MapSwipe Back-End

MapSwipe

Sep 26, 2023

GETTING STARTED:

CHAPTER

ONE

MAPSWIPE BACK-END

MapSwipe is a mobile app that lets you search satellite imagery to help put the world's most vulnerable people on the map. If you are new to MapSwipe it might be good to have a look at the FAQs first.

The MapSwipe Back-End consists of a number of components:

1. Firebase Project
2. MapSwipe Workers
3. Postgres Database
4. Manager Dashboard
5. API

Please refer to the documentation for more information: https://mapswipe-workers.readthedocs.io/

1.1 Resources

- MapSwipe Back-End: https://github.com/mapswipe/python-mapswipe-workers
- MapSwipe App https://github.com/mapswipe/mapswipe
- MapSwipe Website: https://mapswipe.org
- MapSwipe OSM-Wiki: https://wiki.openstreetmap.org/wiki/MapSwipe

1.2 Contributing Guidelines

1.2.1 Feature Branch

To contribute to the MapSwipe back-end please create dedicated feature branches based on the `dev` branch. After the changes create a Pull Request of the `feature` branch into the `dev` branch on GitHub:

```
git checkout dev
git checkout -b featureA
# Hack away ...
git commit -am 'Describe changes.'
git push -u origin featureA
# Create a Pull Request from feature branch into the dev branch on GitHub.
```

Note: If a bug in production (master branch) needs fixing before a new versions of MapSwipe Workers gets released (merging dev into master branch), a hotfix branch should be created. In the hotfix branch the bug should be fixed and then merged with the master branch (and also dev).

1.2.2 Style Guide

This project uses black and flake8 to achieve a unified style.

Use pre-commit to run `black` and `flake8` prior to any git commit. `pre-commit`, `black` and `flake8` should already be installed in your virtual environment since they are listed in `requirements.txt`. To setup pre-commit simply run:

```
pre-commit install
```

From now on `black` and `flake8` should run automatically whenever `git commit` is executed.

1.3 License

This project is licensed under the Apache License 2.0 - see the LICENSE file for details

1.4 Authors

- **Benjamin Herfort** - HeiGIT - Hagellach37
- **Marcel Reinmuth** - HeiGIT - maze2point0
- **Matthias Schaub** - HeiGIT - Matthias-Schaub

See also the list of contributors who participated in this project.

1.5 Acknowledgements

- Humanitarian organizations can’t help people if they can’t find them.

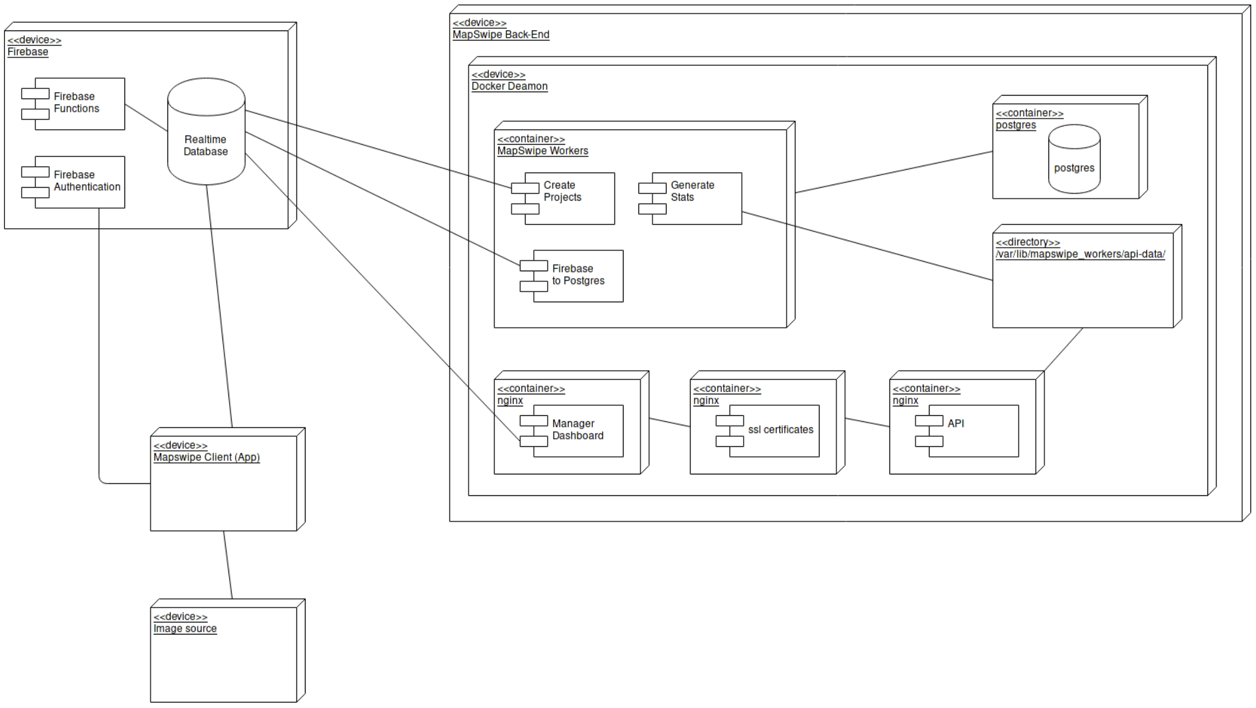
CHAPTER TWO

OVERVIEW

2.1 A typical MapSwipe workflow

1. Project managers upload information about their projects (e.g. area of interest, objects to look for) to firebase realtime database using the **manager dashboard**.
2. The **mapswipe workers** initialize newly uploaded projects and create the project related data (e.g. groups and tasks) in firebase realtime database and postgres database.
3. Project managers "activate" their projects in the **manager dashboard**.
4. The users of the MapSwipe app contribute to the newly generated projects and submit their results to firebase realtime database. The **firebase rules** ensure, that app users can only change pre-defined parts of the firebase realtime database.
5. Once new results are submitted, the **firebase functions** generate real-time statistics and update the progress of groups, compute project level statistics and user statistics in the firebase realtime database.
6. All results are transferred to the **postgres database** by the **mapswipe workers** on defined basis (e.g. every 10 minutes). The postgres database holds all MapSwipe results for long term storage. Once results are transferred to the postgres database, they will be deleted in firebase realtime database by the mapswipe workers.
7. Based on the data in the postgres database the **mapswipe workers** generate aggregated data and statistics (e.g. as csv files). This data is served by the **api**, which uses a simple nginx web server.

2.2 Deployment Diagram



2.2.1 Relations

Mapswipe Client (App) - Realtime Database

- Mapswipe Client is requesting some `projects`, data of a specific `users.userId`. In case of a project selection a group (`groups.projectId.groupId`) and associated tasks (`tasks.projectId.groupsId`) will be requested
- Mapswipe Client will only write to Firebase Realtime Database in case of result generation.
- Mapswipe Client is writing to `results.projectId.groupId.userId1.` in form of `timestamp` and `resultCount` attributes when and how many results were generated.
- The result itself will be written to `results.projectId.groupId.userId1.taskId1.result`.

Manager Dashboard - Realtime Database

- Using the Manager Dashboard user can submitt new project drafts to Firebase (`project_drafts.projectDraftId.`)

Community Dashboard - Aggregated Cached data from Database

- React based static server which uses Django webserver to show overall mapswipe aggregated contribution data.

Mapswipe Workers - Realtime Database

- projectCreation:
  - requests `projectDrafts` from Realtime Database
  - writes to `projects.projectId`, `groups.projectId` and `tasks.projectId`
- tansfer_results - Realtime Database
  - requests `results` from Realtime Database
  - deletes `results` from Realtime Database

MapSwipe Workers - Postgres Database

- projectCreation - Postgres
  - writes projectDraft, project, groups and tasks to Postgres
- tansfer_results - Postgres
  - writes results to Postgres

Django - Stats webserver

- aggregateStatData:
  - requires user contribution related to user_group and project data from Postgres Database

CHAPTER

THREE

DATA

Swiping is just the beginning – MapSwipe data is created by our users and accessible to the entire community. Through the MapSwipe website you can see where we've mapped, which organizations are requesting data, and how many individuals contribute to our impact. When using MapSwipe data, all you have to do is credit the MapSwipe contributors. Here you find a more detailed description of the data available.

3.1 Projects

3.1.1 Files:

- projects.csv
- projects_geom.geojson
- projects_centroid.geojson

3.2 Aggregated Results

This gives you the unfiltered MapSwipe results. This is most suited if you want to apply some custom data processing with the MapSwipe data, e.g. select only specific tasks for machine learning. If you want to use MapSwipe data in the Tasking Manager you might look for the data described below.

3.2.1 Files:

- `aggregated_results_{project_id}.csv`, e.g. agg_results_-M56eeMCZ5VeOHjJN4Bx.csv
- `aggregated_results_{project_id}.geojson`, e.g. agg_results_-M56eeMCZ5VeOHjJN4Bx.geojson

Additionally, project type specific data can be found here. E.g. footprint projects which were created based on OSM data, will have data describing the original OSM object included.

