Dreamjob Deliverer

June 20, 2016

Jacob Davis
Connor Dougan
Graceanna Petty

Table of Contents

1. Introduction

1.1 Client Description and Project Background

Salesforce pioneered the shift to cloud computing, and today they're delivering the next generation of social, mobile, and cloud technologies that help companies become customer companies and revolutionize the way they sell, service, market, and innovate. As a technology company of more than 20,000 employees, they give tests to evaluate job candidates. They generally give the test remotely, freeing the candidates from a commute and allowing the test to be taken in a comfortable environment. To maintain the applicability of the test, they restrict the tests shareability, primarily preventing pre-canned solutions. The current solution involves using GoToMeeting as an interface to control a VM that contains the PDF containing the programming project details. This is extremely cumbersome. Also, on Linux using the GoTo web app, the test-taker is unable to control the VM and cannot pan the document, which mandates the use of a Mac or Windows machine while taking the test.

1.2 Product Vision

The goal of this project is to overhaul the programming test process. Our Salesforce contact envisioned a web interface that empowers the test-giver and restricts the test-taker. This application allows the test-giver to manage a repository of tests, administer a test via a shareable link that is time-locked, and collect/archive solutions to the test. In addition, the application stops test-takers from sharing the test and displays a time-remaining clock for their convenience. Job candidates will utilize this web application by receiving a url to a viewable test which displays the test instructions in a way that prevents them from downloading the materials and is readily accessible from Linux, Mac OS, and Windows.

2. Requirements

2.1 Functional Requirements

The Dreamjob Deliverer web application must have separate administrator and user functionality and access in order to provide the best experience for both parties. Administrators have two main uses for the app. They need to be able to create tests and administer them to candidates and administrators also need the ability to review test submissions and update times of any tests already administered. Users will receive a URL to access the test which displays all information and a method for submission on a web page. Users should not be able to download the instructions, or it should at least be very difficult to do so.

Administrator functionality

- View list of administrable tests
- Create new administrable tests
    - Title
    - Write a description for the test type
    - Upload an instructions file
- View test instructions
- Schedule new test instance
    - Start date and time
    - End date and time
    - Candidate name and/or email for identification
    - Create a shareable URL for the candidate to access the test
- View in-progress and scheduled test instances
    - Edit start date and time
    - Edit end date and time
    - View candidate name and/or email
    - Access shareable URL
    - View remaining time
- View completed test submissions
    - View the start and end times (read only)
    - View candidate name and/or email
    - Access the most recent test submission
- Access candidate test page

User functionality

- Access test page via shared URL
- View test instructions
    - Not downloadable

- View time remaining
    - Toggle to show/hide (hide by default)
- Upload submission
    - Browse filesystem for file to upload
        - Limit file size to 100MB
    - Uploading does not end the test so the user is able to upload additional attempts
    - Confirmation of successful upload

2.2 Nonfunctional Requirements

- Administrator authentication using OAuth 2.0
- Deploy as an application on Heroku
    - Can be any kind of application supported by Heroku (language not specified)
- The test taker can upload any file type for submission
- The test instructions are PDF files
- Tests will be stored in a database
    - Can be any kind of database supported by Heroku and the language chosen
- Use github
    - For version control
    - Accessible by the client

3. System Architecture

3.1 Webpage Architecture

At a high level of abstraction, Dreamjob Deliverer needs to be used by two different groups of users: administrators and test-takers. Due to the differing natures of these groups, this application has separate user interfaces for each group. Figure 1 illustrates how these user interfaces interact differently with each group of users and the test database. Test-takers only need to view a single test and provide a submission file, which can be done in a simple interface. Administrators have the ability to view tests as a test-taker would. In addition, administrators also need to be able to create, schedule, view information, and share tests. All of this is accomplished in an administrator interface, separate from the test-taker's interface.

