

ReMind and CAPITAL Passages

Katie Stasaski and Elsbeth Turcan
Senior Design Homework 2: Component Design

Section 0: What Is ReMind and CAPITAL Passages?

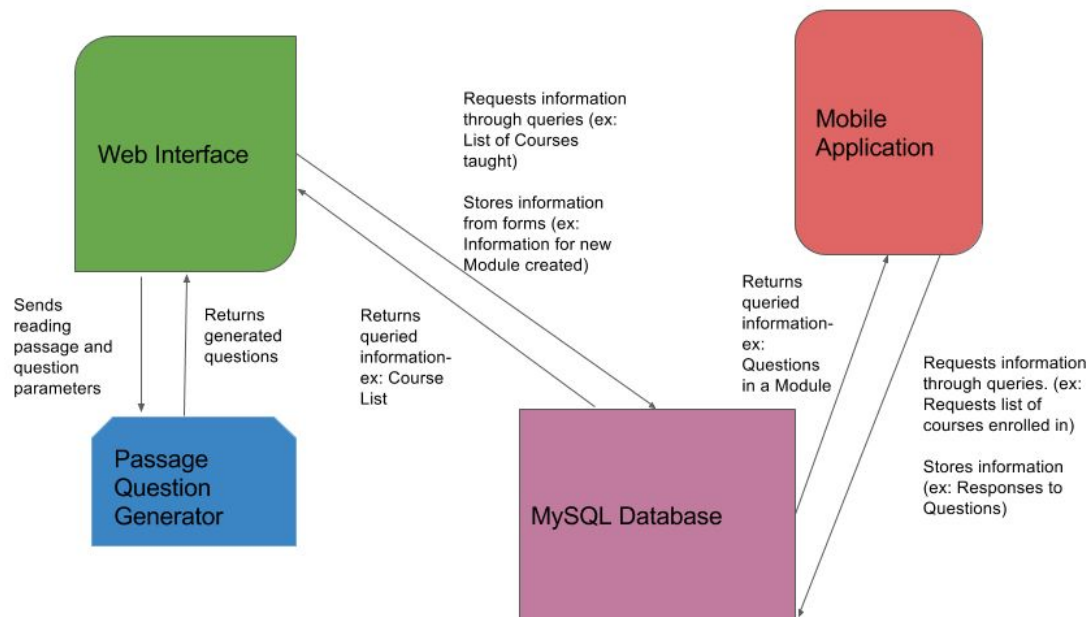
CAPITAL Passages is an application which aims to combat the problem of Adult Illiteracy by automatically generating reading comprehension questions and sending them to a smartphone application. ReMind is its backing user interface, which allows instructors to create educational materials in many fields and send them to their students for practice, evaluation, or survey information.

NOTE: This project is quite large and there are some external members working on the GUI and separate algorithmic components. We, Katie and Elsbeth, will describe CAPITAL Passages in detail because it is our algorithmic contribution, but we will perform a more general overview of other GUI components to which others may contribute.

Section 1: Components

1.1. Components Overview.

The reMind and CAPITAL Passages project has four major components, described in the image below.



A general description of each component is as follows:

- **Instructor Website:** A web interface written in HTML and Scala that Instructor and Administrator users of the reMind system will use to manage their courses, modules, questions, students, and personal profile information.
- **Mobile Application:** A mobile application written in HTML and AngularJS/JavaScript and built in Cordova that Student users of the reMind system will use to view courses, modules, questions, personal progress, rankings, and social information like announcements. Being built in Cordova will allow this application to run on any mobile device platform, including but not limited to iOS, Android, Windows Phone, Windows, BlackBerry 10, and Internet browsers like Google Chrome.
- **MySQL Database:** A database which stores all user information from Instructors and Students and can be maintained by Super Administrator users. The structure of Play will be described below, but the database contains all the information on the Models created in Play Framework (which are things like Courses, Users, Questions, Institutions, and so on).
- **Passages Question Generator:** A back-end plug-in module to reMind that comprises the entirety of CAPITAL Passages. This module will take in a passage of English text and some parameters from the user and generate reading comprehension questions from it.
 - **Question Ranking Component:** A sub-component of the question generator that will evaluate generated questions on several dimensions to determine which ones to show to the user first.

