UpGrade Modules: Standalone Application for Multi-Topic Learnersourced Open-Ended Questions

Brandon M. Garzez
Computer Science and Engineering
University of Michigan
Ann Arbor, MI, USA
bgarzez@umich.edu

***Abstract*—Open-ended assignments are popular throughout schools and universities worldwide [1]. However, it is tedious for instructors to grade open-ended assignments, and it is difficult for students to repeatedly practice them [1]. UpGrade enables automated and real-time feedback for open-ended assignments by generating scalable learning opportunities based on previous students' solutions [1]. It is important that the student-facing side of UpGrade is easy to navigate, intuitive to use, and useful for feedback. To achieve all of these goals, UpGradeModules allows students to choose between several modules, each with a list of sections. These sections contain a handful of questions related to an overarching topic, which enables repetition for the open-ended assignments.**

***Index Terms*—crowdsourcing; online education; deliberate practice; open-ended assignment; multiple-choice question; python; web design; databases; software engineering; computer science; computer applications; web sites**

I. Introduction

A. Research on Learning Outcomes

Open-ended questions and answers are a popular form of assignments throughout schools and universities [1]. The benefits of open-ended assignments include receiving feedback from instructors which is helpful for preparing students for future assignments, exams, and overall mastery of the subject material [1]. Unfortunately, open-ended assignments are difficult and tedious for instructors to grade [1]. Often times, by the time students have received their feedback for an open-ended assignment, far too much time has passed for the feedback to be useful for future assignments; rather, the material needs to be partially or completely relearned before the feedback can be used to improve learning outcomes, making repeated practice with open-ended questions highly challenging [1].

1) Learnersourcing: UpGrade offers the ability for repeated practice of automatically-generated interactive questions by leveraging an area of research called learnersourcing, which is a branch of crowdsourcing research [1]. Learnersourcing takes advantage of the idea that students provide a large quantity of open-ended content through their completion of open-ended assignments [1]. Students' responses can be automatically logged and used to create scalable, repeatable, and easily gradable questions which students can practice online. When implemented correctly, learnersourcing enables students to learn by answering questions, and their answers are used to create new questions to help future students learn [1].

2) The Issue of Cognitive Load: While UpGrade's learning outcomes are compared to those of traditional open-ended questions, UpGrade does not itself ask students to answer open-ended questions. While open-ended questions offer many benefits for learning outcomes, which I discuss above, they also require a high cognitive output for students to complete the assignment [1]. Because answering open-ended questions requires such a high mental output, students often struggle to learn from the feedback received from an open-ended assignment since the student has already exhausted most of their cognitive resources just to complete the assignment [1]. UpGrade avoids the "overloaded cognitive resources" pitfall by providing students with worked examples which are automatically created using older student responses as inspiration for new open-ended question/answer combinations [1].

3) The Issue of Expert Blind-Spots: In addition to avoiding the issue of cognitive load, UpGrade also attempts to mitigate the issue of expert blind-spots which can plague open-ended assignments [1]. Since the instructors who create and grade open-ended assignments are experts in the relevant fields, they often fail to anticipate some of the amateur mistakes which students will make and thus may not give adequate feedback in those areas [1]. Contrast this with learnersourced solutions, which encapsulate a vast spectrum of answers and capture a more complete picture of the students' understanding for a given topic [1]. UpGrade takes additional advantage of learnersourcing solutions by breaking students' solutions into many parts and showing the strengths and weaknesses of each part to further help future students learn from them.

4) Repeated and Deliberate Practice: UpGrade focuses on enabling students to repeatedly and deliberately practice individual skills in order to achieve mastery. The motivation for deliberate practice is twofold. First, research shows that deliberate practice increases learning outcomes and leads to continued learning [1]. UpGrade facilitates repeated and deliberate practice by presenting information and questions to students in small bits and pieces; by presenting only small chunks, UpGrade mitigates the risk of students being overwhelmed by repeated questions of great length and complexity [1].

5) Repeated Feedback: Not only does UpGrade create the opportunity for repeated and deliberate practice, it also provides students with repeated feedback to maximize learning. Unlike with traditional open-ended assignments, which take a large amount of time and resources for instructors to grade, UpGrade quickly and efficiently provides feedback to students [1]. Because UpGrade provides instantaneous feedback, students can immediately incorporate that feedback into their learning. Furthermore, because UpGrade breaks the learnersourced responses into small chunks before presenting them to students in the form of questions, the feedback a student receives is highly focused, further enabling repeated and deliberate practice [1].

B. UpGrade Workflow

There are three main components to UpGrade's workflow: solution logging, solution segmentation, and question generation. See Fig. 1 for an overview of the UpGrade workflow. In this subsection, I discuss each step of UpGrade's workflow in more detail.

1) Solution Logging: UpGrade utilizes learnersourcing to generate questions and feedback automatically. In the first step of learnersourcing, UpGrade collects the solutions provided by prior students in PDF documents. These PDF documents are structured with specific sections based on the assignment rubric. For the purposes of this research lab, the student solutions have already been graded, and instructors and peers have already provided feedback based on the assignment rubric [1].

2) Solution Segmentation: After student solutions have been collected, UpGrade parses the PDF documents and separates the solutions into sections based on the assignment rubric. Images and figures are included with the text for a given section. If the assignment is given to students via an online form with individual fields for students to respond to each section, this separation step is not necessary; however, for the purposes of this research lab, the assignments and solutions exist in the form of a PDF, and thus UpGrade carries out the segmentation process [1]. Where applicable, UpGrade pairs the students' solutions with the feedback they received, and each section's solutions and feedback are saved in a file.

