Integral Solutions

University of Maine

COS 497 | Capstone I

Code Inspection Report

Digital Program of Study Approval System

Client: Doctor Harlan Onsrud

Team Members:
Vincent King | Peter Riehl | Mac Creamer
Liam Blair | Aaron Wilde

Professor:
Terry Yoo

March 7, 2022

Version 1

Product Name
Code Inspection Report

Table of Contents

1. Introduction

As a means of providing to our client, Harlan Onsrud, a complete and transparent analysis of our [Integral Solutions] code progress to date, we have created this document. We recognize that we have not yet completed all of the modules necessary to have full site functionality at the time of writing this document, however, we intend to keep improving and adding on to our site as we are able.

1.1 Purpose of This Document

The document has a dual purpose. Firstly it serves as a notice from Integral Solutions towards our client, Professor Harlan Onsrud. Additionally, this document outlines the work that needs to be done on improving the site, whose name is Digital Program of Study Approval System (POSAS). The defects listed are meant not to scare those who are reading this document, but instead to give clarity and transparency as to what needs to be improved to ensure a quality end product. Due to this project intending to be largely a "proof of concept" for the idea, the intended readership should be for both our client and for the high-ranking academic staff at the University of Maine, with the former being prioritized to ensure absolute clarity for agility's sake.

1.2 References

- "UMaine Graduate Student Program of Study Creation and Approval System" Proposal
    - *Author: Harlan Onsrud*
    - *Date: September 2021*
- "UMaine Graduate Student Program of Study Creation and Approval System" SRS
    - *Author: Integral Solutions*
    - *Date: October 2021*
- "UMaine Graduate Student Program of Study Creation and Approval System" SDD
    - *Author: Integral Solutions*
    - *Date: November 2021*
- "UMaine Graduate Student Program of Study Creation and Approval System" UIDD
    - *Author: Integral Solutions*
    - *Date: November 28, 2021*
- "UMaine Graduate Student Program of Study Creation and Approval System" CDRD
    - *Author: Integral Solutions*
    - *Date: December 2021*

1.3 Coding and Commenting Conventions

Our coding conventions were straightforward in nature. We used two main programming languages: Python and HTML. Python was used to program the backend while HTML was used to program the frontend. We also employed CSS to style the frontend with significant elements coloring and formatting the HTML that was shown on each page. Python enforced it's standard indentation style conventions, meaning code within loops, classes, and functions had to be indented one tab to help the reader understand which code belongs to what structure. An advantage of Django is that it uses Python which forces the indentation style to assist programmers with control flow and the blocks of code they develop. These elements help debug and review the code when problems arise. We also used an indentation style for the HTML coding for very similar reasons. While not strictly enforced like the Python counterpart within Django, the standard HTML practice is the indentation style. Much like with Python, the choice of the indentation style helps keep sections together and assist the reader in understanding what parts of the code are grouped together. As is standard within the software development profession, we opted to provide names to variables that were indicative of their intended purpose. While HTML does not natively provide many opportunities to use custom variables, Django opened the door for variable names which were representative of the designed intent. Python does natively provide many opportunities which afforded us the ability to properly name variables and ensure clarity.

The commenting style we opted for is one that has elements of standardization, while allowing for some creative freedom towards us developers. For every short function that was defined, we opted to provide a very brief overview of what that function served to do. Larger functions utilized several comments dispersed throughout the function, with the expectation that code that was "hacky" or felt off in nature would include comments on this noted nature and suggest improvements going forward into the future. As for HTML, the structure of the language leads very little to the imagination, and as such comments aren't as needed. However, due to the nature of Django and utilizing HTML documents for template purposes, comments were needed for two separate reasons. The first reason being that each HTML document was passed in a context with variables that could be openly referenced within the document, something that we decided needed to be clearly and completely expressed at the top of the HTML document. The second reason was that these variables present in the context would be referenced in logic and loop statements, and as such explaining them in a different light would be necessary. Single line comments were frequently employed, with multi-line comments being used when something had the potential to be particularly unclear.

