Software Lab EE 461L

Team A7: Board Game Database

TEAM A7

11.30.2020

Cedrik Ho, Allegra Thomas, Grant Ross, Sanne Bloemsma

INFORMATION

GCP Deployed App: https://fall-2020-ee461l-teama7.uc.r.appspot.com/
Github Repo Link: https://github.com/UT-SWLab/TeamA7
Team Canvas Group ID: A7

Team Members:

Allegra Thomas (at35737)
Email: allegrathomas@utexas.edu
GitHub: AllegraThomas

Cedrik Ho (ch45935)
Email: cedrikho@utexas.edu
Github: CedrikHo

Sanne Bloemsma (scb2936)
Email: sbloemsma@utexas.edu
Github: sbloemsma

Grant Ross (ghr344)
Email: ghross@utexas.edu
Github: Grant-Ross

MOTIVATION AND USERS

The motivation of this project was to create a cohesive web application for users to browse board games and be able to find out more about board games, their publishers, and various board game genres. It was, to some degree, inspired by various board game hobby pages. However, it is our hope that this web application will be more intuitive to use and be less fragmented in terms of genres and brands of board games hosted.

We believe that this web application will be useful to a wide range of demographics and be particularly useful to individuals who regularly play board games with others. We anticipate that access to this website will also vary from mobile phones, tablets, personal computers, and other various devices with internet connectivity. Thus ease of use and flow of information is highly important to the team.

USER STORIES

Our user stories were all hosted on Github. The following link will show you each of the user stories created and the current state of the project:
https://github.com/UT-SWLab/TeamA7/projects/1?fullscreen=true

User stories covered in Phase I:

As a user, I want to be able to travel through all main pages of the site without issue (via the navigation bar) so I can intuitively get the information I need.
Engineering Perspective: We want all areas of the site to be easily accessible and users to be able to easily switch between them.
Estimated time: 1 hour
Actual time: 1 hour

As a user, I want to see a page with a list of board games and links to each board game's page
Estimated time: 1 hour
Actual completion time: 1 hour

As a user, I want to see 3 separate board game pages with links to their respective publishers and genres
Estimated time: 4 hour
Actual completion time: 5 hour

As a user, I want to start on a homepage when I start navigating the site and see a clean opening page for a better user experience
Estimated time: 3 hour
Actual completion time: 3 hour

As a user, I want to see a list of publishers page so I can explore more information and games about the publisher of games I've liked before
Estimated time: 7 hour
Actual completion time: 7 hour

User stories covered in Phase II:

As a user, I want to see accurate information from a restful API for each page
Estimated time: 5 hours
Actual time: 6 hours

As a user, I want to see images of publisher logos when I go to publishers' pages, so I can more easily recognize publishers
Estimated time: 1 hour
Actual completion time: 1 hour

As a user, I want to see accurate information from a restful API for each page in order to feel the site is a more credible source for the information I'm looking for
Estimated time: 5 hours
Actual completion time: 6 hours

User stories revisited in Phase II:
As a user, I want the home page to be aesthetically pleasing (no blank page, some pictures)

As a user, I want to be able to look at information about a specific game/genre/publisher and see links to other information related to that specific game/genre/publisher in order for the site to be useful to me.

Engineering Perspective: Each collection will have 2 lists of objects from the other 2 models to show how they are related. We will populate the pages with these objects and set up links between them.
Estimated time: 4 hours
Actual time: 7 hours

User stories covered in Phase III:

As a user, I want to select sorting or filtering with one click, and I want to filter and sort in two clicks only.
Estimated time: 10 hrs
Actual time: 6hrs

As a user, I want to be able to filter on one single click from a checkbox. Checkbox will stay to show selected filter
Estimated time: 7 hrs
Actual time: 7hrs

As a user, I want to be able to clear my filter and revert to the base page. I also want to be able to stack filters.
Estimated time: 8hrs
Actual time: 10hr (Stacking ended up not being requirement after further clarification)

As a user, I want the site to be clean and readable as well as aesthetically pleasing in order to navigate the site better and overall have a more positive experience with the site.
Estimated Time: 3hrs
Actual Time: 3hrs

As a user, I want to sort board games/genres/publishers based on alphabetical order so I can find the instances I want easier
Estimated Time: 4hrs
Actual Time: 4hrs

