Atomaddict

Grzegorz Janik
Piotr Lechowicz
Mikalai Barysau

Project coordinator:
dr Marek Woda

Department of Electronics
Wroclaw University of Technology
Wroclaw, Poland

June 4, 2015

1 Introduction

Atomaddict is a project, designed to simplify access to information. In the environment full of different sorts of information the fastest way to collect only information you are interested in is to have tool, which will know your needs and will automatically gather actual information for you, so you are always up-to-date in everything you are interested in.

The main goal of the **Atomaddict** project was to create a narrow-purpose tool with a low entry barrier, nice interface and modular architecture, available for any user regardless of operating system or device type.

2 Project goals

The aim of the project is to create a web application, which aggregates syndicated web content, such as online newspapers, blogs or podcasts. User

will receive information and news based on his own preferences, that is, he will be able to choose topics or particular websites and subscribe to the groups of RSS and Atom feeds. The second goal was creating a tool, which will be easy to use regardless of target platform or users skill in IT applications. **Atomaddict** has simple intuitive interface and customizable functionality, which is supposed to satisfy any user expectations. The application is useful either for new and for hard users. We have achieved it by creating two kinds of views - simple title look-like view and more advanced table view. The principles of the User Interface are nice flexible design, which will look fine on any sort of display, and extremely narrowed functionality, which will allow user to be focused on main application features and not be bothered by useless buttons and clumsy bars.

3 User requirements

- user should be able to sign in, sign out and sign up
- user should be able to view feeds as a table of articles;
- feeds should be shown depending on the category, that user have selected
- each element of table should have title, publication date and link to the original page
- after clicking on the element of table user should be redirected to the original page
- user should be able to refresh feeds
- user should be able to mark page as read
- in '*Options*' section user should be able to choose language of the page between English and Polish
- in '*Options*' section user should be able to change font size (possible sizes: Small, Regular, Big, Mixed)
- All interface elements should adapt (rearrange) to the display size

4 Project management

To make work on the **Atomaddict** project efficient and multitask we took an advice from the project coordinator and used several additional tools as GitHub (`https://github.com`) and Trello (`https://trello.com`). Also Gantt diagram was used to plan the workflow.

4.1 GitHub

GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as wikis, task management, and bug tracking and feature requests for every project.

GitHub offers both plans for private repositories and free accounts, which are usually used to host open-source software projects. As of 2015, GitHub reports having over 9 million users and over 21.1 million repositories, making it the largest code hoster in the world.

GitHub is mostly used for code. In addition to source code, GitHub supports the following formats and features:

- Documentation, including automatically-rendered README files in a variety of Markdown-like file formats (see README files on GitHub)
- Issue tracking (including feature requests)
- Wikis
- Small websites can be hosted from public repositories on GitHub. The URL format is `http://username.github.io`
- Nested task-lists within files
- Visualization of geospatial data
- Gantt charts

- 3D render files which can be previewed using a new integrated STL file viewer which displays the files on a 3D canvas. The viewer is powered by WebGL and Three.js
- Photoshop's native PSD format can be previewed and compared to previous versions of the same file. [**?**]