1.2. Pseudocode and Component Operation.

We will give the pseudocode of a significant section for each component; some components are larger than the work we will personally do on them for this project, and if this is the case we will also present a general overview of the segments that are less relevant to our contributions. The segments that deal with GUI operations will also be described in more detail in *Section 4: GUI*. A notable exception is the MySQL database, which we do not program to perform any specific operations and only exists to store data; we will describe it more thoroughly in *Section 2.2: Data Structures*.

i. Website

- This component is largely GUI-based. We will describe a sample workflow (the most important workflow relevant to our contributions) and refer back to it in *Section 4: GUI*.
- The website communicates with the database to fetch, modify, delete, and store information through Play Framework. When new Questions are created, for example, they are stored in the database; when a user updates their profile information, their User object is modified in the database; when an instructor deletes a course it is deleted from the database, and so on.

Example sub-component: Passages Interaction with User

Begin: User is on the question-creation page and clicks a button indicating that they wish to generate questions with Passages.

Passages screen is displayed (See *Section 4: GUI* for graphical representation)

Display passage input box and “browse passages” button

Display options for how many questions to generate and which types (true/false, multiple choice, and factual)

Display submit button

If user clicks browse passages:

Display a list of all passages currently stored in the system in a modal

If the user clicks on a passage:

Close the modal

Paste the selected passage into the passage-selection box

Else if the user clicks exit:

Close the modal

End if

End if

If the user modifies a parameter:

Copy that parameter change over to be passed to the question generator

End if

If the user clicks the submit button:

If the passage is not filled in:

Alert the user that input is required

Else:

Send the passage to the question generator

Receive questions

End if

End if

When questions are received:

For each question in the list of questions:

Display question to user with a button to delete it

End for

Display re-generate button

Display submit button

If the user clicks a delete button:

Remove the selected question from the working list

End if

If the user clicks the re-generate button:

Goto “send the passage to the question generator”

End if

If the user clicks the submit button:

For each question still in the accepted list:

```

        Save the question to the database
    End for
    Close Passages modal
End if
End: User is returned to the question-creation page.

```

ii. Mobile Application

- The mobile application communicates with the database through HTTP: protocols, routed by Play Framework as described in *Section 5.2. Security*. When a student submits a completed module, the record of their performance is sent to the database; when a student views their courses, the list of their courses is sent from the database, and so on.
- This pseudocode is a combination of a workflow and an algorithm. The question answering is an interaction with the user, while the question evaluation (including how many questions to display, etc.) is more of an algorithm that can have pseudocode.

Example Sub-component: Question Answering

Input: List of questions from the server

While question list has more questions:

 Get next question from question list

 Display question.text

 For answer choice in question.choices

 Display choice

 End for

 Set answer_listener on submit button

End while

answer_listener(click event e):

 If (first time clicking button):

 Flip boolean for first time clicking button

 Create new QuestionResponse() object

 Set user response to be the choice the user selected

 Add this QuestionResponse to a List of QuestionResponses

 Else:

 Flip boolean for first time clicking button

 Prepare responses to server

 Submit responses to server

 Prepare new question

 End if

End listener

Output: Final ModuleRecord and QuestionRecords responses to server

iii. Passages Question Generator

Pseudocode:

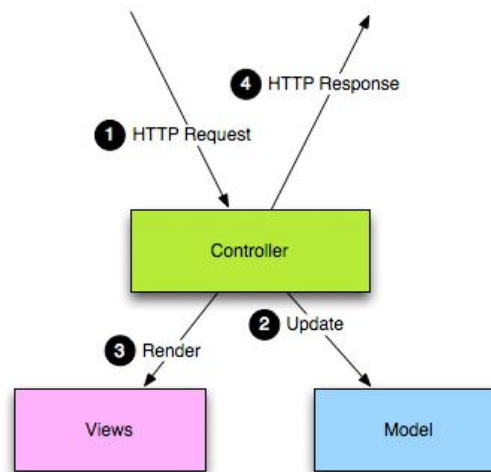
```
Input: English-text passage, user parameters described above
  Clean passage and create uniform formatting
  Parse passage with Stanford CoreNLP
  Tag passage with Stanford CoreNLP
  If user selected factual questions
    Generate as many factual questions as user wants and we have
  End if
  Create fact matrix (described in more detail in Section 2.2: Algorithms)
  If user selected true/false questions
    Generate as many true/false questions as user wants and we have
  End if
  If user selected multiple choice questions
    Generate as many multiple choice questions as user wants and we have
  End if

  create a new SupplementaryContent object in the database
  assign this content object the passage as its value
  for every question generated:
    create a new Question object in the database
    question.prompt = generated.prompt
    question.extraContent = passageContent
    question.choices = generated.choices
    add this new Question object to a list
  end for
Output: List of questions
```

1.3. Model-View Controller Framework.

i. How does MVC Work?

Model-View-Controller Framework is a system design which divides our software into three distinct parts--models, views, and controllers. Models represent data that is used in the application. These are each represented as tables in the MySQL database and manipulated in the code as Java objects. The views represent what the user sees when they use our system. Each one of these is an HTML web page with scala to access data passed in from the database. The controllers control data and user actions; for example, a controller may be used to save new questions and courses that a user creates, and it may also be used to match the user's front-end action of clicking a "generate questions" button to the back-end action of the Passages generator accepting input and generating questions.



A graphical description of MVC framework.

ii. What are Our Models?

Our system includes the following models: (Red models are important to our contribution to the system)

- AnswerDecimal - an answer choice of type decimal
- AnswerInteger - an answer choice of type integer
- AnswerText - an answer choice of type text
- AnswerWord - an answer choice of type word
- Basis - the basis (or, correct answer choice) of a question
- Choice - a choice for a given question
- Content - content associated with a question
- ContentFile - a file, e.g. a picture, associated with a question
- ContentText - a text string, e.g. a passage, associated with a question
- Course - a course, created by an instructor, that can have enrolled students
- FlaggedQuestion - a question that a student has flagged as incorrect
- Institution - an educational institution, e.g. Washington Literacy Center
- ListRecord - a Record of a student's completion of exercises in a list, e.g. a module score
- Message - a message from an instructor to a student
- Module - a collection of questions. A course consists of many modules
- ModuleRecord - a record of a student's success in a given module
- ModuleList - a list of modules that comprise a course
- Prompt - a prompt the student is given to answer a question
- Question - A question, of either type evaluation, practice, or survey
- QuestionList - a list of questions, used in modules
- QuestionRecord - Represents a recorded answer that a student has answered the question with

- **Response** - Represents a response to a given module that a student has completed on the app
- **StudentInInstitution** - represents a student in a given institution
- **StudentQuestion** - represents a question that has been submitted by a student
- **Tag** - used to tag a question as a certain category
- **User** - user of a system, either a super admin, an admin, an instructor, or a student

Section 2: Algorithms and Data Structures

2.1 Algorithms.

Cleaning and pre-processing a passage input to the question generator:

Function: ParsePassage

Input: English passage

```

    Clean passage to be formatted with a uniform format
    Parse passage with Stanford CoreNLP
    Tag passage with Stanford CoreNLP
return parsedPassage
```

The function below will generate simple factual questions from a parsed passage. This will only generate simple questions. This is mainly a back-up in case our question generation method does not work for a certain passage OR if a teacher wants to generate significantly easier questions for a passage.