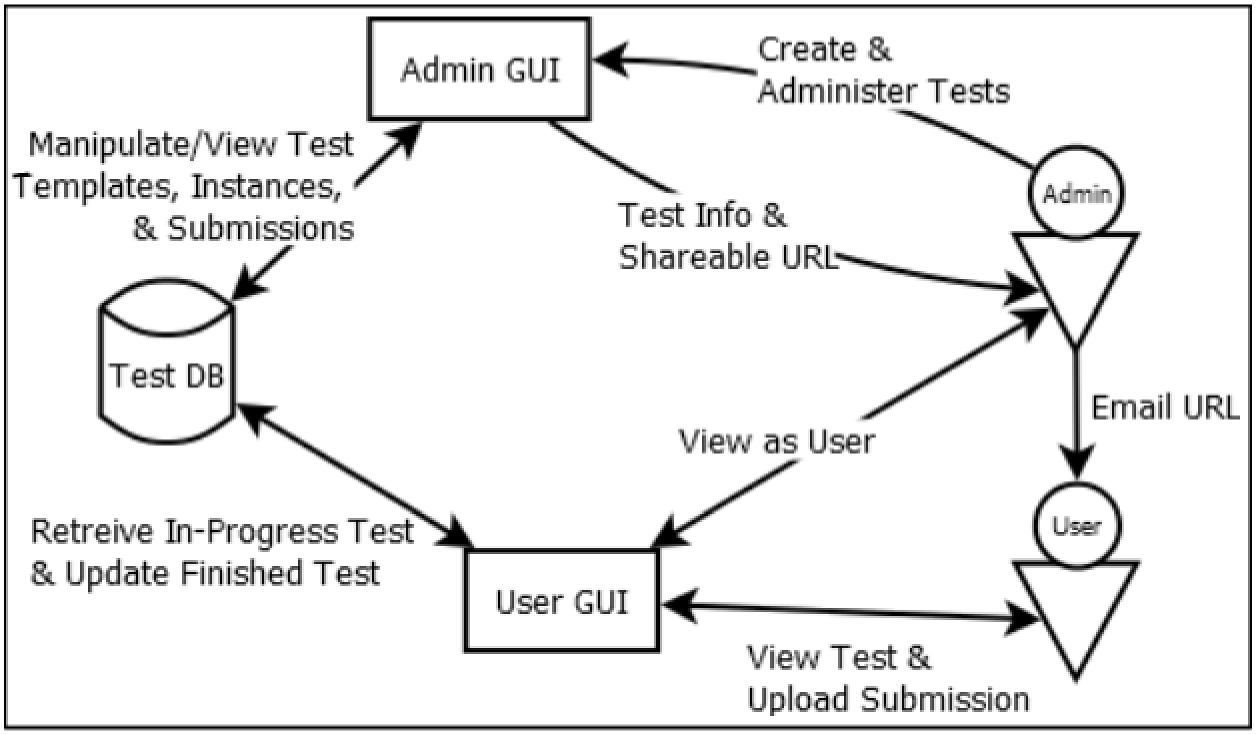

Admin GUI
Create & Administer Tests
Manipulate/View Test Templates, Instances, & Submissions
Test Info & Shareable URL
Admin
Test DB
Email URL
View as User
User
Retreive In-Progress Test & Update Finished Test
User GUI
View Test & Upload Submission

Figure 1 - Webpage Architecture

While, to the users, Dreamjob Deliverer appears to be two separate applications, it is unified into one application through the database. Figure 1 shows how all of the data is stored in the same database and the user interfaces interact with the same data. To separate the two, the test-takers cannot change data through their interface that the administrators have control of and vice versa.

3.2 Database Architecture

When creating the structure for the database used by Dreamjob Deliverer, the various types of data that need to be stored had to be considered. It was necessary to consider the structure of the database because entire files need to be written to the database. These files can be varying sizes and potentially large so a data structure for storing under these circumstances was chosen. Figure 2 shows the entity relationship diagram of the database and how these considerations affected the end result. Two tables were constructed, one for test templates and one for test instances that are currently scheduled. Many instances will have the same set of instructions and title, so separating that data from the test instances saves a significant amount of memory. This saves memory because the same set of instructions doesn't need to be saved in the database multiple times for multiple test instances, instead it can be saved in one place in the database and accessed multiple times.

