kademlia protocol by prefixing existing 160 bits ID of node with 3 bits continent code and 10 bits country code. On the one hand, geographical position has been considered by computing the ID of node. On the other hand, the existing protocol of Kademlia has been respected seamlessly.

Our project, SANE Dashboard is designed for visual management of single SANE node. The project is found on existing project SANE and provides features of checking current node status, looking up user or device information, displaying statistical data, geographical information and submissions of crowdsourcing clients. In addition, the system is also capable of presenting the last geographical position of users and hierarchy of neighbors within the K-Buckets.

2 Related Work

2.1 Related Work

2.1.1 MapBiquitous

The system MapBiquitous was designed by Thomas Springer, Tenshi Hara, Gerd Bombach and Sina Grunau. MapBiquitous aims to provide integrated system for indoor/outdoor location-based services. It plays the role of middleware that provides uniform interface for applications built on different positioning technologies to bridge the heterogeneity between these technologies. It uses open standards for modeling building data with geometric and semantic information. MapBiquitous is based on C/S architecture.

The fat client is responsible for:

- 1 Storage of building data based on hybrid location model(both geometric and semantic location information)
- 2 Render gets necessary building data from loader module and enables the visualization of the data, the building data is represented on top of map layer.
- 3 Locator is built on top of a set of different positioning technologies. A common high level interface is provided on top of these technologies to bridge the heterogeneity between them.
- 4 Loader could be triggered by render module to retrieve building data from the building server that is responsible for the current building. The search for building servers is via directory service.

MapBiquitous Server: The building data and positioning infrastructure are stored on building servers, including building geometric information(floors, rooms, stairs, et cetera) and semantic information(room type, room number, et cetera). User is able to navigate a building using a MapBiquitous client that connects to a building server which maintains building information. Whether some areas of the building is public or not, depends on owner. This means, private information could be well protected.

Directory Service: These building servers are architecturally decentralized and could be found via Directory Service. Directory service is based on WGS84 coordinates and has been implemented using J2EE specifications(Tomcat, JSP) and MySQL. Directory Service supports search for building servers by name, longitude and latitude box. Open source project MapServer is applied for building data provisioning.

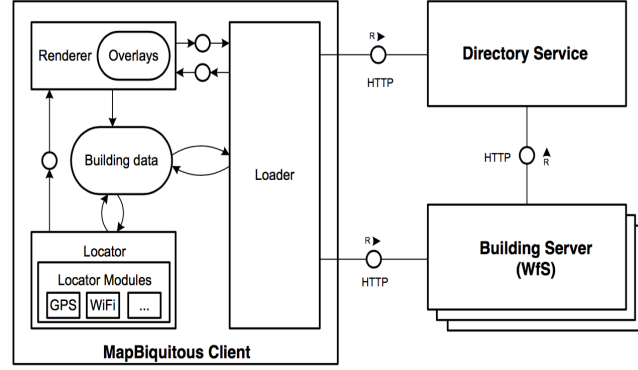


Figure 1: Architecture of MapBiquitous

2.1.2 SANE

SANE(Server Access Network Entity) has been put forward and designed by Tenshi Hara in his paper “Decentralised Approach for a Reusable Crowdsourcing Platform Utilising Standard Web Servers”. SANE plays the role of middleware platform that simplifies development of crowdsourcing applications. SANE is architecturally based on Distributed Hash Table that is a typical decentralized structure. Each node within a DHT is responsible for partial data storage and partial routing. Figure 2 shows the mechanism of DHT of SANE. SANE 1 is responsible for area[0-3], as well as area[c-7]. SANE is responsible for area[4-7], as well as area[0-b]. SANE 3 is responsible for area[8-b], as well as area[0-f]. SANE 4 is responsible for area[c-f], as well as area[8-3]. The hashed userid of client begins with character ‘b’, the node of SANE that is responsible for area ‘b’ is SANE 3 [8-b]. Its upper neighbor SANE 2 and lower neighbor SANE 4 act as backups.

There is no central node within a DHT, this property eliminates bottleneck at central node in a traditional centralized network. DHT is an efficient approach to organize huge numbers of nodes and construct a decentralized network. A node could be easily added to or removed from a DHT network. This means, SANE is well scalable to deal with vast numbers of clients. Given fault tolerance, each node is responsible for backups of its neighbors via storage of partial extra data. SANE enables user management, client management, server management, DHT maintainer and security. SANE ensures that all internal communication between modules are signed and encrypted. SANE also ensures that all write-access to crowdfunding servers must be re-signed and re-encrypted to keep the submitted data traceable. SANE store all additional information related to the submissions, the submissions are stored on crowdfunding servers. SANE project has been im-

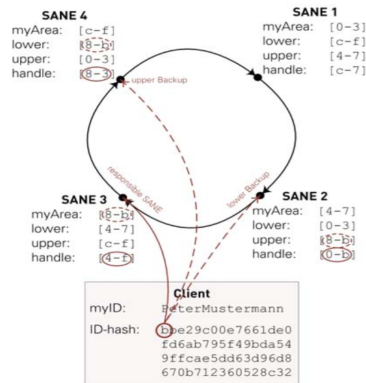


Figure 2: DHT of SANE

plemented using standard technologies LAMP(Linux, Apache, MySQL, PHP). MapBiquitous's Android client could communicate with SANE proxy by simply adding a SANE library, this means, SANE acts as a mediator between MapBiquitous clients and MapBiquitous servers.

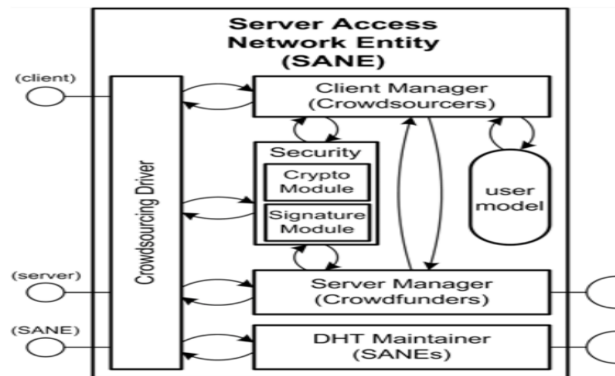


Figure 3: Architecture of SANE

2.1.3 Location based DHT

Kademlia(Kad) is a typical structured P2P overlay network. It aims to solve information storage and retrieval in distributed environment. In Kademlia network, all information is stored in the form of entries within a hash table. These entries are discretely stored on each node, thereby constituting a huge distributed hash table. The huge distributed table could be considered as a large dictionary. As long as we know the key of the information, we will be able to query its corresponding

value information through the Kademlia protocol regardless of the physical position of the node. Each node within a Kad network has a unique ID that is a 160 bits hashed value. It is considered almost impossible that two nodes have the same ID. In the Kad network, the distance between two nodes don't depend on their physical distance. In fact, the distance is calculated by the XOR operation. We assume that there are two nodes, a and b, the distance $d = a \text{ XOR } b$, the larger the value of d, the larger the distance between a and b. The distance calculated by XOR is only a logical distance regardless of physical position.