3) Question Generation: Once student solutions are collected and separated into sections, UpGrade automatically creates new questions based on instructor-defined schema. Instructors can specify several schemas using four components: Question, Answer, Explanation, and Feedback [1]. See Fig. 2 for a visual demonstration of how these components would appear in an UpGrade-generated question. Questions are the open-ended prompts which students answer in their assignments. Answers are the student responses to the given Questions. If required by the instructor, students give an Explanation to justify the rationale for their Answer. Finally, if the assignment has been graded, the Feedback component includes the instructor's feedback to the students [1]. Some common examples of schemas made from the four components include the Question-Answer schema (Fig. 3), the Question-Answer-Explanation schema (Fig. 4), and the Answer-Feedback schema (Fig. 5).

II. UPGRADE MODULES

In this section, I compare the current state of the UpGrade application to the ideal state of the program. Then I evaluate two options for converting the application to the ideal state, and I make a decision on which strategy to use in developing the UpGrade Modules application.

A. Modules

1) Ideal State: As part of the solution segmentation process, an instructor creates various modules to align with the rubric of an open-ended assignment. When students sign in to the UpGrade web system, they must be able to navigate to any module to which they have been assigned in order to answer questions created by UpGrade pertaining to that module. Within each module, different sections further break down the learning into manageable and focused chunks. Each section has a set of questions which are drawn from a large pool of generated questions for that module and section. Students have an unlimited number of attempts to achieve a satisfactory score on a given section, so students can repeatedly practice and work towards mastery of a subject.

2) Current State: In the current state of the UpGrade web system, each module exists as an individual application which is run on a server and accessed by students. See Fig. 6 for a visualization of the current state of the UpGrade system. Because the modules run independently, there is no way for students to navigate between modules without having to switch between applications. This is functionally the same as switching websites, even though the content of each site is directly related, and the user's credentials and data are shared between the sites. Put another way, this would be like if Google Search, Google Images, Google Videos, and

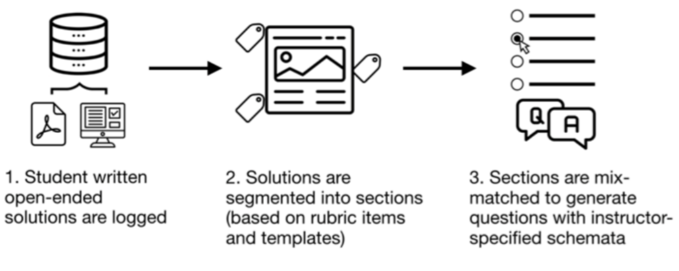

1. Student written open-ended solutions are logged
2. Solutions are segmented into sections (based on rubric items and templates)
3. Sections are mix-matched to generate questions with instructor-specified schemata

Fig. 1. UpGrade Workflow [1].

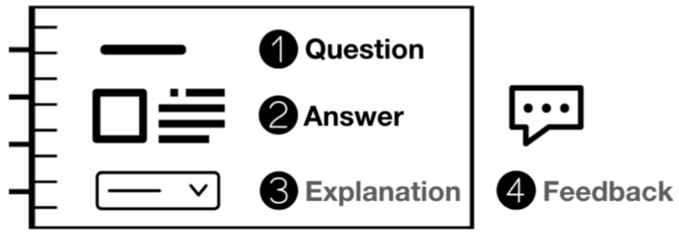
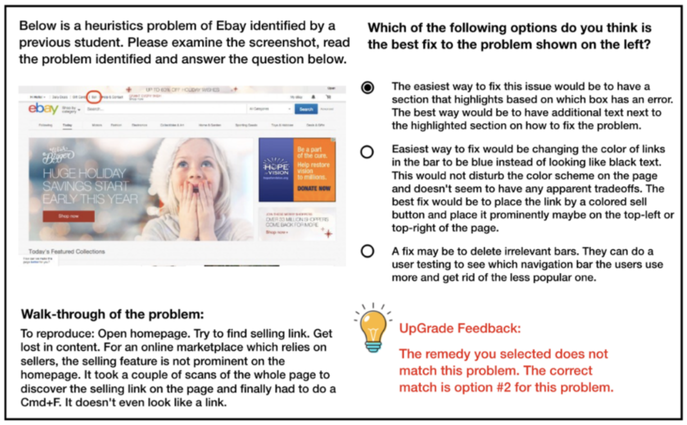


Fig. 2. Simple mockup of the four components used in UpGrade question schemata [1].


Below is a heuristics problem of Ebay identified by a previous student. Please examine the screenshot, read the problem identified and answer the question below.

ebay
Search
HUGE HOLIDAY SAVINGS START EARLY THIS YEAR
Shop now
HOPE VISION
Be a part of the cure. Help restore vision to millions.
DONATE NOW
OVER 33 MILLION SHOPPERS COME BACK FOR MORE
Today's Featured Collections

Walk-through of the problem:

To reproduce: Open homepage. Try to find selling link. Get lost in content. For an online marketplace which relies on sellers, the selling feature is not prominent on the homepage. It took a couple of scans of the whole page to discover the selling link on the page and finally had to do a Cmd+F. It doesn't even look like a link.

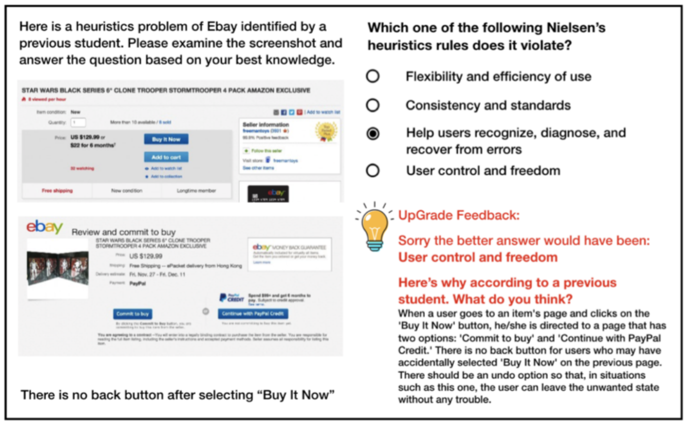
Which of the following options do you think is the best fix to the problem shown on the left?

