Sprint 4 Deliverables
Group: Got any grapes?
July 1 2022

Product Interface MockUp

Here are the links to the video presentation and PowerPoint slides to our Product Interface MockUp.
Sprint 4 Product Interface MockUp
Sprint 4 Product Interface Mockup

User Stories

Story 1: As a productive student, I want to spend as little time as possible on course selection, so that I don't have to spend those time on the 'add or drop class' page dealing with class time conflicts.

Note: This user story highlights the scheduling feature of BetterQ. With given classes, BetterQ can work out all valid section combinations without time conflicts.

Story 2: As a very health-conscious person, I want to know what courses best fit my schedule and commute habits, so that my daily routine can be maintained in which way I have been comfortable for a long time.

Note: This user story highlights the general preferences filter of BetterQ. The user can limit the time interval of courses by specifying that in the general preference filtering process.

Story 3: As a decidophobia student, I want to know what options are avaiable to be within the scope of program curriculum, so that I don't have to switch back and forth among multiple different department pages.

Note: This user story highlights the specific requirements filter of BetterQ. The program can show all the valid options for the student among certain departments/credits/levels.

Story 4: As a rising senior, I want to find the courses that best match my interests and study style, since I have completed most of the required courses and have more freedom in electives.

Note: This user story highlights the general preferences filter of BetterQ. The user can get courses that is in his/her interested areas and matches his/her expectations by specifying their need in general preferences filter.

Use Cases

Use Case: SelectTerm	
Indentifier:	UC1
Description:	The SelectTerm is the first interact of the user and BetterQ. By user choosing his/her current term, the program selects the corresponding term database for later use.
Actors:	The User BetterQ program
Precondition:	The user launchs BetterQ on his/her device and the SelectTerm is the first interface.
Flow of events:	1. The use case starts with the user clicking on the drop-down menu. 2. From the drop-down menu, the user clicks on his/her current term and then hit ‘next’.
Postconditions:	The program reads the user’s selection of term and correctly choose the term database for later process.

Use Case: ImportCompletedClass	
Indentifier:	UC2
Description:	This use case is for user to let the program know his/her completed classes, for later suggesting and prerequisite algorithm.
Actors:	The User BetterQ program
Precondition:	The User get his/her transcript from sis.rpi.edu and have it in clipboard as plain text. Or if he/she don’t want to bother to get them in clipboard, the program also has the option ‘select manually’.
Flow of events:	1. The use case starts with the user at Import Completed Courses page. 2. a. If the user has the transcript in clipboard, just paste it to the blank. b. If the user does not have the transcript in clipboard, click on Select Manually and will be forwarded to SelectClass page. 3. After the user either paste the transcript or select all the completed courses. Click ‘next’.
Postconditions:	The program reads the user’s completed courses, and the corresponding courses will be added to ‘CompletedCourseList’.

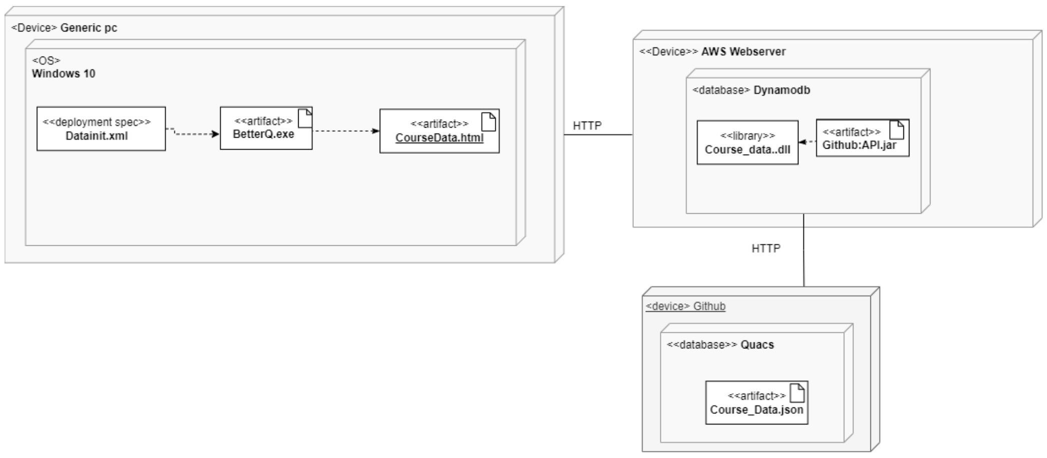
Use Case: SelectClass	
Indentifier:	UC3
Description:	The user then is brought to class selection page with all classes grouped by department. The user chooses the corresponding department and a page with all classes in that department will shows on the page and the user click on the desired class to either add the class to the list.
Actors:	The User BetterQ program
Precondition:	The user has selected the term and knows what the required/completed class(es) for him/her this semester is/are.
Flow of events:	1. The use case starts with the user at the department menu. 2. The user clicks on the desired department and will be brought to a page showing all the classes in that department. 3. The user checks the box next to the specific class he/she wants to 'AddToList' and the class is added to the chosen class list. 4. After that, the user can either click on the 'Back' botton to go back to department menu to add more class (go back to event 1) or hit next to finish selection.
Postconditions:	All the desired class is added to 'CompletedCourseList' if the user gets here from UC2's 'Select Manually' or to 'SelectedCourseList' if from UC2's 'next'.