In Kad network, each node maintains 160 lists, each of which is called K-Bucket. In list i, neighboring nodes that are $2^i \sim 2^{(i+1)}$ away from current node are stored in the form of (Node ID|IP|UDP port) tuple. Default value of K is 20.

The update of K-Buckets follows these principles:

- 1 The list is not full (number of nodes less than K), if the queried node is not in the list, the node will be added to the tail of list. If the node is already in the list, then the node will be moved to the tail of list.
- 2 The list is already full, and the queried node is not in the list. If the head node is still active, the new node will be thrown, the head node will be moved to tail, else the head node will be removed and the new node will be added to tail of list.

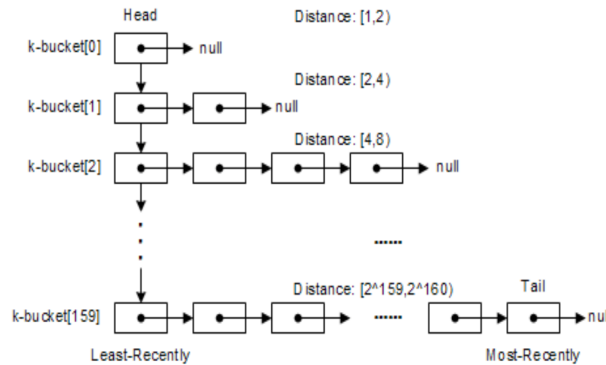


Figure 4: K-Buckets

Kademlia network doesn't take real position into consideration. This means, service node that is responsible for the current client might locate far away from the client. This will cause unnecessary network overhead and compromise user experience. For this reason, Hara proposed a solution in his paper "Decentralised Approach for a Reusable Crowdsourcing Platform Utilising Standard Web Servers". Basically this solution is based on DNS to retrieve nearby service node for the client. Steps of this solution are shown in the following figure:

- Step 1: A client that locates in US contacts service node locating in Germany.

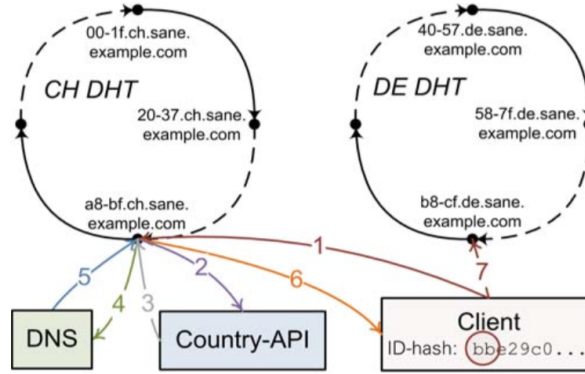


Figure 5: Solution based on DNS

Step 2, 3: Service node determines where the client comes from via Country-API.

Step 4, 5: Service node gets another service node locating in US via DNS.

Step 6, 7: Client contacts the new service node that is responsible for its area.

Given the complexity due to additional DNS, Tan Dung Nguyen proposed a new solution in his paper “Auswahl und Implementierung eines geeigneten Distributed Hash Table Verfahrens fuer die selbstaendig regionale Verteilungen beruecksichtigende Organisation einer verteilten Prokura-Kommunikation in MapBiquitous” basically without modification of original routing algorithm. Main idea is prefixing the existing 160 bits hash code with additional positional information that doesn’t affect routing algorithm. The K-Bucket of a node in normal Kademlia network without considering real position looks like the following:

Table 1: K-Buckets Considering No Real Position

Node-ID	K-Buckets				
0001	distance: 0	distance: 1	distance: 2-3	distance: 4-7	distance: 8-15
	0001	0000	0010	0100	1000
			0011	0111	1101

If nodes 0001, 0000, 0010, 0111, 1000, 1101 locate in Germany, other nodes 0011, 0100 locate in US. We assume that nodes locating in Germany are prefixed with 0, nodes locating in US are prefixed with 1. Logical distance is still calculated via XOR operation. Then the K-Buckets look like the following: The nodes locating in US have been moved to K-Bucket 16-30, other nodes remain in original K-Buckets. The routing algorithm remains unchanged. In Dung’s paper, he proposed 13 bits code to prefix 160 bits hash code. 3 bits are used to identify 5 continents. 10 bits are used for identify all countries(nowadays 241 countries in the world)

Table 2: K-Buckets Considering Real Position

Node-ID	K-Buckets					
	Dis: 0	Dis: 1	Dis: 2-3	Dis: 4-7	Dis: 8-15	Dis: 16-31
00001	00001	00000	00010	(0100)	01000	10011
			(0011)	00111	01101	10100

according to ISO-3166 (International Organization for Standardization) without considering secondary-level region, z.B. province, city due to workload. In the following table, we show a sample of this model, we assume that Europe is coded as 010, Germany is coded as 0100010100. Our DHT module is based on this model.

Table 3: 173 Bits Hash Code

continent	country	160 bits hash code
010	0100010100	010001010001000101000100010100....

2.2 Organization

There are three members in our team, Guanghui Ji, Kaijun Chen and me. We met every week to share the designs for our project during analysis period. After this period, we had to meet more frequently to discuss protocol between frontend and backend, user interface design, technology selection, allocation of tasks, etc. Due to the detailed designs we could focus on implementation in the next period.

The two main tools that we applied for management and organization were Google Drive and Trello. Google Drive, formerly Google Docs, is a file storage and synchronization service created by Google. It allows users to store files in the cloud, share files, and edit documents, spreadsheets, and presentations with collaborators. All of our analysis and designs documents, protocols, related papers, database configuration, etc. are stored in Google Drive. Given the commerciality and closed source of JIRA, we took Trello as our collaboration tool. Trello is a web-based free application that organizes our project into boards, lists, cards and gives user a visual overview of “doing”, “to do” and “done” tasks and the persons in charge of these tasks. Trello also provides notification through Email, so that we could keep up with project progress and assign tasks conveniently.