As for our development conventions, we opted to keep things relatively simple and not dive too deep into GitHub conventions. The format we followed was to create a branch off from the master branch, call it "<nickname>-dev" and work on our own issues within these branches. Other developers would then review the branch, and, if it looked good, gave permission for it to be merged in with the master branch.

1.4 Defect Checklist

Note, not all potential defects are present within the code, this serves as a list of potential current and future defects that could be encountered.

Potential Module Defects		
Violation Name	Category	Explanation
Incomplete Content	Coding Convention	Module under inspection has elements that are not yet complete
Improper coding style	Coding Convention	Element or module utilizes techniques that feel they could be improved or seem "hacky" in nature
Documentation Mismatch	Coding Convention	Model, form, or other element does not match one or more articles of documentation
Missing Integration	Logic Errors	Does not utilize Django at all (HTML only)
Django Violation	Logic Errors	Does not follow proper Django integration conventions (HTML only)
Missing Components	Logic Errors	Module is missing key HTML elements necessary for functionality (HTML only)
Injection Attack Vulnerability	Security Oversights	Editing the HTML allows for certain aspects of the script to be edited and subsequently altered to potentially access database
Permissions Error	Security Oversights	Users are able to perform functions that they otherwise should not have privileges to do
Authentication Error	Security Oversights	Modules of the site are either accessible or not accessible when they should/shouldn't be
Incomplete Sign Off	Security Oversights	A form could be possible to be signed off even when not all necessary parties may have given approval
Lack of Documentation	Commenting	General lack of comments throughout a given file
No context comments	Commenting	No context information on variables used at the top of a template HTML file
Unprofessional comments	Commenting	Comments provided do not maintain a standard of professionalism with the language used
Server Incompatibility	Miscellaneous	Some element of our website when executed causes an issue not described above that impacts the performance of the virtual server that the platform is being hosted on
Other	Miscellaneous	Some element of a module, or a module as a whole, includes some other defect not mentioned or categorized above. Please provide full explanation and description of said bug in a locatable notes area

2. Code Inspection Process

2.1 Description

Our code inspection process at Integral Solutions followed a model that straddled the line between formal inspection and a walkthrough. For the first hour of our meeting, code that was written up followed the inspection format, where our Reader, Mac, discussed the code and paraphrased elements of it, walking through the entirety of the file that we were inspecting, while the Inspectors, Liam and Vincent, took notes and provided feedback on the inspected file after the reading had been completed. Throughout the entire meeting we had established, Aaron and Pete had assumed their roles of Recorder and Moderator, respectively. With the notes and recorded information that Aaron took, we were able to complete section 1.3 in its entirety. Pete, as moderator, made sure to ask clarifying questions to expand inspector understanding of the modules that had been written up.

The second hour followed a more hybrid format between a walkthrough and inspection, where we still had Pete and Aaron assuming their roles of Moderator and Recorder, but Liam and Mac swapped roles when it came to discussing files that each had predominantly worked on. Vincent still assumed his role as inspector, however, Mac took on the role of inspector when Liam was reading files he worked on, and vice versa with Mac for files he wrote up instead. We opted into this diversion of the inspection process because it gave those who had not yet touched certain types of files the ability to better understand how those specific files were written up. Ultimately, this route brought a more natural flow to the conversation and led to more discussion on how the code should be written going forward.