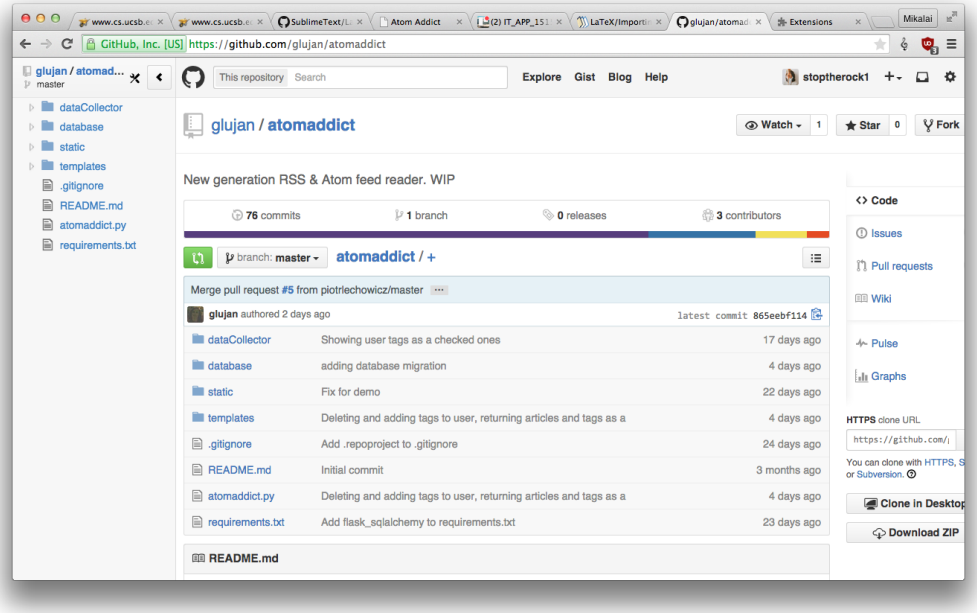

Figure 1: **Atomaddict** project repository on **GitHub**

4.2 Trello

Trello is a free web-based project management application originally made by Fog Creek Software in 2011, that spun out to be its own company in 2014. It operates a freemium business model, as well as being cross-subsidized by other Fog Creek Software products. A basic service is provided free of charge, though a Business Class paid-for service was launched in 2013.

Trello uses the kanban paradigm for managing projects, originally popularized by Toyota in the 1980s for supply chain management. Projects are

represented by boards, which contain lists (corresponding to task lists). Lists contain cards (corresponding to tasks). Cards are supposed to progress from one list to the next (via drag-and-drop), for instance mirroring the flow of a feature from idea to implementation. Users can be assigned to cards. Users and boards can be grouped into organizations.

Trello has a variety of work and personal uses including real estate management, software project management, school bulletin boards, lesson planning, and law office case management. A rich API as well as email-in capability enables integration with enterprise systems, or with cloud-based integration services like IFTTT and Zapier. [**?**]