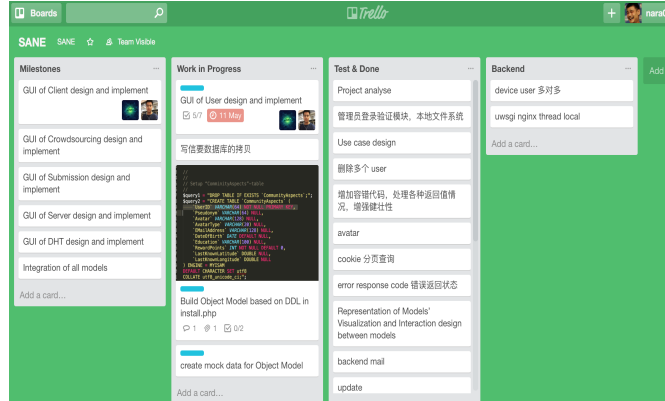


Figure 6: Overview of Trello

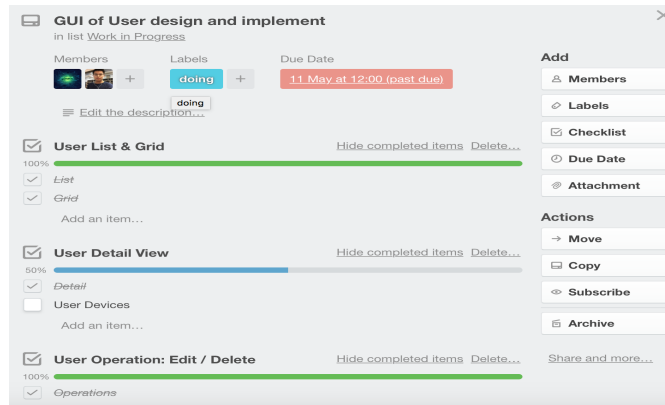


Figure 7: Overview of Trello 2

2.3 Research of Existing Project

The project SANE Dashboard is based upon the project SANE. Therefore we need to analyze SANE to acquire the important information, e.g. schema of tables in database, restful API provided by SANE, directory service of crowdsourcing servers, DHT neighbors within K-Buckets etc.

Schema of tables locate in PHP module “install.php”, e.g. table “UserData” and “Devices” as following:

```
CREATE TABLE UserData (UserID VARCHAR(64) NOT NULL PRIMARY KEY, Username VARCHAR(64) NOT NULL, Password VARCHAR(64) NULL, PublicKey TEXT NULL, RegistrationDate DATETIME NOT NULL ) ENGINE=MYISAM DEFAULT CHARACTER SET utf8
```



```
CREATE TABLE Devices (DeviceID VARCHAR(64) NOT NULL PRIMARY KEY, UserID VARCHAR(64) NOT NULL,
DeviceSpecs BLOB NULL, RegistrationDate DATETIME NOT NULL, ModificationDate DATETIME NOT NULL,
FOREIGN KEY (UserID) REFERENCES UserData(UserID) ) ENGINE=MYISAM DEFAULT CHARACTER SET
utf8
```

Directory Service can be found under “libs/Crowdfunder.php”:

```
$_CONFIG['DS']="carlos.inf.tu-dresden.de"
$_CONFIG['DSport']="80"
$_CONFIG['DSpath']="DS/"
$_CONFIG['DSssl']=false
```

RESTful APIs of SANE are available at <http://sane.hawk310.startdedicated.de>. We can acquire some useful information through the RESTful APIs, e.g. “get-Location”, “getDHTNeighbors”, “getDHTArea”, “getAdmin”. With the help of these APIs, it is possible to construct our DHT module.

2.4 Analysis and Usecases

A use case is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal. Use case analysis technique has been widely applied in requirements analysis of modern software engineering. Our project also began with detailed use case analysis. Basically our system consists of six parts, namely User Management, Device Management, Server Management, DHT Management, Submission Management, Device Management. User will login as administrator of current SANE node. User Management includes userinfo retrieval, userinfo update, userinfo removal. Device Management includes deviceinfo retrieval, deviceinfo removal. Server Management includes serverinfo retrieval. So are DHT Management and Submission Management. The Administrator could also notifies crowdsourcing clients through Email.

We applied PlantUML as the UML tool for modeling usecase. PlantUML is a open-source tool that allows user to generate UML diagram from plain-text language. PlantUML uses well-formed and human-readable script to render the diagrams. PlantUML is based on Graphviz that is responsible for layout of diagrams. PlantUML supports many popular IDE and Editors as plugin or extension, e.g. Eclipse, IntelliJ IDEA, NetBeans, Microsoft Word, LaTeX, etc.

A simple example of PlantUML script as following:

```
@startuml
title SANE Management
actor Admin:
Admin--(SANE Management)
(SANE Management)--(User Management):include
(SANE Management)--(CS Management):include
(SANE Management)--(Server Management):include
```

```

(SANE Management)..(Submission Management):include
(SANE Management)..(DHT Management):include
@enduml

```

The diagram generated by the the above script looks like the following:

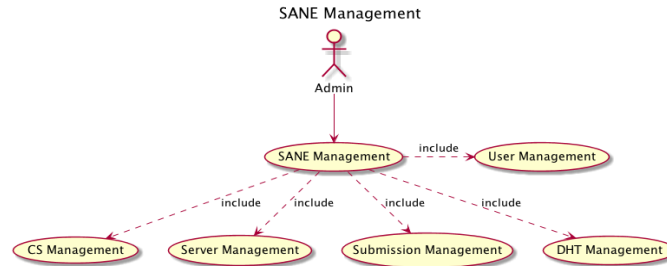


Figure 8: use case diagram

Original complete usecase diagram is available in our project folder on Google Drive.

2.5 User Interface Design

Our project is a web application that enables the administrator to manage SANE node visually. Therefore Graphical User Interface is a critical factor to user experience. GUI must be easy for use, style-consistent. In order to achieve compatibility between desktop and mobile devices, responsive layout has been applied at frontend of our project. This will promote user experience greatly. We have used Mockups as the prototype design tool for GUI at the beginning of the project. Mockups is a awesome prototype design tool for web application that is able to run on different platforms, e.g. Windows, Linux, FreeBSD, Mac OS, etc. Mockups also supports online/offline mode. Therefore we could construct a web application prototype in a short time to instruct our future development effectively. Figure 4 is user view prototype and figure 5 is statistical diagram of device view:

Original complete mockups prototype is available in our project folder on Google Drive.

