Team 11: “MiniWatt!”
Product Backlog

Dylan Brashear
Grant Jochums
Craig Wilhite
Adam Worthington
Robbie Mantzoros
Chris Doak

Product Backlog

Problem Statement:

- The amount of time spent filling out low level class assignments is not proportional to the benefit gained from completing them as most questions are looked up online and answers are copied without any discretion.

Background Information:

- We have all gone through introductory level classes while at university. Different people have different learning styles, and not everyone benefits from monotonous worksheets. Our goal is to save valuable time for those who do not find a reasonable amount of benefit in these worksheets.

Environment:

- C# is the programming language of choice to develop the client-side application. Development will use the .NET framework and target the Windows platform. ITextSharp is an external library that will be used to aid in reading in PDF files and extracting input for our application.
- The backend of the application shall reside within Google App Engine and shall utilize Java as the programming language of choice. All relevant database work shall use MySQL as the database language.

Requirements:

Functional		
Backlog Id	Functional Requirement	Hours
001	As a user, I would like to upload a PDF document containing questions.	2
002	As a user, I would like to upload a PDF document containing a source that is relevant to the questions document.	2
003	As a user, I would like to enter individual question/queries	2
004	As a user, I would like to receive the answer to an individual question in a text box	2
005	As a user, I would like to upload an image of a document containing questions.	20
006	As a user, I would like to upload an image of the source material	20
007	As a user, I would like to receive a PDF document containing answers to my questions	200
008	As a user, I would like to have a functional user Interface	4
009	As a user, I would like to view my previously submitted and received documents	15

Non-Functional Requirements:

- The program must provide answers of a reasonable quality.
- The program must run efficiently enough to save the user time.
- The program must be testable, and be tested thoroughly.
- The program must be compatible with Windows platforms.
- The program must have a simple and elegant user interface

Use Cases:

User Actions	Program Response
Case: Upload a file containing questions without a source file	
1) Select “Open Question File” Button	2) File Explorer box is displayed
3) File is selected	
4) File is confirmed	5) File Explorer disappears
	6) File is converted into strings.
	7) Strings are sent to server for processing
	8) Strings are parsed into questions
	9) Questions are converted into searchable queries.
	10) Scour the internet for potential answers to questions
	11) Determine best result for each question
	12) Return the best result for each question in a collective PDF document
Case: Uploading questions using a source file	
1) Select “Open Question File” Button	2) File Explorer box is displayed
3) File is selected	
4) File is confirmed	5) File explorer disappears
6) Select “Open Source File” Button	7) File explorer box is displayed
8) File is selected	
9) File is confirmed	10) File explorer box disappears
	11) Question file is converted into strings
	12) Source file is converted into strings
	13) Strings for both files are sent to the server

	14) Questions are converted into searchable queries
	15) Scour the ‘source’ strings for potential answers to questions
	16) Determine best result for each question
	17) Return the best result for each question in a collective PDF document.
Case: Uploading questions with an image source file	
1) Select “Open Question File”	2) File Explorer box is displayed
3) File is selected	
4) File is confirmed	5) File explorer disappears
6) Select “Open Source File” Button	7) File explorer box is displayed
8) File is selected	
9) File is confirmed	10) File explorer box disappears
	11) Question file is converted into strings
	12) Read from the image source file into strings
	13) Strings for both files are sent to the server
	14) Questions are converted into searchable queries
	15) Scour the ‘source’ strings for potential answers to questions
	16) Determine best result for each question
	17) Return the best result for each question in a collective PDF document
Case: Uploading an image questions file	
1) Select “Open Question File” Button	2) File Explorer box is displayed
3) File is selected	
4) File is confirmed	5) File Explorer disappears

	6) Read from the image file into strings
	7) Strings are sent to the server for processing
	8) Strings are parsed into questions
	9) Questions are converted into searchable queries.
	10) Scour the Internet for potential answers to questions
	11) Determine best result for each question
	12) Return the best result for each question in a collective PDF document
Case: Asking individual question	
1) Select “one question”	2) Text box is displayed
3) Enter question to be asked	4) Parse the question
	5) Find searchable query
	6) Search the internet using queries
	7) Determine best result
	8) Return best result to question
Case: Viewing submission and answer history	
1) Select “submission history”	2) List view of submissions is displayed
3) Select an individual submission	4) Display files involved in submission (questions, source [if applicable], answers)
5) Select one of the files	6) Open the file