<table>
<thead>
<tr>
<th>1</th>
<th>Content</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Technology Stack</td>
<td>4</td>
</tr>
<tr>
<td>1.3</td>
<td>Design and Architecture</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>API Documentation</td>
<td>7</td>
</tr>
<tr>
<td>1.5</td>
<td>Security</td>
<td>11</td>
</tr>
<tr>
<td>1.6</td>
<td>Installation</td>
<td>11</td>
</tr>
<tr>
<td>1.7</td>
<td>Discussion and Evaluation</td>
<td>14</td>
</tr>
<tr>
<td>1.8</td>
<td>Future work</td>
<td>17</td>
</tr>
<tr>
<td>1.9</td>
<td>References</td>
<td>17</td>
</tr>
</tbody>
</table>
This is a documentation for AuroraAlarm project. This documentation is not organized in a classical way, but as a report. This project was developed as a part of Design of Dynamic Web Systems course, taught at the Luleå University of Technology, Sweden.

Instead of a blog we have been using Github’s issue tracking system. We have commented what it has been done, what we are going to do and where we have problems. You can find issues here:

https://github.com/Gupi/AuroraAlarm/issues
1.1 Introduction

1.1.1 Problem

We are exchange students here in Luleå and we have already realized that here you can see an extraordinary natural phenomenon, called aurora borealis or northern lights. When we saw it for the first time, we were impressed. But after a couple of months living here in northern Scandinavia, we have realized that it is very hard to know, when aurora is visible. You can miss it very easy. We were searching for different applications which can notify you, when aurora is active, but we were not able to find it. That is why we decided to develop a web application which can help every aurora enthusiast to never miss northern lights anymore.

1.1.2 Project description

AuroraAlarm is a simple web application for monitoring auroral activity and to inform users, when Aurora can be seen at their current location. The application consists of four main parts:

1. **Aurora Activity** - check history and forecast of auroral activity
2. **Aurora Alarms** - set aurora notifications
3. **Aurora Gallery** - upload and share the nicest images of aurora
4. **Aurora Map** - check where you can see the nicest aurora

1.1.3 Use cases

We have provided a few use cases, what a user can do with AuroraAlarm application:

1. A user can check, when Aurora was active in the past and when it will be active in the future (up to five days).
2. A user can set a threshold for auroral activity. When auroral activity exceeds this threshold, user receives an email notification.
3. A user can notify others, if he/she can see aurora activity. The user has to set up his/her current location. All other users who wants to receive this kind of alarms are notified as soon as possible.
4. A user can upload an image of aurora and set location, where the image was taken.
5. A user can search over the map, where are collected different aurora images. Here a user can see, where you can see the nicest aurora.
6. A user can sign up with Facebook or Google+ account.

1.2 Technology Stack

AuroraAlarm is built with quite new and popular, open source technologies. In this project we wanted to learn and try, how to use all these technologies and how they interact between each other. Further we would like to list all technologies we have used and write a short description for each and why we have chosen it. Usually the main reason for technology choice was a good documentation. We think this is the main advantage, when you have to select between different technologies.

1.2.1 Backend

Django (https://www.djangoproject.com/) Django is a high-level Python web framework. We have chosen it, because it is Python based, you can write fast, you don’t have to repeat yourself and it has a really good documentation.

Django REST Framework (http://www.django-rest-framework.org/) Django REST framework is a powerful and flexible toolkit that makes it easy to build Web APIs. It has a really good web browseable API which can help you, when you develop API. Documentation is also very good and community is active.

Django Photologue gallery (https://github.com/jdriscoll/django-photologue) A powerful image management and gallery application for the Django web framework. You can upload photos, group them into galleries, apply effects such as watermarks. We have chosen it, because it is highly customizable and easy to integrate.

Celery (http://www.celeryproject.org/) Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well. All our backend jobs are executed with celery: sending emails, collecting aurora forecast data, alerting system,...

Highcharts API (http://www.highcharts.com/) Highcharts is a charting library written in pure HTML5/JavaScript, offering intuitive, interactive charts to your web site or web application. We used this library at aurora history/forecast chart. We have decided for it, because it is for free and at the moment very popular library for drawing different charts.

1.2.2 Graphical User Interface

Twitter Bootstrap (http://getbootstrap.com/) Bootstrap is sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development. It is purely written in HTML5/CSS3. This framework is the most popular project at Github at the moment. Nowadays a lot of web pages looks very similar, because of Bootstrap popularity. That’s why we have decided to buy a prepared theme, which looks a bit different. Wrapbootstrap website offers really good themes for a very good price and we have found this theme: https://wrapbootstrap.com/theme/pixma-responsive-multipurpose-template-WB0B348C6. When we bought it, it was new and almost for free. It was a very good decision. We have realized how easy you can create a very nice web interface without big effort.

jQuery (http://jquery.com/) jQuery is a fast, small, and feature-rich JavaScript library. We had to use it, because jQuery is dependency for Twitter Bootstrap. We have also used it for making AJAX requests.