Use Case: SelectSpecificRestriction	
Indentifier:	UC4
Description:	After have all required class in the SelectedCourseList, the program will help user to find good-fit electives. This use case is the first filter for the electives and the program expects user to fill in the blank according to his/her need. Skip the blank if there is no restriction on that aspect.
Actors:	The User BetterQ program
Precondition:	The user knows what conditions should be put in the blanks.
Flow of events:	1. The use case includes 4 blanks. They are, 'I want to take between [] and [] credits. I want to take a level [] course in the [] department.' The third and fourth blank allows multiple input, the user should separate inputs with comma. 2. The user keys in their according need to the blanks. 3. After entering the restriction, hit next.
Postconditions:	The program narrows the possible recommendations down to classes that fit all the entered restrictions.

Use Case: SelectGeneralPreference	
Indentifier:	UC5
Description:	After SelectSpecificRestrictions, the user then needs to give opinions on some general preferences so that the program can better filter out the best fit class for user.
Actors:	The User BetterQ program
Precondition:	The user has in mind what filtering conditions they care the most.
Flow of events:	1. The use case starts with a page of a list of statement, some with blank. 2. The user can check the box next to the statements those he/she wants them to be filtering conditions. To check the box next to a statement with a blank, the user has to fill the blank first. 3. After checking all the desired preferences, click on ‘next’.
Postconditions:	The program reads all the user’s input of general preferences on courses and will then use them as filter conditions to find top best-match classes for the user.

Use Case: FindBestMatch	
Indentifier:	UC6
Description:	After clicking on ‘next’ from UC5, the program will rank the courses that meet the requirements in UC4 according to the number of match preferences in UC5, from most to least.
Actors:	The User BetterQ program
Precondition:	The user has chosen restrictions and preferences on courses and click on ‘next’ from UC5.
Flow of events:	1. The use case starts with a page listing top 20 coursess that meet all the restrictions in UC4 and has most to least match preferences in UC5. 2. The user can check the box next to the course(s) from the page to add the course to ‘SelectedCourseList’. 3. After selecting all the desired course, click on ‘next’.
Postconditions:	Now the SelectedCourseList should includes the required course(s) and the chosen best matching course(s).

Deployment Diagram



In this deployment diagram, the required hardware and software is a computer running a windows 10 operating system so that the BetterQ executable can deploy directly onto the user's computer, as well as a dedicated Amazon cloud-based webserver with its own dedicated dynamo DB database setup, and a connected Amazon Lambda account which is able to call REST API's connecting it to Quac's existing Github repository and allowing the BetterQ executable to call its own REST API, allowing Communication association between the user's computer, the AWS server and Quac's databases through HTTP.

Supplemental Specification

In this part we will includes all the requirements of BetterQ with FURPS+ model. In this requirement, the blackened key words show their MoSCoW priority.

1. Functionality

With the major and semester information, BetterQ shows the course structure provided by RPI for his/her current semester and possible choices. The filtering algorithm handles at most 6 courses (21 credits) at a time. And with further preferences entered, BetterQ narrows down choices that do not cause time conflict and rank them from best match to least match for users to pick from. This is a requirement this can easily be measured and tested, as it without completion of this there is no viable end product for the user, and as such it is a **must have**.

2. Usability:

BetterQ is a Python program, and it runs on Windows/iOS/Linux systems that have a basic Python environment ready. This is also an easily testable requirement as a failure to run on the users system would mean a complete lack of product, and therefore it is a **must have**.

BetterQ provides course recommendations according to the academic information and personal preferences the users provide. This requirement, while easily testable and incredibly important, is only a **should have**, as there is still some schedule creation, and therefore product functionality even if we do not find the best course recommendations based on user preferences.

3. Reliability:

BetterQ's algorithm makes use of the database that is made with the course data from RPI's official website which has the most accurate and up-to-date course status. This requirement is not quantifiable but is easily testable as we can compare our generated schedule to both QuaCS and sis submitted schedules, and it is a **must have** as the product has no real application if it does not generate schedules with correct class data.

