hr wallingford

Enhancing multi-scalar mapping and research of food security risk, due to climate change in Jamaica

Jamaica Food Security & Climate Resilience Portal: Technical Manual

FWR7056-RT005 R01-00
13 August 2025

Document information

Document permissions	Confidential - client
Project number	FWR7056
Project name	Enhancing multi-scalar mapping and research of food security risk, due to climate change in Jamaica
Report title	Jamaica Food Security & Climate Resilience Portal: Technical Manual
Report number	RT005
Release number	01-00
Report date	13 August 2025
Client	Rural Agricultural Development Authority
Client representative	-
Project manager	Dr Gina Tsarouchi
Project director	Nigel Walmsley

Document history

Date	Release	Prepared	Approved	Authorised	Notes
13 Aug 2025	01-00	MCP	GMT	MCP	Initial Draft

Document authorisation

Prepared	Approved	Authorised
Michael Panzeri	Gina Tsarouchi	Michael Panzeri

Contents

Figures

1 Introduction

1.1 Overview

The Jamaica Food Security & Climate Resilience Portal was implemented by the Rural Agricultural Development Authority (RADA), with support from the United Nations Climate Technology Centre & Network (CTCN). The portal was designed and implemented by HR Wallingford during the period 2024–2025.

This web-based information portal integrates agricultural, socio-economic, and climate data to support national food security and climate resilience planning. It operates as an interactive map viewer that enables users to visualise spatial data related to agriculture and socio-economic conditions, while also serving as a community-based reporting interface that facilitates the collection and sharing of localised information. Additionally, the portal includes a risk simulation tool designed to assess crop suitability under both current and projected climate scenarios, thereby supporting strategic planning and adaptive decision-making.

This technical reference manual is intended for RADA personnel responsible for the administration, maintenance, and operational oversight of the portal.

1.2 Purpose of the Jamaica Food Security & Climate Resilience Portal

The Jamaica Food Security & Climate Resilience Portal is a strategic tool in Jamaica's efforts towards agricultural resilience to climate change. It serves as a centralised repository and dissemination platform for data related to agriculture, food security, and climate risk.

By centralising this vital information, the portal empowers evidence-based decision-making to address climate change challenges, through planning, preparedness and emergency response activities.

The Jamaica Food Security & Climate Resilience Portal is a web-based system, hosted on an Amazon Web Services (AWS), ensuring accessibility and data security. The RADA IT Department will manage the hosting, security and updates of the system and user management. The user management tasks undertaken by RADA includes granting access to individual users from their own organisations, granting access to other organisations and creating Superusers for those other organisations.

1.3 Pre-requisites

The system is a cloud hosted website, therefore in order to access and use the Jamaica Food Security & Climate Resilience Portal, it is necessary to have a computer or mobile device with a web browser that has access to the internet.

Users should have a basic understanding of web-based platforms and familiarity with interpreting spatial data to effectively navigate the system.

To use and manage the Jamaica Food Security & Climate Resilience Portal, users must have authorised access granted by an "Administrator" or "Superuser".

For "System Administrators" (IT Department specialists who are responsible for keeping the service live and updated), expertise in managing web servers, handling Docker containers, and performing database administration are essential capabilities for maintaining and troubleshooting the platform.

1.4 Important notes on the Jamaica Food Security & Climate Resilience Portal

The Jamaica Food Security & Climate Resilience Portal has been designed with the following caveats in mind:

1. Any data in the repository is designed to be made available to other users of the system, potentially including the general public. As such, it should not be the main store of the data, but a place to share certain data with other departments and agencies
2. Consideration should be given to the level of information that is shared, so that no personal or sensitive data is accessible to members of the public
3. The repository is not designed to replace existing data storage that takes place by data owners and their organisations

2 System Architecture

2.1 High-level architecture

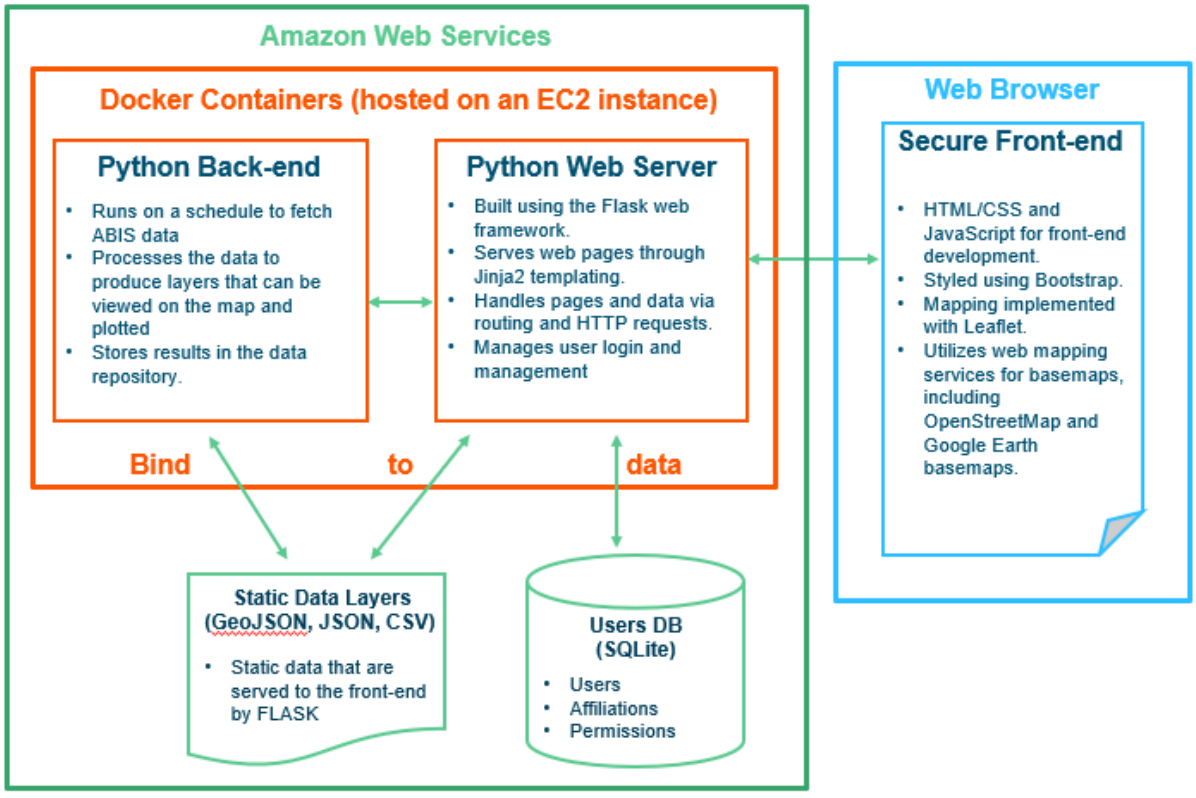


Figure 2.1: Schematic Overview of the components of the Web-app. Please note that some simplifications have been made for clarity

Source: HR Wallingford

The high-level architecture diagram (Figure 2.1) provides an overview of the web application's key components and their interactions. It includes the following elements:

