RATES, Inc.

Hydraulic Model Execution Prototype Report

Project Deliverable ID 1.2.1.4.3.2.4

Andrew N.S. Ernest, Ph.D., P.E., BCEE, D.WRE
William Kirkey, Ph.D.
Christopher B. Fuller, Ph.D.
Linda Navarro, M.S.
Ivan Santos-Chavez, M.S.

Feb 28, 2023

Approval Page

Technical Review By:

DocuSigned by:
BC4713A8EA924AA...

2/28/2023

Andrew N.S. Ernest, Ph.D., P.E., BCEE, D.WRE
President and CEO

Final Approval For Submission:

DocuSigned by:
William Kirkey
CC59C65904A2440...

2/28/2023

William Kirkey, Ph.D.
Chief Research and Technology Development Officer

CONTENTS

Boobrie is a StormWater Management Model Wizard, made up of a series of tools that facilitate construction and execution of a *SWMM* model domain using cloud resources. Boobrie is currently under active development with goals of:

- auto-generating *SWMM* domain input files from data available on REON data sources;
- running *SWMM* against domain specific data inputs on the cloud;
- augmenting the mode code to integrate real-time sensor data;
- proposing real-time sensor network layout by optimizing locations with least confidence/greatest sensitivity; and
- providing preliminary storm sewer layouts using multi-objective optimization and localized design constraints.

The StormWater Management Model Wizard is a component in the Water Wizard ecosystem of water resource management tools. Water Wizard tools are typically named after relevant mythological entities, such as the Boobrie, a malevolent shapeshifting water creature said to inhabit the lochs of the west coast of Scotland.

Requires

WaterWizard/Mage[1], Celery[2], RabbitMQ[3], django-tables2[4]

[1] https://water-wizard.org/
[2] https://docs.celeryq.dev/
[3] https://www.rabbitmq.com/
[4] https://django-tables2.readthedocs.io/

CHAPTER

ONE

MODULES

boobrie	Boobrie Application
project	Modules for debugging the boobrie web app.

1.1 boobrie

1.1.1 Boobrie Application

Boobrie's goals can be accomplished in many ways, however, the most straightforward is utilizing the *FOSS* "solver" engine, compiled and containerized. Access to the source code[5] also opens up the potential for minor code modifications to enable integration of *RTHS* data into the computational environment for various purposes. The process for developing and deploying *SWMM* in the cloud begins with a proof of concept implementation of the "solver" engine on a bare-metal server using available testing data, followed by containerizing the application and making it available for cloud deployment, and finally a web interface for cloud access.

1.1.1.1 Command Line Interface

Boobrie incorporates a command-line utility ("spyce") that serves as the entrypoint for the Boobrie ecosystem. The *CLI* simplifies the production process by abstracting each step into a subset of easily understood commands.

This module provides tools for bare-metal hosts acquiring and running EPA-SWMM, including:

- acquiring the source code from it's GitHub repository;
- compiling the source code to produce the executable binary;
- acquiring the example data sets from their repository;
- running the examples against the compiled *SWMM* source code ("run_example"); and

[5] https://github.com/USEPA/Stormwater-Management-Model

- providing a mechanism for running the *SWMM* "solver" engine ("run").

The default directory structure is:

```
build
 |__ SWMM
 |__ examples
 |__ run
 |__ build
```

boobrie.**code_get**(**args*, ***kwargs*)

Get the EPA-SWMM source code from github. Clones the EPA-SWMM[6] into the project build directory.

```
spyce boobrie.code-get -d <SWMM_DIR>
```

Where <SWMM_DIR> is the target directory to place the source files.

boobrie.**code_compile**(**args*, ***kwargs*)

Compile the source code. If the source code is not avaiable, it is cloned first, the compiled using CMake[7] configuration files.

boobrie.**examples_get**(**args*, ***kwargs*)

Get the Applications Manual the includes example datasets into the `build/examples` folder. The Applications Manual is downloaded and decompressed.

```
spyce boobrie.examples-get -d <DIRECTORY>
```

Where <DIRECTORY> is the target directory for the examples to be downloaded to.

boobrie.**examples_run**(**args*, ***kwargs*)

Run Example Dataset. This Spell copies a single *SWMM* input file into the run folder, and then triggers a run. The example defaults to `*.*`, but can be over-ridden with a `-E <ex>` flag.:

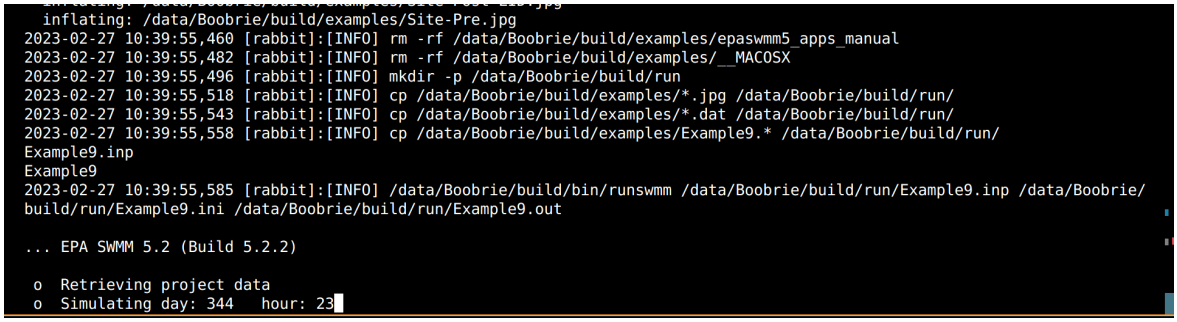
```
spyce boobrie.example-run -E 9
```

or, for all examples:

```
spyce boobrie.example-run
```

Note: Example 9 is a continuous run, excluded from the `*.*` flag

[6] https://github.com/USEPA/Stormwater-Management-Model
[7] https://cmake.org/



boobrie.**swmm_run**(**args*, ***kwargs*)

Run EPA-SWMM. Input files (`*.inp` and `*.ini`) must be placed in the <RUN_DIR> directory.

```
spyce boobrie.run-swmm -d <RUN_DIR>
```

Input files in the <RUN_DIR> folder will be acted on upon execution of the *SWMM* binary. Multiple runs can be executed by placing the appropriate files in the `build/run` directory.