1.2.3 Authentication

Python Social Auth (https://github.com/omab/django-social-auth) Python Social Auth aims to be an easy to setup social authentication and authorization mechanism for Python projects supporting protocols like OAuth (1 and 2), OpenId and others. At the moment our site supports two OAuth2 providers: Facebook and Google+, but
Python Social Auth supports more than 20 different OAuth2 providers. That’s why we can easily add other providers.

Facebook and Google+ (https://www.facebook.com/ and https://plus.google.com/) Facebook and Google+ are the biggest social networks. A lot of users over the world uses their services and have their accounts. Also APIs are very good documented and very easy to use.

1.2.4 Map

Google Maps API (https://developers.google.com/maps/documentation/javascript) Google Maps is a web mapping service application and technology provided by Google. It is very simple to use, with powerful API and good documentation.

Geocomplete (http://ubilabs.github.io/geocomplete/) Geocomplete is an advanced jQuery plugin that wraps the Google Maps API’s Geocoding and Places Autocomplete services. We decided for it, basically because it works naturally with jQuery and has a good documentation.

1.2.5 Database

PostgreSQL (http://www.postgresql.org/) PostgreSQL is a powerful, open source object-relational database system. We use it in deployed version at Heroku, because this the main choice and it works good. We didn’t want to complicate with NoSQL databases, because our data scheme is very simple and SQL works good enough.

sqlite (http://www.sqlite.org/) SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is very easy to use for development purposes. It has been already installed with Django and it works very good, when you have to develop in a team.

1.2.6 Version control

Github (https://github.com/) GitHub is a web-based hosting service for software development projects that use the Git revision control system. We decided for it mainly because of git. It is really good tool for source control and also team work. We have really good experience with it and we recommended it!

1.2.7 Documentation

Read the docs (https://readthedocs.org/) Read the Docs hosts documentation, making it fully searchable and easy to find. You can import your docs using any major version control system, including Mercurial, Git, Subversion, and Bazaar. Final documentation looks really nice and integration with Github is perfect.

Sphinx engine (http://sphinx-doc.org/) Sphinx is a tool that makes it easy to create intelligent and beautiful documentation. In combination with Read the docs and Github it enables you to create documentation sites, very easy and efficient. It has a bit complicated syntax, but you know how to use it, it’s very straightforward.

1.2.8 Deployment

Heroku (https://www.heroku.com/) Heroku is a cloud platform as a service (PaaS) supporting several programming languages. We have decided for it, because we wanted to try this a very popular platform. We had some troubles, but after a couple of days we have succeeded. Maybe it would be easier to use Amazon, but at least we learned something new and see how Heroku works.


1.2. Technology Stack
1.3 Design and Architecture

Our application is a typical client-server application. We have provided a scheme below, where you can see the most important parts of the application. We have two set-ups, development version and version for Heroku. If you want to install or deploy this application, read the instructions in Installation chapter.

Further description is written for a deployed version at Heroku.

1.3.1 Server side

Django server (Gunicorn)

This server contains Django framework. Django has a model-view-controller (MVC) architecture:

- **Model:** in model we defined all objects. Above model there is RESTful API, created with Django REST Framework. All ajax calls from web browsers go through API.
- **View:** here are different HTML5/CSS3 templates, built with Twitter Bootstrap framework.
- **Controller:** logic part of the application is simplified. The main logic is actually moved to JavaScript/jQuery scripts and is loaded in browsers.

Celery and Queue

AuroraAlarm has a lot of background/asynchronous jobs. These jobs are performed with Celery, an asynchronous task queue. For example: sending emails, alerting, calculating distances, etc. At the moment we haven’t implemented all backend tasks yet, but we prepared everything for further development.

PostgreSQL or SQLite

Data is stored in PostgreSQL (deployed version at Heroku) or SQLite (development version). Django uses Object-relational mapping (ORM) and supports all well-known databases. If you want to switch between different databases, you just have to set correct settings in settings.py file, which you can find it in aurora_alarm/aurora_alarm folder.