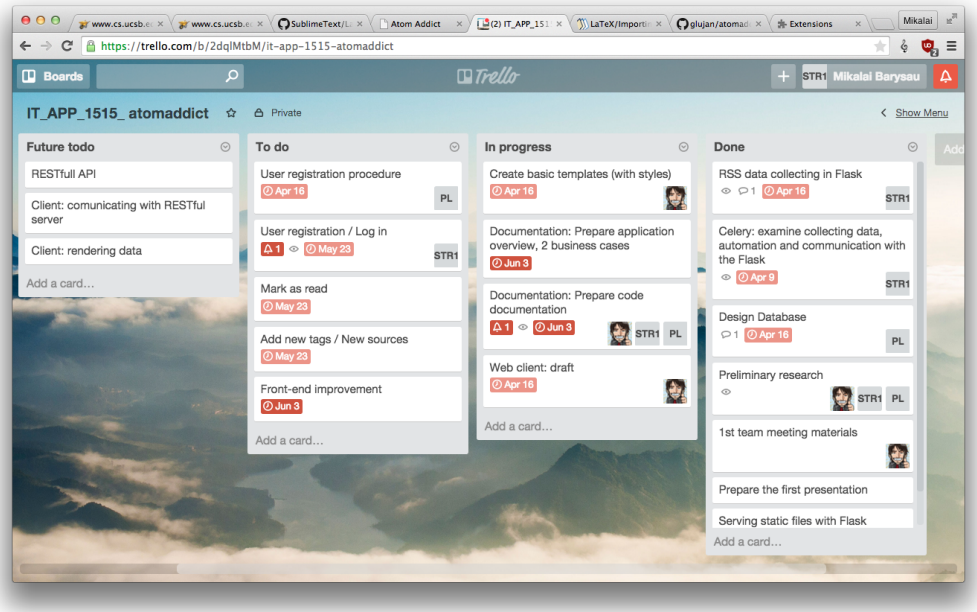

Figure 2: **Atomaddict** project board on **Trello**

4.3 Gantt diagram

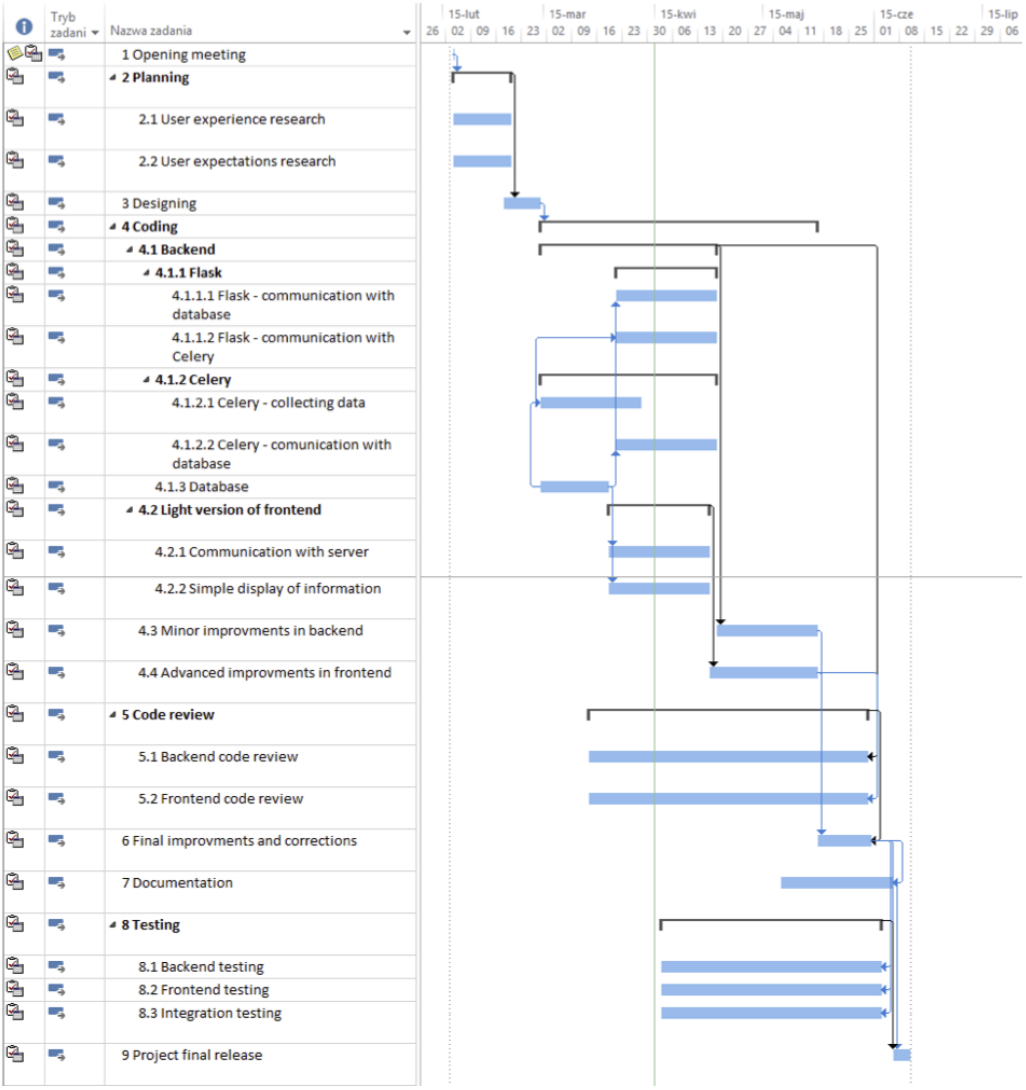

Figure 3: **Atomaddict** project Gantt diagram

4.4 Work breakdown structure

Tryb zadani	Nazwa zadania	SPP	Czas trwania	Rozpoczęci	Zakończen	Poprzedniki
	1 Opening meeting	WBS01	1 godz.	wto, 15-02-0	wto, 15-02-0	
	2 Planning	**WBS02**	**12 dn**	**wto, 15-02-03**	**czw, 15-02-19**	**1**
	2.1 User experience research	WBS02.a	12 dn	wto, 15-02-03	czw, 15-02-19	
	2.2 User expectations research	WBS02.b	12 dn	wto, 15-02-03	czw, 15-02-19	
	3 Designing	WBS03	8 dn	wto, 15-02-1	pią, 15-02-27	2ZR-2 dn
	4 Coding	**WBS04**	**55 dn**	**pią, 15-02-27**	**pią, 15-05-15**	**5**
	4.1 Backend	**WBS04.a**	**35 dn**	**pią, 15-02-27**	**pią, 15-04-17**	
	4.1.1 Flask	**WBS04.a.1**	**20 dn**	**pią, 15-03-20**	**pią, 15-04-17**	
	4.1.1.1 Flask - communication with database	WBS04.a.1.1	20 dn	pią, 15-03-20	pią, 15-04-17	14ZR-5 dn
	4.1.1.2 Flask - communication with Celery	WBS04.a.1.2	20 dn	pią, 15-03-20	pią, 15-04-17	12RR
	4.1.2 Celery	**WBS04.a.2**	**35 dn**	**pią, 15-02-27**	**pią, 15-04-17**	
	4.1.2.1 Celery - collecting data	WBS04.a.2.1	20 dn	pią, 15-02-27	pią, 15-03-27	14RR