1.1.1.2 Docker Container

Containerizing the *SWMM* "solver" engine is a critical step in cloud deployment. The tools provided in this module allow for the construction, deployment and execution of a containerized *SWMM* model.

boobrie.**docker_build**(**args*, ***kwargs*)

Build the docker image. A Docker image is built using the "app" module.

```
# syntax=docker/dockerfile:1
# FROM python:3.10
FROM ubuntu:22.04 AS build
ENV YOUR_ENV=PYTHONFAULTHANDLER=1 \
  PYTHONUNBUFFERED=1 \
  PYTHONHASHSEED=random \
  PIP_NO_CACHE_DIR=off \
  PIP_DISABLE_PIP_VERSION_CHECK=on \
  PIP_DEFAULT_TIMEOUT=100 \
  POETRY_VERSION=1.3.2

RUN ["apt-get", "update"]
# RUN ["apt-get", "-y", "upgrade"]
RUN ["apt-get", "-y", "install", "python3-invoke", \
  "python3-dotenv", "python3-pip", "git", "wget", "unzip", \
  "build-essential", "cmake"]

# System deps:
```

(continues on next page)

(continued from previous page)

```
RUN pip install poetry==$POETRY_VERSION

# Project initialization:
RUN ["poetry", "config", "virtualenvs.create", "false"]

WORKDIR /data/Boobrie

COPY ["./Boobrie", "/data/Boobrie/"]
COPY ["./Spyce", "/data/Spyce/"]
COPY ["./Mage", "/data/Mage/"]

RUN ["poetry", "lock"]
RUN ["poetry", "install", "--no-interaction", "--no-ansi"]

RUN ["/usr/local/bin/spyce", "boobrie.code-compile"]

ENTRYPOINT ["/usr/local/bin/spyce", "boobrie.swmm-run"]
```

```
spyce boobrie.docker-build
```

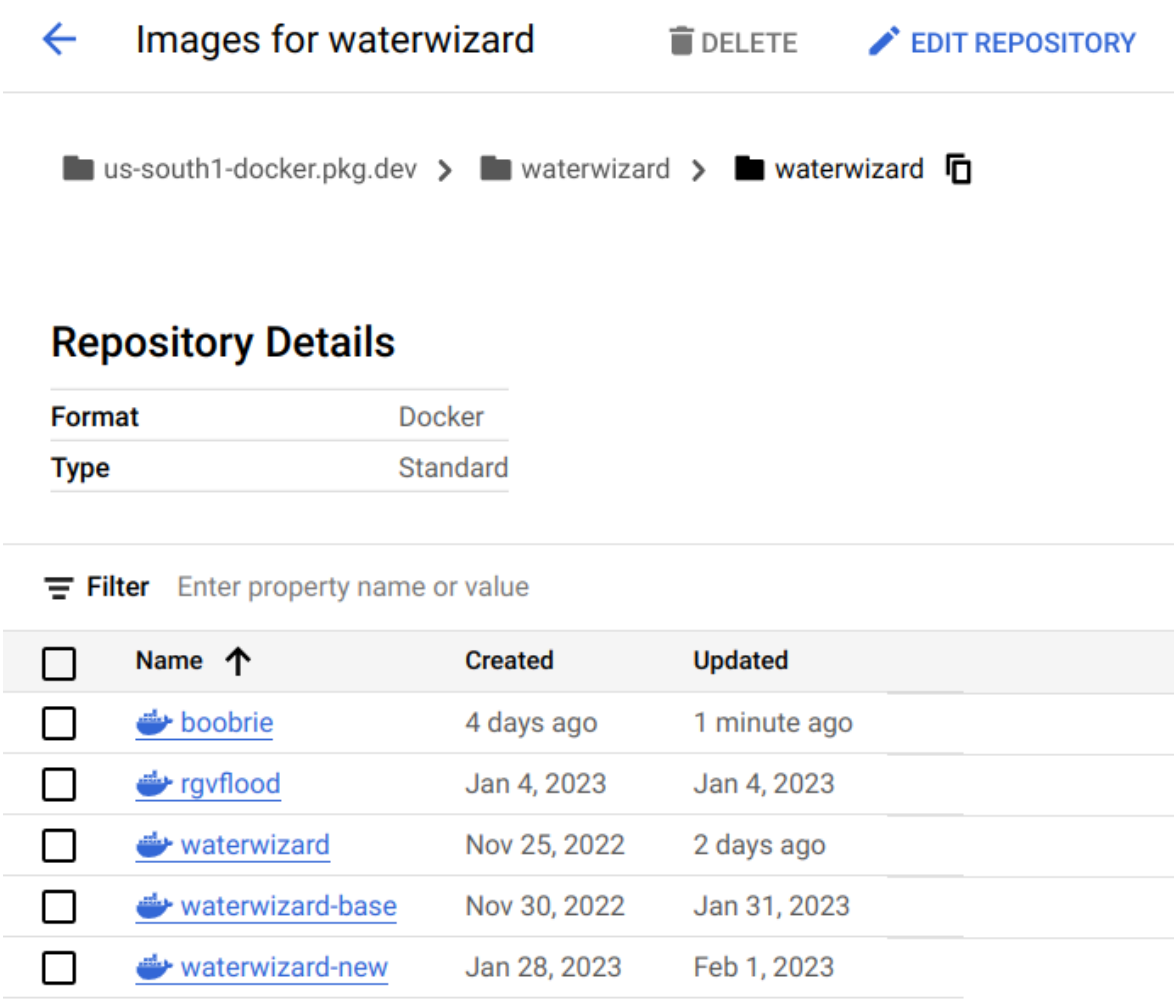
```
#18 15.08 [ 91%] Building C object src/solver/CMakeFiles/swmm5.dir/transect.c.o
#18 15.14 [ 93%] Building C object src/solver/CMakeFiles/swmm5.dir/treatmnt.c.o
#18 15.20 [ 95%] Building C object src/solver/CMakeFiles/swmm5.dir/xsect.c.o
#18 15.38 [ 96%] Linking C shared library libswmm5.so
#18 15.44 [ 96%] Built target swmm5
#18 15.46 [ 98%] Building C object src/run/CMakeFiles/runswmm.dir/main.c.o
#18 15.50 [100%] Linking C executable runswmm
#18 15.55 [100%] Built target runswmm
#18 DONE 15.7s

#19 exporting to image
#19 exporting layers
#19 exporting layers 2.8s done
#19 writing image sha256:f6c3a06548c497b91d841b47d20c4d069fccfcb8fcd9e77f81acb0a381e0314a done
#19 naming to us-south1-docker.pkg.dev/waterwizard/waterwizard/boobrie:latest done
#19 DONE 2.8s
○ (boobrie-py3.10) anernest@rabbit:/data/Boobrie$
```