2.2 Impressions of the Process

Our code inspection process here at Integral Solutions was effective in finding defects in our software as well as in our development process on issues such as lack of documentation, potential security oversights, as well as logical errors within the software. The recorder for our code inspection, Aaron, documented and organized each of these defects throughout our code inspection, resulting in a well organized spreadsheet document containing each defect uncovered in our code inspection. This aspect of our code inspection was a great success, giving our group a well organized spreadsheet which exposes our softwares defects, allowing us to go down the list and correct them. Alongside uncovering defects in our software and development, our code inspection process succeeded in helping members in the group better understand and grasp how to develop software in HTML and how it all connects to our database and server. In our code inspection process at the beginning of each new file, Mac would read through the file in its entirety in an effort to explain each line of code and how the code functioned. This walkthrough allowed the inspectors to search for defects in the software, as well as served as a sort of HTML learning process for members of the group which were unsure of their HTML development skills up until this point. Additionally, Liam switching over to the role of reader towards the end of our

code inspection to discuss files that involved the database and server allowed group members to better understand how all of our software connects and comes together for our product as a whole.

We reviewed 19 modules during our inspection. Some at varying levels of completion. With Django, some of the files are automatically generated and some files may only have a couple of lines of code. A couple of these files include the posas/urls.py, accounts/urls.py, and settings.py. These are least likely to contain errors since the urls.py files only have a couple of lines and the settings.py file only has the Django settings. Other Django files like the accounts/views.py, accounts/managers.py, and accounts/forms.py consist of more complex python code to communicate between the HTML template and the Django/PostgreSQL backend. These modules are the most likely to contain errors because any discrepancies in variable names between the database models and the variables in the python code will cause an error. These files need to handle all the user's personal and login information as well as all the fields in each of the Graduate program of study forms. That's a lot of data and variables that increase the chance of flaws in the code. The majority of the flaws we found in our HTML templates were small coding convention problems. Django has a little different syntax and functions in their template system that is different from traditional HTML. Most of our problems arose from not following Django's specific template conventions. Those flaws are easy to fix and we had a good discussion about the conventions and nuances of the Django template language. Overall it was a very informative meeting and we learned a lot about Django and our product.

2.3 Inspection Meetings

Date	Location	Time Start	Time End	Moderator	Reader(s)	Inspector(s)	Recorder
3/6/22	Zoom	2 PM EST	4 PM EST	Pete	Mac, Liam	Liam, Vince, Mac	Aaron

There was precisely one meeting that Integral Solutions had for Code Inspection. During this meeting, all members were in attendance, with the roles being as follows above. All modules referenced within section 3 are the modules inspected during this meeting, with this being all of the modules that are currently written up.

3. Modules Inspected

3.1 Incomplete/Unstarted Modules

Modules Not Started/Not Completed		
Name	Description	Expected Completion Date
User -> General -> Log In	Only frontend needs to be completed	3/18/2022
User -> Student -> Create DPOS	Frontend for form layout, backend for creation of forms	4/1/2022
User -> Student -> Submit DPOS	Frontend for comment form layout, backend for comment form	4/1/2022
User -> Student -> View DPOS	See Create DPOS, GET method instead of POST method	4/1/2022
User -> Student -> Modify DPOS	See View DPOS, POST method on the viewed form instead of GET	4/1/2022
User -> Student -> Save DPOS	See Create DPOS, POST Method that doesn't take the user to next page	4/1/2022
User -> All -> Resend Reminder to Review DPOS	Add option to main page, do backend work to pull and send messages	4/1/2022
User -> Non-Student -> View Student DPOS	Same as student, but set permissions so only specific students can be viewed	4/8/2022
User -> Non-Student -> Comment on Student DPOS	Similar to "Submit DPOS", don't attach approval status, just the message	4/8/2022
User -> Non-Student -> Approval Status for DPOS	Similar to Comment on DPOS, Attach approval status	4/8/2022
User -> Advisor/Grad Coord -> Modify Student DPOS	Similar to modify DPOS for student, set permissions similar to view DPOS	4/8/2022
Server -> Save DPOS	Automatically POST form to save it on the backend, don't update web page	4/1/2022
Server -> Notify Users of Updates	Backend, get user emails and then execute the email function of Django	4/8/2022
Server -> Verify Password -> Display Error Message	Mimic create account form, display error message on failed pass verification	3/18/2022

3.2 Reviewed Modules

No differences between the reviewed modules and the modular unit's design from the SDD were noticed or reported. Therefore, we opt not to report the differences as there we believe there were none.