BetterQ's algorithm gives the same results every time when given the same input and class data. This is an easily testable requirement, as we can just automatically repeat the same inputs on a loop and compare outputs, however this is only a **should have**, as it could still produce an acceptable schedule even if it is not the same each and every time.

4. Performance:

All features of Better Q require less than ten seconds to run on any valid device. This requirement is easily testable and quantifiable, as we can simulate each individual function on a multitude of different simulated user devices, however it is only a **could have** as it would still have functionality and be a viable product even if it took 11 seconds for each schedule generation.

Usage of BetterQ will not use an incredibly large amount of resources to run on the users computer. This will most likely not be very easy to test, as we would need to try out the program on a multitude of computer architectures, some of which, will most likely cause unexpect and unforeseeable results, however this requirement is a **must have** as we do not want a product which could actively cause harm to a user computer.

BetterQ must be able to easily and quickly update course local course information once we receive new course data. This will be easy to test as we can modify our own course information database to see how it reacts during data updates, however this is only a **could have** as we do not expect the data to have many significant changes throughout the school year, and because most of our userbase will be using the application during or around registration, during which most schedules have been finalized anyway,

5. Supportability:

As an open-source project, BetterQ has accessible testability, extensibility, portability, configurability, and maintainability. This will be very difficult to quantify or even test, but as an open-source project, this is a **must have** as it will likely have to be able to deal with changing data sources, formats, and developers and will therefore have to be able to continue to grow to stay alive as a project.

6. + Other:

Must optimize and clean out old memory usage when data is out of date. This is easily quantifiable, but is also only a **could have** as each semester only requires less than 2MB of data in its .json format.

Work Breakdown Structure

A detailed list of the tasks associated with your project based upon the vision, scope, and user scenarios. The list of tasks should be as granular as possible (work package level) so that the team can clearly see everything that needs to be accomplished in order to complete the project. Tasks should incorporate **development, testing, documentation, communication, and any other activities being performed on your project.**

The following is the list of development tasks in order. It is a fairly linear process but there are some exceptions, listed in their item description.

Data Structure This is the core of the program, how the data on courses will be stored. It consists of the following. This is required for the project and must be done before any of the back-end development can begin.

Course class as the core data element of the program, this is the first thing that needs to be completed since it is how the parts of the program communicate. Each instance will describe all the relevant data for a given course. This includes course name, times, professor, course ID, whether it is communication intensive, department, seats remaining, level, and a "ranking" variable. This "ranking" variable is the only thing that should be modified after each instance is created. It will be used by the algorithm to measure how relevant a course is to user preferences.

Time class Used to manage the sections and times for an instance of the course class. It holds the individual sections as well as their meeting times. It will contain functionality to be quickly compared against other instances of the time class returning whether a conflict was found, and which combinations are valid.

Data procurement The intake of data from the same git hub the QuACS uses. It serves to access the github and quickly parse data into a list of course class instances. This should happen on program start up to keep data updated.

Back End The portion of the program that the algorithm applies for filtering, giving course suggestions and Scheduling.

Filtering The algorithm filters out the course options that meets the requirments the user enters.

Suggesting&Schedule The algorithm that will assign each instance of the course class a rating based on relevancy to user preferences. The specifics are subject to change as development. Currently each instance of the course class is planned to have a "ranking" variable between 0 and 100. A rating of 0 is given to courses that will not be displayed on any calendar due to a conflict. A rating of 100 is given to a course that must be displayed due to being a required course. Each user preference will modify the rating of each instance of the course class. Once schedules are generated, the algorithm seeks to generate schedules with the highest average rating per class avoiding conflicts.

Front end The portion of the program that the user will see and interact with.

Appearance What the program looks like, this can be completed at any point in the development and is based on the mockup that has already been created.

Interactions Connecting the front end to the back end, once the algorithm is finished this will actually send the contents of the input fields in appearance to the algorithm to manage data.

Save Schedules Implementation of a save functionality that allows the user to save

schedule sets to come back to later or to switch between multiple terms. It will need to manage the fact that course data may have changed since the schedule was created. Thus, it may be best to save a set of preferences and regenerate schedules based on selection.
Import Prereqs Functionality for completed courses to be imported and saved permanently across all schedules. There will be a check box on the page with generated schedules for whether prereqs are taken into account when making a schedule. It may make sense to have this as an option on the landing page rather than the preferences menu for an individual schedule.
Stretch goals To be completed once the above functionality is fully implemented and tested. The goal is to have requirements updated on program start up but this may not be realistic given accessibility and organization of the data on HASS and degree requirements. Thus, it may have to be done manually and hard coded into the program.