boobrie.**docker_push**(**args*, ***kwargs*)

Push the docker image to the artifact registry.

```
spyce boobrie.docker-push
```

```
1a3f0a0e1979: Waiting
9cda7e11e2f5: Waiting
c5ff2d88f679: Waiting
5a94d0143ada: Pushed
804b322093ae: Pushed
61eb47105950: Pushed
7475036c680d: Pushed
449ac63dd878: Pushed
0efa46eaf457: Pushed
25a41314532c: Pushed
c5ff2d88f679: Layer already exists
db3e123b914c: Pushed
9cda7e11e2f5: Pushed
1aee649312c4: Pushed
1a3f0a0e1979: Pushed
latest: digest: sha256:a8755247e57647d4c7fa06214986791c0bf27672c021422ed25436cc70c92840 size: 2850
○ (boobrie-py3.10) anernest@rabbit:/data/Boobrie$
```

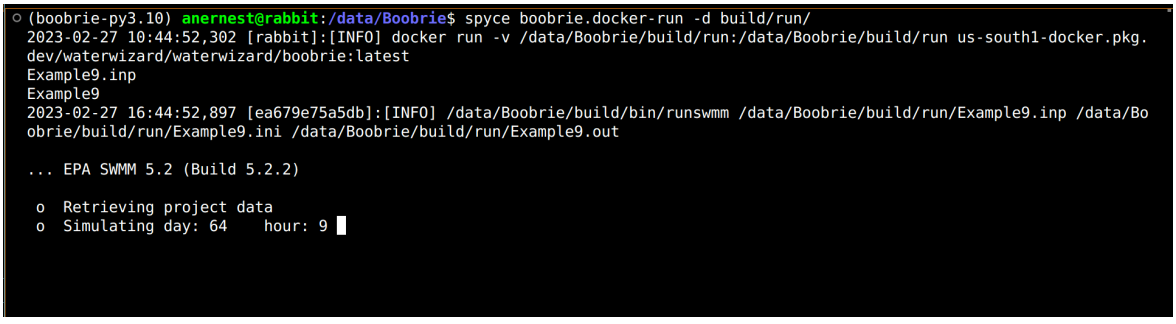


boobrie.**docker_run**(**args*, ***kwargs*)

Runs the docker container, pulling the immage from the artifact registry if necessary.

```
spyce boobrie.docker-run -d <RUN_DIR>
```

Where <RUN_DIR> is the directory containing the input files. Defaults to the current directory.