Function: GenerateSimpleQuestions

Input: ParsedPassage (passage that has been run through Stanford CoreNLP parser)

```

for each pattern in searchPatterns (collection of parse tree patterns which make a good
question. Ex: NP=noun .. (VP=verb < VBD !< VP << SBAR=because) to generate
a “because” question. This will match sentences like Katie was very excited because
her senior design was progressing quite nicely.)
```

```

    for each sentence pattern is found in parsedPassage
        apply parse manipulation that matches with foundPattern to
        generate question (ex: delete because; insert (WRB why) >1 verb; move
        noun >3 verb;
        SaveQuestion();
```

Function: StoreFactInfo This function takes in a passage and stores the encountered facts in a fact matrix, a data structure we created to store relevant question generation info

Input: ParsedPassage (Passage that has been run through the Stanford CoreNLP parser)

```

FactMatrix factMatrix = new FactMatrix();
for each independent clause encountered in ParsedPassage
    subject = encounteredSubject;
    predicate = encounteredPredicate;
    if(factMatrix.contains(subject))
```

```

        factMatrix.subject.add(predicate);
    else
        factMatrix.add(subject);
        factMatrix.subject.add(predicate);

```

Function: GenerateComplexQuestions This function utilizes our data structure, the fact matrix, to generate questions

```

for each subject in the FactMatrix
    generateQuestion using two different facts (T/F question)
    generateQuestion using other subjects in passage as distractor

```

2.1 Data Structures.

- Fact Matrix:** Our fact matrix stores associations between subjects and predicates that are found in a passage. From there, we have a collection of the facts encountered through the passage and can use this information to generate more complex question.
 - The length of the matrix is of size n , where n represents the number of total subjects in a passage.
 - The row of a matrix is a linked list of size m , where m is the number of predicates, or facts, associated with the subject. This will be different for each row in the matrix.
- Database:** Our system stores information in a MySQL database. Play Framework integrates automatically with MySQL to turn a model into a table in the database. In order to do this, you just have to create a variable in a model, and Play will automatically create the variable in the MySQL table. For example, this is the script Play generates to create the question table. Each of the values (message_id, is_global, submitter_id, etc) corresponds to a variable in the Question model described in *Section 1.3 MVC Framework*.

```

create table question (
  id                bigint auto_increment not null,
  retired           tinyint(1) default 0,
  message_id        bigint,
  submitter_id      bigint,
  has_subquestions  tinyint(1) default 0,
  is_global         tinyint(1) default 0,
  created_time      datetime(6) not null,
  updated_time      datetime(6) not null,
  constraint uq_question_message_id unique (message_id),
  constraint pk_question primary key (id))
;

```


Section 3: External APIs

- Stanford CoreNLP
 - Takes a passage as input and returns a parsed passage. Comes with a variety of tools, including proper noun parser, which can distinguish if a proper noun is a person, place, or organization.
 - Parse tree is returned in XML format and can be used with other StanfordNLP tools to form questions
- Tregex
 - Language which is created by same group that created Stanford CoreNLP
 - Matches parse tree structure in parsed passage. This language is used to determine which parts of a sentence
- Tsurgeon
 - Created by same group as Stanford CoreNLP
 - Used in conjunction with Tregex
 - Once the parse tree that matches a pattern has been found, Tsurgeon scripts manipulate the sentence to form a question
- Play Framework
 - <https://www.playframework.com/documentation/2.4.x/api/java/index.html>
 - Play Framework is a Model-View-Controller system
 - Views- different graphical views of our system
 - Model- represents the different objects of our system (ex: Users, Questions, Courses, etc)
 - Controllers- control user actions (ex: controls automatic generation of questions when user requests it)
 - Play Framework uses a library called Ebeans to communicate with the MySQL database.
- FasterXML.Jackson-Databind
 - <http://fasterxml.github.io/jackson-databind/javadoc/2.4/>
 - Java library which handles JSON parsing and mapping to Objects, and vice versa
 - Used to translate data between the server and mobile application
- Ebean
 - <http://ebean-orm.github.io/docs/>
 - Library used by Play Framework to communicate with the database programmatically in Java classes

Section 4: GUI

1. Website Pages

Link to video demo for how users should navigate through the site:

https://www.youtube.com/watch?v=PO_pMtX1m40

Login page: This is the page the instructor will see when they first navigate to our site:

The screenshot shows the login page of the CAPITAL Instructor Site. At the top, there is a dark blue header with the site's logo and name. Below the header, a yellow banner displays a message about redirection. The main content area features a 'Please Login' box with input fields for email, password, and a dropdown menu for 'CAPITAL Words'. A 'Login' button is positioned below these fields. A separate box asks 'New to CAPITAL?' and provides a link to register. The footer contains the university's name and copyright information.