3.3 HOT Tasking Manager Geometries

This gives you filtered MapSwipe data ready to be imported to the HOT Tasking Manager. Currently, the geometries in this dataset consist of maximum 15 MapSwipe Tasks, where at least 35% of all users indicated the presence of a building by classifying as "yes" or "maybe".

3.3.1 Files:

- `hot_tm_{project_id}.geojson`, e.g. hot_tm_-M56eeMCZ5VeOHjJN4Bx.geojson

3.4 Users

This gives you data on the users which contributed to a project.

CHAPTER

FOUR

DEVELOPMENT SETUP

In this document some tips and workflows for development are loosely collected. Those are independent of the production setup using Docker-Compose. A working Firebase Project (Including Firebase Functions and Database Rules) is presupposed. Get in touch with the MapSwipe team (e.g. in Slack) to get access to an existing Firebase Instance for development purposes.

Check list:

1. Github: Clone repo from GitHub.
2. Requirements: Install GDAL/OGR and GDAL for Python on your machine.
3. Configuration: Set environment variables and get a Service Account Key File.
4. Database: Set up local Postgres database using Docker.
5. Python-Package: Install MapSwipe Workers Python package.
6. Run MapSwipe Workers.

4.1 Installation

4.1.1 Clone from GitHub

. . . and switch to development branch.

```
git clone https://github.com/mapswipe/python-mapswipe-workers.git
cd python-mapswipe-workers
git checkout dev
```

4.1.2 Requirements

MapSwipe Workers requires GDAL/OGR (`gdal-bin`) and GDAL for Python (`libgdal-dev`, `python-gdal`) to be installed. Furthermore, we rely on Docker to set up Postgres.

4.1.3 Configuration

All configurations values are stored in environment variables. Please refer to the documentation on Configuration for further details.

Service Account Key

The MapSwipe Workers requires a Service Account Key (`serviceAccountKey.json`) to access Firebase database. Request yours from the MapSwipe working group.

The path to the Service Account Key is defined in the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.

You could also set up your own Firebase instance. However, this is not recommended. If you still want to do it, get your Service Account Key from Firebase from Google Cloud Service Accounts.

Directories

MapSwipe Workers needs access to a data directory for logs and data for the API:

To create this directories run:

```
mkdir --parents ~/.local/share/mapswipe_workers
```

Note: XDG Base Directory Specification is respected

4.1.4 Database

Setup a local Postgres instance for MapSwipe Workers using the Dockerfile provided for development purposes (`postgres/Dockerfile-dev`). The Dockerfile for production (`postgres/Dockerfile`) does need additional setup for build-in backup to Google Cloud Storage, which is not needed for local development. That is why a simplified Dockerfile for development is provided. Make sure that the specified port is not in use already. If so, adjust the port (also in the `.env` file).

```
docker build -t mapswipe_postgres -f ./postgres/Dockerfile-dev ./postgres
docker run -d -p 5432:5432 --name mapswipe_postgres -e POSTGRES_DB="$POSTGRES_DB" -e␣
↪POSTGRES_USER="$POSTGRES_USER" -e POSTGRES_PASSWORD="$POSTGRES_PASSWORD" mapswipe_
↪postgres
```

Or set up Postgres using the `initdb.sql` file in the `postgres/` folder.

4.1.5 Mapswipe-Workers Python Package

1. Export environment variables to current shell.
2. Create a Python virtual environment with `system-site-packages` option enabled to get access to GDAL/OGR Python packages
3. Activate the virtual environment.
4. Install MapSwipe Workers using pip.
5. Run it.

```
export $(cat .env | xargs)
cd mapswipe_workers
python -m venv --system-site-packages venv
source venv/bin/activate
pip install --editable mapswipe_workers/
mapswipe_workers --help
```

Yeah! If you reached this point, you are ready to get into coding. Below you find some more information on Logging, Firebase Functions and Database Backup. However, you don't need this to get started for now.

CHAPTER

FIVE

FURTHER INFORMATION

5.1 Logging

Mapswipe workers logs are generated using the Python logging module of the standard library (See Official docs or this Tutorial. To use the logger object import it from the `definitions` module:

```
from mapswipe_workers.definitions import logger
logger.info('information messages')
logger.waring('warning messages')

# Include stack trace in the log
try:
    do_something()
except Exception:
    logger.exception('Additional information.')
```

Default logging level is Info. To change the logging level edit the logging configuration which is found in the module `definitions.py`. Logs are written to STDOUT and `~/.local/share/mapswipe_workers/mapswipe_workers.log`.

Per default logging of third-party packages is disabled. To change this edit the definition module (`mapswipe_workers/defintions.py`). Set the `disable_existing_loggers` parameter of the `logging.config.fileConfig()` function to `False`.

5.2 Firebase Functions

Firebase functions are used to increment/decrement or calculate various attribute values which are used by the Map-Swipe App. This includes:

- project.progress
- project.numberOfTasks
- project.resultCount
- project.contributorCount
- group.progress
- group.finishedCount
- group.requiredCount
- user.projectContributionCount

- user.groupContribtionCount
- user.taskContributionCount
- user.timeSpentMapping
- user.contibutions{.projectId.groupId.{timestamp, startTime, endTime}}

Those functions will be directly or indirectly triggered by incoming results from the MapSwipe App.

By using Firebase functions those attributes can be calculated in real-time and be accessed by users immediately. The use of those functions also reduces the data-transfer between the Firebase Realtime Database and MapSwipe Workers.

On how to setup the development environment and how to deploy functions to the Firebase instance please refer to the official Guide on Cloud Function for Firebase. For more information refer to the official Reference on Cloud Function for Firebase. For example function take a look at this GitHub repository.

5.2.1 OSM OAuth 2

Firebase functions are also used to allow users to login to MapSwipe with their OpenStreetMap account. Refer to the notes in the app repository for more information.

5.3 Database Backup

5.3.1 Firebase

Manual Backup

- curl https://.firebaseio.com/.json?format=export
- ref: https://stackoverflow.com/questions/27910784/is-it-possible-to-backup-firebase-db

5.3.2 Postgres

Manual Backup

- Backup database in compressed splited files of specified size:
  - `pg_dump -U mapswipe -d mapswipe -h localhost -p 5432 | gzip | split -b 100m - mapswipe.pgsql.gz`
  - ref: https://www.postgresql.org/docs/9.1/backup-dump.html
- Copy the backup to your local machine when logged into your local machine:
  - `scp username@ipadress:mapswipe.pgsql.gz* /path/to/destination`
  - ref: https://unix.stackexchange.com/questions/106480/how-to-copy-files-from-one-machine-to-another-using-ssh
- Restore database backup from multiple compressed files
  - `cat mapswipe.pgsql.gz* | gunzip | psql -U mapswipe -d mapswipe -h localhost -p 5432`

CHAPTER SIX

TESTING

6.1 Tests

- run tests locally during development

```
python -m unittest discover --verbose --start-directory mapswipe_workers/tests/
↪unittests/
python -m unittest discover --verbose --start-directory mapswipe_workers/tests/
↪integration/
```

CHAPTER SEVEN

PROJECT TYPES AND DATA MODEL

7.1 MapSwipe's Crowdsourcing Approach

The MapSwipe crowdsourcing workflow is designed following an approach already presented by Albuquerque et al. (2016). The main ideas about MapSwipe's crowdsourcing approach (and many other crowdsourcing tasks) lies in

1. **Defining** the mapping challenge by posing a simple question (e.g. "Which areas are inhabited in South Kivu?")
2. **Dividing** the overall challenge into many smaller manageable components (e.g. *groups* and *tasks* based on satellite imagery tiles)
3. **Distributing** *groups* and *tasks* to many users redundantly (e.g. every area gets mapped by at least three different users)
4. **Aggregating** all responses (*results*) per *task* from different users to reach a final solution (e.g. by choosing the majority vote)

The MapSwipe back end currently supports 3 **project types**. Each project type formulates a specific kind of mapping challenge.

7.2 Data Model

This way of formulating the overall crowdsourcing challenge and it's subcomponents shapes the **data model** we use. The data model is depicted in *Figure 1* and consists of the following parts:

- project drafts
- projects
- groups
- tasks
- results
- users
- announcement (can be set manually in firebase -> banner at the top of the app)

|

Below you can see the structure on the example of a Build Area project. The project manager supplies a bounding polygon, which is then divided into multiple groups, which are in turn divided into multiple tasks. Results are always bound to a task and what a result means differs by project type.

As a project manager you have to care about the **Project Drafts** only. The information you provide through the **Manager Dashboard** will be used to set up your project. You should provide the following information.

7.2.1 Project Drafts

The project drafts contain all information needed to set up your project. Only MapSwipe user accounts with dedicated **project manager role** can create projects. Make sure to get the rights before submitting project drafts.

After project managers defined their mapping challenges in the very first step, they can generate **project drafts** through the manager dashboard. The project drafts contain all information about your mapping challenge that you need to initialize a project in MapSwipe. For instance, the project draft defines which area you want to get mapped and how many users should work on each task.

7.2.2 Projects

The **project** holds all information provided by the project drafts, but adds additional information which are needed for the MapSwipe app such as progress and number of users who contributed. A project consists of several groups.