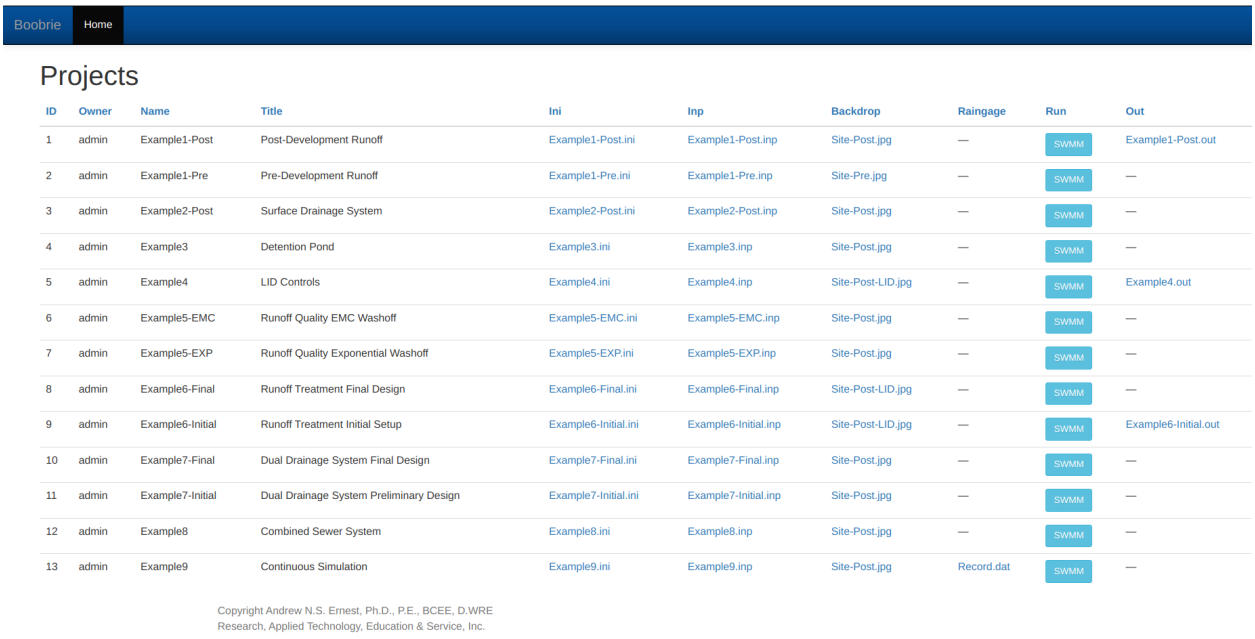


1.1.1.3 Web Interface

The initial cloud deployment of Boobrie is through a Django web application. The functional components are as follows:

1. Ingestion of *SWMM* input files into Water Wizard.
2. Storage of the input files in a session run directory.
3. Triggering a run of the Boobrie docker image, pointing the the session run directory.
4. Provision of Error and/or Output files for view or download.

Boobrie is a *Django* web application that serves as the basis for automated server-side data file construction and *SWMM* execution. In it's current prototype form, it provides the ability to upload datafiles, execute *SWMM*, and download the output.



The Boobrie web application is currently embedded in Water Wizard[8] development service and will be included in the next deployment of RGVFlood[9] once end-user access and execution permissions criteria have been finalized. Integration into the Flood Wizard[10] will be initiated in sequence.

`boobrie.broker_run`(**args*, ***kwargs*)

Run the Message Broker. The message broker is used for inter process communication during asynchronous task execution.

```
spyce docker.broker-run
```

[8] https://water-wizard.org
[9] https://rgvflood.com
[10] https://wizard.rgvflood.com

Parameters
broker (*str*) – Broker name, 'redis' or 'rabbitmq'. Defaults to 'redis'

boobrie.**broker_kill**(**args*, ***kwargs*)

Stops and removes the Message Broker docker container.

```
spyce docker.broker-kill
```

boobrie.**worker_run**(**args*, ***kwargs*)

Run the Celery Worker. Asynchronous task execution is provided by Celery, allowing applications to run in the background in case of long execution times.

```
spyce docker.worker-run
```

boobrie.**worker_kill**(**args*, ***kwargs*)

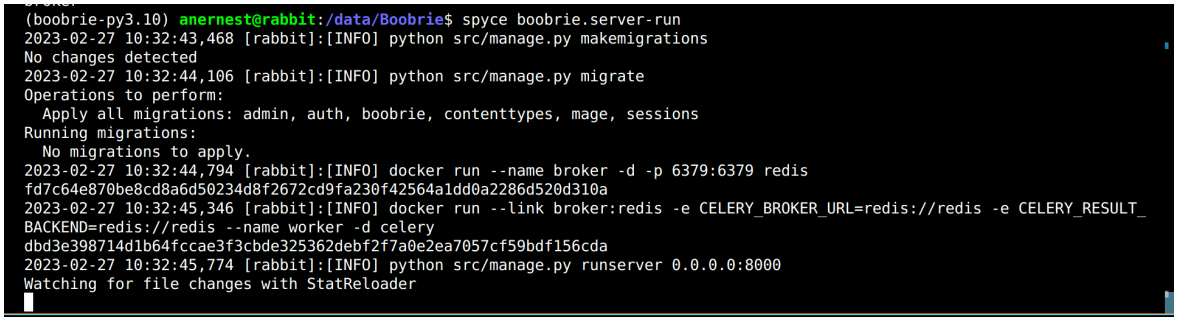
Stops and removes the Celery Worker docker container.

```
spyce docker.worker-kill
```

boobrie.**server_run**(**args*, ***kwargs*)

Run boobrie in debug server. Use Django's built-in "runserver" functionality to spin up a web server to demonstrate and debug boobrie functionality.

```
spyce boobrie.server-run
```



Functions

task(*args, **kwargs)	Marks wrapped callable object as a valid Invoke task.

Modules

boobrie.admin	ModelAdmin Objects.
boobrie.apps	AppConfig Objects.
boobrie.forms	
boobrie.migrations	Database Change Propogation.
boobrie.models	Model Objects.
boobrie.tables	
boobrie.tasks	
boobrie.tests	Code Operation Tests.
boobrie.urls	
boobrie.views	User Interface Classes & Functions.

1.1.2 boobrie.admin

ModelAdmin Objects. Representations of the database models instantiated in the app for rendering and interaction in the Django project administration page.

Classes

`Project`(*args, **kwargs)	Information regarding each project.

1.1.3 boobrie.apps

AppConfig Objects. Defining a subclass of AppConfig configures the application for use in the Django project.

Classes

`AppConfig`(app_name, app_module)	Class representing a Django application and its configuration.
`BoobrieConfig`(app_name, app_module)	

1.1.4 boobrie.forms

Classes

`ModelForm`([data, files, auto_id, prefix, ...])	

`Project`(*args, **kwargs)	Information regarding each project.
`ProjectForm`([data, files, auto_id, prefix, ...])	

1.1.5 boobrie.migrations

Database Change Propogation. Mechanism for propagating changes made to models (adding a field, deleting a model, etc.) into the database schema.

Modules

boobrie.migrations.0001_initial
boobrie.migrations.0002_project_out

1.1.5.1 boobrie.migrations.0001_initial

Classes

`Migration`(name, app_label)

1.1.5.2 boobrie.migrations.0002_project_out

Classes

`Migration`(name, app_label)

1.1.6 boobrie.models

Model Objects. Contains the essential fields and behaviors of the data being store, with each model generally mapping to a single database table.