2.6 RESTful API Negotiation

We choose RESTful API to provide data for front-end of our system. REST(Representational State Transfer) is a kind of architectural style or design style, but not a concrete

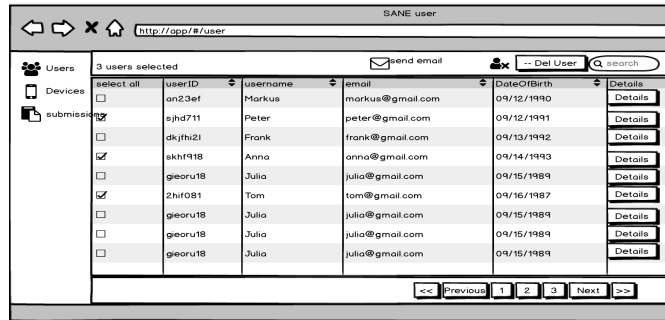


Figure 9: Prototype User View

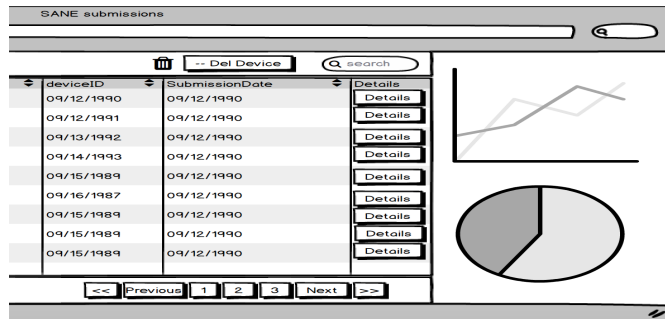


Figure 10: Prototype Statistical Diagram of Device view

standard. All of the Resources could be identified by Uniform Resource Identifiers. RESTful systems typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers. RESTful API is widely applied in client-server architectural software to achieve loosely coupled systems. Front-end could be easily separated from back-end, both sides could be developed independently. This benefit has been proven by our later development.

The RESTful APIs have been published formally using swagger as a extension of Flask web framework. This tool will be introduced later in Technology Selection subsection. The request message format and the return value format have been negotiated and defined before development. Following figures show the RESTful APIs and message format of some APIs. The complete protocol is available both in file “API_Specs.md” and on <http://127.0.0.1:5000/api/spec.html>

spec : SANE API Show/Hide List Operations Expand Operations Raw

DELETE	/api/v1.0/users
GET	/api/v1.0/users
DELETE	/api/v1.0/users/{user_id}
GET	/api/v1.0/users/{user_id}
POST	/api/v1.0/users/{user_id}
DELETE	/api/v1.0/devices
GET	/api/v1.0/devices
DELETE	/api/v1.0/devices/{device_id}
GET	/api/v1.0/devices/{device_id}
DELETE	/api/v1.0/submissions
GET	/api/v1.0/submissions
DELETE	/api/v1.0/submissions/{submission_id}
GET	/api/v1.0/submissions/{submission_id}
GET	/api/v1.0/servers
POST	/api/v1.0/email

Figure 11: API Overview

GET /api/v1.0/submissions

Implementation Notes
This method returns a submission list

Response Class
Model **Model Schema**

```

{
  "data": [
    {
      "crowdsourcing": "",
      "deviceused": "",
      "id": 0,
      "serverID": "",
      "submissionDate": "",
      "submissionID": "",
      "submitterID": "",
      "type": "",
      "userID": ""
    }
  ]
}

```

Response Content Type application/json

Error Status Codes
HTTP Status Code Reason

Figure 12: Model Schema

2.7 Technology Selection

Our system mainly consists of three layers, namely frontend presentation layer, business-logic layer, persistence layer. Backend is responsible for most of the heavy lifting and integration work of our product. It handles data persistence, retrieval, caching, relationship, email delivery, user management, cookie, session, etc. Frontend is responsible for presentation and virtual management. Frontend is somehow like single page application, e.g. Gmail. We use NPM(node package management) to manage our tools at frontend. NPM is the package management tool on Node.js. Install, Uninstall, Update, Query, Distribution of Node.js package could be completed automatically. NPM has liberated programmer from heavy package management. Programmers are able to focus on development of product features. The best way to apply NPM is to create a package.json file. This file has afforded us lots of good things. We didn't have to install a package step by step. What we need to do is to write a package record to package.json. Furthermore, package.json allows us to specify the version of a package. Package.json helped sharing of package between different developers a lot.

Gulp and Bower were installed automatically using NPM. Gulp is a Node.js-based frontend tool for development and deployment automation. It's not only capable of optimizing website resources, but also automating many repetitive tasks during development. There is a file `gruntfile.js`, where we could define our tasks, in JavaScript or CoffeeScript. e.g. linking JavaScript and CSS, parsing templates, compiling LESS.

Figure 8 shows a script snippet of self-defined gulp tasks:

```

52  /**
53   * Handle custom files
54   */
55   gulp.task('build-custom', ['custom-images', 'custom-js', 'custom-css', 'custom-templates']);
56   gulp.task('build-custom-unminified', ['custom-images', 'custom-js-unminified', 'custom-css-unminified', 'custom-templates']
57
58   gulp.task('custom-images', function() {
59     return gulp.src(paths.images)
60       .pipe(gulp.dest(path.join(paths.dist, 'img')));
61   });
62
63   gulp.task('custom-js', function() {
64     return gulp.src(paths.scripts)
65       .pipe(minifyJs())
66       .pipe(concat('sane-dash.min.js'))
67       .pipe(gulp.dest(path.join(paths.dist, 'js')));
68   });
69   gulp.task('custom-js-unminified', function() {
70     return gulp.src(paths.scripts)
71       .pipe(concat('sane-dash.min.js'))
72       .pipe(gulp.dest('dist/js'));
73   });

```

Figure 13: Gulp Tasks

Bower is a opensource package management tool for frontend development developed by twitter. It enables automatic installation of JavaScript and CSS dependencies. Bower and NPM overlap in functionality to some extent. Nowadays NPM was known not only as Node Package Management, but also as JavaScript Package Management. This means, NPM could be applied both in frontend and backend development. NPM supports nested dependency, this means, each package might have its own subdependencies. This is really great on the server where you don't have to care much about space and latency. But one obvious drawback is that multiple copies of the same dependencies in different versions may be downloaded. We used Bower as our frontend package management due to its flat dependency tree. Bower requires only one version for each package to reduce page load to a minimum. We just need to write a dependency into `bower.json` file. The package will be installed automatically. Figure 9 shows a snippet of `bower.json`:

Bootstrap comes from Twitter, it is a very popular front-end framework currently. Bootstrap is built on HTML, CSS, JAVASCRIPT, it is simple and flexible, so that Web development has been greatly simplified. Bootstrap contains a wealth of Web components, with these components, we could quickly build a beautiful, fully functional website. Bootstrap has been applied to build our frontend UI and responsive layout. Desktop client and mobile client could be perfectly compatible with each other.

```

12  },
13  "main": "index.html",
14  "license": "MIT",
15  "ignore": [
16    "node_modules",
17    "bower_components",
18    "test",
19    "tests"
20  ],
21  "dependencies": {
22    "angular": "~1.5.0",
23    "angular-bootstrap": "~1.0.3",
24    "angular-cookies": "~1.5.0",
25    "angular-ui-router": "~0.2.15",
26    "bootstrap": "~3.3.6",
27    "Chart.js": "~2.1.2",
28    "font-awesome": "~4.5.0",
29    "jquery": "~2.2.3",
30    "normalize-css": "~4.1.1",
31    "rdash-ui": "1.0.0",
32    "slick-carousel": "~1.6.0",
33    "sweetalert": "~1.1.3"
34  },
35  "repository": {
36    "type": "git",
37    "url": "https://github.com/rdash/rdash-angular"
38  },
39  },

```

Figure 14: bower.json

AngularJS was developed by Misko Hevery in 2009 and then purchased by Google to address many of the challenges encountered in developing single-page applications. AngularJS provides many awesome features, e.g. client-side modelview-controller (MVC), modelviewviewmodel (MVVM), Two-way Data Binding. AngularJS presents a higher level of abstraction to simplify application development. Like other abstract technology, angularjs will also lose some flexibility. In other words, AngularJS is not suitable for all applications. It is mainly used to build CRUD applications. Fortunately, at least 90% of WEB applications are CRUD applications. Our project is no exception.



Figure 15: Frontend Route MVC

Chart.js is a HTML5-based chart library that uses the canvas element to show a wide range of client charts, line charts, bar charts, radar charts, pie charts, ring maps. In each chart, it contains a large number of customization options, including animated display format. Chart.js is relatively lightweight (gzip version only 4.5k), and does not depend on other libraries. In our project, Chart.js is used to draw statistical results of SANE node, including registered user distribution, age

distribution, education registration, etc.

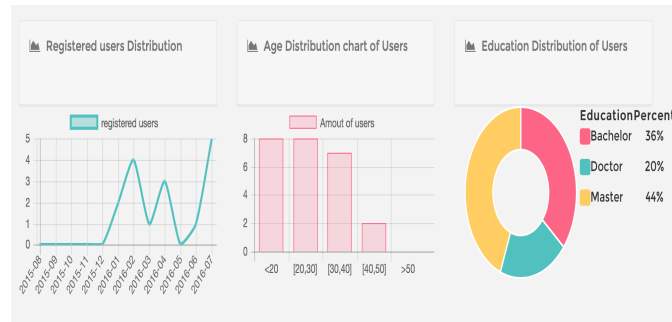


Figure 16: Chart.js

We applied Python as programming language in backend. Python is very easy to learn, it has a simple and elegant syntax. Usually code in Python that provides some features is much simpler than other languages. This means, the development cycle could be largely shortened.

Because of teamwork, we had to unify our development environment, including Python version, dependencies versions. Pyenv is a Python version management tool that enables coexistence of multiple versions and allows user to switch Python versions easily. This affords us a development version of Python without affecting system Python environment. We used Python 2.7.11 as the development version.

Dependency management is a critical factor that affects teamwork greatly. Pip is a popular package management tool of Python. Many packages can be found in the Python Package Index (PyPI). Pip support management of a full list of packages through a “requirement” file. This enables the efficient recreation of the original packages in a new environment. The command looks like “pip install -r requirement.txt”. But pip is only responsible for dependency management, it has no concern about project build and deployment. We need a tool like Maven of Java that permits automatic project build and deployment as well as dependency management. zc.buildout has fulfilled these tasks. Buildout is a tool developed by Zope Corporation’s JimFulton that aims to create cross-platform reusable program. Buildout could build, compose, deploy an application from multiple parts through a configuration file “buildout.cfg”. Buildout could not only automatically install, update dependencies like setuptools, but also construct a separate environment like virtualenv. Figure 12 shows a script snippet of our buildout.cfg file:

Our project is a web application. The world of Python web frameworks is full

```

1 [buildout]
2 parts = dependencies start_server
3 develop = sane-backend
4 versions = lib_versions
5
6 [dependencies]
7 recipe = zc.recipe.egg
8 eggs = flask
9 flask-RESTful
10 flask-RESTful-swagger
11 sane_backend
12 MySQL-python
13 Flask-SQLAlchemy
14 Flask-Cache
15 Flask-Mail
16 Flask-Login
17 Flask-Script
18
19 [start_server]
20 recipe = zc.recipe.egg:scripts
21 eggs = ${dependencies:eggs}
22 entry-points = start=manage:run
23 scripts = start
24
25 [lib_versions]
26 flask = 0.10.1
27 flask-RESTful = 0.3.5

```

Figure 17: buildout.cfg

of choices. Django, Flask, Pyramid, Tornado, Bottle, Diesel, Pecan, Falcon, and many more are competing for developer mindshare. It seems like that Django dominates the world of Python web. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Django consists of a wide range of solutions for almost all fields in web development, e.g. template engine, ORM, form, route distribution, login, cookie, session, etc. Community of Django, both official and unofficial(Stackflow and IRC) community are very active. Basically any problems we encountered will have mature solutions.

Flask is just the opposite of Django. Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine. It is BSD licensed. Flask framework itself contains only routing distribution, request/response encapsulation, plugin system, Werkzeug(process WSGI), Jinja2(template engine) and Flask-SQLAlchemy extension(Model layer). These form a complete MVC framework. For Flask itself, a micro- framework, it was enough, it can only provide so much. If more additional features are needed, we just need to extend Flask by Flask plugins, namely extensions of Flask. If we need to process forms, just install Flask-WTF, if we need login authentication, just install Flask-Login, if we need Administration, just install Flask-Admin. Due to the lightweight of Flask, we used Flask as our web framework.