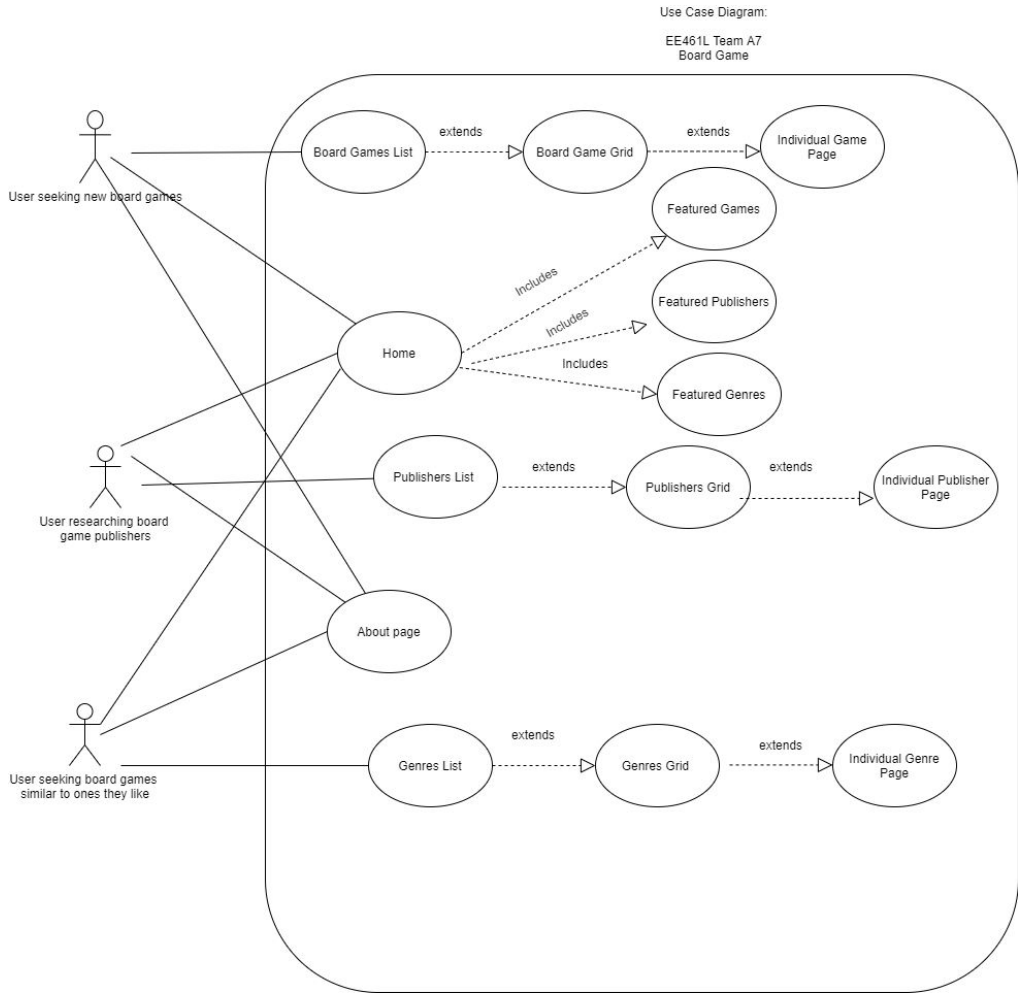
As a user, I want to be able to access lots of accurate information about board games/genres/publishers so that I can learn more about games/genres/publishers that might interest me.
Estimated Time: 10 hrs
Actual Time: 6 hrs

As a user I want to be able to search for board games/genres/publishers by name so I can find information about that instance quickly.
Estimated Time: 5hrs
Actual Time: 5hrs

As a user I want to filter by field types of games/genres/publishers so I can find instances that match my preferences.
Estimated time: 5hrs
Actual Time: 6hrs

As a user I want to be able to search for board games by a particular genre, description phrase, (other instance attributes) in order to find games that I would want to play.
Estimated Time: 6hrs
Actual Time: 7hrs