Functions

`upload_to`(instance, filename)	

Classes

`Project`(*args, **kwargs)	Information regarding each project.
`User`(*args, **kwargs)	Extending User Model

1.1.7 boobrie.tables

Classes

`Project`(*args, **kwargs)	Information regarding each project.
`ProjectTable`([data, order_by, orderable, ...])	
`TemplateColumn`([template_code, ...])	A subclass of *.Column* that renders some template code to use as the cell value.

1.1.8 boobrie.tasks

Functions

`run_task`(name)	

1.1.9 boobrie.tests

Code Operation Tests. Set of tests for automated verification of the code operation. Reserved for use upon integration in Water Wizard.

Classes

`TestCase`([methodName])	Similar to TransactionTestCase, but use *transaction.atomic()* to achieve test isolation.

1.1.10 boobrie.urls

Functions

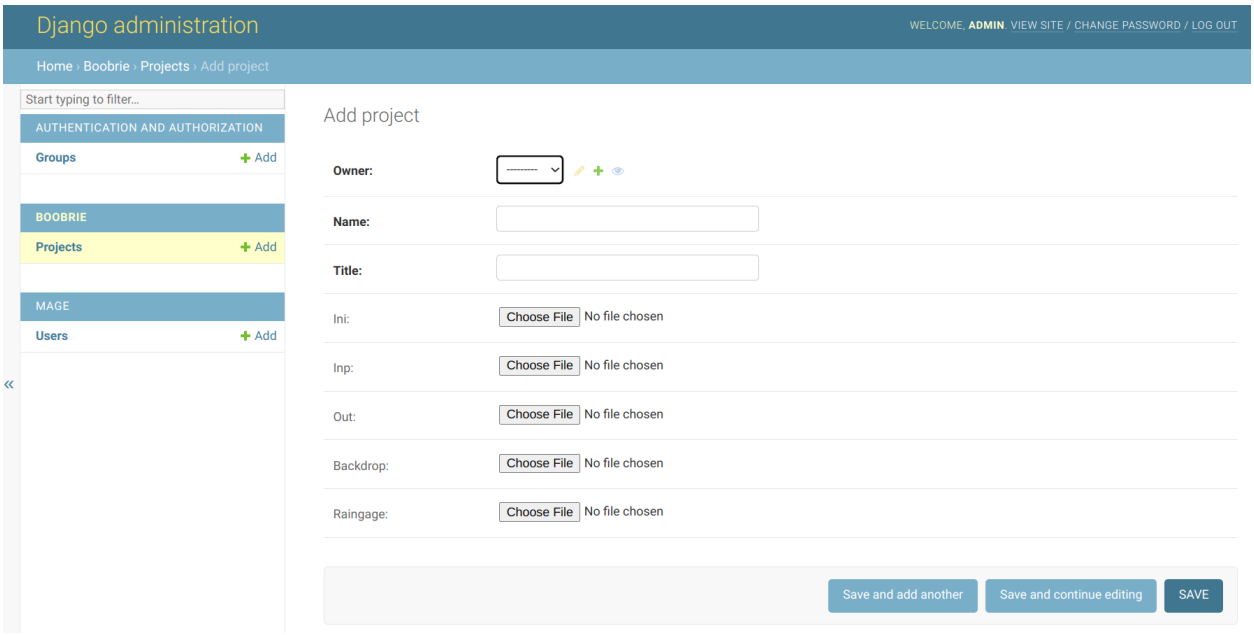
`path`(route, view[, kwargs, name, Pattern])	
`run`(request, project_id)	Run SWMM on the selected Project.

Classes

`ProjectDetailView`(**kwargs)	
`ProjectListView`(**kwargs)	List the Projects uploaded into Boobrie.

1.1.11 boobrie.views

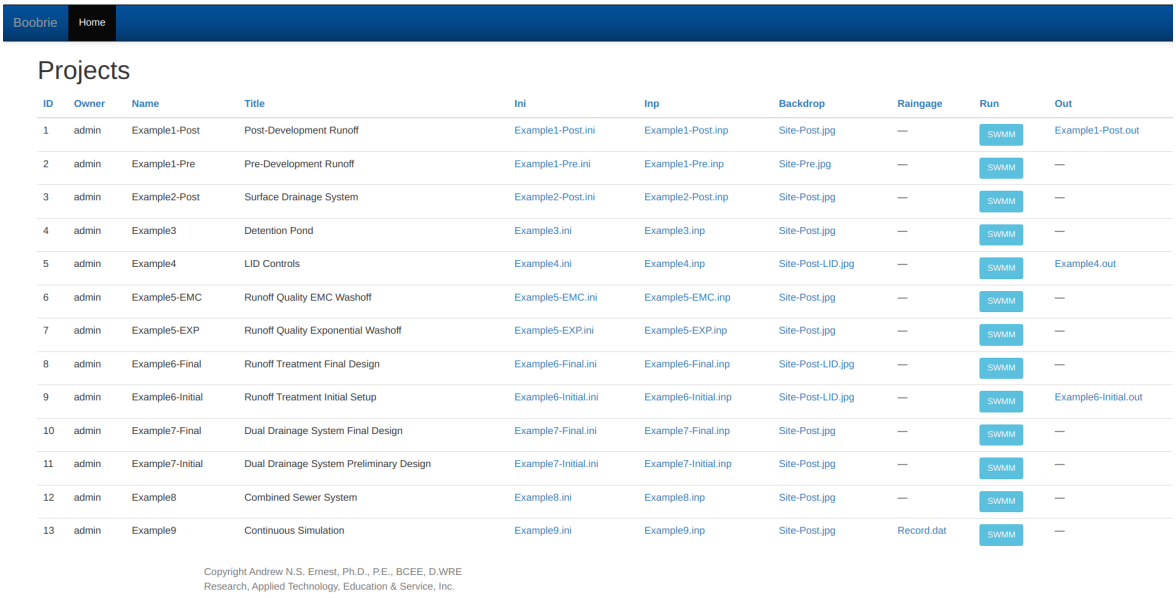
User Interface Classes & Functions. Boobrie Views presents the user interface on the web. Project data entry is currently handled via the Django admin view:



The project name and title, along with the ".inp", ".ini", and optional backdrop image and raingage data files are uploaded.

class boobrie.views.**ProjectListView**(***kwargs*)

List the Projects uploaded into Boobrie. The main Boobrie page lists all project uploaded into the system. The "Run" column provides a "SWMM" button for each project, while the "Out" column will list a clickable download link for the output file, if the project has been run.