- **Python Back-end**: Manages the logic for fetching the Agricultural Business Information System (ABIS) (refer to Section 4.2) data for the current month and loading it into the Jamaica Food Security & Climate Resilience Portal's database

- **Web Server**: Built using Flask, it serves web pages through Jinja2 templating and handles routing and HTTP requests
- **Secure Front-end**: Developed with HTML/CSS and JavaScript, styled using Bootstrap, and incorporates mapping with Leaflet. It utilises web mapping services for basemaps, including OpenStreetMap and Google Earth basemaps
- **Public Front-end**: Developed with HTML/CSS and JavaScript, styled using Bootstrap
- **Static Data Layers:** GeoJSON formatted vector data that provide contextual map layers in the Secure Front-end
- **Users DB:** SQLite database of user information that contains usernames, affiliations, access levels and salted and hashed passwords

3 Jamaica Food Security & Climate Resilience Portal Front-End

3.1 User Interface

When a user opens the Jamaica Food Security & Climate Resilience Portal, an interactive map opens with a sidebar on the right hand side for interacting with map layers. The default first page is the “Welcome” page to provide the users with some context for the portal. An example screenshot of the Jamaica Food Security & Climate Resilience Portal is shown in Figure 3.1.

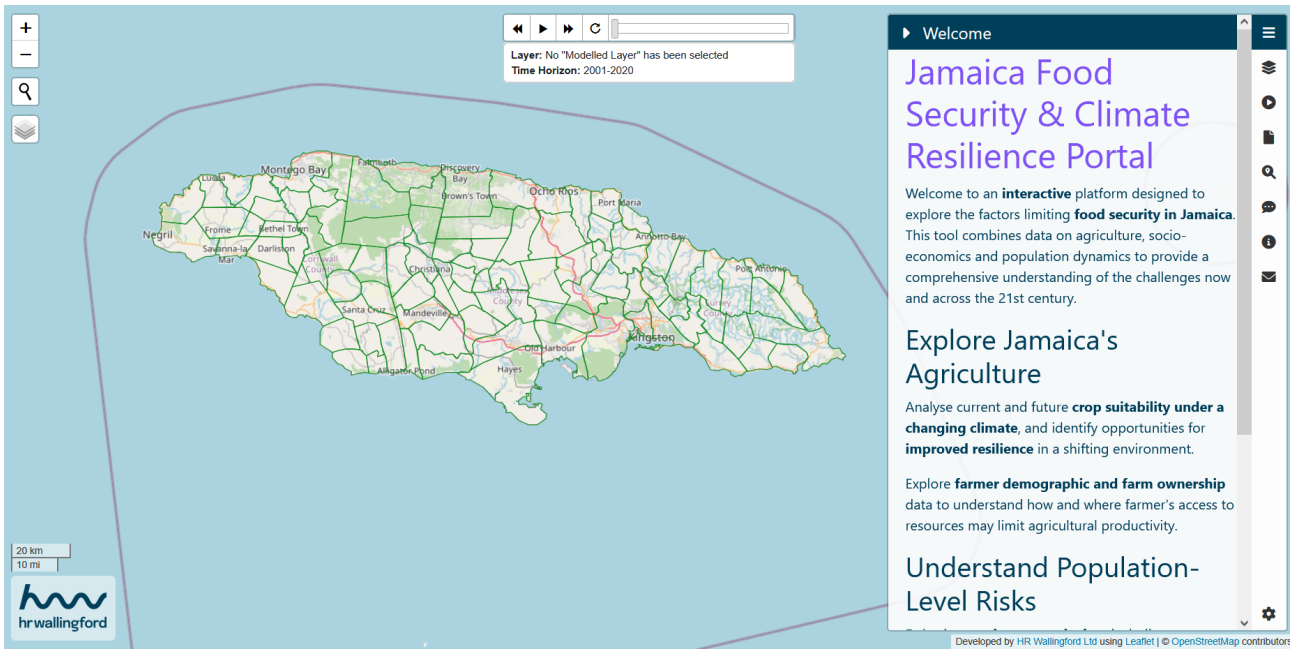


Figure 3.1: Example view of the Jamaica Food Security & Climate Resilience Portal User Interface

Source: HR Wallingford

The user interface (UI) is designed to be intuitive and user-friendly. It leverages Jinja2 templating within the Flask framework to dynamically generate the web pages.

More details on the user interface can be found in the Jamaica Food Security & Climate Resilience Portal User Manual, (HR Wallingford, 2025).

3.2 Technologies Used

The front-end of the web application is developed using a combination of HTML, CSS, and JavaScript.

The front-end of the web application employs several open-source libraries to provide various functional aspects:

- Leaflet.js: Utilised for the mapping components, enabling interactive and customizable maps
- Leaflet-Geotiff.js: A plugin for Leaflet.js. Used to display GeoTIFF model outputs directly on the map, enhancing the visualisation of spatial data
- Bootstrap.js: Employed for webpage layout and responsiveness, ensuring the site adapts to different browser sizes. Note that the page is primarily designed for use on PC-sized screens, though has been tested on mobile devices. Bootstrap is also used for certain styling aspects
- Plotly.js: used to create interactive graphs of climate data

3.3 Map Integrations

Mapping functionality is implemented using the Leaflet library, which provides an interactive and customisable map interface. The application integrates various web mapping services for basemaps, including OpenStreetMap, Google Earth and ArcGIS online satellite basemaps, to offer users detailed and accurate geographical information.

HR Wallingford have developed a system for configuring many of the system customisations using a JSON formatted config file. The `config.json` file is included in the source code at "`skn_crid/static/data/config.json`". Some of the characteristics of the system that can be configured using this file are described below. For certain of the customisations, additional data or code is required that the `config.json` file references; these are also described below.

3.3.1 Changing Initial Location of the Map

Site Administrators at the IT Department can change the initial location of the map by modifying the `initial_position` object in `config.json` file.

Figure 3.2 shows an example of the config file. "`lat`" and "`lon`" are the latitude & longitude of the initial map position, "`zoom`" is an integer where 0 is the whole world, and zoom level doubles each time the zoom level is increased by one value:

```
"initial_position": {
        "lat": 17.27,
        "lon": -62.70,
        "zoom": 11
    },
```

Figure 3.2: Code Snippet to change map's initial location

3.3.2 Adding and Changing Base Layers

Site Administrators at the IT Department can add or change base layers by modifying the "`baseLayerReference`" object in the `config.js` file.

The `config.js` file is included in the source code at "`/skn_crid/static/components/config.js`"

An example of a new basemap layer (in this case OpenStreetMap) is included(Figure 3.3):

```
baseLayerReference = {
    "OpenStreetMap": {
        "url": 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
        "maxZoom": 18,
        "attribution": '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap<
        "id": 'osm',
    },
}
```

Figure 3.3: Code Snippet to update/modify the base layers in config.js

After specifying the new layer, it also needs to be added to the `base_layer` array in `config.json` as seen in Figure 3.4.

```
"base_layer": [
        "OpenStreetMap",
        "GoogleEarth",
        "ArcGISOnline"
    ],
```

Figure 3.4: Code Snippet to add the modified or new base layer to the map using config.json (**This references the base layer in config.js**)

This allows users to easily customise the map's contextual basemapping appearance by selecting different base layers according to their preferences. For more options and the required JavaScript code for different map types, users can refer to:

https://leaflet-extras.github.io/leaflet-providers/preview/index.html

This link was available at the time of writing this document and provides a comprehensive list of available basemaps and their configurations.