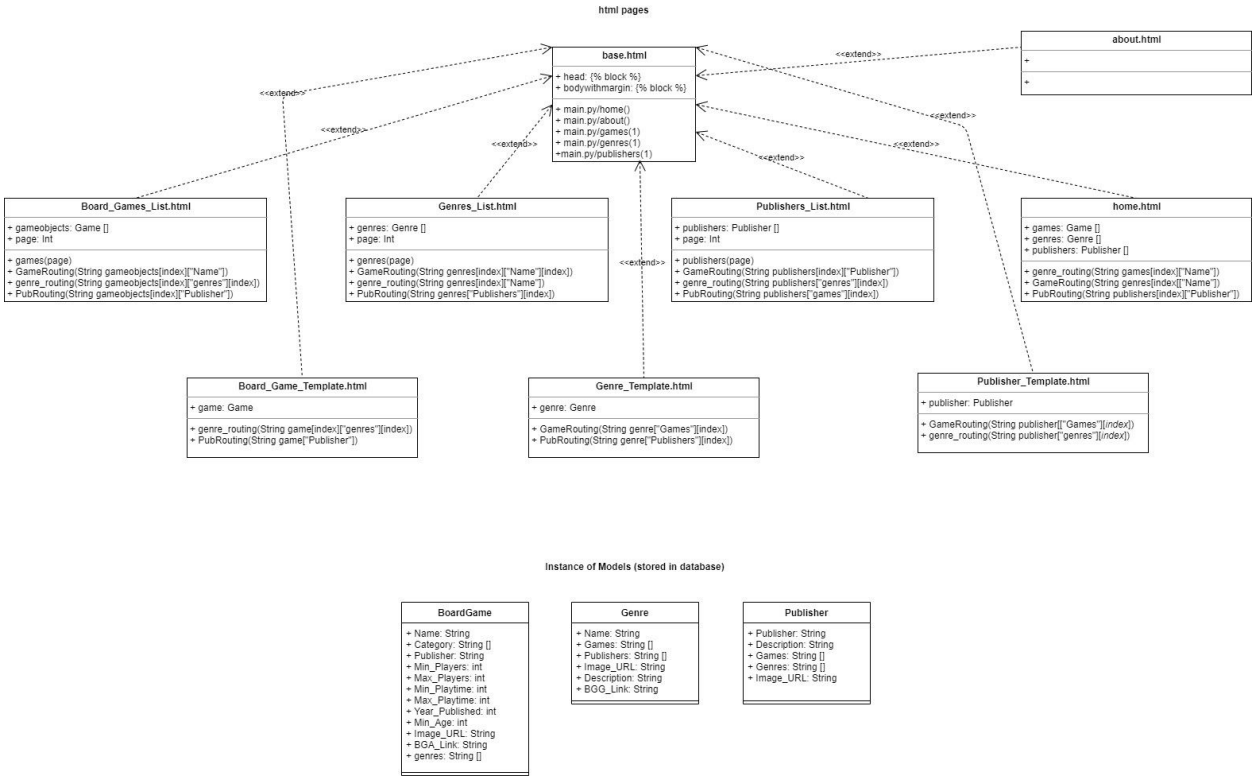
USE CASE DIAGRAM



DESIGN

The design of this website includes a home page with general information about site content, an About page with information about the website itself and the team behind it, and 3 list pages for each of the 3 models in our database. There is also a header on each page that the user can use to get to any of these pages so that every other page is readily available.

UML Diagram



When designing our site, we thought a lot about the html pages as classes or objects with variables passed in from our main.py file using jinja2 and linking, routing, and POST methods as a way of implementing methods in each html diagram. In the end the class diagram for our site looks like the above. As we move forward in Phase III, it is likely we will revisit this diagram and make changes. For starters, as it stands accessing a specific game via our site is dependent on accessing a page where there is a visible link to its instance. In other words, you would not be able to type in the site url and add the name of a game at the end in the same format our site routes to and get to the instance of that game. For Phase III we will approach new changes with the addition of a behavioral state machine diagram while continuing to update our current diagram.

MODELS

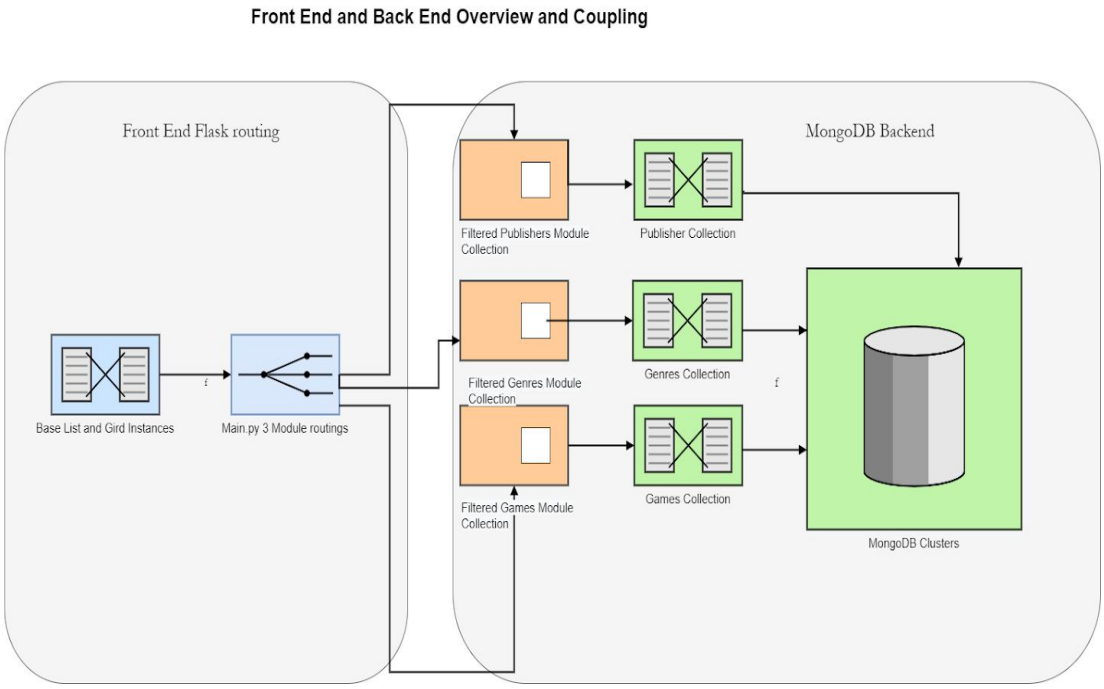
Board Games: This model contains names, general information, pictures, and gameplay descriptions of various board games. Board games are published by publishers, and can be categorized into one or more distinct genres. We pulled pictures of the games using the API for Board Game Atlas.

Publishers: This model contains the names, general information, logos, and history of various board game publishers. These publishers produce and sometimes design different board games, and often specialize in publishing games under one or more genres. We used Google's Custom Search JSON API to generate images for the publisher entries, and manually added some that our search algorithm didn't get right.

Genres: This model contains names, descriptions, and pictures of various established genres of board games. These genres can categorize broader groups of board games with similar aspects, and publishers usually specialize in producing board games under certain genres. We used Google's Custom Search JSON API to generate images for the genre entries.