ID	Owner	Name	Title	Ini	Inp	Backdrop	Raingage	Run	Out
1	admin	Example1-Post	Post-Development Runoff	Example1-Post.ini	Example1-Post.inp	Site-Post.jpg	—	SWMM	Example1-Post.out
2	admin	Example1-Pre	Pre-Development Runoff	Example1-Pre.ini	Example1-Pre.inp	Site-Pre.jpg	—	SWMM	—
3	admin	Example2-Post	Surface Drainage System	Example2-Post.ini	Example2-Post.inp	Site-Post.jpg	—	SWMM	—
4	admin	Example3	Detention Pond	Example3.ini	Example3.inp	Site-Post.jpg	—	SWMM	—
5	admin	Example4	LID Controls	Example4.ini	Example4.inp	Site-Post-LID.jpg	—	SWMM	Example4.out
6	admin	Example5-EMC	Runoff Quality EMC Washoff	Example5-EMC.ini	Example5-EMC.inp	Site-Post.jpg	—	SWMM	—
7	admin	Example5-EXP	Runoff Quality Exponential Washoff	Example5-EXP.ini	Example5-EXP.inp	Site-Post.jpg	—	SWMM	—
8	admin	Example6-Final	Runoff Treatment Final Design	Example6-Final.ini	Example6-Final.inp	Site-Post-LID.jpg	—	SWMM	—
9	admin	Example6-Initial	Runoff Treatment Initial Setup	Example6-Initial.ini	Example6-Initial.inp	Site-Post-LID.jpg	—	SWMM	Example6-Initial.out
10	admin	Example7-Final	Dual Drainage System Final Design	Example7-Final.ini	Example7-Final.inp	Site-Post.jpg	—	SWMM	—
11	admin	Example7-Initial	Dual Drainage System Preliminary Design	Example7-Initial.ini	Example7-Initial.inp	Site-Post.jpg	—	SWMM	—
12	admin	Example8	Combined Sewer System	Example8.ini	Example8.inp	Site-Post.jpg	—	SWMM	—
13	admin	Example9	Continuous Simulation	Example9.ini	Example9.inp	Site-Post.jpg	Record.dat	SWMM	—

boobrie.views.**run**(*request*, *project_id*)

Run SWMM on the selected Project. The run is executed by clicking on the "SWMM" button for the desired project on the Boobrie home page. A successful run will result in the output

file being listed in the last column. The ".out" binary file can then be downloaded.

Functions

redirect(to, *args[, permanent])	Return an HttpResponseRedirect to the appropriate URL for the arguments passed.
run(request, project_id)	Run SWMM on the selected Project.
run_task(name)	

Classes

DetailView(**kwargs)	Render a "detail" view of an object.
Project(*args, **kwargs)	Information regarding each project.
ProjectDetailView(**kwargs)	
ProjectListView(**kwargs)	List the Projects uploaded into Boobrie.
ProjectTable([data, order_by, orderable, ...])	
SingleTableView(**kwargs)	Generic view that renders a template and passes in a *.Table* instances.

1.2 project

Modules for debugging the boobrie web app.

Modules

project.asgi	ASGI config for project project.
project.celery	Celery is a distributed task queue for UNIX systems.
project.settings	Django settings for project project.
project.urls	project URL Configuration
project.wsgi	WSGI config for app project.

1.2.1 project.asgi

ASGI config for project project.

It exposes the ASGI callable as a module-level variable named `application`.

For more information on this file, see https://docs.djangoproject.com/en/4.1/howto/deployment/asgi/

Functions

`get_asgi_application`()	The public interface to Django's ASGI support.

1.2.2 project.celery

Celery is a distributed task queue for UNIX systems. It enables offloading work from a Python app. Once integrated into the app, Celery can send time-intensive tasks to Celery's task queue, allowing the web app to continue responding quickly to users while Celery completes expensive operations asynchronously in the background[11].

Celery is used in Boobrie to offload SWMM for background execution.

Classes

`Celery`([main, loader, backend, amqp, ...])	Celery application.

1.2.3 project.settings

Django settings for project project.

Generated by 'django-admin startproject' using Django 4.1.5.

For more information on this file, see https://docs.djangoproject.com/en/4.1/topics/settings/

For the full list of settings and their values, see https://docs.djangoproject.com/en/4.1/ref/settings/

[11] <https://realpython.com/asynchronous-tasks-with-django-and-celery/>

Classes

`Path`(*args, **kwargs)	PurePath subclass that can make system calls.

1.2.4 project.urls

project URL Configuration

The *urlpatterns* list routes URLs to views. For more information please see:
https://docs.djangoproject.com/en/4.1/topics/http/urls/

Examples: Function views

1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path(‘’, views.home, name=’home’)

Class-based views

1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path(‘’, Home.as_view(), name=’home’)

Including another URLconf

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path(‘blog/’, include(‘blog.urls’))

Functions

`include`(arg[, namespace])	
`path`(route, view[, kwargs, name, Pattern])	
`static`(prefix[, view])	Return a URL pattern for serving files in debug mode.