1.3.2 Client side

Web browser

When browser makes first request, it receives HTML content and JavaScript/jQuery scripts. In these scripts is written a main logic of the application. All data from database is collected with AJAX calls through API.
1.3.3 Files structure

Our project has a recommended Django file structure. Here is a short description for each directory:

- **aurora_alarm**:
  - **api**: application for REST api needs
  - **aurora_alarm**: main urls router, context processor and celery settings
  - **frontend**: graphical user interface for a web application, static files
  - **templates**: HTML5/CSS3 templates created with Twitter Bootstrap

- **deploy**: all files which are necessary for deployment to Heroku
- **docs**: documentation files created with Sphinx

1.4 API Documentation

AuroraAlarm application has a RESTful API, built with Django REST Framework, a powerful and flexible toolkit that makes it easy to build web APIs. Here are some reasons, why we have decided to use this framework:

- The web browseable API is a huge usability win for developers.
- Extensive documentation and great community support.
- Highly customizable with regular function-based views.
- Native authentication support for OAuth2.

If you want to extend this API, we recommend to check the official documentation for Django REST Framework, which can be found here:

http://www.django-rest-framework.org/api-guide/requests

1.4.1 The web browseable API

At the moment API is in development mode and can be accessible without root permission. This a big security issue and has to be fixed. This issue has been added to the roadmap. But on the other hand, you can try and change values and you can interact with the website through API. You can find it here:

http://aurora-alarm.herokuapp.com/api/

The web interface is very simple. You can navigate between different object via links or through URLs. Through URLs you can also use filtering options (check the official documentation). This interface enables you to execute all CRUD (create, read, update, delete) operations. If you make requests through browser, you will get HTTP response, otherwise you will get a classical JSON response. You can also switch between HTTP and JSON through the interface, a button for switching is at top-right corner.
1.4.2 API resources

Aurora daily forecast

Here are saved all aurora activity levels. These values are added automatically with parsing script.

Methods

- GET /api/aurora_daily_forecast/:id
- POST /api/aurora_daily_forecast/
- PUT /api/aurora_daily_forecast/:id
- DELETE /api/aurora_daily_forecast/:id

Attributes
- id [integer] - unique identification number
- date [date] - for which day is value
- first_value [integer] - value entered first time (not updated)
- current_value [integer] - the latest entered value
- created [date] - date when was created
- modified [date] - date when was modified

Example of object:
```
{
  "date": "2014-01-01",
  "first_value": 5,
  "current_value": 5,
  "created": "2014-01-10T16:29:21Z",
  "modified": "2014-01-10T16:29:21Z",
  "url": "http://aurora-alarm.herokuapp.com/api/aurora_daily_forecast/1/",
  "id": 1
}
```

**Users**

Collection of all authenticated users. User is added through OAuth2 authentication protocol.

Methods
- GET /api/users/:id

Attributes
- id [id]
- username [string]
- email [string]
- groups [group]
- userprofile [integer]

Example of object:
```
{
  "id": 1,
  "username": "admin",
  "email": "zupec.nejc@gmail.com",
  "groups": []
}
```

**User profiles**

Here are saved settings for each user.

Methods
• GET /api/user_profiles/:id
• POST /api/user_profiles/

Attributes
• id [integer]
• user [integer]
• threshold [integer]
• receive_daily_alarms [boolean]
• receive_real_time_alarms [boolean]
• radius [integer]
• longitude [decimal number]
• latitude [decimal number]

Example of object:
{
  "user": 1,
  "threshold": 4,
  "receive_daily_alarms": true,
  "receive_real_time_alarms": true,
  "radius": 50,
  "longitude": "22.26990333632807000",
  "latitude": "65.59864468311174000"
}

Photos with location

API for photos with location. These photos are shown in gallery and map.

Methods
• GET /api/photo_with_location/:id
• POST /api/photo_with_location/
• PUT /api/photo_with_location/:id
• DELETE /api/photo_with_location/:id

Attributes
• id [integer]
• image [string]
• date_taken [date]
• view_count [integer]
• crop_from [string]
• effect [string]
• longitude [decimal number]
• latitude [decimal number]

Example of object:
1.5 Security

In this chapter we describe all security mechanisms we included in AuroraAlarm application. Our application is protected against classical threats, such as cross-site scripting (XSS), SQL injections and cross-site request forgeries (CSRF).

1.5.1 Authentication and Authorization

For authentication and authorization we use OAuth2 protocol, which implements high security standards. We didn’t implement this service on our own, but we have used implementation developed and tested by community. In our database we save minimal information about users. All important data is stored at OAuth2 providers.

1.5.2 Cross site scripting (XSS) protection

XSS attacks allow a user to inject client side scripts into the browsers of other users. Against these attacks Django has very good protection. We have used escape filters and default forms.