- [✓ The easiest way to fix this issue would be to have a section that highlights based on which box has an error. The best way would be to have additional text next to the highlighted section on how to fix the problem.]
- Easiest way to fix would be changing the color of links in the bar to be blue instead of looking like black text. This would not disturb the color scheme on the page and doesn't seem to have any apparent tradeoffs. The best fix would be to place the link by a colored sell button and place it prominently maybe on the top-left or top-right of the page.
- A fix may be to delete irrelevant bars. They can do a user testing to see which navigation bar the users use more and get rid of the less popular one.

UpGrade Feedback:

The remedy you selected does not match this problem. The correct match is option #2 for this problem.

Fig. 3. Example of Question-Answer schema [1].


Here is a heuristics problem of Ebay identified by a previous student. Please examine the screenshot and answer the question based on your best knowledge.

There is no back button after selecting "Buy It Now"

Which one of the following Nielsen's heuristics rules does it violate?

- Flexibility and efficiency of use
- Consistency and standards
- [✓ Help users recognize, diagnose, and recover from errors]
- User control and freedom

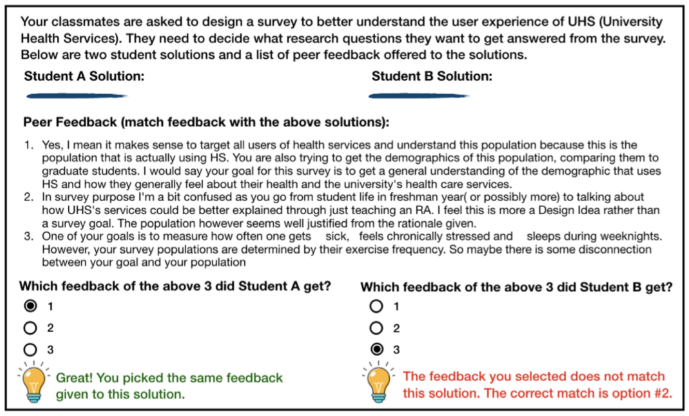
UpGrade Feedback:

Sorry the better answer would have been: User control and freedom

Here's why according to a previous student. What do you think?

When a user goes to an item's page and clicks on the 'Buy It Now' button, he/she is directed to a page that has two options: 'Commit to buy' and 'Continue with PayPal Credit.' There is no back button for users who may have accidentally selected 'Buy It Now' on the previous page. There should be an undo option so that, in situations such as this one, the user can leave the unwanted state without any trouble.

Fig. 4. Example of Question-Answer-Explanation schema [1].


Your classmates are asked to design a survey to better understand the user experience of UHS (University Health Services). They need to decide what research questions they want to get answered from the survey. Below are two student solutions and a list of peer feedback offered to the solutions.

Student A Solution: [redacted]

Student B Solution: [redacted]

Peer Feedback (match feedback with the above solutions):

1. Yes, I mean it makes sense to target all users of health services and understand this population because this is the population that is actually using HS. You are also trying to get the demographics of this population, comparing them to graduate students. I would say your goal for this survey is to get a general understanding of the demographic that uses HS and how they generally feel about their health and the university's health care services.
2. In survey purpose I'm a bit confused as you go from student life in freshman year(or possibly more) to talking about how UHS's services could be better explained through just teaching an RA. I feel this is more a Design Idea rather than a survey goal. The population however seems well justified from the rationale given.
3. One of your goals is to measure how often one gets sick, feels chronically stressed and sleeps during weeknights. However, your survey populations are determined by their exercise frequency. So maybe there is some disconnection between your goal and your population

Which feedback of the above 3 did Student A get?

[✓ 1] 2 3

Great! You picked the same feedback given to this solution.

Which feedback of the above 3 did Student B get?

1 2 [✓ 3]

The feedback you selected does not match this solution. The correct match is option #2.

Fig. 5. Example of Answer-Feedback schema [1].

Google Scholar were all independent websites as opposed to one website. Seamless navigation is impossible under the current state of the system, as switching modules requires a transition of browser data, session information, and account information every time a new module is requested.

In addition to the poor user experience of navigating between modules, having each module exist as an independent website leaves the system vulnerable to security threats. If a threat is discovered on one of the modules, all modules are potentially at risk of the same threat, and thus they all must be patched to mitigate the risk of a security breach. However, since each module exists as its own web application, a developer needs to patch each module independently, and a mistake on just one module could potentially leak information which could give a hacker the ability to access all modules with a student or instructor's credentials. Furthermore, as previously mentioned, having each module run independently requires session data to be transferred between websites when the user switches modules. If just one of the modules has a vulnerability in its cookie management, a potential hacker could gain access to a student's session data and use that to access all of the modules while impersonating the student. These security risks pose a substantial threat to the web system and further emphasize the need for a single website to manage all of the modules.

Finally, because each module runs as an individual application, each module also has its own database to house all of its questions, progress, accounts, and other data. Each of these databases are completely independent, meaning that changes in one database cannot be directly accessed by another module without substantial changes to the web system. This poses multiple blockers to a seamless user experience and security. Regarding user experience, if a user updates any of their account information, perhaps their password for example, their changes are not reflected in the other modules. Thus, the user would need to change their password for each module to which they have been assigned.

The user experience segues perfectly into the security risks of multiple databases. If a user's passwords are leaked in a data breach and the user, like many, uses the same leaked password for multiple websites, including UpGrade, they would need to change their password to protect their account. If the user

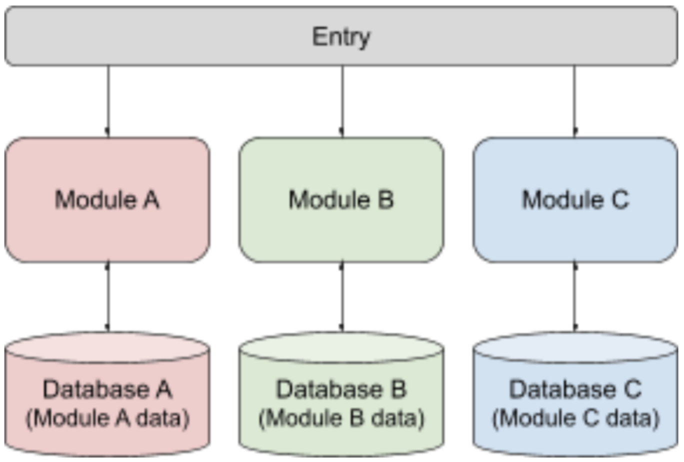

Entry
Module A
Module B
Module C
Database A (Module A data)
Database B (Module B data)
Database C (Module C data)

Fig. 6. Visualization of the current state of the UpGrade system.

forgets to change their password for just one of the modules, it would leave a vulnerability that a hacker could exploit to access their account.