3.3.3 Updating a static map layer

The map contains several static data layers that provide very useful contextual information to the users. These are:

- Parish boundaries
- Parish extensions boundaries
- Land-use
- Socio-economic datasets (income, povery, food poverty)
- Population and population density
- Parish extension officer survey results
- Farmer demographics (ABIS*)
- Farm tenure and ownership status (ABIS*)
- Crop cultivation (ABIS*)
- Livestock count (ABIS*)

**Note that the layers from the ABIS database are updated automatically using the python back-end script. Refer to Section 4.2 for further information.*

These layers are loaded into the map from GeoJSON files that are saved in the "`/home/fwr7081/app_files/layers/`" on the server. It is possible to update these files by taking a copy from the server, loading the copy into a desktop GIS (such as QGIS), making edits to the vector features, then replacing the live data with the updated copy of the data. Note that it is wise to take a backup of any data before replacing the live files.

3.3.4 Adding a new or changing an existing static layer

System Administrators can add or change contextual GIS layers (eg drainage network) by modifying the config.json file.

To add a static layer to the map:

1. Prepare the GIS data in GeoJSON format, and store it in the "`/home/fwr7081/app_files/data/`" folder
2. To use complex styles, create an SLD file for the layer (this can be done in desktop GIS software, such as QGIS and the specification of the SLD files is provided at https://docs.geoserver.org/main/en/user/styling/sld/reference/index.html). The SLD file should be exported into the "`/home/fwr7081/app_files/styles/`" folder. Otherwise, a simple style can be defined in the config (eg Figure 3.5)
3. Create a new layer entry in the "`config.json`" file (eg Figure 3.6). This is located in the folder "`/home/fwr7081/app_files/data/`"

An example layer entry in `config.json` is shown in Figure 3.6, several of the fields require specific values to be entered:

```
"style": {
    "radius": 7,
    "fillColor": "#ffffff",
    "color": "#000000",
    "weight": 1,
    "opacity": 0.8,
    "fillOpacity": 0.2
}
```

Figure 3.5: Example of a "simple" style for a contextual layer

```
{
    "id": "population",
    "z_index": 200,
    "legend_group": "contextual_data",
    "display_name": "Population Density",
    "description": "Approximate population values.",
    "layer_type": "geojson",
    "data_src": "/static/data/population.geojson",
    "sld_src": "/static/styles/population.sld",
    "feature_type": "Polygon",
    "visible_on_load": false,
    "click_event": "popup",
    "popup_format": "{{popup_value}} people/km"
},
```

Figure 3.6: Example config entry for a contextual layer

- The value entered in the "`id`" field should be unique among all layers in the configuration file
- The "`z_index`" entry should be a value between 200 and 300. Layers with a higher value are displayed above those with a lower value on the map

- "`data_src`" should reference the GeoJSON and "`sld_src`" the SLD file created above. If you are using a "simple" style, do not add an entry for "sld_src"
- "`feature_type`" should be one of "`Point`", "`Polyline`" or "`Polygon`". Layers with more than one feature type are not supported
- "`popup_format`" is string that defines the html format of the feature on-click popups. Field names are given using {{curly braces}} and the value of the field will be substituted into the popup at runtime. For example if the layer has a "`site_name`" field, an entry of "`Name: {{site_name}}`" would display the feature's site name when the feature is clicked

More details on how to configure map layer are given in the markdown file `Configuring_the_map.md`

There are notes to consider when adding static layers to the map:

- It is **strongly** recommended that a backup of the original `config.json` file is taken before making modifications
- File size should be considered when adding a new layer to the map. While the app is capable of serving large GIS files, including multiple large map layers will result in slower loading of pages, and increased hosting costs
- Only the GeoJSON format is supported. If the data are in another GIS format (eg ESRI ShapeFile), it will be necessary to convert them to GeoJSON. Most desktop GIS software (eg QGIS) packages are capable of making this conversion. Note that GeoJSON is a verbose text based format and files sizes can be large. It is advisable to simplify complex vector geometries before conversion, using for example, Douglas Peucker generalisation, to save the vertices in WGS84 coordinate reference system and to save vertices with no more than 5 decimal places in the lat and lon coordinates. `Pretty Print` formatting is not recommended
- When choosing a style for the layer, it is recommended to use high levels of transparency (particularly for polygons), to avoid obscuring other layers

3.4 User Interactions

The map layers (even those that are not initially visible) are loaded onto the Front-end when the site loads. User interactions are handled through a combination of routing and HTTP requests managed by the Flask back-end. The application supports various user actions such as submitting, updating and archiving hazard data, and loading specified results from Climate and Agriculture Simulator. UI Map interactions are described below.

Map Interactions:

- **Pan and Zoom**: Users can navigate the map by panning and zooming to explore different areas. The background mapping tiles and Geotiff layer data are requested from the web server whenever a user pans or zooms. The resolution of these layers change on zoom; more details are visible in the map when the user zooms in
- **Layer Control**: Users can toggle different basemap layers, such as switching between OpenStreetMap and other available maps. Users can also toggle on and off any of the different map layers that are listed in the Layers tab of the sidebar by checking and unchecking the layer's visibility checkbox. Checking the box to make a layer visible will expand a key for the layer
- **Markers and Popups**: Interactive markers and popups provide additional information about specific locations on the map

4 Participatory GIS

Participatory GIS (PGIS) is about empowering communities to draw their own maps. It is a concept that helps with knowledge sharing and planning.

Within the Jamaica Food Security & Climate Resilience Portal, the PGIS feature allows users to log a hazard related to agricultural activities. This can be completed by any person with user access.

Users can add a hazard to the map by clicking the button on the 'Upload Hazard' tab of the sidebar. The user provides the hazard category (from a dropdown list), the location of the hazard, a short description and its current status. This allows all users to see the spatial distribution of hazards across Jamaica, monitor and track hazards as logged by local users and identify areas in need of interventions.

Upon submission, the new hazard is added to the one of the two map hazard layers:

- If the 'Active Hazard' checkbox was selected, the hazard is loaded into the Live hazards layer
- If the 'Active Hazard' checkbox was not selected, the hazard is loaded into the Historic hazards layer

Users can click an existing hazard on the map and can use the form on the sidebar to update information about the selected hazard including to change its Active status (which switches the hazard between the live and historic map layers). Every change that is made is recorded within the PGIS data model, enabling a full audit trail of changes to be reviewed by an IT Administrator using a SQLite Database browser such as the Open Source software DB Browser for SQLite https://sqlitebrowser.org/.