CAPITAL Instructor Site
CAPITAL Passages

You have been redirected. Please log in to continue.

Please Login

Email: jenhill2

Password: *****

Words: CAPITAL Words

Login

New to CAPITAL?

Click [here](#) to register for an instructor account.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC
Copyright © 2013. All rights reserved.

Note that the fields of the login page will be checked against users existing in the database. if they are correct, the user may go to the dashboard; otherwise, an error message appears on the same page.

Edit profile page. This is the page instructors can use to edit their information.

The screenshot shows the 'Editing Instructor' profile page. On the left is a sidebar with navigation links: Dashboard, My Courses, Inbox (with 3 messages), and ADMIN. The main content area is titled 'Editing Instructor' and shows the user's name 'Klaz Unent'. Below this is a form with fields for First Name, Last Name, Email, Username, Password, and Confirm Password. A 'Submit' button is at the bottom of the form.

CAPITAL

Messages 3 Klaz Unent

Dashboard

My Courses

Inbox 3

ADMIN

Editing Instructor

Klaz Unent

First Name
Klaz

Last Name
Unent

Email
klaz@email.com

Username
klaz

Password
Password

Confirm Password

Submit

Note that each field of the form is associated with an aspect of the User model.

This is the page used to create a new Module Page. Each course has a collection of modules, which each have a collection of passages with questions. (note the CAPITAL Passages button)

{Module #}

{Type} ▾

Add a Question

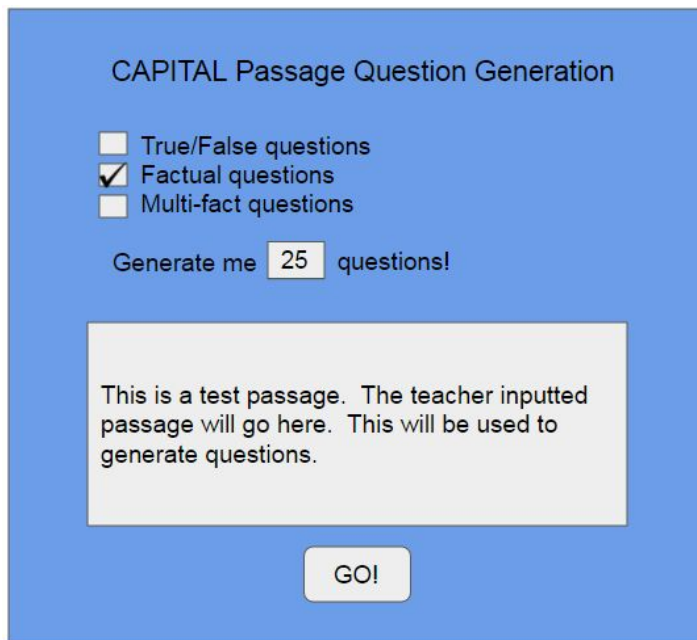
Search

Type	Question
1 <div>▢</div> Multiple Choice ▾	<div>Prompt:</div> <div>Which of the following colors has the highest wavelength?</div> <div>CAPITAL Passages</div> <div>Add content</div> <div>Choices: 2 ▾</div> <div><div><input type="radio"/> Blue</div><div><input checked="" type="radio"/> Red</div></div> <div><div>tag tag tag</div><div>another tag</div></div>

Note that this is the screen described by the Passages algorithm in *Section 1.3 Pseudocode and Component Operation*, and clicking it leads into the screens described below.

2. Passages Interface

Generate Questions from Passages Screen. The instructor is given an option of parameters and how many questions they want to generate.



CAPITAL Passage Question Generation

☐ True/False questions
☒ Factual questions
☐ Multi-fact questions

Generate me questions!