Clearly, separate modules raise several issues pertaining to the user experience and security of the web system. In the next section, I discuss various ideas for consolidating the modules to correct the aforementioned problems.

B. Consolidation Strategies

1) Parent and Child Programs: The simplest approach to consolidating the modules is to create a parent program which oversees the operation of the child module programs. The user logs into the parent program, and upon navigation to a module, the parent program redirects the user to the module's respective child program. The parent program manages all session and account data, and the user can navigate to a new module by first returning to the parent program and then redirecting to a different module's child program. See Fig. 7 for a visualization of the Parent and Child strategy. The issues with this approach include the tedious development work, lack of generality, and security risks.

When developing under the Parent and Child paradigm, any code modifications which are made to one module do not affect the other modules. While this independence between modules enables more control over the functionality of each module, it requires any desired universal changes to be reimplemented for each module. In terms of efficiency, reimplementing code for each module increases the likelihood of mistakes and requires more hours of development work.

In addition to the tedious nature of implementing changes to the module codebases, the Parent and Child strategy also suffers from a lack of generalization. Because the module code is not generalized at all, new modules cannot be easily added to the web system. Since each module runs on its own web application, any new modules require the creation of a new application with its own database, and then that application needs to be integrated into the parent program for it to be accessible by the students. Not only is it impossible to efficiently add new modules, but non-generalized code also introduces variation in the functionality of each module. Since there is no template code upon which to run each module, small differences in each module can add up to create a disorganized and inconsistent user experience.

Finally, the Parent and Child strategy shares another vulnerability with the current state of the UpGrade system. Because each module and its database are independent from the rest of the modules and databases, a vulnerability in one of the modules puts the entire system at risk. Likewise, if a vulnerability is discovered, the fix needs to be implemented for each module, which is inefficient and runs the risk of a mistake being made within one of the modules, rendering the security patch useless. Due to the issues with development, generalization, and security, the Parent and Child Programs strategy was not used for this project.

2) Single Templated Application: The more complicated approach requires the UpGrade system be redesigned from the ground up with generalization and templating in mind for modules to run under one application. Under the Single Templated Application strategy, there is one instance of the program running to handle all modules, and there is one database which houses the accounts, questions, feedback, and all other requisite data for all of the modules. See Fig. 8. for a visualization of the Single Templated Application strategy. After consolidating all of the modules into one program, I can use templated HTML to give each page an identical structure while enabling dynamic web pages which change depending on the user's module, section, and question. The server keeps track of the state of the system: which page the user is on, which questions the user has answered, the user's score for a given section, and which modules the user has access to. Based on those state parameters, a small set of functions can be adapted to fit any number of modules, users, sections, and questions.

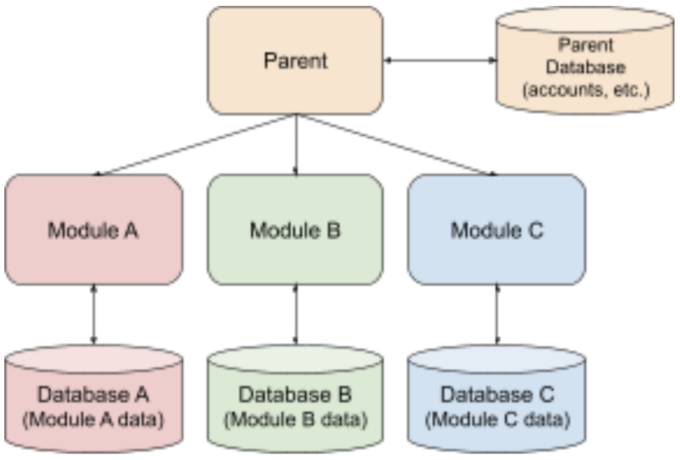

Parent
Parent Database (accounts, etc.)
Module A
Module B
Module C
Database A (Module A data)
Database B (Module B data)
Database C (Module C data)

Fig. 7. Visualization of the Parent and Child Programs strategy.

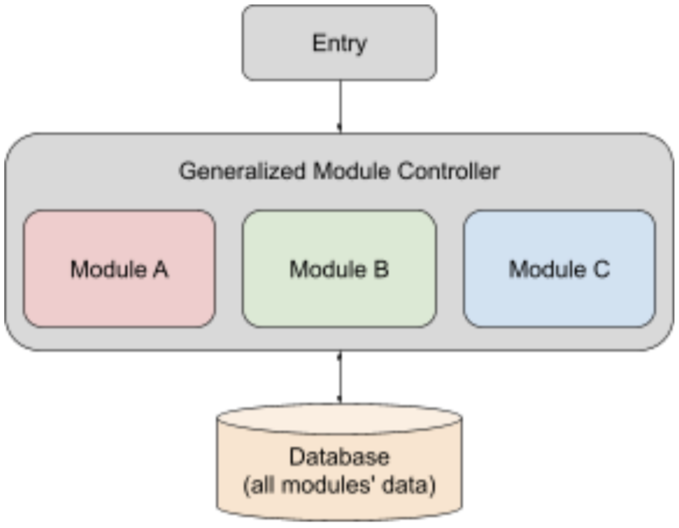


Fig. 8. Visualization of the Single Templated Application strategy.