7.2.3 Groups

The **groups** are an intermediary between projects and tasks. Each group belongs to a single project and consists of several tasks.

Single MapSwipe projects can contain up to several hundred thousand tasks. This can pose a challenge to fast and performant communication between clients and server if many volunteers contribute data at the same time. Therefore, groups have been introduced to reduce the amount of client requests on the backend server.

Groups consists of several tasks, that will be shown to the user in one mapping session. They are the key to distribute tasks to MapSwipe users in a way that we can ensure that everything gets mapped as often as required in an efficient manner.

7.2.4 Tasks

The **tasks** are the smallest component in our data model. Each task formulates an easy and quick to solve mapping challenge. In many cases this challenge can be put into a simple question, e.g. *Can you see a building in this satellite imagery tile*. Tasks always belong to a specific group and project.

Tasks are usually gzip compressed on firebase to save space. That is why this information is not readable by humans in firebase.

7.2.5 Results

The **results** hold the information you wanted in the very beginning. For each task you will receive several results by different users. A result is the simple answer to your initial question. For instance, it's a simple "yes" to the question "can you see a building in this satellite imagery tile".

7.2.6 Users

The **users** provide the results to your tasks. They are the key to solve your mapping challenge. For each user we generate mapping related statistics, e.g. the number of projects a user has been worked on.

CHAPTER

EIGHT

BUILD AREA

8.1 Project Drafts

To initialize a Build Area Project as a Project Manager you only need to upload a bounding polygon as well as fill in some information about your mission. Details on the basic information you need to fill in to describe you mission can be found on the main project type site.

```
{
  "createdBy": "TestCreator",
  "geometry": {
    "type": "FeatureCollection",
    "features": [{
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [34.975833892822266, -15.899098066386088],
            [35.089302062988274, -15.899098066386088],
            [35.089302062988274, -15.820002241903946],
            [34.975833892822266, -15.820002241903946],
            [34.975833892822266, -15.899098066386088]
          ]
        ]
      },
      "properties": {}
    }]
  },
  "image": "",
  "lookFor": "buildings",
  "name": "test - Malawi (1)\ntest",
  "projectDetails": "This is a test project",
  "verificationNumber": 3,
  "groupSize": 120,
  "tileServer": {
    "name": "bing",
    "credits": "© 2019 Microsoft Corporation, Earthstar Geographics SIO"
  },
  "projectType": 1
}
```

8.2 Projects

Below you can find an example for a created Build Area project in firebase.

```
{
    "contributorCount": 1,
    "created": "2021-12-23T13:47:27.346088Z",
    "createdBy": "X0zTSyvY0khDfRwc99aQfIjTEPK2",
    "groupMaxSize": 0,
    "groupSize": 25,
    "image": "https://firebasestorage.googleapis.com/v0/b/dev-mapswipe.appspot.com/o/
↪projectImages%2Fbuildarea.png?alt=media&token=07505c0e-0f80-454c-b446-9b82a73d9d3e",
    "isFeatured": false,
    "lookFor": "Buildings",
    "name": "Build Area with Bing Imagery Z18 - Kenya (1)\nMapSwipe Devs",
    "progress": 0,
    "projectDetails": "This is a \"normal\" Build Area project. The project uses Bing␣
↪Imagery at zoom level 18",
    "projectId": "-MrbXgHx8YJDt6cTIyGA",
    "projectNumber": "1",
    "projectRegion": "Kenya",
    "projectTopic": "Build Area with Bing Imagery Z18",
    "projectType": 1,
    "requestingOrganisation": "MapSwipe Devs",
    "requiredResults": 148158,
    "resultCount": 0,
    "status": "active",
    "tileServer": {
        "apiKey": "",
        "credits": "imagery credits of project",
        "name": "bing",
        "url": "https://ecn.t0.tiles.virtualearth.net/tiles/a{quad_key}.jpeg?g=1&
↪token={key}"
    },
    "tutorialId": "tutorial_-MnNaUEShyefFtMG6_5-",
    "verificationNumber": 3,
    "zoomLevel": 18
}
```

8.3 Groups

The grouping algorithm uses the extent of a project as an input and generates chunks of tasks lying next to each other. Each group has a height of three tasks and a width of approximately 40 tasks.

```
{
  "finishedCount" : 0,
  "groupId" : "g101",
  "numberOfTasks" : 54,
  "progress" : 0,
  "projectId" : "-MrbXgHx8YJDt6cTIyGA",
  "requiredCount" : 3,
  "xMax" : "160239",
  "xMin" : "160222",
  "yMax" : "129763",
```

(continues on next page)

(continued from previous page)

```
  "yMin" : "129761"
}
```

8.4 Tasks

Tasks are only saved for tutorials, since their spatial inforamtion can be derived from the spatial extent of the corresponding group.

Below is an example json for a tutorial project, as can be seen on the three extra attributes screen, referenceAnswer and taskID_real.

```
{
  "groupId" : 101,
  "projectId" : "tutorial_-MGwrwsP9cTYf6c_Nbg3",
  "referenceAnswer" : 0,
  "screen" : 1,
  "taskId" : "18-100-131072",
  "taskId_real" : "18-65040-120545",
  "taskX" : 100,
  "taskY" : 131072,
  "url" : "https://ecn.t0.tiles.virtualearth.net/tiles/a023313133022210002.jpeg?
↪g=7505&mkt=en-US"
}
```

8.5 Results

Results contain information on the user classifications. However, only “Yes” (1), “Maybe” (2) and “Bad Imagery” (3) classifications are stored as results. Whenever users indicate “No building” by just swiping to the next set of tasks, no data entry is created. “No Building” classifications can only be modelled retrospectively for groups where a user also submitted at least one “Yes”, “Maybe” or “Bad Imagery” classification.

CHAPTER

NINE

CHANGE DETECTION

9.1 Project Draft

The Change detection project type is initialized in the same way as the standard buildArea project.

Project Draft example for a Change Detection project:

```
{
  "createdBy": "Sample Manager",
  "geometry": {"type":"FeatureCollection","features":[{"type":"Feature","properties":
↪{},"geometry":{"type":"Polygon","coordinates":[[[-175.21785736083984,-21.
↪122295110595505],[-175.2367401123047,-21.148873980682744],[-175.21339416503906,-21.
↪152716314425852],[-175.19931793212888,-21.15239612375494],[-175.19588470458984,-21.
↪147913381670975],[-175.1931381225586,-21.136385707660683],[-175.1934814453125,-21.
↪129660817021904],[-175.21785736083984,-21.122295110595505]]]}}]},
  "image": "http://www.redcrosseth.org/media/k2/items/cache/
↪5a05a447acfdf6fcc40548cc4c1cea8d_L.jpg",
  "lookFor": "DESTROYED BUILDINGS",
  "name": "Change Detection Sample Project",
  "projectDetails": "This project uses Bing as the tile server and zoom level 18 for␣
↪the before image. For after we use imagery from open aerial map.",
  "verificationNumber": 3,
  "groupSize": 15,
  "projectType": 3,
  "tileServerA": {
    "name": "bing",
    "apiKeyRequired": true,
    "caption": "Before",
    "credits": "© 2019 Microsoft Corporation, Earthstar Geographics SIO"
  },
  "tileServerB": {
    "name": "custom",
    "url": "https://tiles.openaerialmap.org/5b3541802b6a08001185f8b1/0/
↪5b3541802b6a08001185f8b2/{z}/{x}/{y}.png",
    "apiKeyRequired": false,
    "apiKey": "",
    "caption": "After",
    "date": "2018-02-21",
    "credits": "© OpenAerialMap"
  }
}
```

Examples of other initialization options can be found in the mapswipe-backend repository at mapswipe_workers/tests/integration/fixtures/change_detection/project_drafts.json.

9.2 Project structure

Project Structure example for a project which was created via HOT Tasking Manager Project ID.

```
{
  "contributorCount" : 0,
  "created" : "2021-12-23T14:14:52.179930Z",
  "createdBy" : "X0zTSyvY0khDfRwc99aQfIjTEPK2",
  "groupMaxSize" : 0,
  "groupSize" : 25,
  "image" : "https://firebasestorage.googleapis.com/v0/b/dev-mapswipe.appspot.com/o/
↪projectImages%2FEQ%2BEarthquake.png?alt=media&token=6e82ba52-8adb-4214-8f81-
↪4b7030c00946",
  "isFeatured" : false,
  "lookFor" : "damaged buildings",
  "name" : "Earthquake - Experimental Damage Assessment - Les Cayes (Haiti)␣
↪(1)\nSimon BA",
  "progress" : 0,
  "projectDetails" : "In attempt to provide a rapid damage assessment for the 7.2␣
↪magnitude earthquake on August 14, please slowly compare the images to determine if␣
↪damage is visible in the post-event scene. This methodology is still being tested␣
↪and should not replace traditional damage assessment methods. Imagery is provided␣
↪through [Maxar's Open Data Programm](https://www.maxar.com/open-data) and hosted by␣
↪[MapBox](https://www.mapbox.com/).",
  "projectId" : "-Mrbd5ArF4lb_GoYG2I5",
  "projectNumber" : "1",
  "projectRegion" : "Les Cayes (Haiti)",
  "projectTopic" : "Earthquake - Experimental Damage Assessment",
  "projectType" : 3,
  "requestingOrganisation" : "Simon BA",
  "requiredResults" : 3636,
  "resultCount" : 0,
  "status" : "inactive",
  "tileServer" : {
    "apiKey" : "",
    "credits" : "© 2019 Maxar",
    "name" : "maxar_premium",
    "url" : "https://services.digitalglobe.com/earthservice/tmsaccess/tms/1.0.0/
↪DigitalGlobe%3AImageryTileService@EPSG%3A3857@jpg/{z}/{x}/{y}.jpg?connectId={key}"
  },
  "tileServerB" : {
    "credits" : "© Maxar, MapBox",
    "name" : "custom",
    "url" : "https://api.mapbox.com/v4/mapboxsatellite.haiti-post-2021/{z}/{x}/{y}.
↪webp?sku=101Fw3jtBuWI5"
  },
  "tutorialId" : "tutorial_-MhJtd9ePFOw8Vs6xwZ2",
  "verificationNumber" : 3,
  "zoomLevel" : 19
}
```