Basically our project is a CRUD web application. Database access is a basic feature. We chose SQLAlchemy as our ORM(Object Relation Mapping) framework. Despite strong influence of Django SQLAlchemy is de facto standard of ORM framework in the world of Python. SQLAlchemy has been widely used in the next generation Python Web framework built on WSGI standard. One advantage of using SQLAlchemy is that it permits developers to first consider the data model, and then decide later visual data mode(using the command-line tool, Web GUI framework or frame).

Our backend published RESTful API for frontend. This means, backend and frontend could be developed simultaneously. Flask Framework is capable of publishing RESTful service. A sample of Flask published API looks like the following:

```
from flask import abort
@app.route('/todo/api/v1.0/tasks/<int : task_id>', methods=['GET'])
def get_task(task_id):
    ____task = [task for task in tasks if task['id'] == task_id] ____if len(task) == 0:
    _____abort(404):
    ____return jsonify('task': task[0])
```

But we chose Flask-RESTful as our extension instead of native Flask. Flask-RESTful simplifies creation of RESTful service and it encourages best practices with minimal setup. A sample of Flask-RESTful looks like the following:

```
from flask import Flask
from flask.ext import restful
class HelloWorld(restful.Resource):
    ____def get(self):
    _____return 'hello': 'world'
```

After the API on server side completed, we should inform clients how the API could be used, how are the details of API. We chose Flask-RESTful-Swagger as our RESTful API tool instead of maintaining a single document manually. The obvious shortcoming of the latter approach lies in that it is difficult to maintain the accuracy of all documents, as the server-side API updates. We have to make extra effort to track changes of API. Swagger makes it easy to publish RESTful API with projects. It supports many popular programming language including Python. Flask-restful-swagger is a wrapper for Flask-RESTful which enables swagger support. In essence, we just need to wrap the API instance and add a few python decorators to get full swagger support.

Usually during the development of RESTful service, we must consider the cache to improve the performance of service. We chose Flask-Cache as our extension for caching. If we want to use simple caching for test, the built-in SimpleCache is a good choice. Data will be cached in memory of Python interpreter. If we want more powerful caching system, e.g. memcached, make sure to have one of the memcache modules supported and a memcached server running somewhere. This is how we connect to such an memcached server:

```
from werkzeug.contrib.cache import MemcachedCache
cache = MemcachedCache(['127.0.0.1:11211'])
```

In our project we used the built-in SimpleCache for caching. If we want to promote performance of data access in the future, it's very easy to switch to other dedicated caching system.

As a web-based management dashboard, administrator must be able to notify users by Email when something happens. Flask-Mail has been applied as our Email module. We could set SMTP in our application and add attachments easily.

A important task of backend is to provide login related features for frontend, e.g. logging in, logging out, and "remember me" for a session or cookie for a long period. Flask-Login provides a nice solution for us. We could integrate these features into our system. But Flask-Login is a little more complex than other extensions.

We used Bitbucket as our project hosting server and Git as our version control system. Because the original database was not available during development, we had to mock up data according to schema described in "install.php" for test. All of data could be accessed on the remote DB server.

3 Concept

3.1 Backend Concept

RESTful API:

GET /api/v1.0/users ___get all users from database

PUT /api/v1.0/users/user_id ___update user specified by userid

DELETE /api/v1.0/users ___delete users specified by userids

DELETE /api/v1.0/users/user_id ___delete user specified by userid

DELETE /api/v1.0/devices ___ delete devices specified by deviceids

DELETE /api/v1.0/devices/device_id ___ delete device specified by deviceid

GET /api/v1.0/devices ___ get all devices from database

GET /api/v1.0/devices/device_id ___ get device specified by deviceid

DELETE /api/v1.0/submissions ___ delete submissions specified by submissionids

GET /api/v1.0/submissions ___ get submissions specified by deviceid, userid, serverid

DELETE /api/v1.0/submissions/submission_id --- delete submission specified by submissionid

GET /api/v1.0/submissions/submission_id --- get submission specified by submissionid

GET /api/v1.0/servers --- get all crowdsourcing servers

GET /api/v1.0/servers/server_id --- get single crowdsourcing server by serverid

POST /api/v1.0/email --- send email to one or multiple users

GET /api/v1.0/dht --- get neighbors within K-Buckets

GET /api/v1.0/admin --- get administrator of this SANE

GET /api/v1.0/owner --- get owner of this SANE

Objectmodel and Relation in page 14:

Package Class Diagram in page 15:

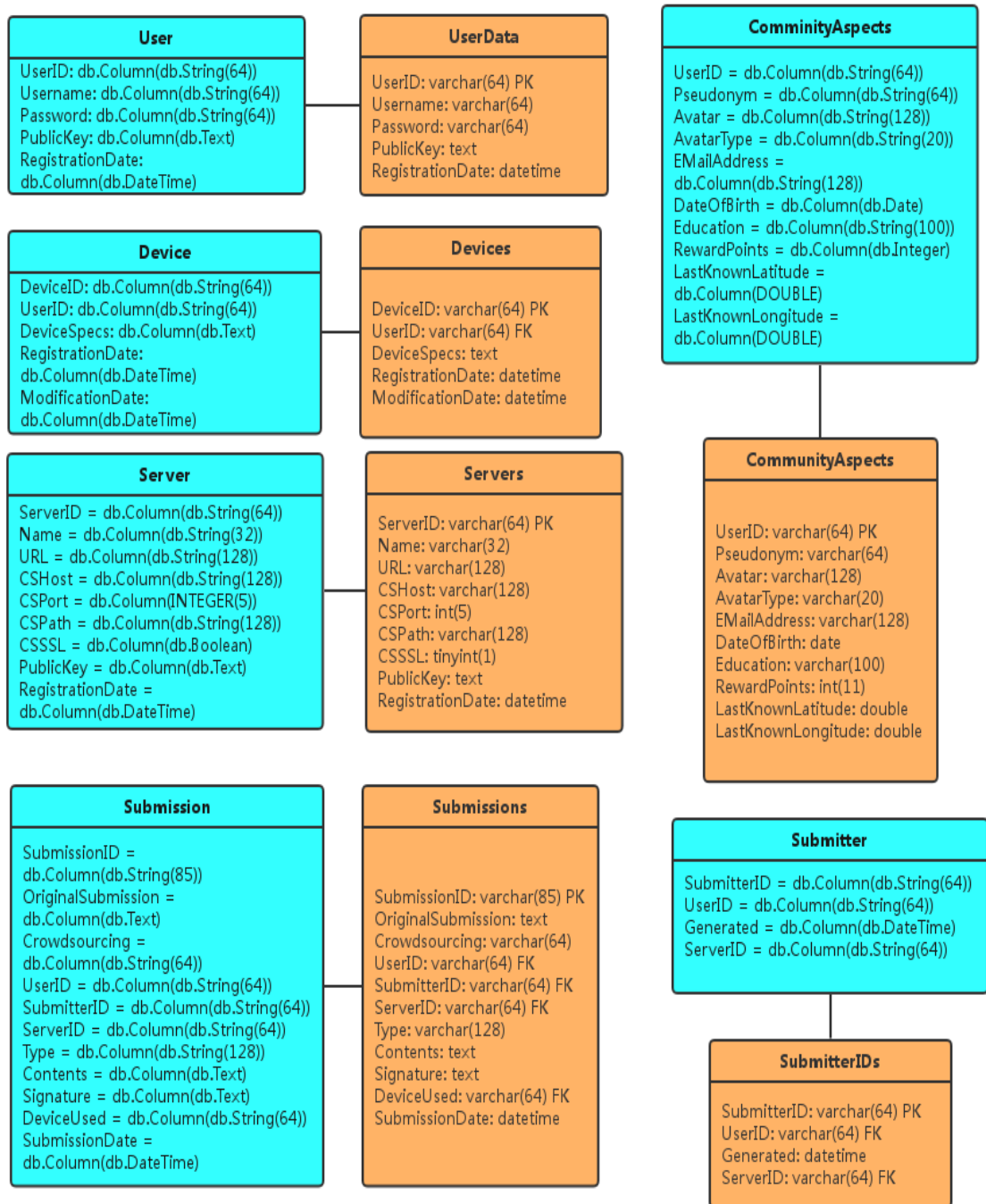


Figure 18: Object Relation Mapping

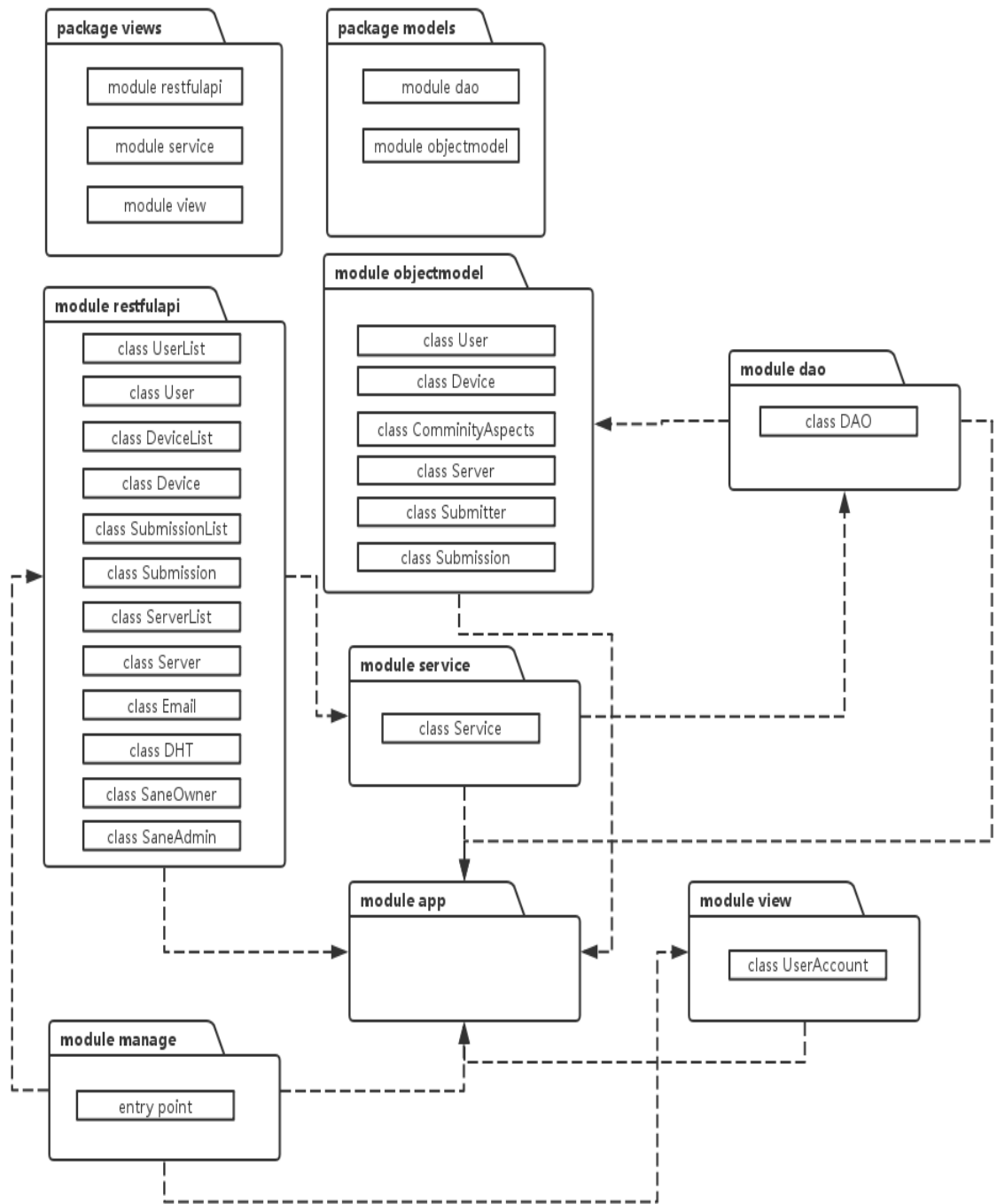
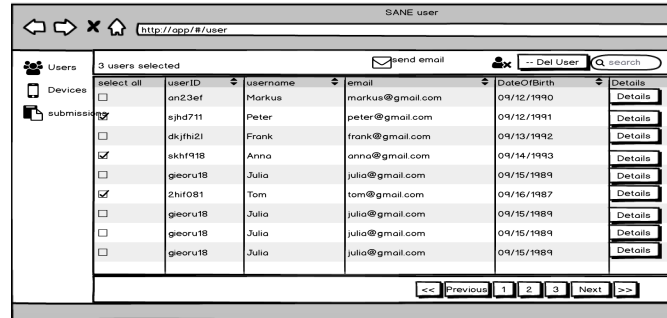