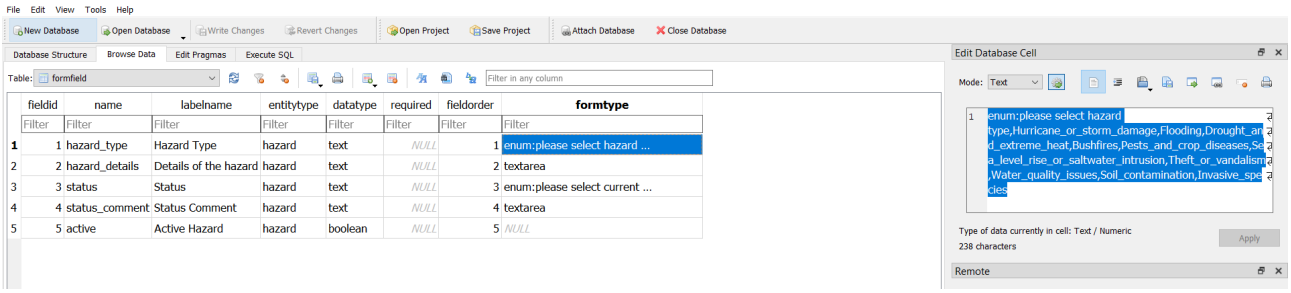
The PGIS database has been implemented in the form of a GeoPackage (a spatially aware form of SQLite) and has been applied to the Jamaica Food Security & Climate Resilience Portal where JS code has been written to implement the functionality on the client side and python on the server side.

4.1 Summary of how the PGIS functionality works

The following points refer to the structure of the PGIS database and how it is used to give an administrator friendly means of updating the fields that are available to users on the sidebar form for the hazards:

1. 'entity_type' is used to define the type of a map feature, such as a hazard. Within the PGIS it is conceptually used to define how we lay out an feature's comments form, ie the fields, the field types and their order, for example, using the 'form_field' table
2. 'form_field' gives a set of rows that that define the fields that are loaded into the comments tab of the side panel (see Figure 4.1)
3. When the user clicks on a hazard (or they open the form to create a new one), the data from 'form_field' are used to build the form (using a third party Open Source JS library called jsonforms). The form is dynamically built using the rows for the hazard 'entity_type_id' (see Figure 4.2) and so it is possible to add more fields to the form and to change or remove existing ones. 'form_field' also contains information about the options in select lists; these are defined in the database as "enum" types followed by the list of options as comma separated strings with no spaces. (see Figure 4.1)
4. When a hazard is submitted, it is posted into the table comment. The Primary Key for the comment is 'comment_id' but 'entity_type_id' and 'entity_id' must be unique since these are used as Foreign Keys to relate the comment to a single Hazard Point on the map and they are Primary Keys for the Hazard Point. The comment itself (ie the data from the form) are stored as a json formatted text string in the field 'entity_data' in the 'comment' table. All other fields provide information about the comment such as date it was submitted, by whom and so on

5. The location of the hazard are stored in the table 'geometry_geojson' with the geometry itself stored in the field 'geojson' as a string formatted geojson object
6. Selecting a hazard feature in the map populates the form with the values for the selected feature
7. When another comment is submitted on the same feature the original comment is loaded into the 'comment_history' table and the new comment is loaded into the 'comment' table which now has a value added to the 'modified_on' field. (the 'created_on' remains the date of the first comment). This facilitates the loading of the most recent comment via the comment table but allows an audit trail by searching all of the 'comment_history' rows for a hazard



fieldid	name	labelname	entitytype	datatype	required	fieldorder	formtype
1	hazard_type	Hazard Type	hazard	text	NULL	1	enum:please select hazard ...
2	hazard_details	Details of the hazard	hazard	text	NULL	2	textarea
3	status	Status	hazard	text	NULL	3	enum:please select current ...
4	status_comment	Status Comment	hazard	text	NULL	4	textarea
5	active	Active Hazard	hazard	boolean	NULL	5	NULL

Figure 4.1: Screenshot of the participatory GIS .gpkg file viewed in DB browser. The *formfield* and hazard_type are highlighted

Source: HR Wallingford

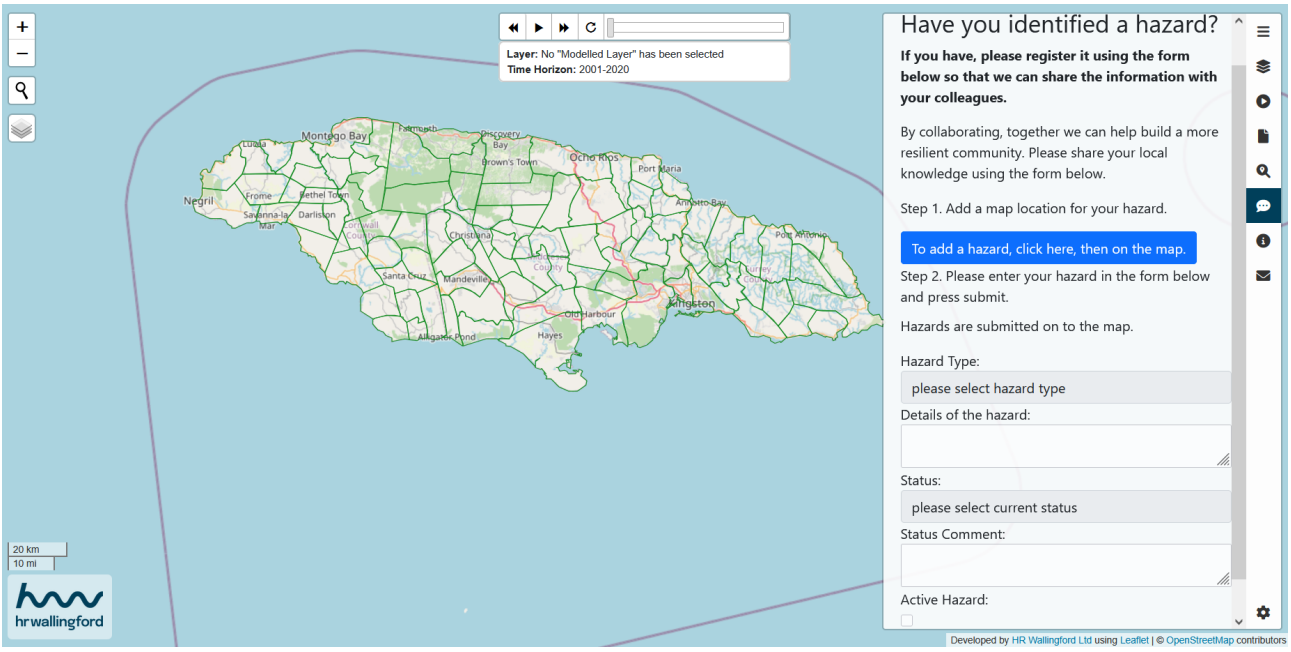


Figure 4.2: Screenshot from the "Upload Hazard" feature within the Jamaica Food Security & Climate Resilience Portal

Source: HR Wallingford

4.2 Archiving hazards

Administrators and Superuser are able to archive hazards. When a user with one of these permission levels selects a hazard, the data for the selected hazard is loaded onto the Hazards sidebar, and they will see that there is an additional section at the bottom of the panel with a button as shown in Figure 4.3.