9.3 Group structure

```
{
  "finishedCount" : 0,
  "groupId" : "g101",
  "numberOfTasks" : 24,
  "progress" : 0,
  "projectId" : "-Mrbd5ArF4lb_GoYG2I5",
  "requiredCount" : 3,
  "xMax" : "154722",
  "xMin" : "154715",
  "yMax" : "235151",
  "yMin" : "235149"
}
```

9.4 Task structure

```
{
  "groupId" : "g101",
  "projectId" : "-Mrbd5ArF4lb_GoYG2I5",
  "taskId" : "19-154715-235149",
  "taskX" : "154715",
  "taskY" : "235149",
  "url" : "https://services.digitalglobe.com/earthservice/tmsaccess/tms/1.0.0/
↪DigitalGlobe%3AImageryTileService@EPSG%3A3857@jpg/19/154715/289138.jpg",
  "urlB" : "https://api.mapbox.com/v4/mapboxsatellite.haiti-post-2021/19/154715/
↪235149.webp?sku=101Fw3jtBuWI5"
}
```

9.5 Result Structure

Results contain information on the user classifications. However, only “Yes” (1), “Maybe” (2) and “Bad Imagery” (3) classifications are stored as results. Whenever users indicate “No Change” by just swiping to the next task, no data entry is created.

CHAPTER

TEN

FOOTPRINT

10.1 Project Draft

Footprint projects can be supplied with geometries in three seperate ways.

1. by specifying a HOT Tasking Manager Project ID and an object filter
2. by specifying an url to the data (e.g. an ohsomeAPI call)
3. by uploading an aoi and an object filter

Project Draft example for a footprint project which was initialized with an aoi and a filter:

```
{
  "createdBy" : "Sample Admin",
  "filter" : "building=* and geometry:polygon",
  "geometry" : {
    "type": "FeatureCollection",
    "features": [
      {
        "type": "Feature",
        "properties": {},
        "geometry": {
          "type": "Polygon",
          "coordinates": [[[9.18032169342041, 48.790552471542284],[9.187102317810059,
↪48.790552471542284],[9.187102317810059,48.79407236257656],[9.18032169342041,48.
↪79407236257656],[9.18032169342041,48.790552471542284]]]}
      }
    ]
  },
  "groupSize" : 25,
  "lookFor": "Buildings",
  "image": "http://www.fragosus.com/test/Javita.jpg",
  "projectDetails": "This is a template for a GeoJSON AOI project. We use Bing as the␣
↪tile server.",
  "inputType" : "aoi_file",
  "name" : "Test Footprint GeoJSON AOI",
  "projectTopic" : "Test Footprint GeoJSON AOI",
  "projectType" : 2,
  "verificationNumber": 3,
  "tileServer" : {
    "credits" : "© 2019 Microsoft Corporation, Earthstar Geographics SIO",
    "name" : "bing",
    "url" : "",
    "wmtsLayerName" : ""
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Examples for the other initialization options can be found in the mapswipe-backend repository at mapswipe_workers/tests/integration/fixtures/footprint/projectDrafts.

10.2 Project structure

Project Structure example for a project which was created via HOT Tasking Manager Project ID.

```
{
  "TMId" : "11193",
  "contributorCount" : 1,
  "created" : "2021-12-10T18:05:26.090515Z",
  "createdBy" : "X0zTSyvY0khDfRwc99aQfIjTEPK2",
  "filter" : "building=* and geometry:polygon",
  "groupMaxSize" : 0,
  "groupSize" : 30,
  "image" : "https://firebasestorage.googleapis.com/v0/b/dev-mapswipe.appspot.com/o/
↪projectImages%2Fimage.jpeg?alt=media",
  "inputType" : "TMId",
  "isFeatured" : false,
  "lookFor" : "Buildings",
  "name" : "OSM Building Validation - Indonesia (1)\nAmerican Red Cross",
  "progress" : 0,
  "projectDetails" : "The Red Cross Climate Centre, Indonesian Red Cross (Palang␣
↪Merah Indonesia/PMI), IFRC, British Red Cross and Australian Red Cross are␣
↪implementing a programme where the data contributed will be used by the Red Cross␣
↪to assist in forecasting future disaster impacts, by knowing in advance what is␣
↪likely to be impacted and its exposure and vulnerability. The information will help␣
↪implementation of early action activities to take place before a disaster strikes,␣
↪contributing to reduce risk, prepare for effective response and ultimately to␣
↪strengthen community resilience.",
  "projectId" : "-Mq_IVluLteQRS75gWej",
  "projectNumber" : "1",
  "projectRegion" : "Indonesia",
  "projectTopic" : "OSM Building Validation",
  "projectType" : 2,
  "requestingOrganisation" : "American Red Cross",
  "requiredResults" : 286302,
  "resultCount" : 0,
  "status" : "private_active",
  "teamId" : "-Mq_EQlzqmYytCspuFSq",
  "tileServer" : {
    "apiKey" : "ca613e76-811f-46e7-9e1d-84f6795441c2",
    "credits" : "© 2019 Maxar",
    "name" : "maxar_premium",
    "url" : "https://services.digitalglobe.com/earthservice/tmsaccess/tms/1.0.0/
↪DigitalGlobe%3AImageryTileService@EPSG%3A3857@jpg/{z}/{x}/{y}.jpg?connectId={key}"
  },
  "tutorialId" : "tutorial_-MO3ky5z--RY8PC1lONa",
  "verificationNumber" : 3
}
```

10.3 Group structure

The footprint groups follow the standard group structure.

```
{
  "finishedCount" : 0,
  "groupId" : "g100",
  "numberOfTasks" : 30,
  "progress" : 0,
  "projectId" : "-Mq_FxTdV2QJHsxQcvFk",
  "requiredCount" : 3
}
```

10.4 Task structure