The database setup sees only one unit which contains all of the pertinent data for all modules. When a user wishes to navigate between modules, the server requests the information about that module from the database to present the user with the appropriate web page. Adding a new module to the database only requires the module's metadata to be inserted into the database without requiring a complete restructuring of the web system. Where before each website had its own specialized code, under the Single Templated Application strategy, the database holds all of the information about all of the modules, and the server plugs a requested module's metadata into a templated function to return the proper page which is populated with that module's data.

The Single Templated Application strategy also enables developers to make widespread changes to all modules by implementing a change to the templated function only. Furthermore, only the templated HTML file and its corresponding CSS file need to be changed to redesign or restyle the layout of the page. All changes to the templated functions and files are immediately reflected across all modules and subsections within those modules. Furthermore, any security patches only need to be implemented once, and the entire system falls under one security protocol, requiring all attacks on the site to go through the same security measures, eliminating the chance that one weak module puts the whole system at risk. Finally, the singular application maintains all of the session data; no longer do multiple websites need to keep track of a user's cookies or credentials.

The downsides to the Single Templated Application strategy are twofold. First, the strategy requires that the entire UpGrade program be redesigned with templated code replacing the existing code which is designed to only execute one module. While the generalized code is similar in many ways to the existing code, I must undergo copious planning and testing to ensure that all modules, even modules with different schema (Question-Answer, Answer-Feedback, etc.), run together without errors in the templated functions. The planning and testing required to modularize the code properly is more time-consuming and complicated than simply creating a parent program to oversee the individual module applications.

The other downside to the Single Templated Application strategy is that it is more difficult to implement different functionality between the modules. While generalized code is excellent at maintaining consistent behavior between modules, which is critical for this project, it creates challenges for implementing modules which have distinctly different behavior. In order to accommodate such differences between the modules, I must set up the templated functions and database schema in a way that enables multiple functionalities. As an example, some modules require that the multiple choice options are sentences, and others require that the options are images. For the application to handle both cases, the database must allow a question to be stored with both a text answer and an image answer, and the server function which handles displaying the questions must account for questions which have text options, image options, or potentially both.

Despite the aforementioned downsides, the Single Templated Application strategy enables highly scalable, consistently-behaving, and intuitive modularization for the UpGrade system which allows students to easily navigate between modules and strengthen their skills. For this reason, despite the complexity, this project followed the Single Templated Application strategy.

III. Implementation

In this section, I specify the implementation details for the UpGrade Modules application. First, I describe the backend server code, and I list each server function with a detailed description of its effects. Then, I discuss the database schema, and I list and describe each table in detail.

1) URL Handling: Each of the UpGrade module applications handle their own URLs separately; however each module handles the URLs in the same way with subtle differences to account for the number of sections in each module, the name of the module, etc. UpGrade Modules handles the URLs in a nearly identical way with one key difference. The original applications encode the module directly into the URL, e.g. ".../modulename/section1", and UpGrade Modules encodes the URL as a query parameter which is passed to the backend server with every new HTTP request, e.g. ".../upgrade/section1?module=2". With the former encoding, each module requires its own set of URL paths, and the server has to redirect each URL to the same templated function. However, the latter encoding uses the same URL for all modules and passes the module parameter as a GET argument to the templated function to indicate the current module to the server.

2) Templated Server Code: When a user makes a request to the server, the URL handler redirects the request to one of the server's backend functions to process the request and return a web page to display to the user. There are many backend functions, and the URL handler picks one depending on the structure of the request. However, the server functions do not know in advance which user is requesting a page. For example, if User A requests section number 1 for module "Storyboard," and User B requests section number 3 for module "Logdata," the server does not know in advance where the users are coming from. The users include which module, section, and question they are requesting in the HTTP request itself, and the backend function receives this information in the form of parameters. The backend functions then use the parameters from the HTTP request to pull the correct data from the database, build the response, and send the response back to the user. See Fig. 9 for a simplified example of how this request-response cycle would work. The server has seven function types, and I describe each of them in more detail in the following subsections.

Note: each of the following functions first ensure that the user is logged in before performing the described task; if the user is not logged in, each page directs the user to the login page before continuing.

a) index and moduleselect: The index function simply redirects to the moduleselect function. Perhaps the most straightforward function of the application, the moduleselect function returns the module selection form and then redirects the user to the homepage for the module they selected.

b) home: The home function first checks that the user has signed the consent form for the website, and if not, it prompts the user to sign the consent form. If the user has consented to use the site, the function displays the list of sections encapsulated by the current module which the user can navigate to, and it gives the user a brief overview of the module. Both the list of sections and the module overview are queried from the database.

c) section: The section functions give a brief overview of the selected section and displays the user's best score for that section. To find the user's score, the section function performs a database query and uses the returned data to compute the user's best score for that section. The returned page also has a button which takes the user to the questions for that section.

d) section_questionpage: The section_questionpage function shows the question which the user is currently answering, starting with the first question. The function queries the current question number from the database along with that question's prompt, answer choices, correct solution, and feedback. The returned page shows the user the question prompt, the list of options for the user to select, and a submit button. Upon pressing the submit button, the user's selected choice is sent to the server's imagefeedback function, and the button prompts the user to go to the next question.

e) imagefeedback: The imagefeedback function compares the user's answer to the correct solution, and it returns feedback according to whether the user answered correctly or incorrectly. The function also updates the user's progress for the current section and marks the question as completed so the user cannot resubmit the question until they finish the section.

f) nextpage: The nextpage function is called whenever the user presses the button from the section_questionpage function after submitting their answer. The function saves the response from the previous question and then redirects the user to the next question's section_questionpage function. If the user has answered all of the questions, the function instead redirects the user back to the current section function.

g) signform: The signform function is invoked whenever the user submits the consent form referenced in the home function description. The function save's the user's response in the database and then redirects the user to the home function of the current module.

