Documentation tools

Sphinx is a widely used documentation generation tool primarily used for Python projects, but it supports several other programming languages as well. It was originally created to document the Python programming language and has since become a popular choice for generating high-quality documentation for software projects of all kinds.

Sphinx is a powerful documentation generator that has many great features for writing technical documentation including:

- Generate web pages, printable PDFs, documents for e-readers (ePub), and more all from the same sources
- We can use reStructuredText or Markdown to write documentation
- An extensive system of cross-referencing code and documentation
- Syntax highlighted code samples
- A vibrant ecosystem of first and third-party extensions

The initial steps are for applying sphinx:

- Importing the project to Read the Docs
- Checking the first build
- Basic configuration changes
- Trigger a build from a pull request

Step 1: Installing Sphinx

We'll need to install sphinx via pip. At a minimum we will need version 1.3, but unless we have good reason, we should install the most recent version.

```
pip install sphinx
```

Step 2: Setup our Project with Quickstart

When we install the sphinx package a number of command line utilities are set up as well.

One of those, sphinx-quickstart will quickly generate a basic configuration file and directory structure for our documentation.

Run this command at the base directory of our project (i.e. the Git repo root). It will ask us a number of questions that will determine its actions. We can generally accept the default values, but here are some suggestions of when to deviate from the default:

```
Root path for the documentation: ./docs
```

```
autodoc: automatically insert docstrings from modules (y/n):y
```

```
coverage: checks for documentation coverage (y/n) [n]: y
```

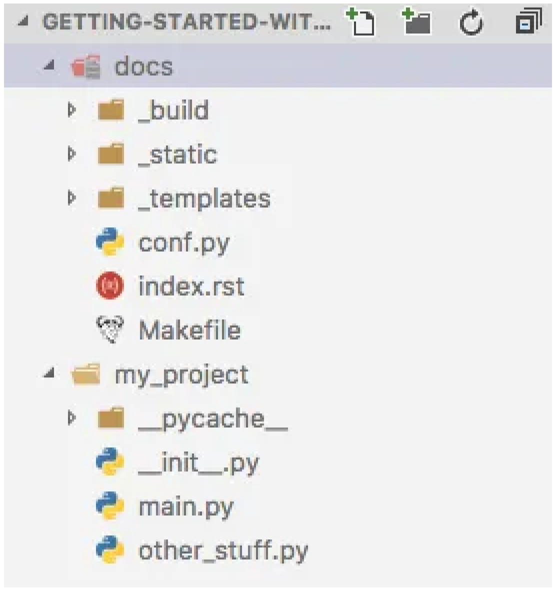
After the program has run, we'll notice a new one ./docs folder exists in our project directory. In addition, there are three new files in that folder: conf.py, index.rst, andMakefile.

Step 3: Adjusting the conf.py File

The default conf.py file generated by the quickstart utility is about 170 lines long, so I won't include the whole thing here. There are, however, a couple of items that we need to update before continuing.

Tell Sphinx the Location of our Python Package

The first thing that we need to do is indicate where the Python package that contains our program code is in relation to the conf.py file. If our directory structure looks like this:



Example Project Directory Structure

We will need to indicate in the conf.py file that Sphinx must go "up" one directory level to find the Python package.

The place to put this is at the end of the first section of the configuration file. Just before the General Configuration settings, we'll see this:

```
# import os

# import sys

# sys.path.insert(0, os.path.abspath('.'))
```

If it wasn't commented out, it would indicate that our package is in the same directory as the conf.py file. We'll need to change it to this:

```
import os

import sys

sys.path.insert(0, os.path.abspath('..'))
```

Add "Napoleon" to the list of Sphinx Extensions to Use

Out of the box, Sphinx only understands docstrings written in traditional reStructuredText. If we've had the, ahem, privilege of working with such docstrings, we’ll know that they are a pain to write and not at all human friendly to read when looking at them directly in the source code.