Figure 19: Package Class Diagram

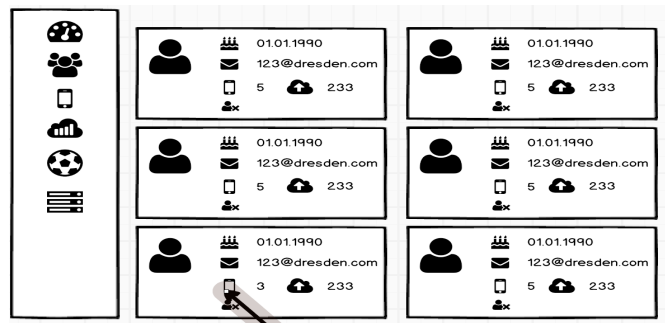
3.2 Frontend Concept

In frontend, there are some main modules, e.g. Dashboard, Overview, Users, Devices, Submissions, Servers, DHT. Each module is responsible for visualization of an aspect of data stored in SANE. There are two basic views for each module, e.g. user module, there are list view and grid view of users.



	userID	username	email	DateOfBirth	Details
<input type="checkbox"/>	an23ef	Markus	markus@gmail.com	09/12/1990	Details
<input checked="" type="checkbox"/>	sjhd711	Peter	peter@gmail.com	09/12/1991	Details
<input checked="" type="checkbox"/>	dkjfh2l	Frank	frank@gmail.com	09/13/1992	Details
<input checked="" type="checkbox"/>	akht918	Anna	anna@gmail.com	09/14/1993	Details
<input type="checkbox"/>	gieoru18	Julia	julia@gmail.com	09/15/1989	Details
<input checked="" type="checkbox"/>	2ht081	Tom	tom@gmail.com	09/16/1987	Details
<input type="checkbox"/>	gieoru18	Julia	julia@gmail.com	09/15/1989	Details
<input type="checkbox"/>	gieoru18	Julia	julia@gmail.com	09/15/1989	Details
<input type="checkbox"/>	gieoru18	Julia	julia@gmail.com	09/15/1989	Details

Figure 20: List View of User



		01.01.1990		123@dresden.com		5		233
		01.01.1990		123@dresden.com		5		233
		01.01.1990		123@dresden.com		5		233
		01.01.1990		123@dresden.com		3		233
		01.01.1990		123@dresden.com		5		233

Figure 21: Grid View of User

In overview module, all statistical data are shown in different chart created by chart.js. In user module, user can be shown with geographical position in Map. In DHT module, tree-style structure has been applied for displaying the neighbors within the K-Bucket of a SANE node.

A sample of statistical data in overview Module.

3.3 Implementation

The source code of our project is stored privately in Bitbucket. My work was mainly backend implementation. Ji and Chen implemented frontend mainly. Frontend files were

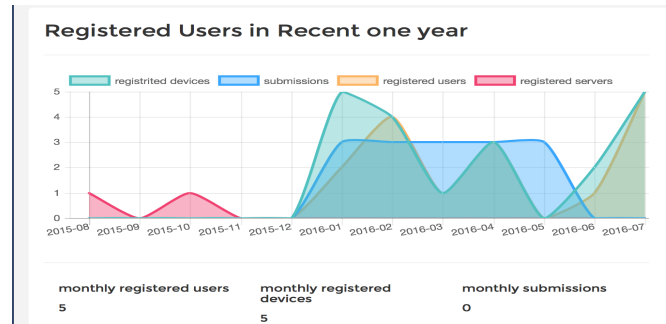


Figure 22: Statistical View in Overview

compiled as static resources and located in the static folder in backend project.

3.4 Evaluation

Horizontal Evaluation:

cohesion	maintainability	correctness	reliability
✓	✓	✓	×

extensibility	fault-tolerance	efficiency	adaptability
✓	×	✓	✓

robustness	scalability	understandability	elegance
×	×	✓	✓

Figure 23: Horizontal Evaluation

From the view of horizontal evaluation, our project meets these criteria:

Cohesion: Our backend is separated from frontend architecturally. It is easy to maintain, reuse, understand. So it is a system with high cohesion.

Maintainability, Extensibility: Besides separation of backend from frontend, backend is based on modular design, frontend is based on AngularJS that has a MVVM (Model-View-ViewModel) pattern. Both sides are easy to maintain and extend.

Correctness: Our system performs basically correctly during development and test. It conforms to users specific needs and expectations.

Efficiency: Our system consumes little resources due to optimized DB operations and

basic cache mechanism. The frontend-backend separated architecture has avoided unnecessary data transmission.

Adaptability: With `zc.buildout` and Python itself, our project could be easily deployed on different platforms almost without modification.

Understandability: Frontend has different views. The same view has different styles. Statistical data are displayed with different charts. UI components are easy to understand.

Limitations:

Robustness, Reliability, Fault-tolerance: Our system performs well, backend has basic fault-tolerance mechanism. But we have not tested the system for a long period of time under highly concurrent access.

Scalability: We have not yet tested scalability of our system in distributed environment.

Vertical Evaluation:

Module	User	Device	Submission	Campaign	Server	DHT
completed	x	x	x		x	
basically completed				x		x
not completed						

Figure 24: Vertical Evaluation

Modules User, Device, Submission, Server have been completed and met the requirements in concept period. Modules DHT and Campaign have been basically completed. We have only demonstrated how the structure should be displayed. Data was not dynamic generated but static mockup data.

3.5 Summary and Outlook

This is my first time doing research project. Before this project i have done many other complex practicums. The biggest difference between research project and practicum lies in that i didn't know what to do, what i should do, what i can do at the beginning. Tasks of complex practicums are usually unambiguous, i just need to implement tasks assigned by my tutor. At the beginning of this project i had to analyze the existing project, i had to have meetings with my team members to make sure what were feasible, which module could be

completed directly, which module could be completed with the help of some schemas or extensions defined by ourselves, which module was not feasible due to currently ambiguous specifications or other conditions that could not be met as soon as possible. For these currently infeasible tasks, we still had to make compromises to ensure the basic features.

Since we analyzed the requirements, modeled the system, specified protocols detailedly at the beginning, we didn't spend too much time on development and modification of protocols or requirements. We have cooperated very well with each other.

The system has basically met our requirements, the future work, e.g. dynamic DHT data, real mobile device specification, many-to-many users/devices(maybe with modification of DB schema), better display and more features for crowdsourcing servers could be integrated in to existing system without more efforts.