3) Templated HTML Files: An HTML file contains the contents of a web page. When a user navigates to any website, their browser builds the page they see from the data encoded in the HTML file. Web frameworks such as Django enable developers to create templated HTML files which are populated with dynamic information depending on the state of the server and the user's request. As an example, a website asks for the user's name, and when the user submits their name, the website responds with the message, "Hello Name!" Using a

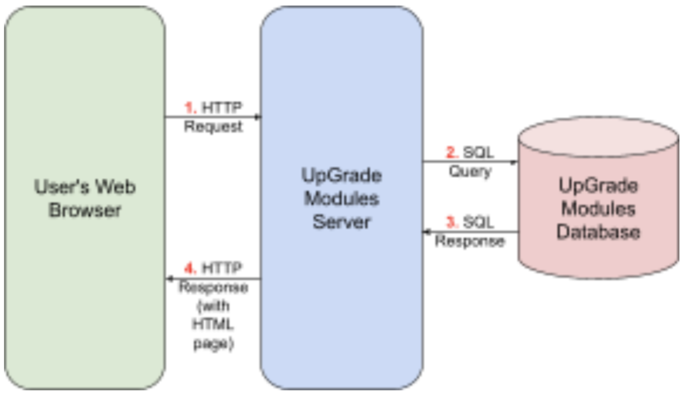


Fig. 9. Diagram of HTTP request-response cycle between a browser, web server, and database.

template makes it possible for the server to fill the user's name into the web page by passing the user's name to the template as a parameter. UpGrade Modules takes advantage of templated HTML files to show the user dynamic web pages based on the user's current module, section, and question. The server has four templated HTML file types, which I describe in detail in the following subsections.

a) base and nobase: The base and nobase HTML files are used by nearly every web page in the UpGrade Modules system. Both files contain the navigation bar which is present on nearly every page. The base file additionally contains the list of sections for a given module, and it is only used for web pages where a module has been selected by the user.

b) welcome: The welcome HTML file describes what the user sees when they select a module, but before they select a section from within the module. The file contains the name and description of the module as well as an overview of the user's progress through each of the module's sections. The file builds on the base HTML file, so it also includes the navigation bar and the list of sections for that module.

c) section: The section HTML file describes the web page which the user sees after selecting a section from the current module, but before starting the questions for that section. The file contains a description of the section, and it also tells the user the number of questions within the section and their best score. The file builds on the base HTML file, so it also includes the navigation bar and the list of sections for the current module.

d) questionpage: The questionpage HTML file creates the question form which the user submits to the website. It contains the question prompt, any relevant images for the question, and the answer choices for the question. Additionally, the file contains a button which the user can press to submit their selected answer and to then move to the next question. Upon submitting an answer, the server will populate the HTML file with relevant feedback depending on whether the user answered correctly or incorrectly.

A. Database Schema

As previously stated, the original UpGrade modular applications each have their own databases. These databases contain the questions, solutions, feedback, registered users, and all of the other data about the module it is paired with. UpGrade Modules consolidates these databases into one universal database containing all of the data for all of the modules. The database schema consists of many tables, each of which contains information about some part of the application. The Django framework creates several tables automatically, and the UpGrade Modules application constructs the rest of the tables. There are six tables which are built by UpGrade Modules, and each will be described in detail in the following subsections.

a) Participant: The Participant table holds all of the data about the students who access the UpGrade Modules site. Each participant has an indicator for whether they have signed the consent form as well as whether or not they gave permission to share their results.

b) Module: The Module table holds all of the modules which have been consolidated into the UpGrade Modules application. Each module in the table has a unique identification number and a name.

c) Section: The Section table holds all of the sections which are parts of various modules throughout the application. Each section is directly affiliated with a module, and each section has a unique identification number, a name, a description, and a total number of questions.

d) Progress: The Progress table tracks the progress of each user through each section and module. Each progress entry is directly affiliated with a participant and a section, and each entry also has an indicator for whether the participant has completed their section, the participant's score, the trial number to track the number of attempts the participant has made, and a field to track when this progress entry was last updated so the most recent trial can be easily found.

e) Question: The Question table contains all of the questions for all of the modules and sections in the application. Each question is directly affiliated with a section, and each has a correct answer and a list of options; additionally, each can have a question stem, an image, and a list of feedback messages.

f) Response: The Response table holds the responses that are recorded as a participant answers questions. Each response is directly affiliated with a participant, question, and section, and each has a copy of the submitted answer, a copy of the optional justification provided for that answer, an indicator for whether the answer was correct, a trial number, a feedback message, and timestamps for when the student submitted their answer, navigated to the next page, and most recently updated their response.

IV. Results

In this section, I describe the results of the project and show some sample screen captures from the UpGrade Modules application for reference. This section also compares the anticipated outcome of the project with the final deliverable and reflects on how well expectations were met throughout the project.

A. Implementation

1) Templated Server Code: I consolidated the server code considerably into one Django application containing three modules: Storyboard, Logdata, and UAR. In addition to the module consolidation, the final product also enables users to seamlessly navigate between the modules and sections once they have logged into the system. Compared to the desired implementation, the final product is not as consolidated as possible, and much of the information is still hard coded directly into the server functions. All server functions are divided into four parts, one for each section. Each module's first section uses the ”function1” code, and its second section uses the ”function2” code, and so on. While each module uses the same section functions, the sections themselves are not fully consolidated to the point where a single function can handle all sections for all modules. However, for the three modules which have been incorporated into the project, the user experience is smooth, and adding additional modules does not require much, if any, modification to the server functions.

2) Templated HTML Files: The HTML files are not fully consolidated; each module has a folder for its respective HTML files. While each module has very similar HTML files, the differences are stark enough that a complete consolidation is not possible within the timeframe of the project. However, all of the HTML files contain template scaffolding, making the consolidation of the files considerably easier now compared to the beginning of the project when such scaffolding did not exist.