Diagram Coupling of Front End and Back End

This diagram gives a high level breakdown of the collections stored in MongoDB and how filtered collections are passed on to List instances on the front end. Each Collection is filtered independently and each filtered collection has a super collection that is defined by the module that it sources. The MongoDB cluster is a single DB that hosts all 3 super collections. Filtered collections are written and destroyed because if we were to store them we would have to store every permutation of filtering and sorting possible. This issue would grow exponentially if stackable filters were implemented.



TOOLS AND FRAMEWORKS

The main tools and frameworks for this project are as follows:

Python Flask: Used in Phase I, Phase II, Phase III, and Phase IV for app routing in order to create links and each page of the site.

BootStrap4: Used in Phase I, Phase II, Phase III, and Phase IV to create basic CSS structure and mobile development.

GCP (Google Cloud Platform): Used in Phase I, Phase II, Phase III, and Phase IV to host the deployed site and also a third party cloud storage solution for code and future data. It was also used to obtain the license key for Google JSON Search API.

GitHub: Used in Phase I, Phase II, Phase III, and Phase IV for version control, team management, and as a host for issue tracking and user stories.

Javascript jQuery: Used in Phase III and Phase IV for button controls in list pages and for search result page pagination, filtering, and advanced searching.

Python unittest: Used in Phase III and Phase IV for testing Flask routing and functionality as well as searching and filtering functions .

Selenium: Used in Phase II, Phase III, and Phase IV for testing the site and button functionality, particularly when the application was deployed.

MongoDB and pymongo: Used in Phase II, Phase III, and Phase IV to create the database and collections for each of our three models as well as searching, sorting, and filtering those databases.

PHASE I REFLECTION

This initial phase was a large first step in developing a cohesive and modular web application. The team as a whole was firm in making decisions and few conflicts arose.

As a team we knew coming in that none of us had much experience with building larger web applications. Therefore early on it was not too surprising that we had trouble with creating a solid basic framework with correct inheritance. Specifically we had issues with extending templates without overriding existing content and achieving consistency of grids and CSS elements across model pages. To solve these issues, we re-worked the Flask framework from the Bootstrap4 tutorials and created a default CSS sheet.

We struggled early on with creating a basic file that could easily be given to each of our models. This lack of planning caused some blockers for other team members early on. Once we adopted a more religious use of GitHub's user stories and a true Scrum asking the three key questions, these issues were immediately solved. There were also some unexpected issues in terms of element behaviours and resizing when the page was not at its default width. We created a larger template and standardized cards width and height to create the proper ratios and regrouped data so that growth of the page was as expected.

When deploying the app there was some confusion amongst naming of the python file, so after weeks of using "app.py" as the name of our file in order for it to properly deploy on GCP and be compatible with the ".yaml" file it had to be renamed to "main.py." Although it caused some confusion initially it was easily resolved after reading some documentation for GCP.

Overall, I believe that due to our lack of experience in web development, we struggled to predict certain issues ahead of time. The rather quick and responsive nature of web applications is a new area of software development that the team as a whole has had to get adjusted to. We feel that our use of GitHub issue tracking and clear user stories were very beneficial in helping create a quick and straightforward process for development goals. We are excited and believe that Phase II will be even more expansive and rich in information and features.

Despite the challenges that arose, we were still satisfied with some aspects of our implementation and workflow. Our weekly stand-ups helped us track progress and stay accountable, and our fair breakdown of work based on the models chosen helped us accomplish our goals in parallel. We believe that each individual will be considered the "manager" of the model they worked on in Phase I, since we found a clear assignment of responsibility for a specific model greatly improves communication and productivity. Also, interactions with APIs went well, as we have registered our company with the API providers and access to the APIs seems to work fine. While we did hard code information, we have a clear path to filling out all three models using our APIs.

As stated earlier our team work structure and communication has been a source of pride and success for the team as a whole. We believe that our team contract has been enforced and integral

to our quick resolutions and continued respect for each other's ideas and work. Overall, this reflection has been greatly beneficial in helping the team analyze the issues we have overcome and the weaknesses we still hold going forward into phase II.

Five Things We Struggled With:

1. Achieving consistency across model pages
2. Extending templates without overriding existing content
3. Adjusting page content with page resizing
4. Using one virtual environment between team members
5. Predicting issues ahead of time

Five Things That Worked Well:

1. Dividing work between models
2. Getting information from sources with APIs
3. Setting up the website using the Bootstrap tutorial from class
4. Syncing code with GitHub
5. Helping each other with code issues in meetings

PHASE II REFLECTION

This phase of the project was much more difficult for us. While the requirements had risen in complexity from Phase I, we also struggled due to a combination of poor planning, a lack of communication, and procrastination.

After our success in Phase I, our team was ready and eager to begin planning for Phase II. This initial planning was brief, however, and our proposed division of the work was not fleshed out enough. As a result, the design of our project going into Phase II was not fully understood or unanimously agreed upon by our team. This meant that while we had a general idea of who would be working where, the specific files each member would focus on were unclear. Another thing we did not plan well was the implementation of the database. We implemented one of our models early on, but the other two were added way too late. We failed to notice that one of the APIs we had intended to use for one of our models limited usage from unverified organizations, meaning it was no longer a feasible option for our project. Because of this, we had to quickly find another API, which we thankfully did. However, we agree that this was unnecessary last minute stress, and a more rigorous research plan and earlier implementation would have been less stressful for all parties.