1.5.3 Cross-site request forgery (CSRF) protection

CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user’s knowledge or consent. Django has built-in protection against most types of CSRF attacks, providing you have enabled and used it where appropriate. To prevent CSRF attacks, we check for a nonce in each POST request.

1.5.4 SQL injection protection

SQL injection is a type of attack where a malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted or data leakage. By using Django’s querysets, the resulting SQL will be properly escaped by the underlying database driver.

1.6 Installation

1.6.1 How to setup a development environment

Here are the instructions, how to setup a development environment.

The following prerequisites are required to run AuroraAlarm:
• Python 2.7
• Virtualenv
• Pip (1.0+)

Note: we are assuming that you are running an UNIX-like operating system.

Create a new virtual environment and activate it:

```
virtualenv --distribute env
source env/bin/activate
```

Clone the repository from Github:

```
git clone git@github.com:Gupi/AuroraAlarm.git
```

Change directory to AuroraAlarm and install all the requirements:

```
cd AuroraAlarm
pip install -r requirements.txt
```

Synchronize the database and django application. The wizard will ask you to create a new superuser. Create it:

```
cd aurora_alarm
python manage.py syncdb
```

Load initial database objects:

```
python manage.py loaddata initial_data
```

Now you are ready to run application:

```
python manage.py runserver
```

Open a web browser and go to:

```
http://127.0.0.1:8000
```

That’s it, you have successfully installed AuroraAlarm application :)

1.6.2 How to deploy AuroraAlarm application to Heroku

Our application is at the moment hosted at Heroku. You can find it here: http://aurora-alarm.herokuapp.com. Sometimes it takes a couple of seconds to show up, because Heroku stops inactive websites and server needs some time to return a website.

First deploy

Let’s assume you want to deploy application for the first time. First you have to register at Heroku and setup an environment. You can follow this simple quickstart tutorial: https://devcenter.heroku.com/articles/quickstart. After step 3 follow our instructions bellow.

Note: we are assuming that you are running an UNIX-like operating system.

Create a new directory, e.g. name it aurora-alarm:

```
mkdir aurora-alarm && cd aurora-alarm
```

Create Procfile for your new app and enter command starting gunicorn server. Check if the name of the application is correct:
echo web: gunicorn aurora-alarm.wsgi > Procfile

Clone the latest code from Github somewhere to your disk:

cd ..; git clone git@github.com:Gupi/AuroraAlarm.git

Copy all files from the folder aurora-alarm (from cloned repository) to your created repository for Heroku. All files should be in root of your repository:

cp AuroraAlarm/aurora-alarm aurora-alarm

Now we have to replace some files which are important for Heroku. In our Github repository you can find deploy folder, where you will find all the important files for Heroku, which are a bit different than the files in development version of the code:

cp AuroraAlarm/deploy/requirements.txt aurora-alarm
cp AuroraAlarm/deploy/settings.py aurora-alarm/aurora-alarm
cp AuroraAlarm/deploy/wsgi.py aurora-alarm/aurora-alarm

Ok, all files and directory structure is now prepared for deployment to Heroku. Create a new git repository and commit all the files:

git init
git add *
git commit -m "Deploy all file to Heroku."

Create a new Heroku app:

heroku create

Deploy everything to Heroku:

git push heroku master

Now you have to setup a database through CLI. Run bash:

heroku run bash

When command line is initialized, you have to run next commands:

python manage.py syncdb
python manage.py migrate photologue
python manage.py syncdb

After first syncdb you will receive an error, because photologue tables are not included. That’s why we have to run migrate command and after a command syncdb one more time.

Well, you have successfully deployed your application to Heroku. Now you can run the deployed application and check if everything works:

heroku open

This command will open your browser and you should be able to see running application. If you have any problems, you can open heroku logs. You will see debug messages:

heroku logs
Update an existing Heroku app

Let’s assume you have already created Heroku app and you wanted to update the latest code from the Github repository. Our application at Heroku is named as aurora-alarm. All further instructions/commands will use that name.

First, clone the repository at Heroku:

```
git clone git@heroku.com:aurora-alarm.git
```

Clone the repository at Github:

```
git clone git@github.com:Gupi/AuroraAlarm.git
```

After these two steps, you will have two directories: aurora-alarm (clone from Heroku) and AuroraAlarm (clone from Github). Now you have to copy the latest code from AuroraAlarm/aurora-alarm to aurora-alarm. Please, check instructions in section above, where you will find instructions, how to prepare directory structure. When you have finished with directory structure, you have to commit everything and upload to the Heroku repository:

```
git add *
git commit -m "Code update to the latest."
git push heroku master
```

Ok, code is now updated. Now we have to update the database. If code for database is changed, than I recommend that you delete the whole database and set it up one more time:

```
heroku pg:reset postgres
```

Now you just have to follow up instructions in the section above, how to setup a database. That’s it, you have successfully updated the code.