```
{
  "feature_id" : 0,
  "geojson" : {
    "coordinates" : [ [ [ 5.15910196973, 13.48686869581 ], [ 5.15937974751, 13.
↪48686869581 ], [ 5.15937974751, 13.48742425137 ], [ 5.15910196973, 13.48742425137 ],
↪ [ 5.15910196973, 13.48686869581 ] ] ],
    "type" : "Polygon"
  },
  "id" : "13564_100_0",
  "properties": "feature_geometries, e.g. attributes from osm"
}
```

10.5 Result Structure

The Result for a footprint project are explicitly given via the “yes”, “no” and “not sure” buttons.

CHAPTER

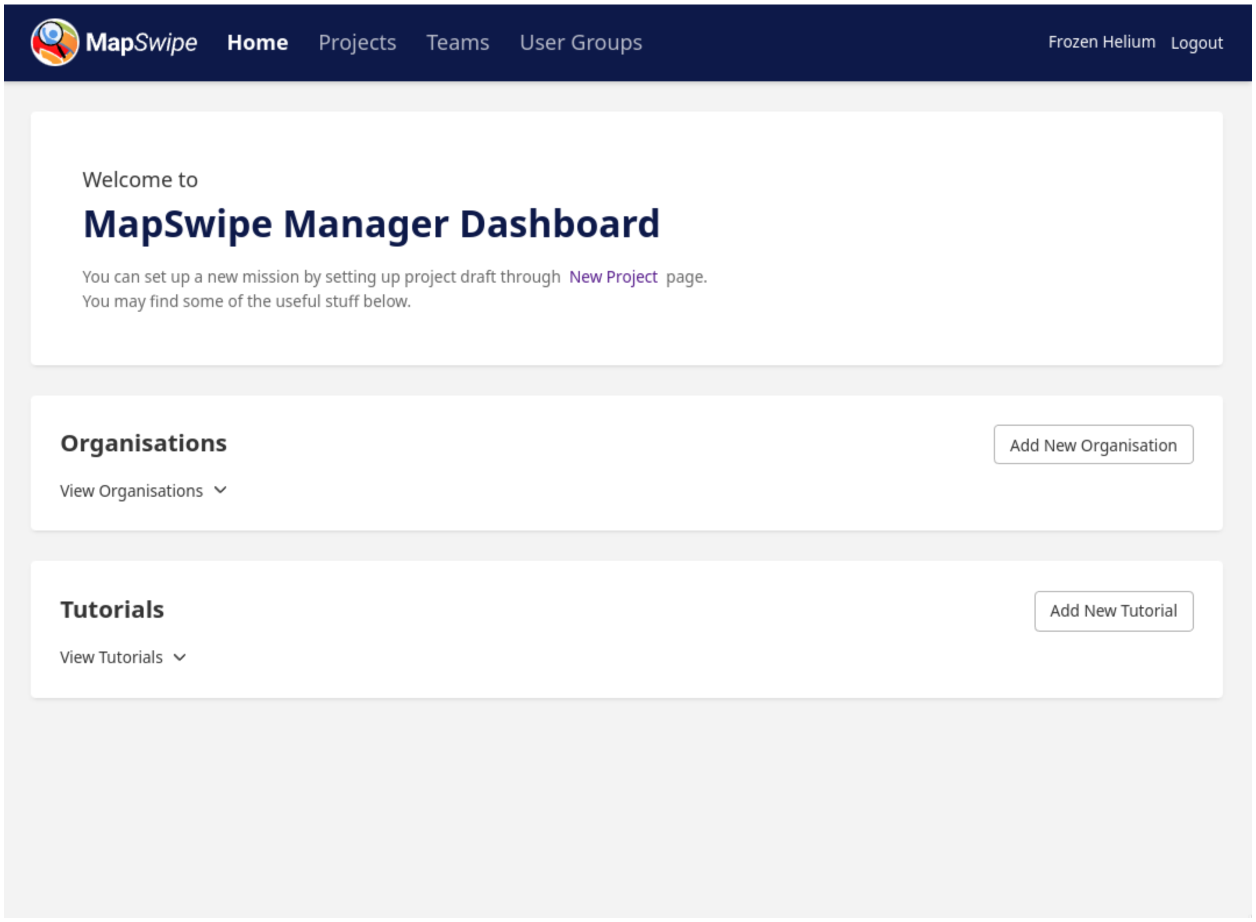
ELEVEN

FOR MAPSWIPE MANAGERS

11.1 Setting up a new MapSwipe mission

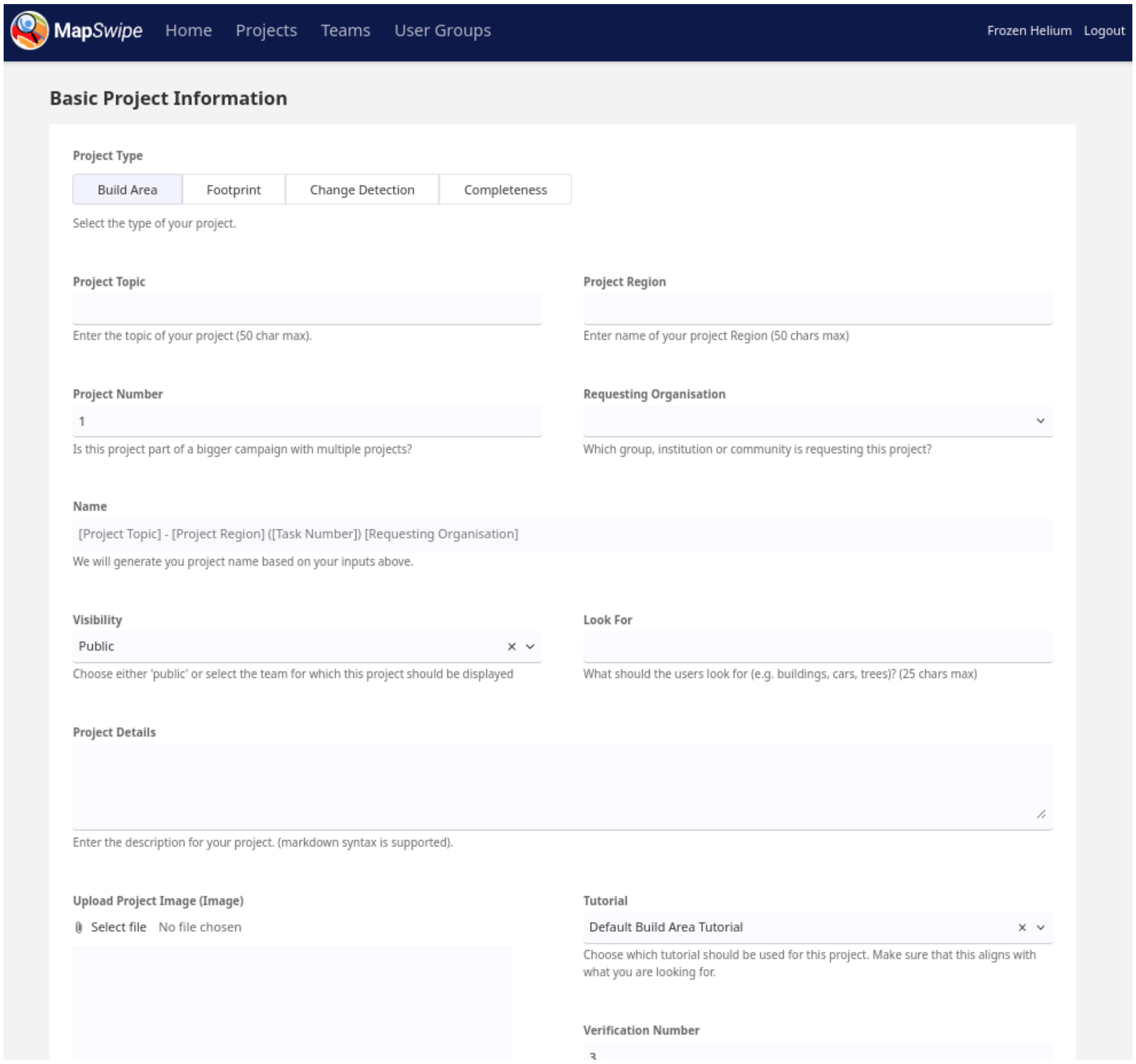
Any of the Missing Maps members can request a MapSwipe mission. If you want to add a new mission, but don't know how to do it, it will be best to reach out to the MapSwipe community via Slack.

To set up a new mission you can create a project draft through the Manager Dashboard. Sign in with your MapSwipe account. In order to create new projects you need dedicated *project manager credentials*. Reach out to the MapSwipe community if you don't have these already. Once you're signed in, you will see a screen similar to the one below.



You can set up a new mission by setting up project draft through New Project page.
You may find some of the useful stuff below.

You can directly go to create **New Project** page from here, or you can navigate to the **Projects** page from the navbar at the top and then click on **Add New Project** button on the top right. You'll be navigate to the New Project Page which will look like following.



Basic Project Information

Project Type

Build Area | Footprint | Change Detection | Completeness

Select the type of your project.

Project Topic

Enter the topic of your project (50 char max).

Project Region

Enter name of your project Region (50 chars max)

Project Number

1

Is this project part of a bigger campaign with multiple projects?

Requesting Organisation

Which group, institution or community is requesting this project?

Name

[Project Topic] - [Project Region] ([Task Number]) [Requesting Organisation]

We will generate you project name based on your inputs above.

Visibility

Public

Choose either 'public' or select the team for which this project should be displayed

Look For

What should the users look for (e.g. buildings, cars, trees)? (25 chars max)

Project Details

Enter the description for your project. (markdown syntax is supported).

Upload Project Image (Image)

Select file No file chosen

Tutorial

Default Build Area Tutorial

Choose which tutorial should be used for this project. Make sure that this aligns with what you are looking for.

Verification Number

3

General points of attention:

- Verify if the project name is short and clear
- The minimum verification count is 3. The algorithm is calculated on that. Less persons looking at one square has implications on the quality of the MapSwipe data. The best results we have with a 5 person verification.