We believe that we should have focused more on creating the databases early on so that they could be seamlessly implemented as we built more pages of the site. What we ended up doing instead was building pages to get their information without their specific collection, then replacing that information using calls to the individual collections as they were being made. In the end, our project required re-design of database access and at times certain team members were blocked by the lack of fully implemented database collections.

Our initial database collection and REST-API usage for a single page worked well, but it failed to scale horizontally. As we added more information, broken links occurred as did confusion about the accessibility of sub fields from database objects. This issue was solved by the segregation of collections and rerouting REST-API calls for all 3 modules separately. While this is a clean and easy solution, it did create more methods and possible app routes to test.

We believe that more frequent checkpoints with smaller goals will help us in Phase III. Monitoring user stories more frequently will also help. We believe that our discussion on the technical aspect of implementing the user stories needs to be cohesive and in depth.

Despite the challenges that arose, we became more satisfied with our implementation and workflow as Phase II progressed. We were able to satisfy the project requirements and communicated well near the end, and the information and code we wrote this time around was much more standardized. Our testing is simple but we believe it is effective and covers the necessary requirements for the stage of development we are in. We will be much more diligent and communicate more often for Phase III, and we look forward to continuing the maintenance and build of this project.

Five Things We Struggled With:

1. Database management
2. Scaling implementation
3. REST-API call structure
4. Appropriate API calls
5. Communication of division of work

Five Things That Worked Well:

1. Documentation of Issues on Github
2. Quick transition to find a new API
3. Syncing code with GitHub
4. Testing framework
5. Teamwork and communication in the later part

PHASE III REFLECTION

This phase took us longer than our initial assessment. We initially underestimated the assignment due to a sense of strong productivity early on and the deadline extension giving us the illusion of there still being plenty of time to complete the assignment and to repeatedly push our initial deadlines back. Delays on implementing filtering and delays on testing had a significant impact on our failure to meet deadlines we established for this phase.

Filtering functionality took longer than expected and some parts of testing should have been implemented earlier in the Phase III timeline. The initial deadline we set for filtering, searching, sorting, and database refinement was met with basic searching, alphabetical sorting, and improved database content. More questions about the implementation of filtering should have been asked early on in the assignment as there was significant confusion around what was expected for filtering functionality. We should have also timed our team meetings earlier in the week to allow more possible TA office hours the group could take their questions to, as our current timing of Wednesday and Friday missed Monday and Tuesday office hours.

In this Phase, we had better team communication and held frequent stand up meetings throughout. Issues and blockers were dealt with relatively quickly in this phase. We met consistently each week to work on code concerns and issues, and adopted a virtual paired programming approach for certain issues. We did a better job of defining everyone's individual assignments early on, which helped team members better understand how their portions of the assignment would interact with others.

Filtering functionality should have been implemented earlier in particular due to its interaction with sorting. Filtering for all 3 modules turned out to be harder and more tedious than expected, which was somewhat due to contradictory clarifications on Piazza. Issues included slow load times for users and lack of stackable filters. We reverted to single use filters after receiving clarification from our professor.

We should have taken a more incremental approach to testing and added testing for different aspects of the project as soon as they were implemented. The team should have initially created a more detailed breakdown of testing rather than just dividing up the labor and relying on individuals to design testing plans solo. Compared with Phase II, Phase III testing is the most comprehensive, structured, and useful to demonstrating project functionality. For division of labor for Phase III, we divided testing so each team member was responsible for testing code they had not primarily written. This forced us to justify our implementation and ensure the entire team is familiar with most of the code and that code can be further used and modified by other team members with less confusion.

We believe that a more in-depth analysis of the actual implementation of features will allow us to determine a more realistic timeline. Solely dividing the features by team members is not enough. With Phase IV requiring design patterns, we believe that in depth discussion will especially be

useful for the team. This next phase will involve much refactoring of the code base. The fact that we are all very familiar with the code base will help us refactor efficiently.

Our final product for this phase meets the requirements and provides useful information to individuals interested in board games. We actually showed it to some peers who are interested in board games, and they liked what it provided. Moving into Phase IV, we will continue to have stand ups every other day and create our project timeline and division of labor earlier than prior phases.

Five Things We Struggled With:

- Testing earlier in the phase
- Breaking down big tasks into subtasks for more incremental due dates
- Getting clarification on expectations from instructor earlier in the phase
- Speed of filtering early on was too slow
- Breakup of tasks may have been unbalanced workload-wise

Five Things That Worked Well:

- Standup meetings on Slack proved helpful
- More activity on GitHub issues
- Meeting earlier in the week allowed more time for questions to Dr. Eberlein and the TAs
- More usage of jQuery and javascript helped the site function better
- A lot of progress was made overall in paired-programming sessions

PHASE IV REFLECTION

This phase went rather smoothly for us. Every team member was able to complete most of their assigned sub section of work in the allotted time and before deadlines. We did not face too many issues or blockers during this phase. Although there were some issues with filtering causing sporadic duplication of key errors, we were able to find and rectify the bug during one of our team work days over Zoom. Overall, our frequent two hours blocks of group work and paired programming is what led to our success in this phase.