1.7 Discussion and Evaluation

In this chapter we would like to present and describe what have we done so far in this project. For starting point we will use the goals/instructions for this project, provided by professor: https://sites.google.com/a/ltu.se/m7011e-2013/assignment. At the end we will check what works and what doesn’t.

1.7.1 Components

Backend (server-part) that stores data persistently and efficiently. Backend is made with Django which uses ORM. It is persistent and efficient. At the moment we use SQL-type database, because we believe that meets the requirements of traffic for this kind of projects.

A clearly defined and documented API for communication with the backend. For API we have used Django REST Framework. You can find it here: http://aurora-alarm.herokuapp.com/api/, documentation is accessible in API documentation chapter.

Easy to use frontend that allows for upload of data via the web and allows the user to see all data stored in an efficient and “nice” way. Our frontend tends to be simple as possible. All forms are AJAX supported and for feedback (when action is committed) a user receives a message (in green, yellow or red color). There is no save buttons. For location based inputs we have provided Google Maps integration and Google address location search. Images with provided locations are shown at map.

All communication with the backend should be via the API from bullet 2. All frontend requests are made with AJAX and jQuery library.

A simplified but well documented example application that communicates with the backend to allow for new users to create new applications. We provided this kind of information in API documentation chapter.
Security on the “right level”. It is recommended that a modern security scheme is used where users can login using alternative services like e.g. Google+ and Facebook accounts. User can login via OAuth2 authentication system. At the moment we support Google+ and Facebook accounts, but we can extend system quite easy with other popular OAuth2 providers.

You should understand how the security part works and be able to motivate why your solution is secure. Authentication security is ensured with OAuth2 protocol. We just save minimal data about users (what OAuth2 protocol require). Django framework supports a lot of security mechanisms. We have implemented further mechanisms: cross-site-scripting (XSS) protection, cross site request forgery (CSRF) and SQL injection protection.

The system should utilize modern HTML5-technologies of your choice. Frontend is completely written with HTML5 code. We have used Twitter Bootstrap framework, which is natively written with HTML5 and CSS3 technologies. Also highcharts library is written in pure HTML5/JavaScript.

1.7.2 Deployment

The system should be deployed in the cloud and be publically available. AuroraAlarm is deployed at Heroku. The application is accessible at: http://aurora-alarm.herokuapp.com/. All instructions, how to deploy the code, you can find it in installation chapter.

1.7.3 Bonus Parts

User administration - add, remove, edit users. User management can be done through RESTful API or Django’s control panel.

Data administration - interfacing databases, editing data via separate graphical interface. CRUD operations for all objects can be done through Django’s control panel. You can find it here: http://aurora-alarm.herokuapp.com/admin/
Connections to social media, e.g. FaceBook and Google+. Users can authenticate with Facebook or Google+ accounts.

Integrate “cool” other APIs We haven’t integrated any cool other APIs. We were searching aurora forecast API’s, but we were not successful. Instead we have developed parsing function which collects data. This data is now accessible via API.

Use your imagination - do that little extra... We bought Twitter Bootstrap theme which helped us to make a nicer graphical interface. We were exploring with different new technologies (asynchronous processing with Celery, jQuery addons for location filtering, AJAX requests, photologue gallery,...)

1.7.4 What works and what doesn’t

We have already successfully developed an user interface and started to develop backend and asynchronous tasks. All main functionalities are developed, except sending emails and API security. Future ideas are described
1.8 Future work

In this chapter we would like to present our ideas, how this project can be extended and upgraded. Some ideas are quite easy to realize and some of them can take a lot of time to implement, e.g. aurora recognition with an USB camera or integrating weather forecast. All ideas and tasks are collected in our issue system at Github repository.

1.9 References

All technologies used in this project are referenced in chapter technology stack. Other important references:

- Github repository: https://github.com/Gupi/AuroraAlarm
- Issues and roadmap: https://github.com/Gupi/AuroraAlarm/issues
- Documentation at Read the docs: https://auroraalarm.readthedocs.org/
- Course site: https://sites.google.com/a/ltu.se/m7011e-2013/
- Deployed version at Heroku: http://aurora-alarm.herokuapp.com/

1.9.1 Contact

- Nejc Zupec, zupec.nejc@gmail.com
- Prisca Bonnet