Clicking on the button will archive the selected hazard point, which means it is no longer loaded into the map as a live or historic hazard. The hazard is not deleted from the database, it is flagged as being 'archived' and it is excluded from the data when hazards are requested from the web portals back-end. Archiving rather than deleting enables IT administrators to reinstate hazards that were archived incorrectly, it enables a full audit trail to be undertaken and it also enables analysis of trends in all reported hazards to be undertaken, including those that have since been archived.

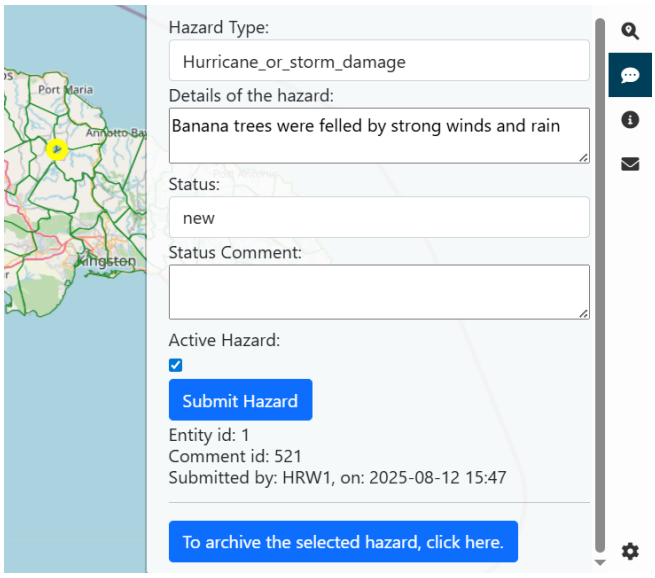


Figure 4.3: Button to allow Superuser and Administrator level users to archive a selected hazard

Source: HR Wallingford

5 Agricultural Business Information System Link

5.1 ABIS API link overview

The Jamaica Food Security & Climate Resilience Portal automatically pulls data from the ABIS database. As a result, there is no need for manual data entry within the portal itself. Data is synchronised monthly, reflecting the status of ABIS at the time of extraction. This process depends on the timely and accurate updating of ABIS. Importantly, no personal information about individual farms or farmers is shared on the portal.

Please note: The update process is automatically run on a monthly schedule, but for resilience reasons, the updated data do not overwrite the live data on the Jamaica Food Security & Climate Resilience Portal. To prevent any unforeseen error in the update process from taking down the live site, it is necessary for an IT administrator to check the updated data and then copy the files to the live data folder that serves the web portal.

The portal captures and displays data across the following categories:

- Farmer Demographics:
  - Number of farmers per parish extension, disaggregated by:
    - Gender

  - Age group (youth farmers (18-35 years), farmers over 35 years)
- Farm Tenure:
  - Number of farms per parish extension, with tenure categorised as:
    - Leased
    - Owned (Registered Title/Other)
    - Rent Free (Family Land/Other)
    - Rented
    - Squatting - Government Land
    - Squatting - Private Land
- Farm Ownership:
  - Number of farms per parish extension, categorised as owned by:
    - Individual
    - Government
    - Partnership
    - Private Company
    - Co-operative
    - Other
    - Not Stated
- Farmland Irrigation:
  - Proportion (%) of irrigated farmland per parish extension, based on:
    - Total farm area (ha)
    - Irrigation status

The downloaded data is reformatted for the app, allowing the user interface to display monthly data – as shown in Figure 5.1. Clicking on an area of interest will show this data over historic period (see Figure 5.2).

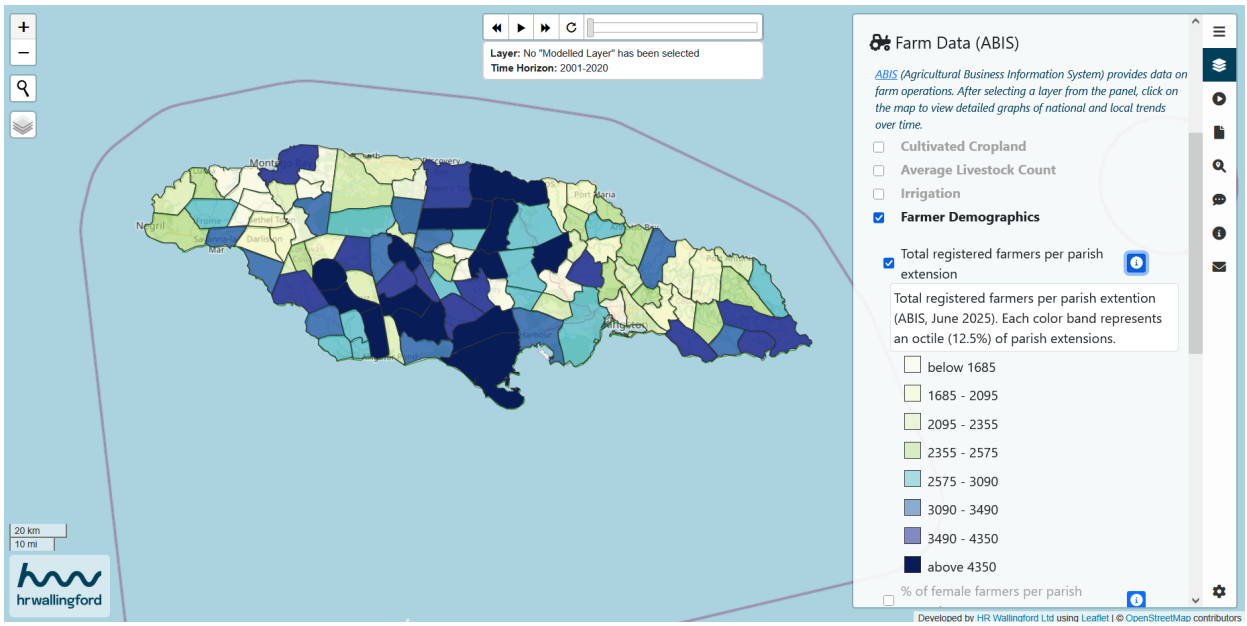


Figure 5.1: Screenshot from the Jamaica Food Security & Climate Resilience Portal highlighting the number of farmers per parish extension in June 2025