- Reread, correct and improve the description given through the request.
- Check if the image is the good format and if the size is not to big - max 1MB. The image will be show as a thumbnail for your mission.
- Tile server import, which imagery are you using is the question here. We only have legal permission to use Bing for now. For custom tile servers (e.g. from OpenAerial Map) check the permission.
- Check the (Bing) imagery available for the area, we learned that good imagery is key to keep a mission going. If the imagery is not good enough or covered by clouds adjust the area. You need to zoom in till level 18.

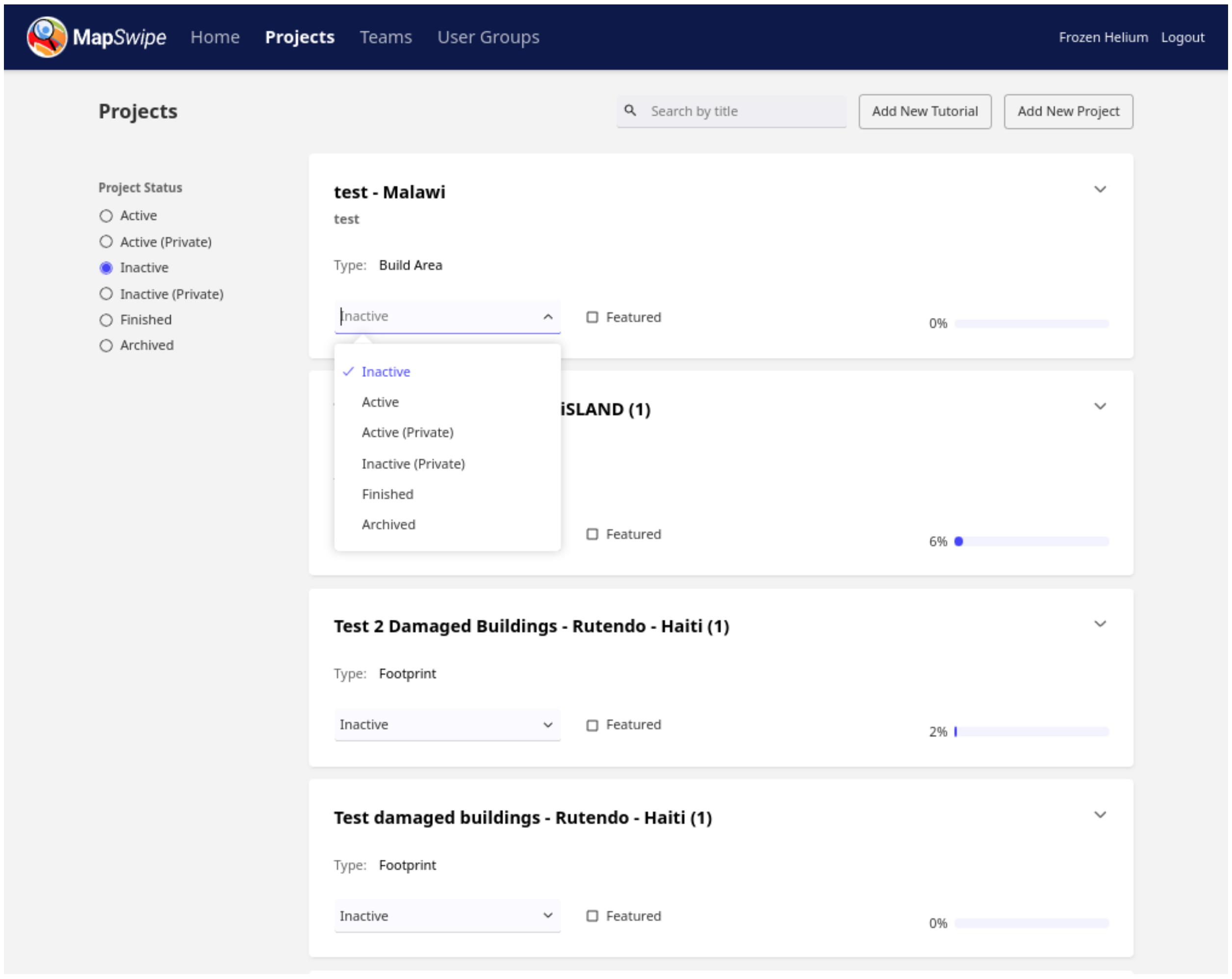
For BuildArea and ChangeDetection projects:

- MapSwipe can only process geographical areas up to 5,000km. Check if the GeoJSON you received has this size. If not, you may need to split up the area into multiple pieces, and create more than one MapSwipe mission out of the area.
- The GeoJSON needs to be a flat polygon, check this and change (in geojson.io or QGIS) if needed.

For Footprint projects:

- The GeoJSON file should contain only simple Polygons. We currently don't support complex Multipolygon geometries (e.g. polygon with holes)

Once you submit, the task should appear relatively quickly in the manager dashboard. You will receive a message in Slack. But it's still not active and not visible to the MapSwipe app users. You need to set the project status to `active` through the manager dashboard. Just navigate to the **Projects** page and then change the Project Status on the left filter pane to show **Inactive** projects. You'll find the project you've just created here. Now, you can set the project status to be **Active** from the dropdown as shown in the screenshot below. If the new project does not appear in the app after about 1 hour, check Slack for an error message, and see Troubleshooting section below.



test - Malawi
test
Type: Build Area
Inactive
✓ Inactive
Active
Active (Private)
Inactive (Private)
Finished
Archived
Featured
0%
iSLAND (1)
Featured
6%
Test 2 Damaged Buildings - Rutendo - Haiti (1)
Type: Footprint
Inactive
Featured
2%
Test damaged buildings - Rutendo - Haiti (1)
Type: Footprint
Inactive
Featured
0%

11.2 Troubleshooting

- If a project does not show up in the app or on the project dashboard, probably something went wrong with the GeoJSON file, visit the imports dashboard - hopefully it will be the last in the list and the error can be determined and fixed. Please ask on the MapSwipe slack if you cannot determine the error.

- If the project is not in the imports dashboard, then it was not successfully submitted and you may try again.
- If you encounter any error like *the % is not rising anymore in the app*, *the number of mappers is not rising anymore in the app firebase functions* might be the reason for this. Check firebase functions logs (Firebase > Functions > Logs) or reach out for help in Slack.

11.3 Becoming a project manager

If you have access to the mapswipe backend server you can grant project manager rights like this:

```
docker run --name mapswipe_workers_local -it pythonmapswipeworkers_mapswipe_workers ␣
↪bash`
mapswipe_workers --verbose user-management --email=your_email@xyz.org --manager=true
exit
docker rm mapswipe_workers_local
```

CHAPTER

TWELVE

COMMAND LINE INTERFACE

The Mapswipe Backend provides a Command Line Interface(CLI) with which the users can interact with the program. They can be used for example to create projects, which were uploaded to the manager-dashboard, or to export statistics on the finished projects. To get a comprehensible lists of the available commands use the `--help` flag.

`mapswipe_workers --help` would get you all possible commands, while e.g. `mapswipe_workers archive --help` would get you additional information on how to use that command.

In our current deployment setup the commands of the MapSwipe Workers CLI are hard-coded in the Docker-Compose File.

You can run these commands also using docker-compose:

```
docker-compose run mapswipe_workers mapswipe_workers --help
```

CHAPTER

THIRTEEN

USE CASES

This document is based on the first version of MapSwipe and contains outdated information.

13.1 How to identify "good" mapping tasks for MapSwipe

MapSwipe projects can cover large areas in comparison to other mapping approaches, e.g. using the HOT Tasking Manager. Nevertheless, the level of detail of the resulting information you can expect from the resulting data will be lower than using the data from OpenStreetMap.

Here is a list of characteristics that apply to many projects we have in MapSwipe and that may also indicate how suitable your project is for MapSwipe:

- the features you want to map are relatively easy to spot and distinguish from other objects on satellite image with a resolution of around 0.3-0.5 meter.
- you are interest only in a limited number of object types, the ideal case would be 1 object type per project
- the area is large, larger than projects you usually see in the HOT Tasking Manager
- the features you want to map cover only some parts of the whole area (e.g. the built-up area is often less than 10% of the whole project area)

13.2 Building Mapping

This has been the focus for most MapSwipe projects. Buildings are relatively easy to spot, since their shape is familiar to most MapSwipe users. However, not all buildings look the same. Some have a rectangular shape, but others are round or build of clay, which makes it difficult to distinguish buildings and the ground. Sometimes also trees may look like a building.

Building mapping can be done at zoom level 18 or higher.

13.3 Landcover Mapping - e.g. Mangroves

MapSwipe can be used to map other land cover classes or features besides buildings. We are currently exploring, how the MapSwipe approach could be used to map mangrove forests in southeas asia. The difficult aspect of this tasks is to differentiated mangrove forest and other vegetation types, such as grassland or land used for agriculture.

Landcover mapping can be done at various zoom levels depending of the size of the features of interest. For forest mapping you may choose a lower zoom level, whereas more fine grained features also require a higher zoom level.

13.4 OpenStreetMap Data Validation

In the initial phase data from MapSwipe can support the detailed mapping in OpenStreetMap. However, we can also turn MapSwipe into an OSM data validation tool. By combining satellite imagery and OSM data into one image we can check the completeness and quality of existing data in OSM.

This approach can be used to quickly assess area, where new satellite imagery became available and an updating of the already existing OSM data is needed.

Custom tiles for these type of MapSwipe project can be generated using MapBox Studio. There you can style the OSM data overlay and choose several base layers.

CHAPTER

FOURTEEN

ANALYSIS OF THE MAPSWIPE TILES – HOW ‘SQUARE’ ARE THEY?

To generate VGI for humanitarian use, project areas are cut into tiles in order for users to search for buildings on aerial imagery. The Mapswipe tiles are created as described by using Level of Detail 18 (task_z = 18 for Mapswipe data). The users assign a ‘Yes’, ‘No’, ‘Maybe’ or ‘Bad Imagery’ to each tile. This information is used to select on positive tiles and create bigger geometries representing a settlement layer. While these tiles look absolutely squared (e.g. in a web map or in mobile applications such as MapSwipe), they **DON’T** represent the same area.

The web mercator projection distorts the image in a way that objects further away from the equator will continuously appear bigger than they are. So always keep in mind: **The area represented by a tile will be getting smaller the further north or south your tile is located.** Table 1 shows that the area of projects further away from the equator have smaller areas and smaller side length. Furthermore, the angles will deviate further from rectangular.

Table 1: Measurements for Mapswipe Tiles.

Figure 1 shows the decrease in size from south to north in project 10836 (Chad) on the northern hemisphere.

In the Mapswipe App, each tile has a size of 256x256 pixels. The translation of pixels into meters is depending on geographical longitude and level of details (see Figure 2).

As a result of the change in sidelength and inner angles, the area of the tiles decreases with distance from equator as well. Have a look at Figure 3.

14.1 Summary

- Area and side length of a tile will continuously decrease with distance to the equator
- within one project the differences in size depend on the north-south extend of the project
- The tile shapes differ from rectangular form the further away they are from the equator
- Find a example geojson file with calculated area and side length for given task_y or the Y-coordiantes of the tile center. You can use this file in QGIS or any other GIS software.

CHAPTER FIFTEEN

CONFIGURATION REFERENCE

Most of the configuration is stored in environment variables. At the root of the GitHub repository (in the same directory as `docker-compose.yml`) an example file (`example.env`) with all possible configuration variables exists. To get started copy this file to `.env` (no name is required) and fill in missing variables. The Docker Compose file will access those variables when needed.

> Note: If you want those variables to be accessible as Environment Variables in your current shell (Eg. Inside a Python virtual environment for development.) your need to parse the file and export the variables: `export $(cat .env | xargs)`

In following chapters configuration values and keys are discussed for each part of the MapSwipe Back-end.

15.1 MapSwipe Workers

All configuration values for MapSwipe Workers are stored in environment variables.

Required environment variables are:

15.1.1 Firebase

- FIREBASE_API_KEY
- FIREBASE_DB
- FIREBASE_TOKEN
- GOOGLE_APPLICATION_CREDENTIALS

15.1.2 Postgres DB

- POSTGRES_DB
- POSTGRES_HOST
- POSTGRES_PASSWORD
- POSTGRES_PORT
- POSTGRES_USER

15.1.3 OSMCha

- OSMCHA_API_KEY

15.1.4 Optional environment variables:

- SLACK_CHANNEL
- SLACK_TOKEN
- SENTRY_DSN

For satellite imagery access to at least one provider is needed. Define the API key as environment variable:

- IMAGE_BING_API_KEY
- IMAGE_MAPBOX_API_KEY
- IMAGE_MAXA_PREMUIM_API_KEY
- IMAGE_MAXAR_STANDARD_API_KEY
- IMAGE_ESRI_API_KEY
- IMAGE_ESRI_BETA_API_KEY

  Notes: When deploying using `docker` or `docker-compose` `POSTGRES_HOST` should have the value `postgres` and the Service Account Key (`serviceAccountKey.json`) should be copied to `mapswipe_workers/serviceAccountKey.json` so that during the build of the image the file can by copied by Docker.

15.1.5 Elaboration

Firebase: MapSwipe Workers use the Firebase Python SDK and the Firebase REST API. Both require the database name (`FIREBASE_DB`) and the API-Key from the Firebase instance. The Firebase Python SDK does also need a Service Account Key. The path to this file is set in the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.

Postgres: MapSwipe Workers writes data to a Postgres database and generate files for the API based data in Postgres.

OSMCha: MapSwipe Workers enriches some Projects with data from OSM changelogs which are requested from OSMCha. Create an account, you will find you api key in your profile e.g. `Token 589adf125234a`

Sentry (optional): MapSwipe workers use sentry to capture exceptions. You can find your project's DSN in the "Client Keys" section of your "Project Settings" in Sentry. Check Sentry's documentation for more information.

Slack (optional): The MapSwipe workers send messages to slack when a project has been created successfully, the project creation failed or an exception gets raised. refer to Python slackclient's documentation how to get a Slack Token.

Imagery: MapSwipe uses satellite imagery provided by Tile Map Services (TMS). If you are not familiar with the basic concept have a look at Bing's documentation.

15.2 Postgres

Required environment variables are (Those are the same as needed by MapSwipe Workers):

- POSTGRES_DB
- POSTGRES_HOST
- POSTGRES_PASSWORD
- POSTGRES_PORT

- POSTGRES_USER

  Notes: When deploying using `docker` or `docker-compose` `POSTGRES_HOST` should have the value `postgres`.

15.2.1 Postgres Backup

On details of how the back-up works please refer to Postgres Backup.

Required environment variables are:

- WALG_GS_PREFIX

To gain access to Google Cloud Storage another Service Account Key is needed. Again refer to Postgres Backup on how to create this file. The Service Account Key (`serviceAccountKey.json`) should be saved to `postgres/serviceAccountKey.json`

15.3 Manager Dashboard

Please refer to the official documentation if you set up your own firebase. Otherwise you can request guidance on the settings from the mapswipe team. The structure of your app.js should look like below.

15.3.1 Firebase

- MANAGER_DASHBOARD_FIREBASE_API_KEY
- MANAGER_DASHBOARD_FIREBASE_AUTH_DOMAIN
- MANAGER_DASHBOARD_FIREBASE_DATABASE_URL
- MANAGER_DASHBOARD_FIREBASE_PROJECT_ID
- MANAGER_DASHBOARD_FIREBASE_STORAGE_BUCKET
- MANAGER_DASHBOARD_FIREBASE_MESSAGING_SENDER_ID
- MANAGER_DASHBOARD_FIREBASE_APP_ID

15.3.2 Sentry

- MANAGER_DASHBOARD_SENTRY_DSN
- MANAGER_DASHBOARD_SENTRY_TRACES_SAMPLE_RATE

15.4 Community Dashboard

15.4.1 Django API

- COMMUNITY_DASHBOARD_GRAPHQL_CODEGEN_ENDPOINT
- COMMUNITY_DASHBOARD_GRAPHQL_ENDPOINT

15.4.2 Sentry

- COMMUNITY_DASHBOARD_SENTRY_DSN
- COMMUNITY_DASHBOARD_SENTRY_TRACES_SAMPLE_RATE

15.4.3 Elaboration

COMMUNITY_DASHBOARD_GRAPHQL_CODEGEN_ENDPOINT: Graphql endpoint of the Django API. Eg: https://api.example.com/graphql/ **COMMUNITY_DASHBOARD_GRAPHQL_ENDPOINT**: Same as COMMUNITY_DASHBOARD_GRAPHQL_CODEGEN_ENDPOINT

15.5 Django API

All configuration values for Django are stored in environment variables.

Required environment variables are:

15.5.1 Django

- DJANGO_SECRET_KEY
- DJANGO_ALLOWED_HOST

15.5.2 Optional environment variables:

- DJANGO_SENTRY_DSN
- DJANGO_SENTRY_SAMPLE_RATE
- DJANGO_RELEASE
- Postgres (NOTE: Database configuration are pulled from postgres configuration directly in docker-compose files.)
  - DJANGO_DB_NAME
  - DJANGO_DB_USER
  - DJANGO_DB_PWD
  - DJANGO_DB_HOST
  - DJANGO_DB_PORT

15.5.3 Elaboration

DJANGO_SECRET_KEY: A secret key for a particular Django installation. This is used to provide cryptographic signing, and should be set to a unique, unpredictable value. **DJANGO_SENTRY_SAMPLE_RATE**: Sample rate by which sentry send transaction metadata. Value should be between 0 to 1. https://docs.sentry.io/platforms/python/guides/django/configuration/sampling/

15.6 NGINX

The configuration for nginx is defined in `nginx/nginx.conf.template` file.

15.6.1 Domains

- NGINX_MAIN_DOMAIN
- NGINX_DJANGO_DOMAIN
- NGINX_MANAGER_DASHBOARD_DOMAIN
- NGINX_COMMUNITY_DASHBOARD_DOMAIN

  NOTE: Make sure the used domain have valid certificates in /etc/letsencrypt/

  NGINX_MAIN_DOMAIN: Domain for main mapswipe static api server. **NGINX_DJANGO_DOMAIN**: Domain for django web server. **NGINX_MANAGER_DASHBOARD_DOMAIN**: Domain for manager dashboard. **NGINX_COMMUNITY_DASHBOARD_DOMAIN**: Domain for community dashboard.

CHAPTER SIXTEEN

INSTALLATION

This document describes how to setup all the parts of the MapSwipe backend in a production environment.

Please consult the Configuration Reference for this setup as well.

1. Firebase
2. Postgres
3. MapSwipe Workers
4. API
5. Manager Dashboard
6. Django API
7. Community Dashboard
8. Lets Encrypt and NGINX as proxy

For this setup the main repository is required:

```
git clone https://github.com/mapswipe/python-mapswipe-workers.git
cd python-mapswipe-workers
```

16.1 Firebase Setup

Download a Service Account Key File for MapSwipe Workers:

1. In the Firebase console, open Settings > Service Accounts.
2. Click Generate New Private Key, download it and use it to set the environment variables in .env.

Configure your API Keys in Google APIs & Services

1. Open Google APIs & Services > Credentials
2. Create API key for MapSwipe workers:
    - set name of api key to `mapswipe-workers`
    - set Application restrictions > IP addresses > set IP addresse of mapswipe workers server
    - set API restrictions > Restrict Key > Identity Toolkit API
3. Create API key for Manager Dashboard:
    - set name of api key to `manager-dashboard`
    - set Application restrictions > HTTP referrers > set HTTP referrer of managers dashboard

   - set API restrictions > Restrict Key > Identity Toolkit API and Cloud Functions API

4. Also make sure to configure the API keys for the App side here.

16.1.1 Deploy Database Rules and Functions

The Firebase setup consists of two parts:

- Firebase Database Rules (`firebase/database.rules.json`)
- Firebase Functions (`firebase/functions/`)

To deploy them to the Firebase Project the Firebase Command Line Tools are required. When running the provided Docker image (`firebase/Dockerfile`) the database rules and the functions will be deployed. For this to work a Firebase Token is needed:

1. On a PC with a browser install the Firebase Command Line Tools (https://firebase.google.com/docs/cli/)
2. Run `firebase login:ci` to generate a Firebase Token.
3. Save the Firebase Token to `.env` at the root of the cloned MapSwipe Backend repository on the server: `echo "FIREBASE_TOKEN=your_token" >> .env`

Once the Firebase Token is set the database rules and functions will be deployed when running the `firebase_deploy` Docker image using `docker-compose`:

```
docker-compose up --build -d firebase_deploy
```

This container needs to run only as long until the `firebase deploy` command inside the Docker container terminates. Use `docker logs firebase_deploy` to find out if the command is still running.

16.2 Postgres Setup

In the `postgres` directory is an `initdb.sql` file for initializing a Postgres database.

When running Postgres using the provided Dockerfile it will setup a Postgres database using the `initdb.sql` file during the build.

The Postgres configuration (eg. password) has to be defined in the environment file (`.env`):

```
POSTGRES_PASSWORD=your_password
```

To run the Postgres Docker container:

```
docker-compose up -d postgres
```

The Postgres instance will be exposed to `localhost:5432`.

16.3 MapSwipe Workers

16.3.1 Configuration

16.3.2 Run MapSwipe Workers

```
docker-compose up -d mapswipe_workers
```

16.4 Manager Dashboard

The Manager Dashboard uses the Firebase JavaScript client SDK to access *Firebase Database* service as authenticated as MapSwipe user with project manager credentials.

1. Open Google APIs & Services > Credentials
2. Create API key for MapSwipe workers:
    - set name of api key to `mapswipe_workers_api_key`
    - set Application restrictions > IP addresses > set IP addresse of mapswipe workers server
    - set API restrictions > Restrict Key > Identity Toolkit API
3. Create API key for Manager Dashboard:
    - set name of api key to `manager_dashboard_api_key`
    - set Application restrictions > HTTP referrers > set HTTP referrer of managers dashboard (e.g. `https://dev.mapswipe.org`)
    - set API restrictions > Restrict Key > Identity Toolkit API and Cloud Functions API
4. Also make sure to configure the API keys for the App side here.

Project-id refers to the name of your Firebase project (e.g. dev-mapswipe). The firebaseConfig in `mapswipe_dashboard/js/app.js` should look like this now:

```
var firebaseConfig = {
    apiKey: "manager_dashboard_api_key",
    authDomain: "your_project_id.firebaseapp.com",
    databaseURL: "https://your_project_id.firebaseio.com",
    storageBucket: "your_project_id.appspot.com"
  };
```

Get **Web API Key**: `> Settings > Project settings > General`. Add the web api key to the `.env` file.

Make sure to set restrictions correctly:

- https://cloud.google.com/docs/authentication/api-keys#api_key_restrictions
- https://console.cloud.google.com/apis/credentials

```
docker-compose up -d manager_dashboard
```

16.5 API

Currently the API is a simple Nginx server which serves static files. Those files are generated by MapSwipe Workers and shared over a Docker volume with the API Container.

```
docker-compose up -d api
```

16.6 Django API

Currently the django API is a web server build using django which provides stats information.

```
docker-compose up -d django
```

16.7 Lets Encrypt and NGINX

To enable SSL for the API and MapSwipe Manager Dashboard use Certbot to issue standalone certificates.

16.7.1 Certbot

To install Certbot follow instructions on https://certbot.eff.org/lets-encrypt/ubuntubionic-other

Create certificates:

```
certbot certonly \
    --standalone \
    --domain dev-api.mapswipe.org \
    --domain dev-managers.mapswipe.org \
    --agree-tos \
    --email e@mail.org \
    --non-interactive
```

> Note: Certbot systemd timer for renewal of certificate will not work for standalone certificates because the service (docker nginx) which occupies port 80 has to be stopped before renewal.

For certificate renewal a cronjob is used. This has to be run as root: `sudo crontab -e`

```
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

0 */12 * * * certbot -q renew --pre-hook "docker stop nginx" --post-hook "docker␣
↪start nginx"
```

16.7.2 Nginx

NGINX serves the MapSwipe API and Manager Dashboard. If you want these point to a specific domain, make sure to set it up.

Once you got your domain name change `server_name`, `ssl_certificate` and `ssl_certificate_key` in the NGINX configuration file (`nginx/nginx.conf`)

Run NGINX:

```
docker-compose up -d nginx
```

CHAPTER

SEVENTEEN

DEBUGGING

17.1 Logs - MapSwipe Workers

Where can I find logs?

- `docker logs mapswipe_workers`
- or
- `cat mapswipe-data/mapswipe_workers.log`

Logs are written to directly to the terminal (stdout). The easiest way is therefore is to run `docker logs mapswipe_workers` to see the logs.

Logs are also writing to file inside the Docker container (`~/.local/share/mapswipe_workers/mapswipe_workers.log`). The parent directory of the file is the data directory of MapSwipe Workers. This directory is mounted (as a Docker volume) locally to disk (`mapswipe-data/`). Logs can therefore be accessed as text file as well: `cat mapswipe-data/mapswipe_workers.log`

17.2 Logs - Django web server.

- `docker compose logs django`
- or
- `cat django-data/django.log`

17.3 Common Errors

Docker containers are always restarting: Take a look at the docker logs (eg. `docker logs container_name`). If you get an `Unable to load configuration file at ./cfg/config.cfg. Exiting.` due to `PermissionError: [Errno 13] Permission denied: './config/configuration.cfg'` error message, you probably have SELinux on your system enabled. If so you have to configure (change mount option of volumes) your docker-compose file. Please read the documentation provided by Docker regarding this configuration (https://docs.docker.com/storage/bind-mounts/ Chapter: "Configure the selinux label").

17.4 Useful Docker Commands

- `docker ps -a`: list all containers and check status

- `docker image ls`: list all docker images
- `docker exec -it mapswipe_workers bash`: open shell in a running container
- `docker exec -t mapswipe_workers tail -100 /var/log/mapswipe_workers/mapwipe_workers.log`: show last 100 lines of the log file
- `docker stats`: show memory usage, CPU consumption for all running containers
- `docker system prune --all`: clean up any resources — images, containers, volumes, and networks — that are dangling (not associated with a container)

CHAPTER

EIGHTEEN

BACKUP

For Postgres backups WAL-G is used. “WAL-G is an archival restoration tool for Postgres”.

For more information please refer to the official docs of WAL-G and the official docs of Postgres on write ahead log (WAL). Current setup took inspiration from this blog post.

18.1 Backup setup

The WAL-G backup setup is integrated into the Postgres Docker image. It will do a baseline backup of the database to Google Cloud Storage every day utilizing a cron job and `wal-g backup-push`. After that the Postgres will push WAL (Write-Ahead Log) files to Google Cloud Storage regularly using `wal-g wal-push`. For exact commands please take a look at following script:

- `postgres/wal-g/make_basebackup.sh`
- `postgres/wal-g/archive_command.sh`

WAL-G is installed alongside Postgres. See the Dockerfile of Postgres (`postgres/Dockerfile`) for details. In the docker-compose postgres command (`docker-compose.yml`) archive parameter of postgres are set needed to make archives.

18.1.1 Configuration

To store backups in Google Cloud Storage, WAL-G requires that this variable is set: `WALG_GS_PREFIX` to specify where to store backups (eg. `gs://x4m-test-bucket/walg-folder`). Please add this to the `.env` file at the root of MapSwipe Back-end (See `.example-env` for environment variables which have to be set)

WAL-G determines Google Cloud credentials using application-default credentials like other GCP tools. Get a Service Account Key file (`serviceAccountKey.json`) for your Google Cloud Storage (See Google Cloud Docs) and save it to `postgres/serviceAccountKey.json`.

18.2 Restore setup

The WAL-G restore setup is realized in a dedicated Docker image (`postgres/recovery/Dockerfile`). The entrypoint is the `ini_recovery.sh` srcipt. This script will create a new Postgres database cluster, fetch latest backup using `wal-g backup-fetch` and create a `recovery.conf` file in the new cluster. `recovery.conf` is used by Postgres during first start to get the `restore_command`. Again the exact commands are to be found in `postgres/recovery/restore_command.sh`. During first start Postgres will get WALs from backup server and restore the database.

18.2.1 Configuration

The same configuration as for the backup setup is requiered. Except the Service Account Key has to be stored at `postgres/recovery/serviceAccountKey`.