This is a test passage. The teacher inputted passage will go here. This will be used to generate questions.

GO!

Edit generated questions page where teachers can review our generated questions.



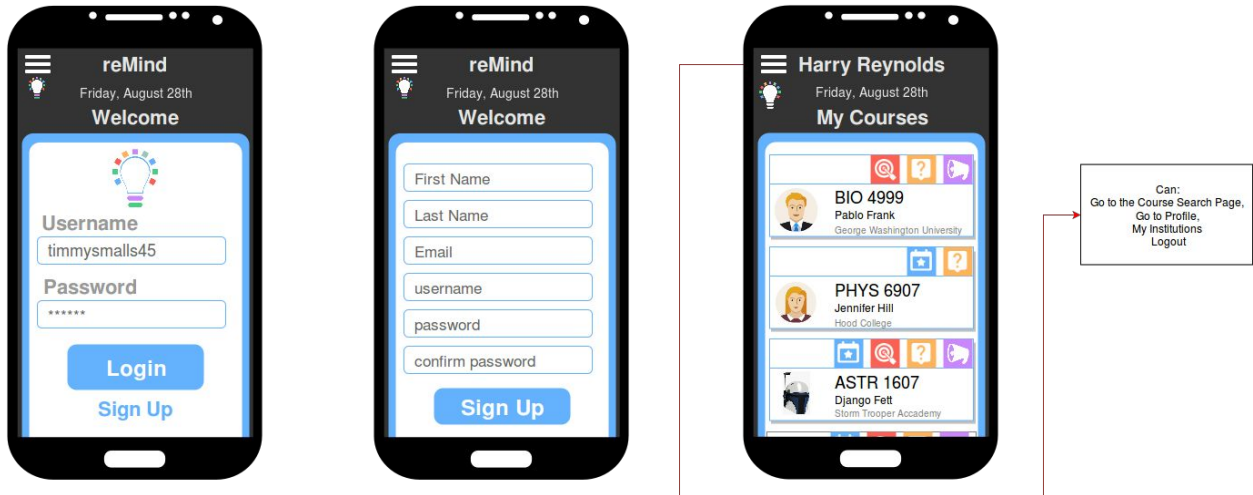
Here are your generated questions:

☐ Who went to the place?
☒ What happened on January 1st?
☒ Why was the event cancelled