Source: HR Wallingford, data from ABIS

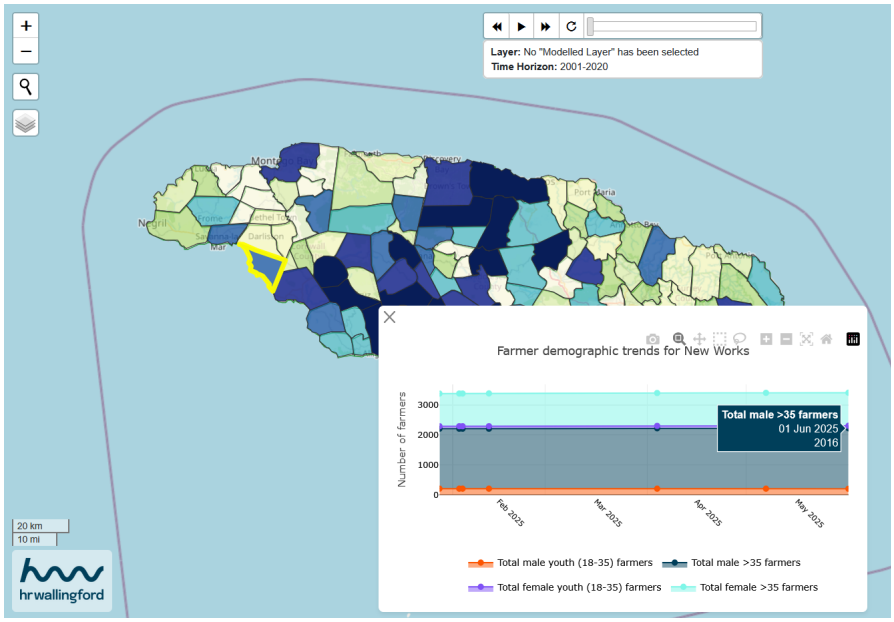


Figure 5.2: Screenshot from the Jamaica Food Security & Climate Resilience Portal highlighting the number of farmers by age and gender in a specific parish extension over time

Source: HR Wallingford, data from ABIS

5.2 How it works

The script begins by downloading the raw data outlined in Section 5.1. This data is then processed and converted into the file formats that are used by the web portal and they are saved in the app_data/pipelines subfolder where they can be manually checked. When the IT Administrator is content with the new files, they can be copied directly to the corresponding app_data folders (ie one level up from the pipelines folder) from where they serve the live web portal.

Many of the static map data layers are updated via new geojson files.

The config.json file is updated to reflect the current dataset date (eg "June 2025"), as illustrated in Figure 5.1.

Data are appended to the csv data files that underpin the chart functionality of the portal.

For the farmer demographics layers, the corresponding SLDs (Styled Layer Descriptor) are updated to maintain a balanced visual distribution across colour bands. This adjustment ensures that as the national number of farmers changes over time, the map continues to accurately represent relative population across Jamaica. Without this update, an increase in farmer numbers could distort the visual clarity of the discrete colour bands.

At the time of writing, the API link access data from the public URL: https://abis.gov.jm/. Should the URL of the portal change, or the data structure of data stored on the portal change, this may cause errors to the API script. This should be resolved by debugging the Python scripts. Effort has been made to ensure the script is well commented so that debugging should be simple. HR Walingford are not responsible for future changes to the API link which may impact the code functionality.

6 Climate & Agricultural Simulator

6.1 Overview

The Climate & Agriculture Simulator tool is a component of the Jamaica Food Security & Climate Resilience Portal. It aims to provide a high-level risk indicator for agricultural suitability at the parish extension scale, under both current and future climate conditions.

Its primary purpose is to support strategic decision-making by informing long-term planning and interventions related to the impacts of climate change on agricultural productivity and national food security. It is not intended for guiding short-term decisions such as seasonal planting or immediate agricultural planning.

Please refer to the User Manual for more information of the methodology of the Climate & Agricultural Simulator.

6.2 How it works

Users can explore different scenarios by selecting from the following options:

- **Climate Change Pathways** (based on Shared Socio-economic Pathways)
- **Cropping Patterns** (distribution of cultivated crops across Jamaica)
- **Interventions** (eg irrigation)

After selecting the desired parameters and clicking “Run Simulation”, the tool loads pre-processed datasets. These data sets are GeoJSONs saved within “static/data/agri-sim”. Each of the datasets is saved in a subfolder according to its cropping pattern and intervention scenario, with the file named according to its time stamp and climate change scenario (eg static\data\agriculture\agri_sim\cp_BAU\irr_0\historic\Jamaica_parish_ext_crop_suitability_score_2001-2020.geojson).

A time slider at the top of the map interface allows users to explore projected changes in agricultural risk across 20-year intervals throughout the 21st century. This works by loading different GeoJSONs according to their time stamp in the filename.

Clicking on a parish extension reveals a detailed graph comparing climate projections and associated uncertainties (based on the 25th–75th percentile range of the climate model ensemble for future climate).

Additionally, the tool displays TIFF layers showing mean monthly rainfall and temperature for each season for the selected scenarios. These layers are also linked to the time slider, enabling users to visualize how climate variables evolve over time. These data sets are in geotiff format and are saved within “static/data/climate”.

7 Website Back-End

7.1 Technologies Used

The back-end of the web application is built using robust and scalable technologies to ensure efficient processing and data management. Key technologies include:

- **Python**: Utilised for its versatility and ease of integration with various libraries and frameworks
- **Flask**: A lightweight web framework used to handle routing, HTTP requests, and server-side logic. It follows the Web Server Gateway Interface (WSGI) standard and can be hosted using any WSGI server. In the provided configuration, the Waitress WSGI server is used

- **SQLite**: A lightweight, file-based relational database system used for storing and managing data. It is integrated with Flask to provide a simple and efficient database solution for development and small-scale applications. SQLite was selected as a cost-effective and robust database as it does not require an additional database server

7.2 Security

Security is a key component of any ongoing software, particularly software open to the internet, such as a web server. Key security features of the application, and considerations while maintaining the application include:

- **Regular Updates and Patches**: The application is provided with up-to-date versions of the libraries required. Part of ongoing maintenance should include regular updates, particularly of any components exposed to the internet (eg the web server)
- **Login System**: The app uses a robust login system to ensure that only authenticated users can access the data housed in the application. Login is facilitated with a username and password. This is implemented using the Flask-Login extension for Flask. The users database contains the users credentials including their salted and hashed passwords
- **Local Hosting of Assets**: All assets, including JavaScript libraries, are hosted locally. This reduces exposure to external risks, enhances data privacy, and ensures full control over content delivery
- **CSRF Protection**: The Jamaica Food Security & Climate Resilience Portal uses Cross-Site Request Forgery (CSRF) protection to protect relevant endpoints (particularly user management) from unauthorised access. This may occasionally cause errors when using the app – particularly if a page has been left open for a long time, as a CSRF token is created on page load, and will eventually expire. If these errors occur, then it is possible to simply refresh the page and try again

7.3 Python Back-end

The Python back-end is being run within a Docker Containers which served the website pages and data.

7.3.1 Web Server

The web server uses the Python web framework Flask to serve web pages and handle HTTP requests for data. This framework follows the Web Server Gateway Interface (WSGI) standard and can be hosted using any WSGI server. The provided deployment uses the Waitress server for WSGI hosting, with an Apache web server as an intermediate proxy.

The app makes use of the Jinja2 templating engine to serve standard content and generate dynamic content based on the database and files produced by the Python back-end. The files served by the Flask app are pre-generated and optimised by the Python back-end. As long as the app has access to the directory of data (or a copy of the data), new instances of the app may be created as required.