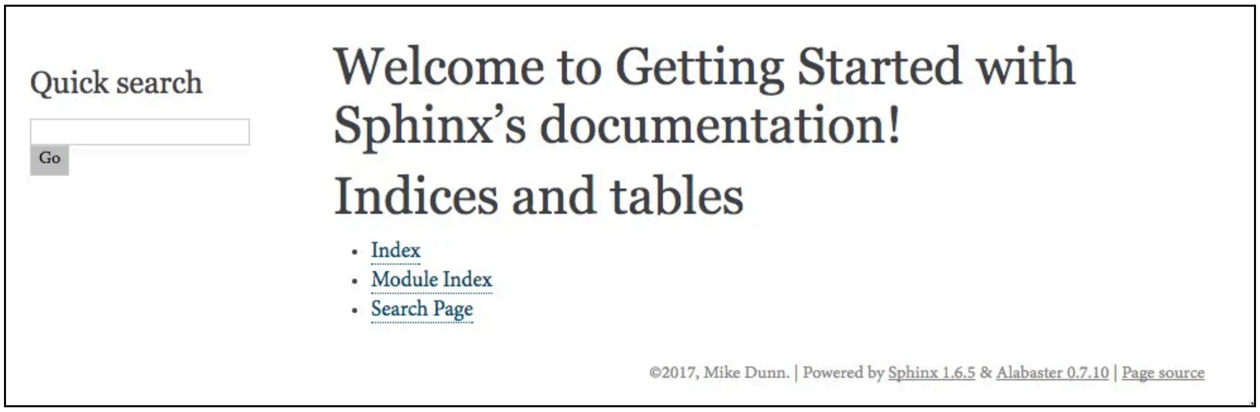
The Napoleon extension enables Sphinx to understand docstrings written in two other popular formats: NumPy and Google.

All we have to do is add sphinx.ext.napoleon to the extensions list. When we are done, it should look like this:

```
extensions = ['sphinx.ext.autodoc', 'sphinx.ext.coverage',
'sphinx.ext.napoleon']
```

Step 4: Update index.rst

At this point, we could actually run the build process to generate our documentation. But it would be pretty disappointing. Here’s is what we’d get:


Welcome to Getting Started with Sphinx’s documentation!

Indices and tables

- Index
- Module Index
- Search Page

©2017, Mike Dunn. | Powered by Sphinx 1.6.5 & Alabaster 0.7.10 | Page source

As much as I would like for Sphinx to go and find our docstrings for us and arrange them nicely without any further configuration, it isn’t quite that magical.

To move forward, we will have to do some minor modifications to our index.rst file, which currently looks like this:

```
.. Getting Started with Sphinx documentation master file, created by
   sphinx-quickstart on Mon Nov 13 11:41:03 2017.
   You can adapt this file completely to your liking, but it should
at least
   contain the root `toctree` directive.

Welcome to Getting Started with Sphinx's documentation!
=======================================================

.. toctree::
   :maxdepth: 2
   :caption: Contents:

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

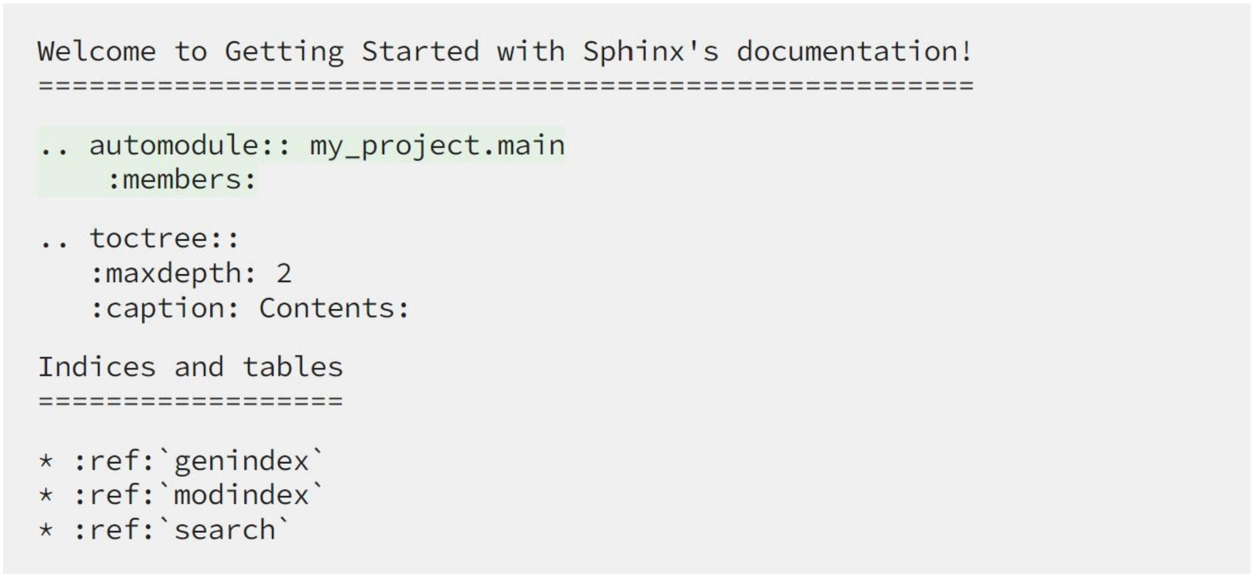
Let’s start by getting rid of the comment at the top which is just noise:

```
Welcome to Getting Started with Sphinx's documentation!
=======================================================

.. toctree::
   :maxdepth: 2
   :caption: Contents:

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Now, while there are a number of things that we could do here, we are going to limit ourselves to the bare minimum to keep this post to a somewhat reasonable length.

Do we see that .. toctree:: line? That is what Sphinx calls a directive. We need to add autodoc directives to our index.rst file so that Sphinx knows what code objects we wish to use the autodoc extension on.

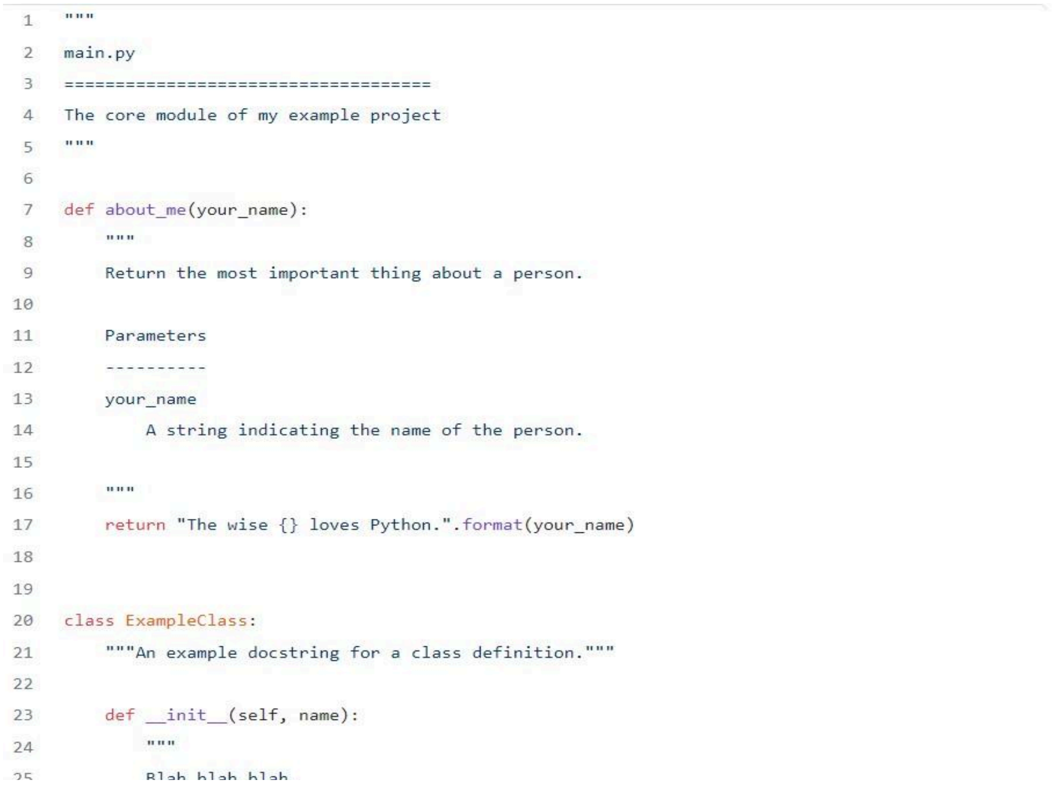
I’ll go ahead and add one indicating to Sphinx that I want it to document the public members of my main.py module inside the my_project package:



```
Welcome to Getting Started with Sphinx's documentation!
=======================================================

.. automodule:: my_project.main
    :members:

.. toctree::
   :maxdepth: 2
   :caption: Contents:

Indices and tables
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Step 5: Write our Docstrings

However, here is the code from main.py which contains a couple of simple NumPy style docstrings that will be picked up by our autodoc directive:



```python
"""
main.py
====================================
The core module of my example project
"""

def about_me(your_name):
    """
    Return the most important thing about a person.

    Parameters
    ----------
    your_name
        A string indicating the name of the person.

    """
    return "The wise {} loves Python.".format(your_name)


class ExampleClass:
    """An example docstring for a class definition."""

    def __init__(self, name):
        """
        Blah blah blah
```

Step 6: Generate our Docs!

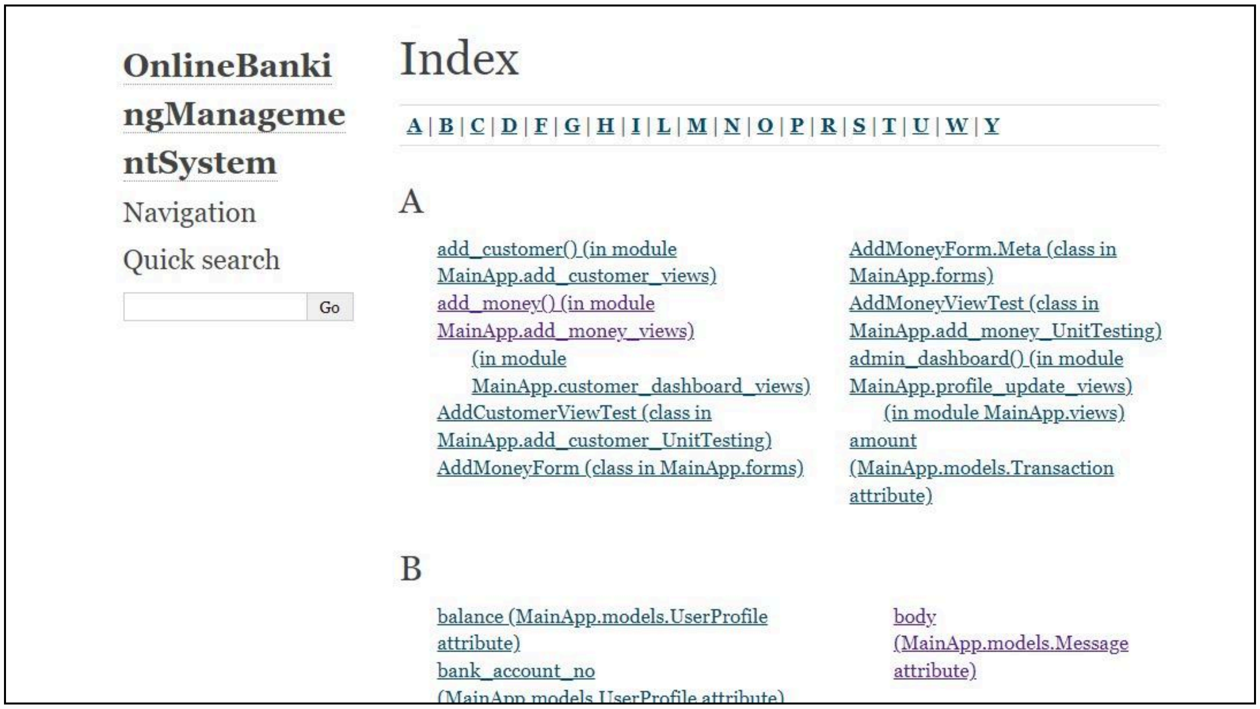
Now it’s time to reap the rewards of our labor. Make sure we are in the ./docs directory and execute the following: make html

If we’ve been following along thus far, we should see something like this:

```
Running Sphinx v1.6.5
loading pickled environment... done
building [mo]: targets for 0 po files that are out of date
building [html]: targets for 1 source files that are out of date
updating environment: 0 added, 1 changed, 0 removed
reading sources... [100%] index
looking for now-outdated files... none found
pickling environment... done
checking consistency... done
preparing documents... done
writing output... [100%] index
generating indices... genindex py-modindex
writing additional pages... search
copying static files... done
copying extra files... done
dumping search index in English (code: en) ... done
dumping object inventory... done
build succeeded.
```

As long as we see that glorious build success message at the end, we are ready to go and behold our beautiful creation.

At the command line, execute open _build/html/index.html (or just open up that page in our browser manually) and we should see something like this:



OnlineBankingManagementSystem

Navigation

Quick search

Go

Index

A | B | C | D | F | G | H | I | L | M | N | O | P | R | S | T | U | W | Y

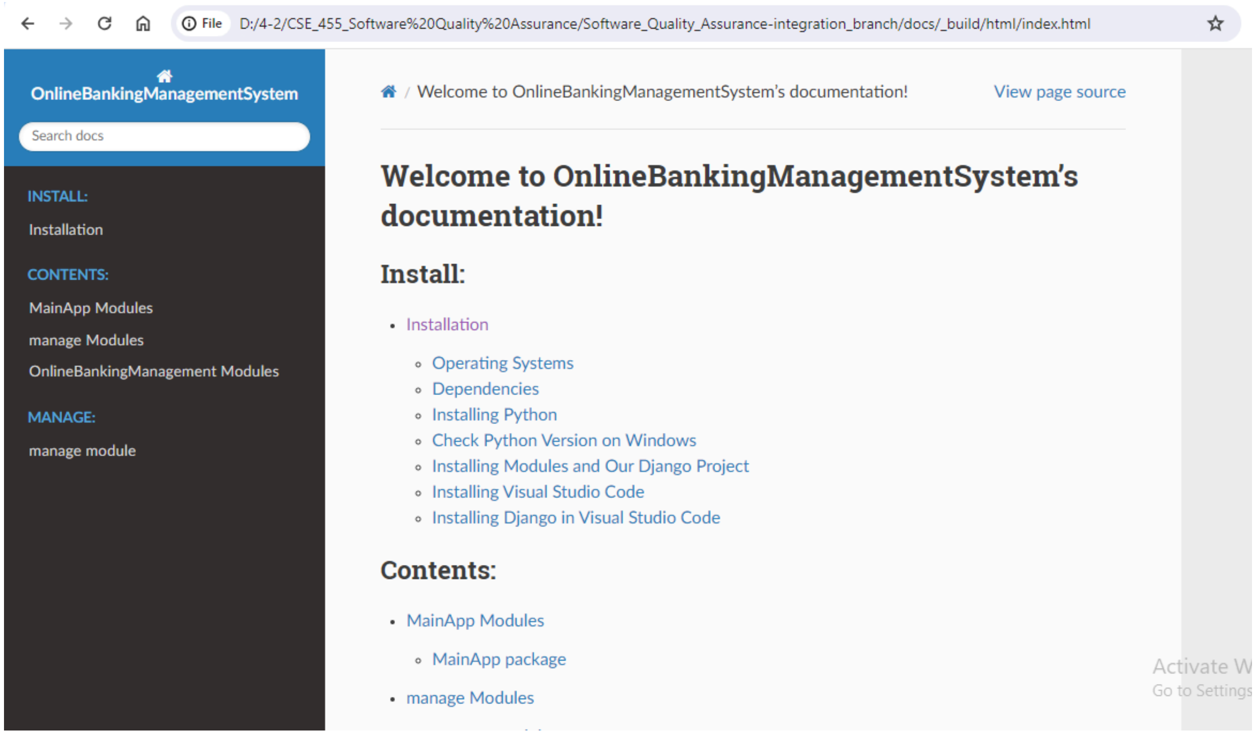
A

add_customer() (in module MainApp.add_customer_views)
add_money() (in module MainApp.add_money_views)
(in module MainApp.customer_dashboard_views)