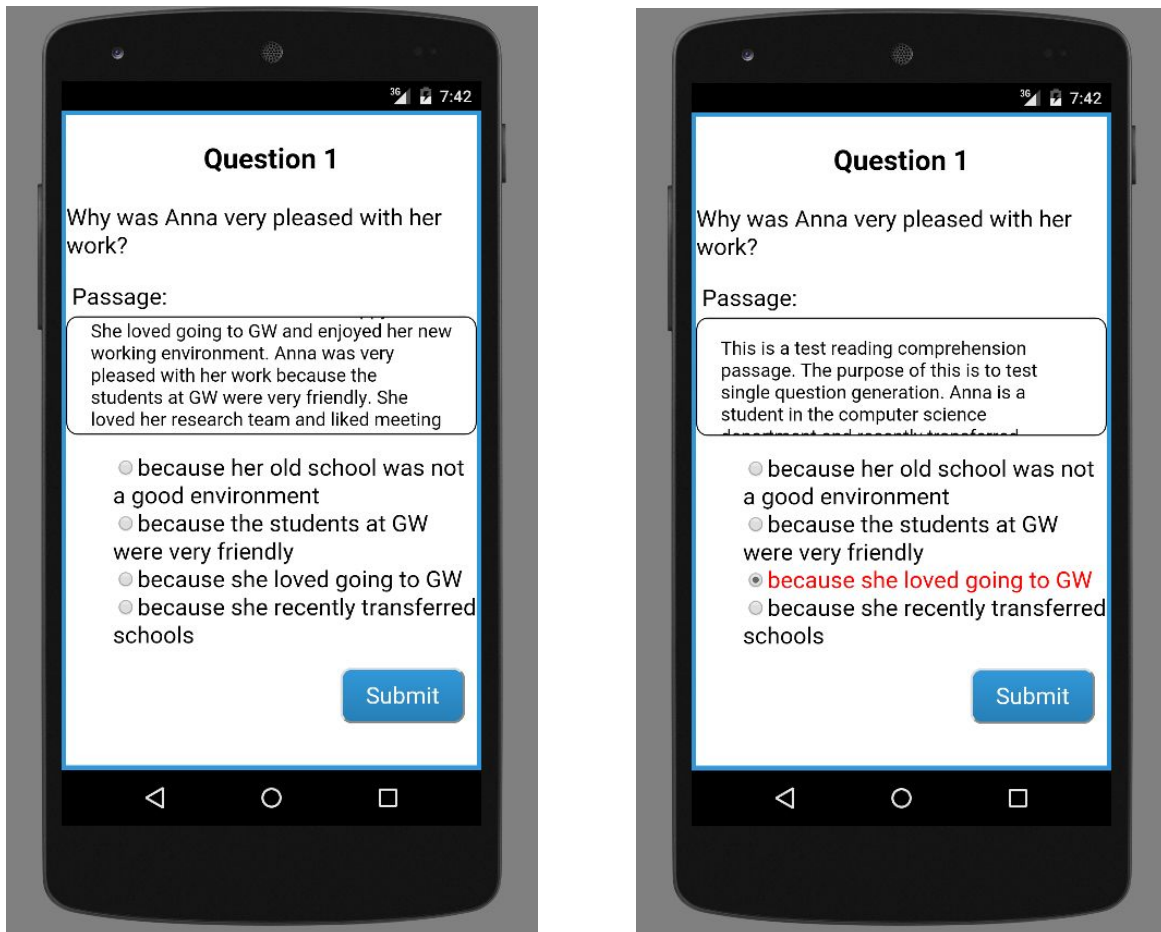
Re-generate me questions!

3. Mobile Application Pages

Setup and course navigation:



Passage display and question answering:



Section 5: Network Components

5.1. *Protocols and Data Formats.*

- Data will be sent between the website and the server by Play Framework, which uses a Model-View-Controller framework to glue together the View (the website) and the database. This means its protocols and data formats are Java objects directly; we will create Models which represent the different units in our system (Users, Courses, Questions,...) and use Java functions to manipulate them, and Play Framework will modify them in the database accordingly.
- Data will be sent between the mobile application and server using JSON. Specifically, we will use JavaScript's natural JSON parsing on the application end, and FasterXML.Jackson (which is included in Play Framework) on the server end.

5.2. *Security.*

- User data will be stored in CAPITAL's servers, which we are responsible for maintaining.
- Users' passwords are encrypted.
- Pages check whether or not users are logged in and whether or not they have permission to view them before rendering.
- Play Framework handles queries in such a way that SQL injections are impossible. Page requests are sent to a routes file, and the routes file directs any requests that require database queries to a Java function in a controller. This Java function accesses the database programmatically, using data from the stored data in the database when possible instead of user input (e.g., the user generally clicks on a course they want to access instead of typing in its name) and being sure to properly cast, authenticate, and truncate data that must be from the user (e.g., when the user inputs their username).

5.3 *Communication Failures & User Feedback.*

- All user interactions with the website are authenticated; if there is any error in user input or in communicating with the server, authentication generally happens before the page is reloaded and will display an error message or modal to the user indicating what went wrong (e.g., "This username is already in use. Please try again.").
- In more extreme cases, the server redirects the erroring page to the index page for the user to try again. In this case we are not likely to be able to display the user a specific error message, but the system will not break.
- When the mobile application cannot communicate with the server, it will display a message indicating that something is wrong with the connection, asking the student to perhaps check their Internet connection, and not load into the rest of the application when it cannot fetch data from the server.
- Data is generally saved to the database as soon as it is received from the user--e.g., when the user creates a new Course, the course creation function will create a new Course object and call Course.save(), which saves the data in the database, before it exits.