The Flask app accepts a large number of configuration parameters. These can be set as environment variables, or specified in a .env file in the root directory of the source code (with environment variables taking precedence over the .env file. The main parameters that may be relevant for deploying the app are:

- `SECRET_KEY` – This is a key used internally by Flask to ensure secure operation and transfer of data. It is set in the deployed version of the application, but will need to be set if a new deployment is created. It should be set to a consistent, secure value for each deployment of the app

- `DATA_DIR` - File path of the directory of data produced by the Python back-end (or a copy of it). Note that the app is designed to be deployed inside Docker, so the volume should be bound to the container, and the bound path specified here
- `LOG_LEVEL` - The verbosity of logs produced by the app. Should be one of "DEBUG" (default), "INFO", "WARNING", or "ERROR"

8 Database Management

8.1 Technologies Used

The system uses SQLite for its database needs, providing a lightweight and efficient solution for data storage and retrieval.

9 Deployment

9.1 Environment Setup

To give an added level of security and resilience, and to facilitate administration, relocation of hosting and software updates, the web server and the update pipelines are run within Docker Containers. These package the software and dependencies so that they are run within an encapsulated environment on the host machine.

9.1.1 Starting and Stopping the Docker Containers

The Docker Containers are run from Docker Images. A container can be stopped and very easily started again via the image. The commands to use are:

- To stop a Docker Container: docker compose down
- To Start a Docker Container: docker compose up -d

The latter command will start a container from its image.

9.1.2 Rebuilding and editing the Docker Container

`docker-compose.yml` configures the application's services and performs the creation and start-up process. If the application is deployed onto a new machine, the application files or the Python environment are changed, or if the `dockerfile` is changed, it is necessary to rebuild the Container. The command to build a Docker Container is:

- to build the Docker Image: `docker compose build`

If the Container is already running, it is necessary to stop it, then rebuild the Image and then to restart the Container.

The `docker-compose.yml` builds a Docker Container from a set of instructions. This includes an image which is specified by a `dockerfile`. The `dockerfile` contains instructions to build the python environment that is used by the app, the environment is specified by a Python `requirements.txt` file. If the container's image or the Python environment requires changing or updating, it is necessary to update the requirements.txt file and to rebuild the Container.

The `dockerfile` contains instructions to copy the application files into the container. Therefore it is also necessary to rebuild the container if any of the application files are edited.

Within `docker-compose.yml`, permanent storage bindings are defined. Whenever the Docker Container is built, the application files specified in the `dockerfile` are copied from the source into the new container, consequently, any changes to files that occur within the Container since

the previous build are lost. It is important therefore for permanent data storage, such as user management databases, live data updates, updates to the config file and styles, and uploads to the hazards layer or other enduring data created by the web app at run-time to be bound directly from storage via the `docker-compose.yml` to avoid these data being overwritten whenever the image is rebuilt. In the case of the Jamaica Food Security & Climate Resilience Portal, the folder containing the database of climate data, the styles, the users database and the hazards database are all bound directly from disk.

9.2 Hosting

The website is hosted on Amazon Web Services. There are two cloud servers as follows:

9.2.1 Web server and backend process server

Web server and backend process server are run within two different Docker Containers on the same AWS instance. The instance is an AWS Elastic Compute 2 (EC2) burstable general-purpose instance type t3.medium[1].

9.2.2 Elastic Block Store

Amazon Elastic Block Store (EBS) is used to provide continuous file storage for the web server and the ABIS update pipeline. This ensures that when an instance is shut down (eg for updates), the data store for those instances is preserved and continues to be accessible.

9.2.3 Website certification and web application firewall

The website uses secure sockets layer (SSL) certification and a web application firewall helps protect the web server by filtering and monitoring HTTP traffic between the web application and the Internet.

9.2.4 Accessing the portal

The Jamaica Food Security & Climate Resilience Portal is accessed by logging in and following the link from the public facing Jamaica Food Security & Climate Resilience Portal page at the URL: https://food-security-jamaica.hrwallingford.com.

9.3 User Management

The system allows for selected users to carry out key user management functionality in the web portal. This can be managed through the “Manage Users” button on most pages of the portal.

9.3.1 Introduction

A user must login before they can access the system.

Users credentials, (username, hashed and salted password, affiliation and access level) are stored within a users’ database on the web server. Users belong to an affiliation (a loose term to represent Ministry, Department, Organisation or any other appropriate grouping of users). Users can be given one of three different access levels:

- Administrators - Highest level of access:
  - Can create new affiliations (organisations) and add, manage and deactivate users in them
  - Management capabilities are extended to affiliations they have created

[1] https://aws.amazon.com/ec2/instance-types/t3/

- Superusers – Medium level of access:
  - Can add, manage and deactivate users within their affiliation, but not create new affiliations
  - Management capabilities are extended to affiliations an administrator in their affiliation has created
- User - Standard level of access:
  - Has no management capabilities

9.3.2 Managing own account

Users can manage their own account – specifically changing their password on the "User Settings" page. This can be accessed from the "User Settings" link in the settings panel of the sidebar when viewing the Jamaica Food Security & Climate Resilience Portal.

This page also displays the current user's username and affiliation. An example of this is shown in Figure 9.1:

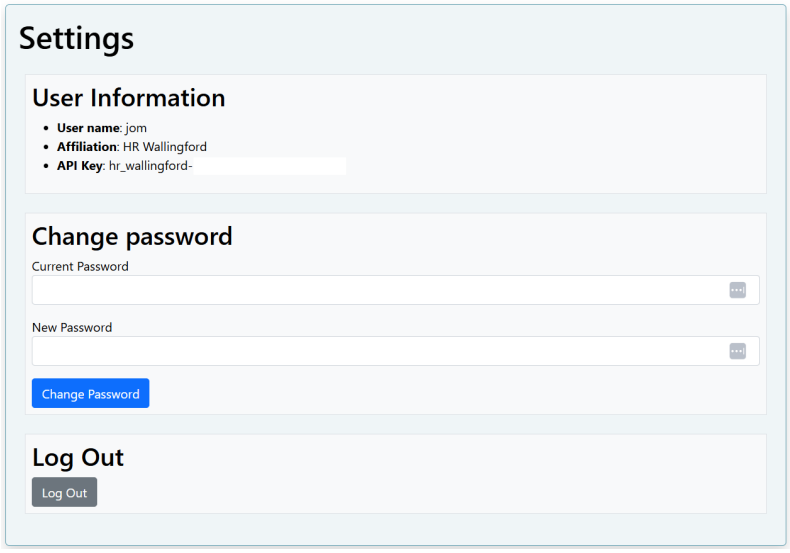


Figure 9.1: User settings page

9.3.3 Account Management

The Account Management page (Figure 9.2) of the Jamaica Food Security & Climate Resilience Portal can be accessed from the "User Management" link in the settings panel of the sidebar when viewing the portal. This page is only available for Superusers and Administrators. There are several different affiliation and user management functions that can be undertaken using this page.