> **Manage HASS requirements** Have the program prompt the user and give suggestions to what courses they want in their HASS pathway and what terms they would take them in.
> **Manage degree requirements** When prompted by the user for their degree, both major and minor, it suggests which terms to take various courses based on what the user has completed already. This may be far more complex than it initially seems.

Testing should be conducted in a team meeting after each arm of development is completed by the members of the team who were not in charge of the development of that given area of development. That way it can be obvious if mistakes are made or if design is not clear to someone other than the programmer. Comparing data results to QuACS results may be beneficial as it should be immediately obvious if some data is left out or disorganized. For example, comparing a random sample of course class instances to their equivalents on QuACS to see if data was imported properly.

Communication should consist of updates at each meeting and major completions of development goals announced on the discord. Minor progress not significance for an announcement will be noted in a tracking document.

Updated Project Schedule

The project is designated to be completed within 10 Sprints in the manner of Scrum master from June 1 to August 5, 2022. Every sprint starts at Thursday and ends on Wednesday and lasts one to two weeks.

Sprint 0   June 1

- Team Establishment

Sprint 1   June 2 – June 8

- Documentation: Create personas and user scenarios for BetterQ.
- Project OverView: Discuss in the group about feature details and database sources.
- Task Distribution: Draft a diagram breaking down the key features for

implementation.

Sprint 2 June 9 – June 15.
- Documentation: Work on the vision statement, including more personas and user scenarios.

Sprint 3 June 16 – June 22
- Documentation: Complete vision statement.
- Documentation: Create all 6 use cases according to the functionalities.
- Documentation: Generates Deployment Diagram for BetterQ.

Sprint 4 June 23 – June 29
- Documentation: Create Interface MockUp
- Documentation: Create 4 User Stories corresponding to the features.
- Documentation: Generate Deployment Diagram and Work Breakdown Structure for BetterQ
- Documentation: Design the program's Data Structure, including the Course Class and Time Class.

Sprint 5 June 30 – July 13
- Database: Work on data procurement and parse data into the designated data structure.
- Front End: Rough draft of user interface appearance.
- Back End: Implement Save Schedule and Import Prerequisite feature.

Sprint 6 July 14 – July 20
- Front End: Improve user interface appearance.
- Front End: Implement the logic between pages and icons.
- Back End: Implement Preference Filter and Schedule feature.
- Interim Release: Complete Interim Release Deliverables.

Sprint 7 July 21 – July 27
- As a group: Test the demo BetterQ and evaluate performance.
- Front End: Pushing toward the perfection of the user interface.

Sprint 8 July 28 – Aug 3
- Beta Release: Complete Beta Release Deliverables.
- As a group: more test and test documents on BetterQ.

Sprint 9 July 4 – July 10
- Final Release: Final Release presentation.

Project Status Report

Progress summary
- **Documentations:** We have a complete vision statement, user personas, user scenarios, use cases, deployment diagram, and work breakdown structure.
- **Front End:** We have a mock-up presentation on the project's features and interfaces.
- **Back End:** We have designed the data structure for the database BetterQ will be using.

Issues we encountered so far

1. **Sprint 1 deliverables.** We had a rather bad grade for Sprint 1 deliverables for lacking details in documents. It was mainly because of a lack of effort and unbalanced task distribution; only one of us was in charge of producing all the documents. In order to fix this mistake, we re-distributed the documentation tasks among the group so that everyone is involved in the work and gives constructive feedback.
2. **Lose one teammate.** By the end of Sprint 4, one of our group members, Kevin, had to drop the class to lighten his workload. For the team, we had to reconsider our desired features and coding task distribution. This incident would inevitably reduce the team's total work power. After discussion, we decide to keep our current project scope and Preston will take over the algorithm part that Kevin was responsible for.

Contribution Summary

John Bartly:

· Supplemental Specifications (of this deliverable)
· Deployment Diagram (of this deliverable)
· Data procurement for the BetterQ database

Matthew Chin:

· Product Interface Mock-Up Presentation (of this deliverable)
· Front End Design including interface appearance and icon interactions.

Jingyang Guo:

· User Stories (of this deliverable)
· Use cases (of this deliverable)
· Updated Project Schedule (of this deliverable)
· Project Status Report and Contribution Summary (of this deliverable)
· Weekly status update
· Project management

Preston Waters:

· Product Interface Mock-Up design and PowerPoint (of this deliverable)
· Work Breakdown Structure (of this deliverable)
· Data procurement for the BetterQ database
· Back End algorithm implementation