	4.1.2.2 Celery - comunication with database	WBS04.a.2.2	20 dn	pią, 15-03-20	pią, 15-04-17	14ZR-4 dn
	4.1.3 Database	WBS04.a.3	13 dn	pią, 15-02-27	śro, 15-03-18	
	4.2 Light version of frontend	**WBS04.b**	**20 dn**	**śro, 15-03-18**	**śro, 15-04-15**	
	4.2.1 Communication with server	WBS04.b.1	20 dn	śro, 15-03-18	śro, 15-04-15	14
	4.2.2 Simple display of information	WBS04.b.2	20 dn	śro, 15-03-18	śro, 15-04-15	14
	4.3 Minor improvments in backend	WBS04.c	20 dn	pią, 15-04-17	pią, 15-05-15	7
	4.4 Advanced improvments in frontend	WBS04.d	22 dn	śro, 15-04-15	pią, 15-05-15	15
	5 Code review	**WBS05**	**56 dn**	**pią, 15-03-13**	**pią, 15-05-29**	
	5.1 Backend code review	WBS05.a	56 dn	pią, 15-03-13	pią, 15-05-29	7ZZ+2 dn
	5.2 Frontend code review	WBS05.b	56 dn	pią, 15-03-13	pią, 15-05-29	19ZZ+2 dn
	6 Final improvments and corrections	WBS06	15 dn	pią, 15-05-15	sob, 15-05-30	18;19;20ZZ
	7 Documentation	WBS07	30 dn	wto, 15-05-05	pią, 15-06-05	23ZZ+5 dn
	8 Testing	**WBS08**	**44 dn**	**czw, 15-04-02**	**wto, 15-06-02**	
	8.1 Backend testing	WBS08.a	44 dn	czw, 15-04-0	wto, 15-06-0	23ZZ+2 dn
	8.2 Frontend testing	WBS08.b	44 dn	czw, 15-04-0	wto, 15-06-0	23ZZ+2 dn
	8.3 Integration testing	WBS08.c	44 dn	czw, 15-04-02	wto, 15-06-02	23ZZ+2 dn
	9 Project final release	WBS09	3 dn	pią, 15-06-05	śro, 15-06-10	23;25;24