AddCustomerViewTest (class in MainApp.add_customer_UnitTesting)
AddMoneyForm (class in MainApp.forms)
AddMoneyForm.Meta (class in MainApp.forms)
AddMoneyViewTest (class in MainApp.add_money_UnitTesting)
admin_dashboard() (in module MainApp.profile_update_views)
(in module MainApp.views)
amount (MainApp.models.Transaction attribute)

B

balance (MainApp.models.UserProfile attribute)
bank_account_no (MainApp.models.UserProfile attribute)
body (MainApp.models.Message attribute)

Some layout of our documentation:



D:/4-2/CSE_455_Software%20Quality%20Assurance/Software_Quality_Assurance-integration_branch/docs/_build/html/index.html

OnlineBankingManagementSystem

Search docs

INSTALL:
Installation

CONTENTS:
MainApp Modules
manage Modules
OnlineBankingManagement Modules

MANAGE:
manage module

Welcome to OnlineBankingManagementSystem's documentation! View page source

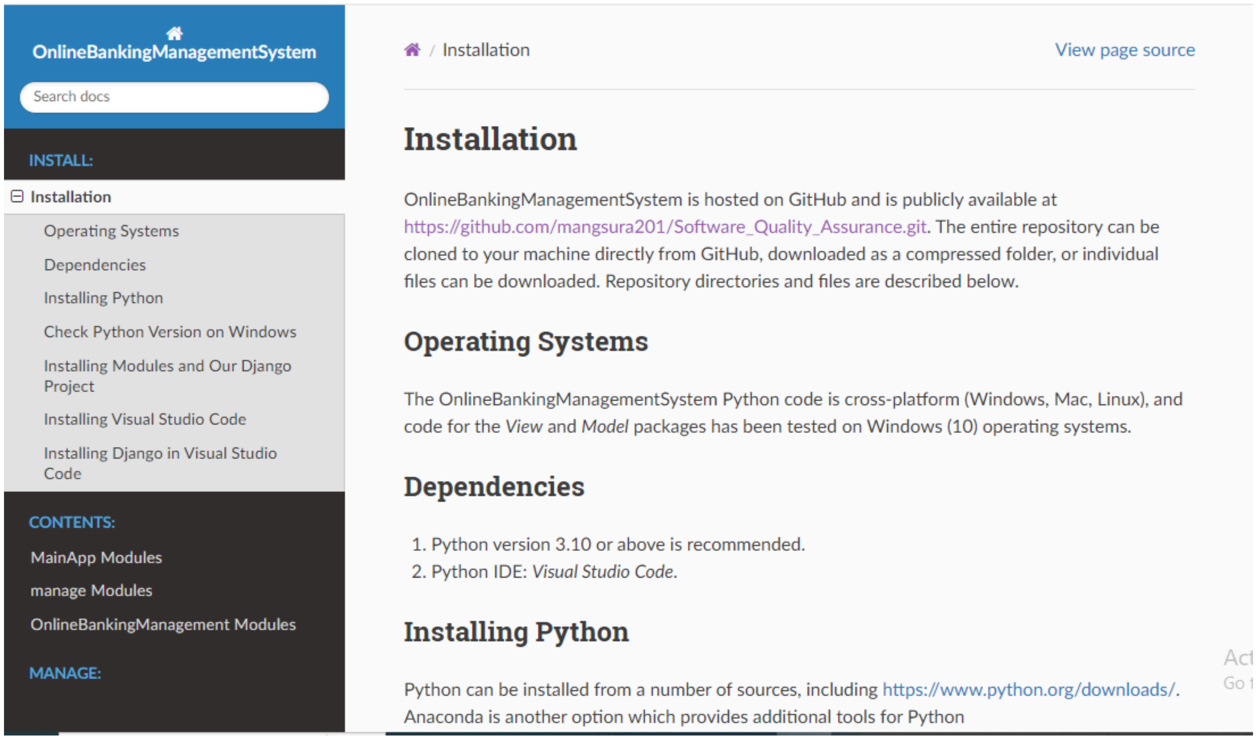
Welcome to OnlineBankingManagementSystem's documentation!

Install:

- Installation
  - Operating Systems
  - Dependencies
  - Installing Python
  - Check Python Version on Windows
  - Installing Modules and Our Django Project
  - Installing Visual Studio Code
  - Installing Django in Visual Studio Code

Contents:

- MainApp Modules
  - MainApp package
- manage Modules



OnlineBankingManagementSystem

Search docs

INSTALL:

Installation

Operating Systems

Dependencies

Installing Python

Check Python Version on Windows

Installing Modules and Our Django Project

Installing Visual Studio Code

Installing Django in Visual Studio Code

CONTENTS:

MainApp Modules

manage Modules

OnlineBankingManagement Modules

MANAGE:

/ Installation

View page source

Installation

OnlineBankingManagementSystem is hosted on GitHub and is publicly available at https://github.com/mangsura201/Software_Quality_Assurance.git. The entire repository can be cloned to your machine directly from GitHub, downloaded as a compressed folder, or individual files can be downloaded. Repository directories and files are described below.

Operating Systems

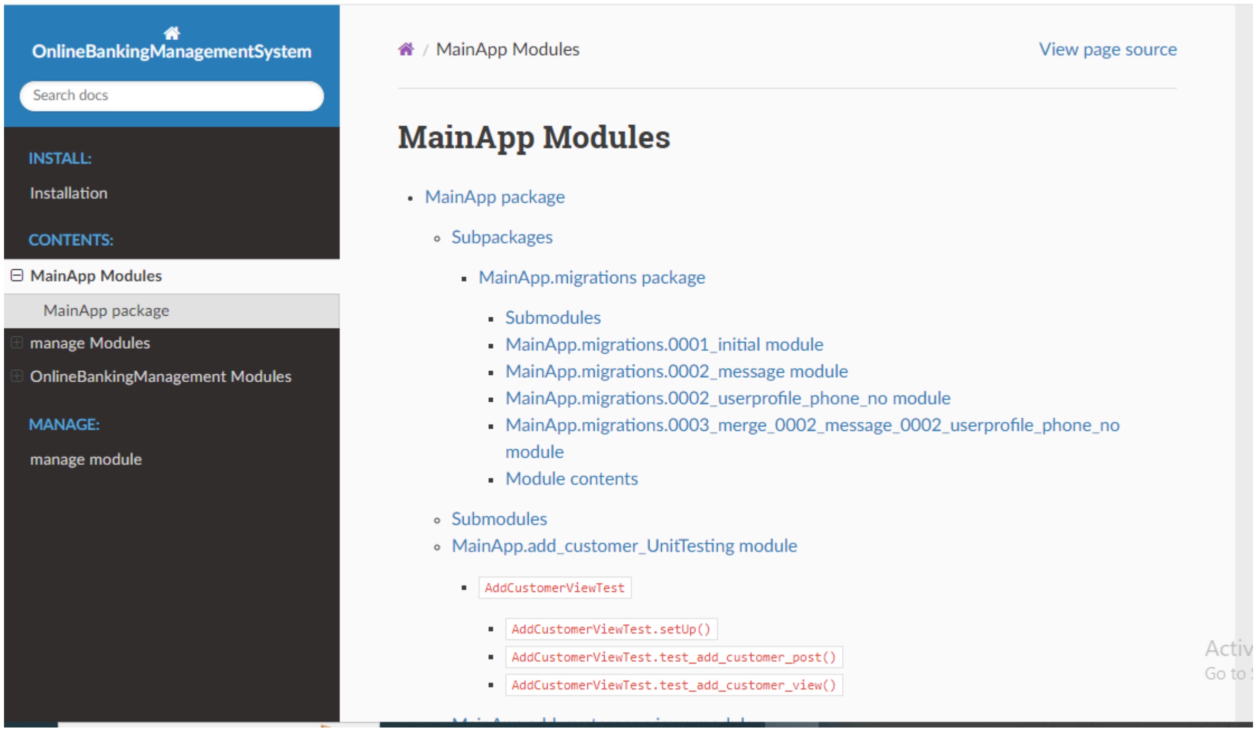
The OnlineBankingManagementSystem Python code is cross-platform (Windows, Mac, Linux), and code for the *View* and *Model* packages has been tested on Windows (10) operating systems.

Dependencies

1. Python version 3.10 or above is recommended.
2. Python IDE: *Visual Studio Code*.

Installing Python

Python can be installed from a number of sources, including https://www.python.org/downloads/. Anaconda is another option which provides additional tools for Python