Reviewed Modules		
Name	Description	SDD Reference (Section 2, Decomp. Diagram)
templates/forms/coursework/coursework.html	Frontend for coursework (PhD/Mast)	User -> Student -> Create DPOS
templates/forms/coursework/gradcoursework.html	Frontend for coursework (Certificate)	User -> Student -> Create DPOS
templates/forms/gradCertificateForm.html	Frontend for Certificate Form	User -> Student -> Create DPOS
templates/forms/mastersDegreeForm.html	Frontend for Masters Degree Form	User -> Student -> Create DPOS
templates/forms/phdDegreeForm.html	Frontend for PhD Degree Form	User -> Student -> Create DPOS
templates/registration/login.html	Frontend for Logging In	User -> General -> Log In
templates/registration/register.html	Frontend for Registration	User -> General -> Create Account
templates/base.html	Frontend for site consistency feeling	Client Side Functionality
templates/home.html	Frontend for home page (visible on log in)	User -> General
accounts/admin.py	Backend for account administration	User -> General -> Create Account
accounts/forms.py	Backend/Frontend for Registration Form	User -> General -> Create Account
accounts/managers.py	Backend for Registration for Account Creation	User -> General -> Create Account
accounts/models.py	Backend for communication of Database Models	User -> General -> Create Account
accounts/tests.py	Backend for testing the full "accounts" module	User -> General -> Create Account
accounts/urls.py	Backend for communicating url paths with the site	Client Side Functionality
accounts/views.py	Backend for creating the frontend page context for registration	User -> General -> Create Account
forms/models.py	Backend for communication of Database Models	User -> Student -> Create DPOS
posas/settings.py	Backend that controls system wide settings and variables	Client Side Functionality
posas/urls.py	Backend for registering all other url paths for site	Client Side Functionality

4. Defects

Present the following for each defect found, including coding and commenting convention noncompliance. A tabular format is strongly suggested.

Categories from above:

- Correctness
- Coding Convention
- Logic Error
- Security Oversights
- Commenting
- User Friendliness
- Miscellaneous

Located Defects		
Name of the Module	Description	Category
templates/forms/coursework/coursework.html templates/forms/coursework/gradcoursework.html	Incomplete Content, Missing Components, Lack of Documentation	Coding Convention, Logic Errors, Commenting
templates/forms/gradCertificateForm.html templates/forms/mastersDegreeForm.html templates/forms/phdDegreeeForm.html	Incomplete Content, Lack of Documentation, Django Violation	Coding Convention, Commenting, Logic Error
templates/registration/register.html	Injection Attack Vulnerability	Security Oversights
templates/registration/login.html	Injection Attack Vulnerability, Incomplete Content	Security Oversights, Coding Convention
templates/base.html	Incomplete Content	Coding Convention
templates/home.html	Authentication Error (Homepage visibility)	Security Oversights
accounts/models.py accounts/views.py	Documentation Mismatch (Lack of Class Features) Improper coding style (Inefficient Code)	Coding Convention

Appendix A – Team Review Sign-off

By signing below, all involved parties confirm that they have reviewed the contents of this document. Additionally, all parties will confirm that they have agreed on the document's content and format.

Team Member Comments:

1. _____
   _____
   _____
2. _____
   _____
   _____
3. _____
   _____
   _____
4. _____
   _____
   _____
5. _____
   _____
   _____

Customer Name: _____ Customer Signature: _____

Date of Signature: _____

Team Names:	Team Signatures:	Date of Signatures:
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

Appendix B – Document Contributions

Mackenzie Creamer - 65%
- Sections 1, 1.1, 1.3, 1.4
- Section 2.1, 2.3
- Section 3
- Section 4

Peter Riehl - 7.5%
- Section 1.3

Vincent King - 10%
- Section 2.2

Liam Blair - 10%
- Section 2.2

Aaron Wilde - 7.5%
- Section 4