Before we began to code, we defined certain areas of code that needed refactoring into their respective types. Each refactoring target was decided as a team and examined as a team before full implementation. These refactoring discussions minimized the possibility of new bugs, increased our working speed, and aided in readability. Our testing was also refactored well and organized more cohesively so that new developers could easily run tests without deep knowledge of the code base. We had an issue where changes in filtering caused the test to break, but this was easily fixed through quick and open communication. We could have possibly avoided this if we moved our testing timeline back one day to allow for changes to be finalized before we started testing, but we still had plenty of time before the due date after all code changes were made.

We are proud of our searching and filtering refactoring as they focused on a larger part of the code base and arguably contained the most complex functionality. Our filtering and searching code is now much more readable and modular. We believe increased functionality and parameters will be much easier to maintain and implement after our refactoring.

Although the team as a whole was rather new to design patterns, we believe we implemented a good design pattern with an acceptable justification for its use in our code base. Discussing design patterns as a team helped us solidify our understanding of certain types of patterns, and going through the process of identifying and implementing them helped further our understanding of them.

Our communication and organization in this phase was a source of success for the team. While we struggled with deadlines in Phase II and Phase III we believe that we vastly improved in Phase IV. We hope that these refactoring will allow for easy future maintenance and increase the lifetime of the code. Overall we feel that we all learned a lot on this project and are proud of the product that we have created for users.

Five Things We Struggled With:

1. Filtering bugs (Duplication Key)
2. Testing refactoring timeline
3. Broken testing (dead code, changed declaration fields)
4. Some issues with filter refactoring in html
5. Some merge errors

Five Things That Worked Well:

1. 2 hour meeting blocks
2. Improved speed of use for user
3. Syncing code with GitHub
4. Cohesive testing framework
5. Teamwork and communication

Citations and Sources

Data and API Resources

https://boardgamegeek.com/wiki/page/BGG_XML_API2
https://www.boardgameatlas.com/api/docs
https://developers.google.com/custom-search/v1/overview

Educational Resources

This includes documentation we used to learn about the frameworks we were using and plan how to implement our vision for the project.

Planning

"Web Development 101." The Odin Project
www.theodinproject.com/courses/web-development-101/

Python, JSON, and xml

https://requests.readthedocs.io/en/master/
https://www.geeksforgeeks.org/response-json-python-requests/
https://docs.python.org/2/library/xml.etree.elementtree.html
https://www.geeksforgeeks.org/xml-parsing-python/
https://stackoverflow.com/questions/1912434/how-do-i-parse-xml-in-python

Bootstrap and HTML

"BootStrap Tutorial." Bootstrap 4 Tutorial, www.w3schools.com/bootstrap4/default.asp.
https://stackoverflow.com/questions/36003670/how-to-put-a-link-on-a-button-with-bootstrap
https://schier.co/blog/html-templating-output-a-grid-in-a-single-loop
HTML input type="checkbox" (w3schools.com)

Google Cloud Platform

"Google Cloud Platform Tutorial" GCP
https://console.cloud.google.com/getting-started?tutorial=toc

MongoDB

https://www.w3schools.com/python/python_mongodb_getstarted.asp

TESTING OVERVIEW AND GOAL

This section outlines the key areas of testing for each phase of Team A7's EE 461L Project. For each phase of the project, the team will outline what user story criteria we would like to test and how we implemented that testing. We hope that this documentation helps the group identify areas that need to be tested for each phase as well as communicate our ideas on testing to not only the group, but other interested parties.

Scope

The scope for testing in Phase II focused on the fundamental transition of pages and elements on our webpage. We believe that this was the most important part to test as the main purpose of the site depends on this aspect to be functional. The rest API calls were debugged when we populated the database, and we don't believe they should be the focal point of our testing for this Phase.
In Phase III we tested the database more thoroughly as well as all flask routing and searching and filtering functions. We ensured after adding more information to the database that there continue to be no duplicate instances and no instances unconnected to instances of the other models. In Phase IV we focused on refactoring our existing tests and consolidating our test files. We tested all of the same things as in previous phases and updated the tests to work with the refactored code.

Objectives

In testing our site, we want to maintain an aesthetically acceptable and functional site that allows users to navigate between our three models and instances of those models without issues. For testing in Phase II, our focus was predominantly front end and focused on the user experience of the site. Our MongoDB database and the calls we made to our APIs have repeatedly been tested in early stages manually (printing to the console to compare) as well as automated (checking the information we added to the database was also in the API we pulled from).
For Phase III we shifted to more automated comparisons for the database as we collected more and more information. Our goal for Phase III is to ensure the database contains uniquely but connected instances for each model, filtering and sorting work separately as well as together, and that our search function works as expected and for all three models individually as well as when all three models are searched at once. In Phase IV our goal was to ensure that all our tests could still be run with our refactored code base. We tested page routing, searching, sorting, filtering, and database backend functionality as well as front end functionality with Selenium.