OnlineBankingManagementSystem

Search docs

INSTALL:

Installation

CONTENTS:

MainApp Modules

MainApp package

manage Modules

OnlineBankingManagement Modules

MANAGE:

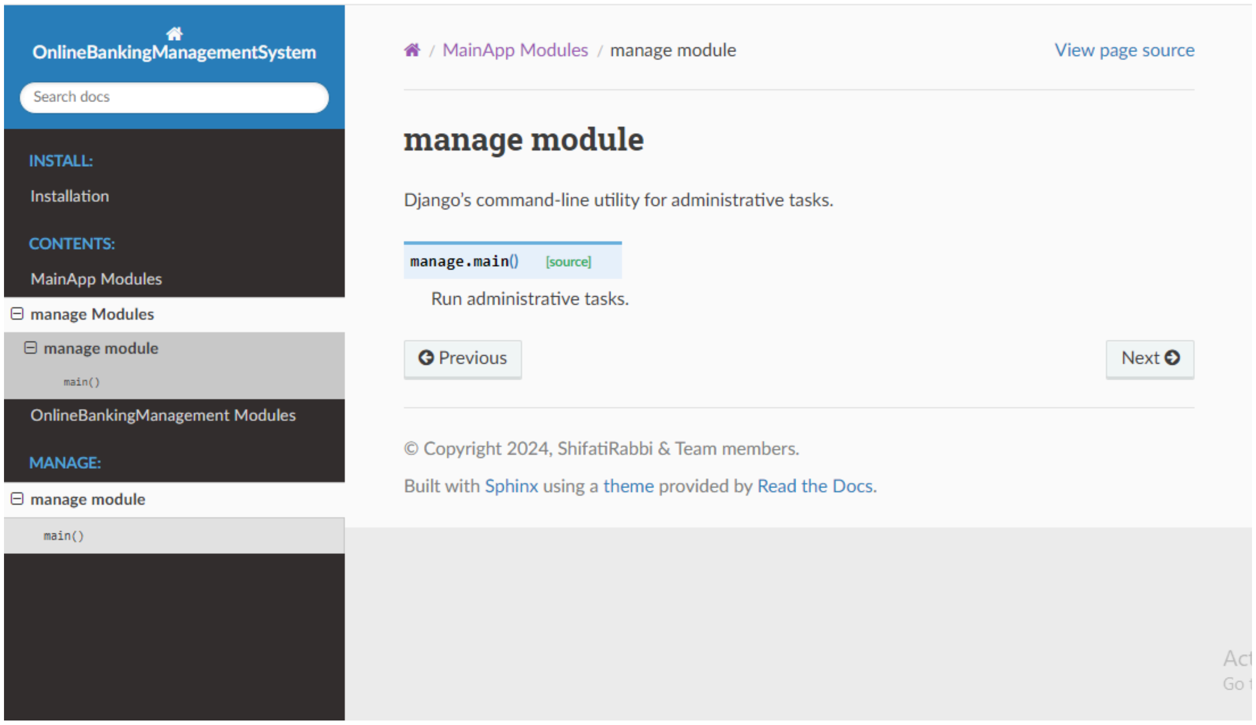
manage module

/ MainApp Modules

View page source

MainApp Modules

- MainApp package
  - Subpackages
    - MainApp.migrations package
      - Submodules
      - MainApp.migrations.0001_initial module
      - MainApp.migrations.0002_message module
      - MainApp.migrations.0002_userprofile_phone_no module
      - MainApp.migrations.0003_merge_0002_message_0002_userprofile_phone_no module
      - Module contents
  - Submodules
  - MainApp.add_customer_UnitTesting module
    - `AddCustomerViewTest`
      - `AddCustomerViewTest.setUp()`
      - `AddCustomerViewTest.test_add_customer_post()`
      - `AddCustomerViewTest.test_add_customer_view()`



MainApp Modules / manage module

View page source

manage module

Django's command-line utility for administrative tasks.

`manage.main()` [source]

Run administrative tasks.

Previous

Next

© Copyright 2024, ShifatiRabbi & Team members.

Built with Sphinx using a theme provided by Read the Docs.

That's all.