3) Database Schema: The database is nearly completely consolidated, although in order to fully consolidate the server code and the HTML files, additional schema likely needs to be added to the database. However, in the current state of the UpGrade Modules application, there is only one database, and all modules store questions, responses, and feedback using one set of schema. The server code interprets the data from the database in various ways depending on the current module and section, but from the database's perspective, all modules use the exact same schema.

B. Application Interface

The module selection screen allows users to select modules from a drop-down menu, as shown in Fig. 10. Upon submitting a module selection, the website will show the corresponding module's homepage, as shown in Fig. 11 and Fig. 12. Finally, the user can select sections and answer the corresponding questions for that section, as shown in Fig. 13.

V. Discussion, Limitations, and Future Work

In this section, I discuss the limitations and potential future work related to the UpGrade Modules application. I also reflect on how I would redo this project differently if given the opportunity.

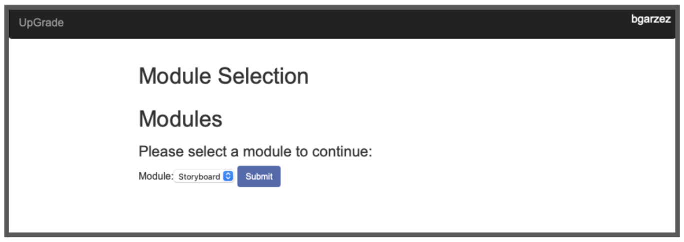
A. Complete Consolidation of Server Code and HTML Files

While I consolidated most of the backend server functions and HTML files during this project, much of the functionality still relies on hardcoded values and special cases to force all modules to conform to a homogeneous model. The time scope of the project is a major reason why the server code and HTML are not completely consolidated. As the sole developer of UpGrade Modules, an application which uses a framework with which I was not familiar at the start of this Capstone, the scope of the project proved to be slightly larger than what could be accomplished in one semester. However, while the scope of the project was larger than anticipated, I also wasted much of the time I spent at the beginning of the project by attempting to learn the Django framework as I consolidated the code. Because of my learn-on-the-job attitude towards this project, most of the changes made in the first month of the project needed to be redone or completely removed later on due to a reevaluation of the workflow. Given the chance to redo this project, I would spend more time planning the development of the templated server functions before I write any code. By planning more in advance, every change would be deliberate and would be made with the foresight of future additions to the project; thus, the changes made would be permanent and no time would be wasted replacing broken code with what should have been written the first time.

B. Storing all Metadata in the Database

While the database is the only part of the project which is completely consolidated into one set of schema, there is still room to further optimize how it is used throughout the project. The HTML files are mostly hardcoded with only a few templated variables incorporated into them. As a result, each module has several HTML files which only it uses, and there are several copies of near-identical files with only slight differences; these copies should be consolidated into one file. In order to remove the discrepancies between the files, the section and module descriptions should be included as part of the database. When the user requests the HTML for a module's welcome page, the server should query the module description from the database and substitute that variable into the universal HTML welcome page template rather than fetching a unique HTML file for that request.

Similarly, the question pages are nearly identical for all modules and sections, with only the number of questions and types of questions being different between any two question-page HTML files. The Question schema can be modified to store the options as a JSON string which includes the text answer and the image path for an image that corresponds to that answer. By storing the options as a JSON string, answers can consist of text and images simultaneously, and all of the questions can fit the same schema regardless of how many options a given question has. The JSON string would be parsed by the server function and then passed as a list of objects to the HTML template. By doing this, the questionpage HTML files could be reduced to just one file for all modules and all sections.



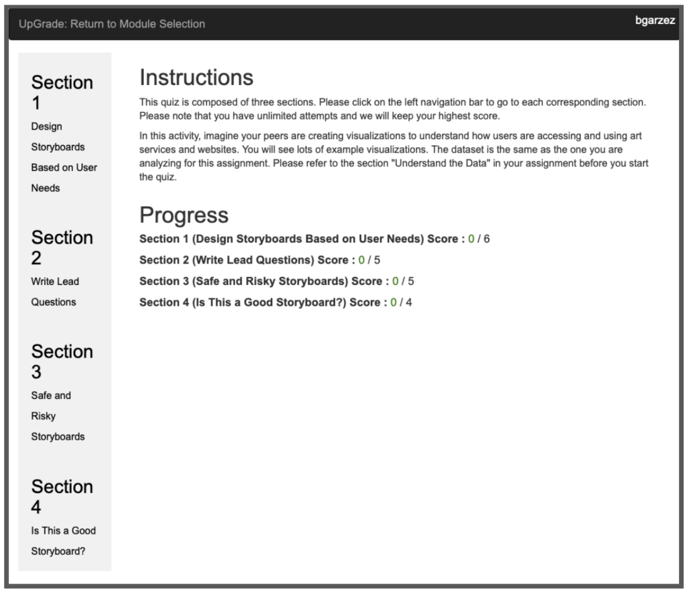
Module Selection

Modules

Please select a module to continue:

Module: Storyboard

Fig. 10. Screen capture of module selection screen interface.


UpGrade: Return to Module Selection
bgarzez
Section 1 Design Storyboards Based on User Needs
Section 2 Write Lead Questions
Section 3 Safe and Risky Storyboards
Section 4 Is This a Good Storyboard?

Instructions

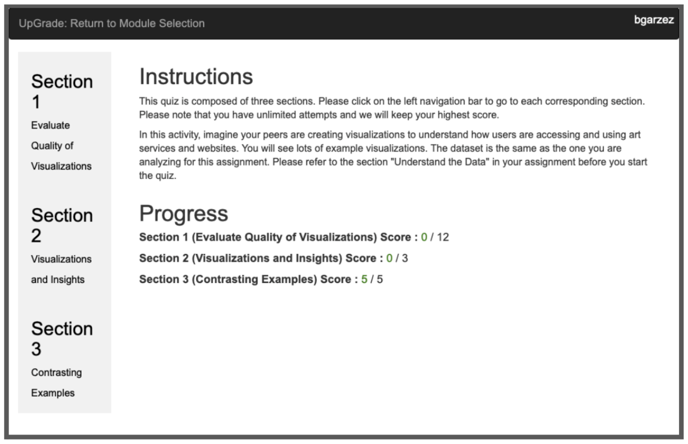
This quiz is composed of three sections. Please click on the left navigation bar to go to each corresponding section. Please note that you have unlimited attempts and we will keep your highest score.