Scalability

As the project grows and we contribute more information to each of our models, we will have to create more tests particularly for database uniqueness and connectivity, sorting, searching, and filtering functions. The goal is for our tests to be cumulative, which means that for each phase we will be able to build on what we have already verified.

Code Coverage

The Searching function is not coverable for every possible input string that the site can receive, but it is covered for the key edge cases as well as expected inputs. Otherwise, the possible inputs to the site are primarily finite and mostly covered.

PHASE II TESTING BREAKDOWN

Front End Testing Goals (Based on User Stories)

- All main navigation bar pages are accessible from anywhere on the site
- All list pages load at a reasonable speed and accurately
- Pagination functions as expected
- All instances of a model shown in list pages link to an instance page
- Instance page links link to related links of each model
- Image and window resizing doesn't compromise the readability of the site
- Frequently clicked elements of page (Carousel, Nav Bar) do not lose functionality with repeated usage
- [Phase II] no external sites aside from about page links to sources

Testing Implementation

Front end testing can be rather difficult to automate fully, particularly when it comes to visuals and aesthetics. The team as a whole spent hours going through links, pages, and buttons in an attempt to find flaws or odd issues with the site. The issues, if not instant fixes, were logged into our GitHub issues tracker.

Below are a few issues we found during testing:

Automated Testing (Selenium, Flask library)

- The genres list and publishers list pages were initially modeled off of the games list page. The title name of the genres list page was the same as the board games list page.
- Content was not properly generating on instance pages for publishers.
- Links to instance pages from the genre list page were broken.
- Typos regarding singular and plural of model names (genres vs. genre, games vs. game, publishers vs. publisher) yielded jinja errors.
- Null publishers were found in the board games collection (API to MongoDB storage issue).
- Duplicate genres were covered by publishers.
- Refreshing an instance page of a genre or publisher caused the relevant games links to disappear.

- General visual issues with image and text resizing which were more difficult for our automated testing to detect.

Testing Frameworks

- Selenium
- Flask Test Library
- Python UnitTest

PHASE III TESTING BREAKDOWN

Front End Testing Goals (Based on User Stories)

- “Beautify” by standardizing images, text and background colors modified for improved readability.
- Filter by single click, and have check box note selection of user
- Sort by single click for A-Z, Z-A, and number of players (Sort and filter simultaneously)
- Have multimedia links such as Youtube and Reddit for each module's elements.

Testing Implementation

Front end testing can be rather difficult to automate fully, particularly when it comes to visuals and aesthetics. The team as a whole spent hours going through links, pages, and buttons in an attempt to find flaws or odd issues with the site. The issues, if not instant fixes, were logged into our GitHub issues tracker.

Below are a few issues we found during testing:

- Filters were not all returning accurate results
- Filter checkboxes were not staying checked when the page was refreshed

Selenium Testing:

- Navigation bar functionality
- Every game instance on the list pages links to an instance page (considering pagination)
- Every genre instance on the list pages links to an instance page (considering pagination)
- Every publisher instance on the list pages links to an instance page (considering pagination)

Back End Testing Goals

For back end testing our primary concerns were the reliability of connection to our MongoDB database and the consistency of the collections, cursor objects, and filtered collections that were returned depending on the calls we created.

- Sorting functions all work on their own and collections returned are organized as expected
- Filter functions work and all collections returned are expected
- No empty collections existing for filtering and sorting after commands given to MongoDB
- Connectivity to our MongoDB account was accepted and responded with 200
- Connectivity to each collection was accepted and sorting and filter testing instances used actual collections objects from MongoDB.

Testing Implementation

Flask Testing:

- Test Home Page
- Test Board Games List, Genres List, and Publisher Routing (no filters, all filters, sorting, all combinations of sorting and filtering)

Database Testing:

- No empty documents or empty "Name" fields for all instances of each model
- Each instance is connected to at least one other instance from each other model
- Each instance's set of instances from the other models also exist in the database as instances of their respective models

Filtering Testing:

- Filters all return accurate results
- When no filters are selected, all items in a collection are shown
- After a filter is selected, the page with filtered results loads in a reasonable amount of time (<5 seconds)
- When a filter is selected, its checkbox (and no other checkboxes) stays checked even when the page is refreshed

Testing Frameworks

- Selenium (FrontEnd)
- Flask Test Library (FrontEnd)
- Python UnitTest (Used in BackEnd)

PHASE IV TESTING BREAKDOWN

Phase IV testing focused on refactoring existing tests. During phase III, each team member wrote tests covering different sections of the app - searching, sorting, filtering, and database functionality - so we ended up with individual files for each area of testing. Additionally, we had other separate testing files from phase II, such as front end Selenium tests and app routing tests. For phase IV, we consolidated all tests into 3 files: test_main.py, test_selenium.py, and test_database.py. test_main.py combined all back end app routing, searching, sorting, and filtering tests into one file. test_selenium.py contains all front end testing through Selenium and checks the validity of every instance page on the website. Lastly, test_database.py contains all of our MongoDB database tests.

During phase IV, we also extensively refactored much of our code base. Many tests had to be updated to work with our new code.