Figure 4: **Atomaddict** project work breakdown structure

5 System diagram

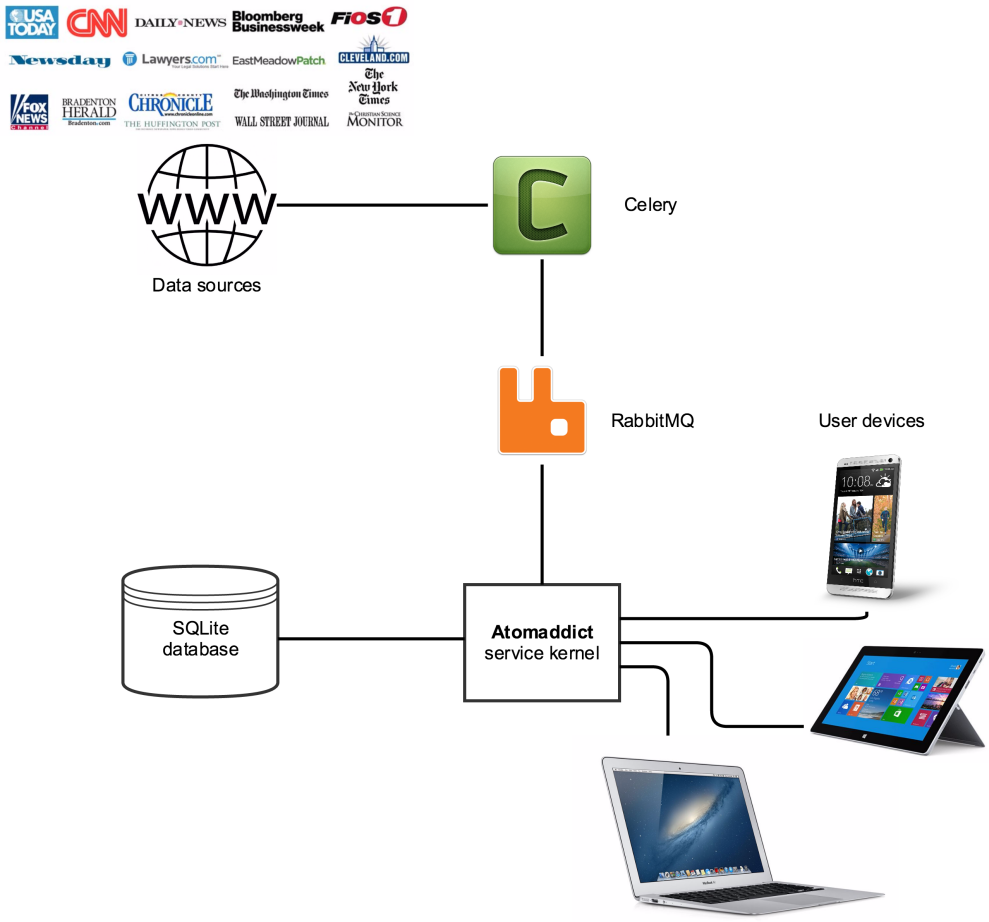

Figure 5: **Atomaddict** system

6 Use cases

6.1 Usecase: Validate user

Brief Description

This use case describes how the user signs in to the **Atomaddict** system.

Actors

- Customer
- Atomaddict system

Preconditions

There is an active internet connection between the Customer and the Atomaddict system. There is no user signed in to the session with Atomaddict system.

Basic Flow of Events

1. User opens the projects webpage in a web browser as a not singed in user
2. User clicks on '*Sign in*' button
3. The Atomaddict website asks User to provide user's credentials: *email* and *password*
4. User provides user's credentials and sends data to the Atomaddict system
5. The Atomaddict system checks user's credentials and replies with successful reply telling if User is signed in
6. User gets redirected to the main page of the Atomaddict website as a signed in user

Alternative Flows

- **Wrong email or password**: the Atomaddict informs User about wrong credentials and asks User to provide user's credentials: *email* and *password* again (The use case resumes at step 4)

- **No response from the Atomaddict system**: The use case ends with a failure condition

7 Technologies used in project

Implementation of **Atomaddict** project was done in Python, JavaScript, HTML and CSS. Database was created with SQLite engine.

7.1 Celery

7.2 RabbitMQ

7.3 Database

To store necessary information for **Attomaddict** was used SQLite database. The figure 6 provides a visual overview of the AttomAddict database and the realtions beetwen the tables. Table 1 and 2 include additional details on the tables and columns.