In this activity, imagine your peers are creating visualizations to understand how users are accessing and using art services and websites. You will see lots of example visualizations. The dataset is the same as the one you are analyzing for this assignment. Please refer to the section "Understand the Data" in your assignment before you start the quiz.

Progress

Section 1 (Design Storyboards Based on User Needs) Score : 0 / 6
Section 2 (Write Lead Questions) Score : 0 / 5
Section 3 (Safe and Risky Storyboards) Score : 0 / 5
Section 4 (Is This a Good Storyboard?) Score : 0 / 4

Fig. 11. Screen capture of a module's home page interface.


UpGrade: Return to Module Selection
bgarzez
Section 1 Evaluate Quality of Visualizations
Section 2 Visualizations and Insights
Section 3 Contrasting Examples

Instructions

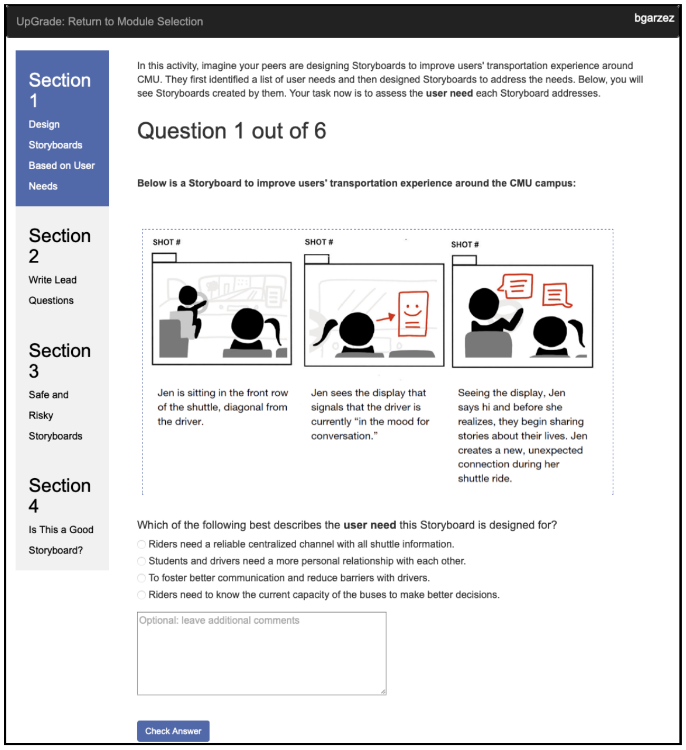
This quiz is composed of three sections. Please click on the left navigation bar to go to each corresponding section. Please note that you have unlimited attempts and we will keep your highest score.

In this activity, imagine your peers are creating visualizations to understand how users are accessing and using art services and websites. You will see lots of example visualizations. The dataset is the same as the one you are analyzing for this assignment. Please refer to the section "Understand the Data" in your assignment before you start the quiz.

Progress

Section 1 (Evaluate Quality of Visualizations) Score : 0 / 12
Section 2 (Visualizations and Insights) Score : 0 / 3
Section 3 (Contrasting Examples) Score : 5 / 5

Fig. 12. Screen capture of another module's home page interface.


UpGrade: Return to Module Selection

bgarzez

Section 1
Design Storyboards Based on User Needs

Section 2
Write Lead Questions

Section 3
Safe and Risky Storyboards

Section 4
Is This a Good Storyboard?

In this activity, imagine your peers are designing Storyboards to improve users' transportation experience around CMU. They first identified a list of user needs and then designed Storyboards to address the needs. Below, you will see Storyboards created by them. Your task now is to assess the **user need** each Storyboard addresses.

Question 1 out of 6

Below is a Storyboard to improve users' transportation experience around the CMU campus:

Jen is sitting in the front row of the shuttle, diagonal from the driver.

Jen sees the display that signals that the driver is currently "in the mood for conversation."

Seeing the display, Jen says hi and before she realizes, they begin sharing stories about their lives. Jen creates a new, unexpected connection during her shuttle ride.

Which of the following best describes the **user need** this Storyboard is designed for?

- Riders need a reliable centralized channel with all shuttle information.
- Students and drivers need a more personal relationship with each other.
- To foster better communication and reduce barriers with drivers.
- Riders need to know the current capacity of the buses to make better decisions.

Optional: leave additional comments

Fig. 13. Screen capture of a section's question page.

VI. Conclusion

In this work, I discuss the research behind UpGrade's learnersourcing approach to open-ended assignment practice with multiple-choice questions automatically generated from prior students' solutions to open-ended problems. I explain the issues with UpGrade's independently-running modules, and I compare two solutions. I then describe the implementation goals of UpGrade Modules, expanding upon the templated server code, templated HTML files, and unified database. Finally, I reveal the results of the project and discuss how UpGrade Modules compares to the target implementation. With additional development and consolidation, UpGrade Modules can stand alone as a fully-functional application and allow instructors to easily add and adjust modules, sections, and questions to maximize the learning outcomes of their students.

Acknowledgment

This work was created as part of the Lifelong Learning Lab at the University of Michigan, led by Assistant Professor Xu Wang. I thank Professor Wang for adding me to the lab team and for guiding the development of the UpGrade Modules application. I also thank Lukas Stauffer and Alyvia Meanor for participating in user tests of the web site and interface.

References

[1] X. Wang, S. T. Talluri, C. Rose, and K. Koedinger, "UpGrade: Sourcing Student Open-Ended Solutions toCreate Scalable Learning Opportunities," Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale. ACM, Jun. 24, 2019. doi: 10.1145/3330430.3333614.