Front End Testing Implementation

test_selenium.py
Selenium Testing:

- Navigation bar functionality
- Every game instance on the list pages links to an instance page (considering pagination)
- Every genre instance on the list pages links to an instance page (considering pagination)
- Every publisher instance on the list pages links to an instance page (considering pagination)

Back End Testing Implementation

test_main.py
Flask Testing:

- Test Home Page routing
- Test Board Games List, Genres List, and Publisher List Routing (no filters, all filters, sorting, all combinations of sorting and filtering)

Filter Testing:

- Filters all return accurate results
- When no filters are selected, all items in a collection are displayed

Sort Testing:

- All sort types return correct results (alphabetical, inverse alphabetical, minimum players, minimum playtime)
- When no sorting is applied, all items in the collection are displayed
- Stacking sorting and filtering doesn't cause incorrect results

Search Testing:

- Exact search terms return correct results
- Partial search terms return correct results
- Nonsensical search terms return no results

test_database.py
Database Testing:

- No empty documents or empty "Name" fields for all instances of each model
- Each instance is connected to at least one other instance from each other model
- Each instance's set of instances from the other models also exist in the database as instances of their respective models

Testing Frameworks

- Selenium
- Flask Test Library

- Python UnitTest

Homework 3: Team 7 Contract

Team Members: Cedrik Ho, Allegra Thomas, Grant Ross [Phase I Lead], Sanne Bloemsma

In this document, we will establish our expectations for team communication, conflict resolution, and task allocation, as well as detail the strengths and working styles of each team member.

Team Communication

Our team's primary forms of communication will be Slack and Zoom meetings. We will use GitHub, integrated with our Slack channel, for code management. Our team will meet through Zoom twice a week, from 3pm-5pm on Wednesdays and 10am-12pm on Fridays. During meetings the team will define clear goals for each team member to complete before the next meeting. We will also hold daily standup meetings via Slack, in which each team member briefly details what they have completed since the last meeting, what they will complete before the next meeting, and any problems they have with their current tasks.

Team Strengths and Working Styles

For the team's strengths and working styles, we have only met once so far and therefore are not familiar with each other's abilities yet. We have identified our own strengths and working styles as well as where we may work really well together or may have points of conflict.

Strengths

Cedrik - Python back end testing and frameworks. Open to any task.

Allegra - Document editing, self-guided learning and research. Familiar with Python and Java.

Grant - Giving constructive feedback. Familiar with Java and Python. Very familiar with Github.

Sanne - Self-motivation, self-guided learning, and following instructions. Familiar with Java.

Working Styles

Cedrik - Clear deadlines and expectations. Individual technical work but frequent team meetings to go over all aspects of the project. Favors a clear hierarchy of task importance and order of completion within personal tasks as well as team tasks.

Allegra - Break tasks up into smaller parts and set a schedule for task completion. Complete individual work in chunks with clear goals and deadlines and get feedback from the team before moving on to the next chunk.

Grant - Starting a project early, working on it in small bursts leading up to the deadline. Make sure each member has a clear idea of what they are working on in order to avoid issues caused by lack of communication. Enjoys paired-programming occasionally, but only for larger, more complex tasks.

Sanne - Working on tasks for one to two hours at a time and compiling a project over a set period of time. Prefers to by the person typing while the other person observes and gives feedback or suggestions.

Conflict Resolution

Since conflict usually stems from a disagreement on how something is going to be done or was supposed to be done, we will be explicit on what tasks we are working on each week, who is responsible for that task, and what all team members expect from that completed task.

If a team member disagrees with actions taken or not taken by another team member, the former will initiate a private conversation with the latter to discuss a compromise. If this meeting doesn't resolve the situation, then the team member will discuss the issue with the rest of the group. If a team member disagrees with the direction, actions, or lack of action by the group, they will address their concerns at the beginning of the next meeting in a constructive manner.

For general decision-making, we will adhere to a strict all-or-none voting system and three-strike disciplinary system to ensure order. If a single team member does not agree with a decision, the proposal cannot pass. When this situation arises, everyone will collectively work together to find a compromise the team can progress with. Additionally, since there is no hierarchy within the organization, discipline for poor communication will be handled using a three-strike system. Actions that will receive a strike include failure to have work ready by a clear deadline and missing a meeting with no advance notice. On the third strike, we will communicate these issues with the professor or a TA to resolve the problem.

Task Allocation

Task allocation will change throughout the course of the project, but initially we have decided to break up the project and responsibility as follows:

-Documentation and Deliverables [All]

-Testing and Quality Assurance [Cedrik Ho]

-Front End UI UX [Grant Ross]

-Data Management, Manipulation and Metrics [Sanne Bloemsma]

-API Calls and DB Management [Allegra Thomas]

Expectations of Team Members

- Be responsible and familiar with their own workload and capabilities outside of this project and this class
- Complete tasks and portion of assignments as promised, both in respect to time and quality
- Be comfortable with voicing their concerns or suggestions for improvement when it comes to the project or pair programming
- Be comfortable taking criticism
- Be understanding of others with less experience who may not be as familiar with the information needed to complete portions of the project
- Be understanding that others' experience does not mean they are responsible for the majority of the project and they are responsible for making sure they do their fair share of work