

COS 497: Capstone II

Code Inspection Report

(Adapted from Susan Mitchell)

Birdspotter
Code Inspection Report

Table of Contents

	<u>Page</u>
1. Introduction	3
1.1 Purpose of This Document	
1.2 References	
1.3 Coding and Commenting Conventions	
1.4 Defect Checklist	
2. Code Inspection Process	5
2.1 Description	
2.2 Impressions of the Process	
2.3 Inspection Meetings	
3. Modules Inspected	7
4. Defects	8
Appendix A – Client Sign-off	12
Appendix B – Peer Review Sign-off	13
Appendix C – Document Contributions	14

1. Introduction

1.1 Purpose of This Document

The Code Inspection Report is a document designed to provide an overview of the coding standards, procedures, and conventions that we, The Penobscot Development Group, adhere to. This document will also provide a report of each completed module of the project and an analysis of the major and minor defects found in each module. In short, the purpose of this document is not to fix or change any existing code, but instead to define our coding standards and report on how well the project has conformed to those standards so far.

1.2 References

This document references:

- System Requirements Specification Document: [link](#)
- System Design Document: [link](#)
- Prospector linting rules: [link](#)

1.3 Coding and Commenting Conventions

The development team followed the PEP8 style guide conventions while developing the product. All code written is analyzed by the linting tool prospector using its default tools. The specific PEP8 and other rules prospector uses are available in its documentation. See Section 1.2.

1.4 Defect Checklist

Blank lines

- variable naming
- Whitespace
- print() vs. logging() vs messages
- Magic numbers
- string structure

Logic Errors

- Overloaded functions
- Repeated Code
- Error Handling
- Outdated code/artifacts
- Code cleanup
- Testing

Security

- Hardcoded credentials
- Hardcoded file paths
- Permissions not enforced correctly

Documentation

- Docstrings
- Commented out code
- Lack of documentation

2. Code Inspection Process

2.1 Description

Our code inspection process took place over two meetings. First, the team took the entirety of the current project codebase and divided it into 7 modules. Then, the team walked through each file in the module, with the developer(s) who worked on each file explaining what each piece of code does and why it was implemented that way. In response, the rest of the team noted defects with the code that were recorded by the recorder. Later, those notes were used to create a full write-up in this document.

2.2 Impressions of the Process

Overall, this process went fairly well. The review went fairly smoothly, though we did end up adding a couple of items (such as "Permissions not enforced correctly") to our list of defects to look for, which obviously isn't ideal. It also would have been ideal to have more of the application complete farther before the Code Inspection, but thanks to a final sprint by the development team leading up to the code inspection, the application was an almost complete product, only missing one of the major functions (that had run into issues due to new/better information from the client) and a couple of smaller features. Since the code inspection, we have fixed all of the small and medium-sized issues, and have a list of the larger issues that we have yet to tackle in order by priority.

The best module of Birdspotter was by far the accounts module - it had the highest code quality, and the highest level of unit test coverage, so the lowest likelihood of remaining flaws. This isn't particularly surprising - this module was mostly written by the development team manager, and was the first module developed when development began. Furthermore, this module is fairly easy to test, as it is a series of form interactions and module manipulation, which is fairly straightforward to unit test, as all logic is happening within this backend code - very little mocking required.

The worst-off module of Birdspotter was the DataIO module - while it had a similar lack of unit tests than some other modules, its higher complexity and a larger amount of code meant that it has a higher likelihood of defects down the line. To be fair, this is one of the most complex modules in the system, and one of the most difficult modules to write tests for, as it is heavily reliant on external systems, such as the file system and NGINX server, in a way that is difficult to design such that it can be easily mocked or the external dependency can be avoided. This module is definitely going to be the focus of our backend development for further testing and validation.

2.3 Inspection Meetings

All inspection meetings were conducted virtually. The team roles, as well as the meeting dates, are listed below.

Roles:

- Moderator: Jacob Morin
- Scribe: Alex Feren
- Reader: Devin Christianson
- Developers: Devin Christianson, Alex Feren Nick Kania, Kyle Walker

Meeting Dates:

- March 1st 2021, 2:00PM - 3:15PM
- March 3rd 2021, 1:00PM - 2:00PM

3. Modules Inspected

Accounts

The accounts module contains all of the code related to creating, modifying, and validating user accounts on the web application.

DataIO

The dataIO module handles organizing, importing, and exporting data from the database, including the forms for importing and exporting shapefiles through the web interface.

Dataviz

The dataviz module controls the page and related systems for displaying the graphics and other visualizations of the data.