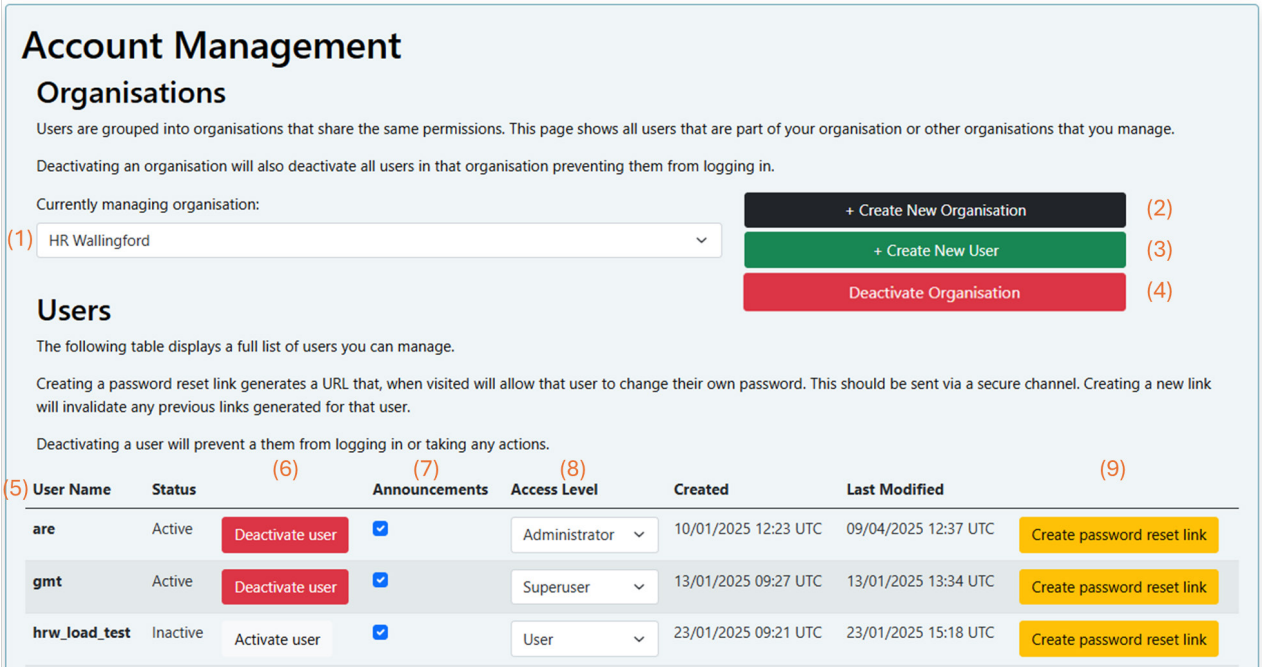


Figure 9.2: Account Management page

1. Select Organisation

The management page shows the users for a single organisation (affiliation) at a time. The user can select the organisation currently being shown with this drop-down menu (select list (1) in Figure 9.2). When new users are created, they are added to the selected organisation.

2. Create New Organisation

Available to Administrators only, the Account Management page shows the organisation that the presently logged in user is currently managing in a drop down menu.

If the user would like to create a new organisation, a button is available on this page (button (2) Figure 9.2) The user will be prompted to insert an organisation name, before being redirected back to the account management page, with the new organisation selected.

3. Create New User

After creating an organisation, users can be added to it (button (3) in Figure 9.2). Superusers and Administrators can add new users to an organisation. The user will be added to the organisation displayed in select list (1) in Figure 9.2.

Selecting the button “Create New User” brings up a form (Figure 9.3) where a new username can be entered for the new user and the users Access Level for the new user can be set. At the top of the Create User form there is a message that shows which affiliation the user will be added to.

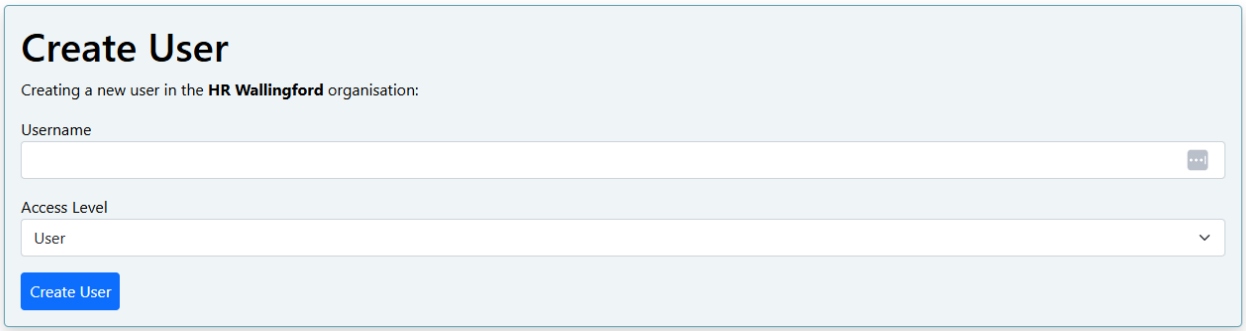


Figure 9.3: Create User form

Source: HR Wallingford Ltd

After creating a new user, a single-use link will be generated for that user to set their password. This should be sent to the user through a trusted channel (eg by email). Until the user has used this link to set their password, they will be unable to log in. This link will need to be used within 24 hours, or it will expire for security reasons. If the link has expired before the user has activated it, generate a new one with button (9) in Figure 9.2.

Note that users with management permissions have permission to create new users within their organisation, so this role should only be given to trusted users. Allowing a user to have management permissions may be useful to delegate management functions rather than relying on a single administrator.

4. Deactivate Organisation

For Administrators only, ((4) in Figure 9.2) is available to deactivate an organisation. While deactivated, no member of the organisation will be able to log in or use the portal. The organisation can be reactivated at any time.

5. User Table

This table ((5) in Figure 9.2) shows a list of all users in the selected organisation. Near the top of the Account Management form there is a select box for choosing which organisation's users are to be managed.

6. Deactivate User

To remove a user's access to the system, deactivate the user by clicking the "Deactivate User" button on their row of the users table.(button (6) in Figure 9.2). The user can be reactivated at any time.

7. Set Public Announcements permission.

Give a user permission to create and manage Public Announcements using the tick box (7) in Figure 9.2.

8. Change a user's Access Level

A user's Access Level can be changed by selecting a different access level using the select list for that user (select list (8) in Figure 9.2). Permissions can be set equal to or lower than the currently logged in user. ie an Administrator can set a user's Access Level to either Administrator, Superuser or User and a Superuser can set a user's Access Level to either Superuser or User but not Administrator.

9. Create a password reset link

A password reset link can be created by clicking on the "Create a password reset link" button (button (9) in Figure 9.2). This should be sent to the user through a trusted channel (eg by email).

This link needs to be used by the user within 24 hours or it will expire for security reasons. If the link has expired before the user has activated it, generate a new one with this button. Creating a password reset link will invalidate the user's current password, meaning that they cannot log in until the reset is complete.

hrwallingford

We design smarter, more resilient solutions across both the natural and built environment to help everyone live and work more sustainably with water.

HR Wallingford
Howbery Park
Wallingford
Oxfordshire OX10 8BA
United Kingdom

+44 (0)1491 835381

info@hrwallingford.com

www.hrwallingford.com

bsi PAS 99 Integrated Management Systems CERTIFIED
IMR 719286

bsi ISO 9001 Quality Management Systems CERTIFIED
FS 516431

bsi ISO 45001 Occupational Health and Safety Management CERTIFIED
OHS 595357

bsi ISO 14001 Environmental Management CERTIFIED
EMS 558310