1.2.5 project.wsgi

WSGI config for app project.

It exposes the WSGI callable as a module-level variable named `application`.

For more information on this file, see https://docs.djangoproject.com/en/4.1/howto/deployment/wsgi/

Functions

`get_wsgi_application`()	The public interface to Django's WSGI support.

APPENDIX

A

GLOSSARY

API
Application Programming Interface

API.RGVFlood.com
RGVFlood.com data assimilation service.

AU
Assessment Unit

AWS
Amazon Web Services

Azure
Microsoft's Cloud Computing Platform

Bernoulli
The Bernoulli equation is a simplification of the Navier-Stokes equations assuming inviscid fluid and steady (non-time-variant) flow.

BLE
Base Level Engineering

Celery
A task scheduling and messaging application used to maximize parallel task processing.

CentOS
A *Linux* distribution

CI
Cyberinfrastructure

CI/CD
Continuous Integration and Continuous Delivery/Continuous Deployment

CLI
Command-Line Interface

Cloud
Servers, software and databases that are accessed over the Internet

Clover
: Cloud Virtual Water Model Executor

COP
: Common Operating Picture

CPU
: Centralized Processing Unit

Crowdsource
: Data collection from open, relatively un-controlled, sources.

CUAHSI
: Consortium of Universities for the Advancement of Hydrologic Science

Cyberinfrastructure
: computing systems, data storage systems, advanced instruments and data repositories, visualization environments, and people, all linked by high speed networks

DEM
: Digital Elevation Model

Deterministic
: Approaches to describing processes that do not rely on randomness.

DFIRM
: Digital Flood Insurance Rate Map

DHS
: Department of Homeland Security

DIKW
: Data, Information, Knowledge, Wisdom

Django
: <https://www.djangoproject.com/>

DO
: Dissolved Oxygen

Docker
: Docker is a container deployment platform that allows for the rapid deployment of a applications in the cloud, independent of the physical infrastructure.

DRF
: *Django ReST* Framework

DSS
: Decision Support System

EC2
: *AWS* Elastic Cloud Compute

Eeyore
: URL: Eeyore.ratesresearch.org CPU: Dual Intel(R) Xeon(R) E-2124 CPU @ 3.30GHz Memory: 16GB HD: 4TB OS: Ubuntu Linux 20.04

FEMA
: Federal Emergency Management Agency

FIF
: Flood Infrastructure Fund

FOSS
: Free and Open Source Software

GAE
: Google App Engine

GCE
: Google Compute Engine

GCP
: Google Cloud Platform

GCS
: Google Cloud storage

GeoNode
: A web-based application and platform for developing geospatial information systems (GIS) and for deploying spatial data infrastructures (SDI).

GeoNode/db
: *PostgreSQL* with *PostGIS* extensions database server storing *GeoNode* *Django* and *GeoServer* data.

GeoNodeGCP
: An implementation of *GeoNode* on *GCP*

GeoServer
: Open source server for sharing geospatial data.

GeoTIFF
: A public domain metadata standard which has the georeferencing information embedded within the *TIFF* file.

GIS
: Geospatial Information System

GitHub
: An online software development platform used for storing, tracking, and collaborating on software projects.

GKE
: Google *Kubernetes* Engine. https://cloud.google.com/kubernetes-engine/

H&H
: Hydrologic and Hydraulic

HAND
: Height Above Nearest Drainage <http://handmodel.ccst.inpe.br/>

HEC
: Hydrologic Engineering Center

HEC-DSS
: *HEC* Data Storage System

HEC-HMS
: Hydrologic Engineering Center Hydrologic Modeling System. <https://www.hec.usace.army.mil/software/hec-hms/>

HEC-RAS
: Hydrologic Engineering Center River Analysis System. <https://www.hec.usace.army.mil/software/hec-ras/>

HEC-RTS
: Hydrologic Engineering Center Real Time Simulation

Hogwarts
: *RATES Kubernetes* on-premise multi-node bare-metal cluster.

HPC
: High Performace Computing

HPCC
: *HPC* cluster

HTML
: Hypertext Markup Language

HUC
: Hydrologic Unit Code

HWMD
: Hidalgo/Willacy Main Drain

IBWCNF
: *USIBWC* North Floodway

IDV
: Integrated Data Viewer from *UniData*

InfoWorks ICM
: <https://www.innovyze.com/en-us/products/infoworks-icm>

IT
: Information Technology

K8s
: *Kubernetes*

KIND
: *Kubernetes* IN *Docker*. https://github.com/kubernetes-sigs/kind

Kubernetes
: An orchestration system facilites the deployment and management of containerized applications, with a specific focus on scaling to increase demand for the provided services. https://kubernetes.io/

LaTeX
: A high-quality typesetting system including features designed for the production of technical and scientific documentation

LiDAR
: Light Detection and Ranging

Linux
: An open source operating system that is made up of the kernel, the base component of the OS, and the tools, apps, and services bundled along with it.

LLM
: Lower Laguna Madre

LLM/BSC
: Lower Laguna Madre/Brownsville Ship Channel watershed.

LRGV
: Lower Rio Grande Valley

LRGVDC
: Lower Rio Grande Valley Development Council

LSM
: Land Surface Models focus on describing the processes driving the exchange of terrestrial water with atmospheric.

Mechanistic
: Formulations describing physical, biological or chemical processes based on a theoretical understanding.

Metadata
: Data that provides information about other data

Microservices Architecture
: A modular approach to developing applications. allowing large applications to be separated into smaller independent parts.