Management

The management module is the system that controls the management of user permissions and administration permissions for the web application.

Map

The map module contains the code for the web application's map view page, including the systems for gathering the map tiles, automatically focusing the globe map on the target area, and displaying the data using collapsing points.

Templates

The templates module is responsible for the .html templates that are used as a base for the web application's pages. This includes the base for the data visualization, map, navbar, etc.

Configuration

The configuration module covers all of the configuration files for the various systems that exist in the web application and its development tools. This includes configuration for the web server, the code linter, and the version control.

4. Defects

Module: Accounts

Defect Category	File(s)	Description
Docstrings	accounts/models.py	The function GroupRequest requires a docstring.
Docstrings	accounts/views.py	All functions in views.py are missing required docstrings.
Docstrings	accounts/tests.py	The form validator and api tests are required missing docstrings.

Module: DataIO

Defect Category	File(s)	Description
Blank Lines	dataio/scripts/import_handler.py dataio/model.py	Function formatting in these files breaks convention by not including a blank line in between function definitions.
Variable Naming	dataio/scripts/import_handler.py	Functions intended to be private should be named starting with an underscore.
Logging	dataio/scripts/import_handler.py	This file uses the print function to log information instead of the preferred logging method.
Docstrings	dataio/scripts/get_user_datasets.py	The function get_dataset_data lacks the required docstring.
Commented code	dataio/scripts/get_user_datasets.py	This file contains commented lines of code, breaking convention.
Outdated code	dataio/scripts/get_user_datasets.py dataio/forms.py	The function get_user_datasets uses an outdated method to retrieve data. The __init__() function in forms.py needs review.
Testing	dataio/tests.py	This module is lacking unit tests.

Module: Dataviz

Defect Category	File(s)	Description
Whitespace	dataviz/views.py	Inline comments must start on the same column as the code above or beneath it.
Outdated code	dataviz/models.py	The models in this module are duplicates of the models in the dataIO module and should be removed. The dataIO models should be used instead.
Outdated code	dataviz/views.py	The method of checking user permission levels is outdated in views.py.
Testing	dataviz/tests.py	This module is lacking unit tests.

Module: Management

Defect Category	File(s)	Description
Lacking Documentation	management/commands/create_groups.py	This file is missing inline documentation to explain the code within it.

Module: Map

Defect Category	File(s)	Description
Magic numbers	map/views.py	The map view has a hardcoded height for elements. This height should be responsive depending on the device viewing it.
Docstrings	map/views.py	The docstrings in views.py should be more descriptive about the functionality of the code they describe.

Module: Templates

Defect Category	File(s)	Description
Hardcoded credentials	templates/index.html	The edit and export functionalities of the index.html template do not check the current permission level of the user. This does not conform to the project requirements.
Outdated code	templates/template.html	This template is unused and should be removed.
Outdated code	templates/result.html	Using result.html to display a status message is obsolete since the inclusion of the toaster plugin, which now handles status messages.

Module: Configuration

Defect Category	File(s)	Description
Logging	views.py	This file uses the print function to log information instead of the preferred logging method.
Magic Numbers	settings.py	The health_check function in this file uses default thresholds for testing status that aren't defined or explained elsewhere.
Hardcoded Credentials	settings.py	This file has hardcoded credentials for accessing the SLURM and IMAP systems.
Lacking Documentation	settings.py	This file has only minimal documentation, lacking functional descriptions of the code within.
Outdated Code	sample_data.csv sample_data_table.csv	These files are artifacts left over from early development testing. They are now unused.
Outdated Code	prospector.yaml	In the prospector.yaml file, the dataviz/ directory is marked to be ignored by the linter. This is

		outdated as the dataviz module has been implemented.
--	--	--


Appendix A – Client Review Sign-off

By signing below, the client acknowledges that they have attended the Preliminary Software Review and have reviewed all articles described in sections 1, 2, 3, and 4 of this document.

Cynthia S. Loftin date: 3/17/2021

Appendix B – Team Review Sign-off

By signing below, all Penobscot Development Group members acknowledge that they have reviewed and understand all articles described in sections 1, 2, 3, and 4.

 date: 3/17/21

Devin Christianson date: 3/17/2021

Kyle Walker date: 3/17/2021

Nick Kania date: 3/17/2021

_____ date: _____

Appendix B – Document Contributions

The below table describes the Penobscot Development Group members contribution to the document for internal purposes.

60% - Jacob Morin
10% - Devin Christianson
10% - Alexandre Feren
10% - Kyle Walker
10% - Nick Kania