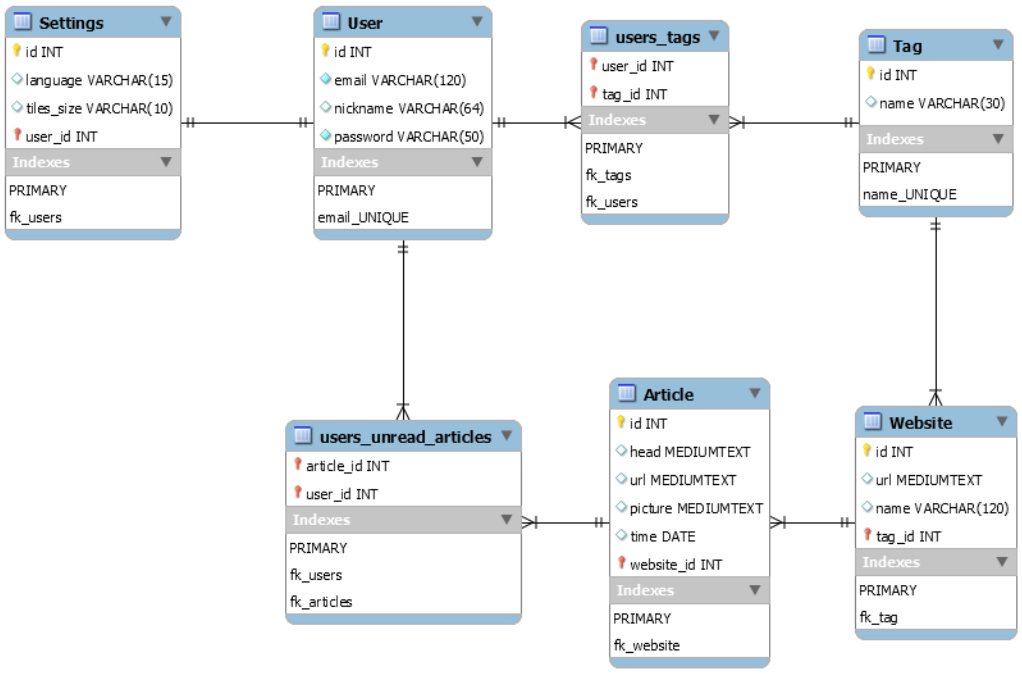

Figure 6: Database model

7.3.1 Table Overview

AttomAddict Tables Overwiew	
User	Basic users data
Settings	User configuration settings
Tag	Tags representing available topics of interest for subscription
Website	Websites providing RSS data
Article	Articles parsed from websites

Table 1: Tables overwiew

7.3.2 Table Details

AttomAddict Table Details		
User	email	email used to log into application
	nickname	optional user nickname
	password	password used to log into application
	tags	tags assigned to particular user
	articles	unread articles by user
Settings	language	user's preferable language
	tiles size	size of user's tiles
Tag	name	unique tag name
	users	users assigned to particular tag
Website	url	unique url definig werbsite
	name	short description of website
Article	haed	Content of an article
	url	address of an article
	picture	url of picture if it is available
	time	time of publication
	users	users which didn't read this article from the website from the tag which they are subscribing

Table 2: Table Details

7.3.3 Implementation

Database was made in python using SQLAlchemy. SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

7.4 Front-end

8 Code documentation

8.1 Database package

Database package is used for management database. It consists of different modules.

- `model.models.py`
  - **description:** The data that is stored in the database is represented by a collection of classes. The ORM layer is doing translations required to map objects created from these classes into rows in the proper database table. The `__init__.py` file in package `model` connects database into flask application.
  - **classes:** User, Settings, Tag, Website, Article.
- `db_repository` package
  - **description:** Following versions of database model are stored in that file providing easy access to them.
- `db_create.py`
  - **description:** Module used to create database. Type and location of the database are stored in `config.py` in that package.
- `db_migrate.py`
  - **description:** Any changes to the database model can be incorporate into the database by executing this module.
- `db_upgrade.py`
  - **description:** This script allows the database to be upgraded to the latest revision, by applying the migration script stored in the database repository.
- `db_downgrade.py`

  - **description:** This script allows the database to be downgraded one revision.

- `session.py`
  - **description:** Changes to a database are done in the context of a session. It allows easy access to database resources, it is adding, deleting, or changing records.
  - **classes:**
    * `Put` - adding resources to the database
    * `Get` - getting resources from the database
    * `Delete` - deleting resources from the database
    * `Add` - connecting relationships between tables - e.g. adding tag to user
  - **methods:** They provide methods for changing user's settings or list of tags. They allow to get tags, articles or settings as a dictionary for easy use in html files.
    Example methods:
    * `def get_user_settings_as_dictionary(user_email)`
      Method is used to get user settings in a dictionary form, it is {'key' : 'value', ... }.
      input parameters: `user_email` – user's email, whose settings will be returned
      return value: settings as a dictionary
    * `set_user_tags(email, tags)`
      Method add checked and remove unchecked tags in the website to the user.
      input parameters: `email` – user's email to whom tags will be added or removed.
      `tags` – list of checked tags

-
  - **description:**
  - **classes:**
  - **methods:**

-
  - **description:**
  - **classes:**
  - **methods:**