PatientBank is a platform for individuals to aggregate their own health information. Especially for high utilizers of health care, comprehensive health records are incredibly valuable. While electronic health records have become more prevalent, patients’ access to their own information remains limited. PatientBank solves this problem for the patient by requesting, gathering, and maintaining a comprehensive and unified record on his or her behalf.

**Background**

The first version of PatientBank was developed for Professor Piskac's CPSC 439 Software Engineering class in the spring of 2014. The project included a flexible framework for storing health records, a personal health record web interface, and a simple XML-based data integration algorithm. The original team included Feridun Mert Celebi, Paul Fletcher-Hill, and Rafi Khan. Over the summer of 2014, Celebi and Fletcher-Hill continued to develop parts of the application, including two satellite applications for administrators to view patient information and providers to upload health records.

In the fall of 2014, Abhishek Chandra and Mikayla Thompson joined the project as collaborators. Chandra gained exposure to the project in the spring of 2014, when he worked on another team in the same CPSC 439 Software Engineering class, and Thompson spent the summer of 2014 at the electronic medical records giant, Epic Systems, where she developed an interest in health information technology. During the fall of 2014, Fletcher-Hill, Celebi, Chandra, and Thompson developed CPSC 490 projects to continue developing the core PatientBank functionality, while each focusing on a single part of the growing platform.

This spring, Thompson will continue her project from the fall. The other team members are continuing to work on developing PatientBank, but are not doing additional semesters of CPSC 490.

**First Semester Work:**
PatientBank’s current model centers on document storage and sharing: patients request their medical records from previous healthcare providers, they’re provided in electronic form or scanned in, and then patients can share the documents with their current providers. Both patients and doctors are sifting through large amounts of information and search capabilities would be very helpful in finding clinically important documents. However, many documents have been scanned from paper copies and are, in effect, image files and not amenable to text searches. This project (a) utilized optical character recognition (OCR) technology to extract searchable text from document images; and (b) found likely-inaccurate OCR artifacts and corrected them to English words (essentially a spell check).

The project used the Tesseract OCR utility to develop a Ruby 2.1.2 command line utility (as well as a Ruby gem for integration into larger projects). Text was recognized from the image file, and the extracted words were compared to both English and medical dictionaries. For words that did not appear in either dictionary, a search was performed to find all dictionary words within an edit distance of 2, and any words found were added to the search text for the document. The error correction functionality improved text coverage by up to 60%.

**Second Semester Proposal:**
As PatientBank is developed further, it will move from basic document storage to more advanced forms of data integration and analysis, so information can be synthesized, condensed, and presented in more informative ways than it’s original form. For instance, test results can be graphed over time, instead of each result only being stored individually in separate documents. Similarly, visit notes can be stored as discrete entries, rather than only as part of document images.

Accurate text recognition is obviously an essential step in that direction, but meaningful information extraction depends on a greater understanding of the document’s structure. Recognizing headers, section titles, narrative paragraphs, and tables, to name a few elements, will be essential to identify extracted information.

HTML is already specifically designed to describe a document’s structure and contents in a well-defined and widely manipulatable way. It would be a major step towards full integration to be able to translate document images into an HTML approximation.
This project's goal will be to develop a tool (in the format of a Ruby Gem) that can accept document images (in PDF form, for instance) and return an HTML document that describes the image. Eventually, it would be great to produce a fully accurate and descriptive HTML version, but for the sake of this semester, the two elements to be identified are paragraphs and tables. Paragraphs (the \texttt{<p>} tag) will include traditional paragraphs, but are also a catch-all category, so all text other than tables will fall into this element (for instance: titles, headers, etc.).

There are five main steps to reach this goal:
1. Find and separate items
2. Classify items (as tables or paragraphs)
3. (For tables) Split into cells (by row or column)
4. Extract text from items and cells
5. Format as HTML

Outlines of algorithmic solutions to each of these steps are described below. Refinement and implementation of the algorithms is the work planned for this semester.

Finding and separating is a visual, image-based process. Generally, it involves finding blocks of connected, related text by proximity and density. One technique, that showed promise when explored slightly last semester, is to blur the document, and then start a rectangle on a dark-colored pixel. Each side of the rectangle is expanded in turn until all of the edges are on light-colored pixels. This rectangle should now outline an area of dense text, for instance a paragraph. All of this can be done using the ImageMagick software and its Ruby interface.

Classifying is the determination of whether a particular item found in the previous step is a table or (by default) a paragraph. The distinguishing characteristic for tables is that they include almost entirely monotonal columns (all light pixels, or all dark pixels) to indicate the boundaries between columns either by leaving a blank space or including a border line. (Horizontal lines are not as distinguishing because paragraphs include blank rows between each line of text.) Items with these properties will be preliminarily classified as tables. Once an item is classified as a table, it must be split into cells, using the same all-light/all-dark principle as was used to identify tables. There should be at least one
such line in each direction (horizontal and vertical), for a minimum of four cells in the
table. (If these can't be found, the item will be reclassified as a paragraph.) The image
will be split along these lines.

Extraction is the most solved step of this process, and closely related to the work from
last semester. The images, split by paragraph and cell, can be run through the
Tesseract OCR engine, using the components of the gem written last semester.
The final step, once items are identified and the text has been extracted, is to reformat it
as a simple HTML document. Based on the vertical coordinates of each item in the
image, they will be turned into HTML and inserted. Paragraphs are simple, as they only
require putting the extracted text between <p> tags. For tables, the structure is slightly
more complex, where a <table> tag surrounds the element, and the internal
components are <tr> tags for rows, and <td> tags for each cell within the row. Given
how the cells were selected from the table item, it should be fairly straightforward to put
the extracted text into this format.

A development priority for this semester will be to use test driven development (TDD)
techniques. In TDD, the behavior of the program to be written is described with
automatic tests, which will, of course, fail before the program is written. As the program
is developed, more and more tests will pass. This practice encourages well-thought-out
design and accountability for program functionality. With this philosophy in mind, the
first project for this semester will be to determine suitable examples and write a
comprehensive set of tests that define the behavior for each step described above.