MIKE Urban+
: <https://www.mikepoweredbydhi.com/download/mike-2019/mike-urban-plus?ref=%7B5399F5D6-40C6-4BB2-8311-37B615A652C6%7D>

MPI
Message Passing Interface

NAT
Network Address Translation

Navier-Stokes
The Navier-Stokes equations are mathematically representations of conservation of mass and momentum for simple fluids such as water.

NCAR
National Center for Atmospheric Research

NetCDF
NetCDF (Network Common Data Form) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. It is also a community standard for sharing scientific data. The Unidata Program Center supports and maintains netCDF programming interfaces for C, C++, Java, and Fortran. Programming interfaces are also available for Python, IDL, MATLAB, R, Ruby, and Perl. Reproduced from NetCDF[12].

NFS
Network File Systemn a protocol that lets users on client computers access files on a network, making it a distributed file system.

NGINX
High performance web server.

NIC
Network interface controller

NLDAS
North American Land Data Assimilation System

NOAA
National Oceanic and Atmospheric Agency

NWC
National Water Center

NWM
National Water Model

NWS
National Weather Service

ODM
Observations Data Model

[12] https://www.unidata.ucar.edu/software/netcdf/

PDF
Portable Document Format

PostGIS
Spatial database extender for *PostgreSQL*

PostgreSQL
Open source object-relational database system, available with *PostGIS* extensions

Primo
Parallel raster inundation model

PWA
Progressive Web Application, an application format that allows installation as native applications onto mobile devices and desktop PCs directly from the web.

Python
<https://www.python.org/>

R
A language and environment for statistical computing and graphics

RabbitMQ
An open-source inter-process message broker

RATES
Research, Applied Technology, Education and Service, Inc., a non-profit technology-based company.

RBAC
Role Based Access Control

REON
River and Estuary Observation Network. A partnership of organizations, supported by cloud software, committed to furthering the Democratization of Water Intelligence by sharing water data, analytics and models for local and regional decision making.

REON.cc
Cloud-based cyber-infrastructure that supports *REON*'s goals.

REON/db
PostgreSQL with *PostGIS* extensions database server storing *REON* specific data for *RTHS*, *REON/WM* & *REON.cc* data.

REON/RGV
Instantiation of *REON* with specific application to the Lower Rio Grande Valley - this includes the collection of *RTHS* stations, the *REON* partners with a stake in the LRGV, and the application of the *REON/WM* to the *LRGV*.

REON/WM
REON Water Model

REONode
REON Development Environment. A live implementation of *REON.cc* extensions and applications under active development.

ReST
REpresentational State Transfer

RGVFlood
Instantiation of the *REON* Cyberinfrastructure specific to the *LRGV*.

RGVFlood.com
The domain name and *URL* for *RGVFlood*.

RTHS
Real Time Hydrologic System

RTHS.us
Cloud server of *RTHS* network data

RVD
Raymondville Drain

RWRAC
Regional Water Resources Advisory Committee

SA
Situational Awareness

SaaS
Software as a Service

SMT
Simultaneous Multi-Threading

SONAR
Sound Navigation Ranging, a technique for detecting and determining the distance and direction of underwater objects by acoustic means.

Sphinx
Documentation generator supporting multiple output formats

SPRNT
Simulation Program for River Networks

Spyce
Smartphone Python Computing Environment

Stochastic
Approaches to describing processes in statistical terms.

SustainRGV
Instantiation of the *REON* Cyberinfrastructure specific to the *LRGV* sustainability.

SustainRGV.org
: The domain name and *URL* for *SustainRGV*.

SWMM
: Stormwater Management Model

SWTF
: Stormwater Taskforce

Tastypie
: a webservice *API* framework for *Django*

TGLO
: Texas General Land Office

Tier I
: Tier I Real-Time Regional Hydrologic Modeling Framework

Tier II
: Tier II On-Demand Sub-Regional Hydraulic Modeling Framework

Tier III
: Tier III Off-Line Urban Stormwater Modeling Framework

TIFF
: Tag Image File Format, a computer file used to store raster graphics and image information.

Tigger
: URL: Tigger.water-wizard.org CPU: Dual Intel(R) Xeon(R) CPU E3-1245 v3 @ 3.40GHz Memory: 16GB HD: 4TB OS: Ubuntu Linux 20.04

TIN
: Triangular Irregular Networks are a form of vector-based digital geographic data and are constructed by triangulating a set of vertices.

TOML
: Tom's Obvious Minimal Language

TWDB
: Texas Water Development Board

TWDB/FIF
: The Texas Water Development Board Flood Infrastructure Fund.

Ubuntu
: A *Linux* distribution

UCAR
: University Corporation for Atmospheric Research

UI
: User Interface

UniData
: A *UCAR* community program focused on sharing geoscience data and the tools to access and visualize that data.

URL
: Uniform Resource Locator

USACE
: United States Army Corps of Engineers

USGS
: United States Geological Survey

USIBWC
: United States International Boundary Water Commission

UTRGV
: University of Texas Rio Grande Valley

vCPU
: Virtual *CPU*

VCS
: Version Control System

VIC
: Variable Infiltration Capacity (VIC) Macroscale Hydrologic Model. <https://vic.readthedocs.io/en/master/>

VM
: Virtual Machine

Water Wizard
: A suite of decision support tools designed for regional decision makers.

Wizard.RGVFlood.com
: A web, mobile and desktop client-side application that, working with the server-side components at *RGVFlood.com*, provides the end-user with the up-to-date analytics, visualization and decision support services from the core *REON.cc CI*.

WPP
: Watershed Protection Plan

WPS
: WRF Preprocessing System

WRDA
: Water Resources Development Act

WRF
: Weather Research and Forecasting Model

WRF-Hydro

WRF Hydrological modeling system. <https://ral. .edu/projects/wrf_hydro/overview>

PYTHON MODULE INDEX

b

p

INDEX

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W