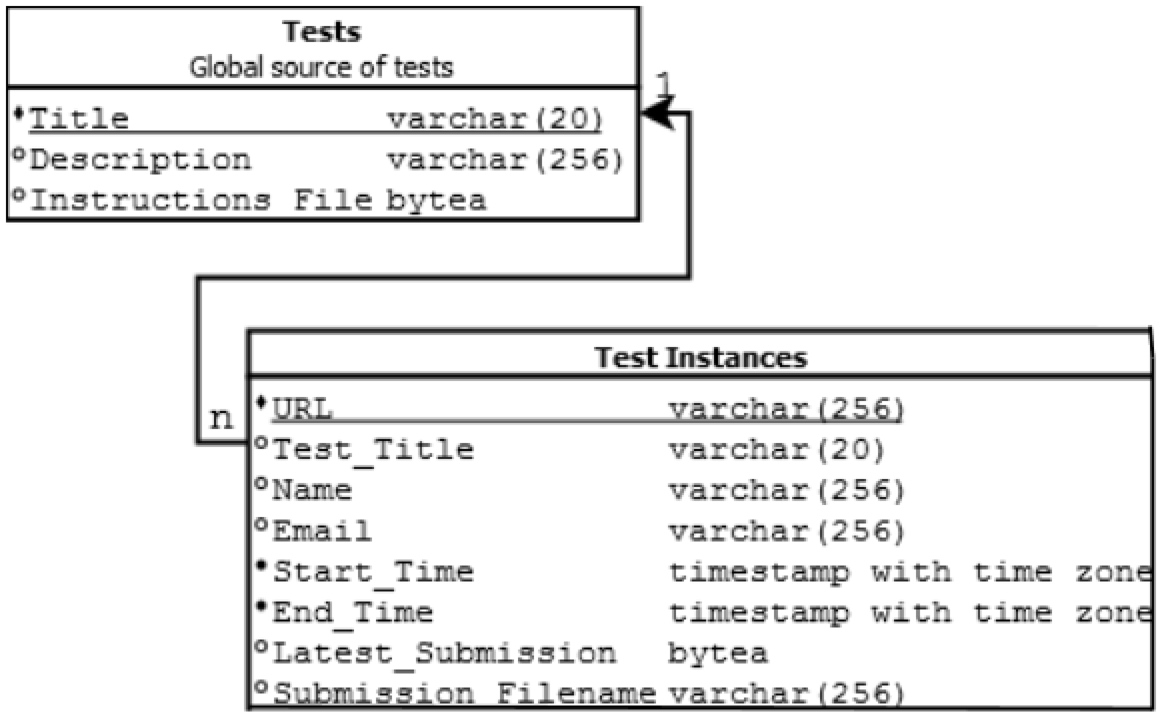


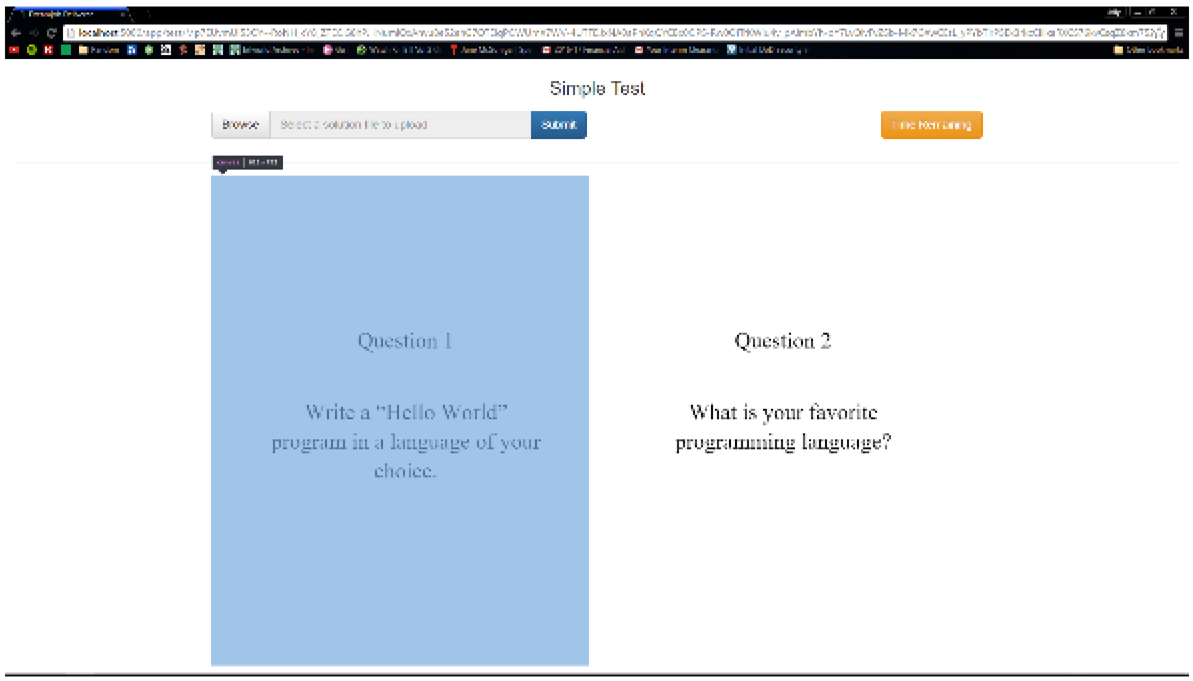
Figure 2 - Database Entity Relationship Diagram

As stated in Section 3.1, the test-taker and administrator interfaces each allow modification of different pieces of data. More specifically, the test-taker interface only grants the ability to alter the latest submission and submission filename entries. The administrator interface grants the ability to modify all other attributes in both tables apart from the URL which is created automatically.

4. Technical Design

To test-takers, the Dreamjob Deliverer application is a webpage that hosts a set of test instructions and allows the test-taker to upload a file for submission. The application's purpose is to provide a test of the candidate's capabilities, therefore it is important that the set of instructions cannot be easily distributed. All instruction files are PDFs stored on the database and are simple to embed in web pages. All browsers embed PDFs differently, consequently it is impossible to prevent the test-taker from simply downloading the file if it is displayed in this manner.

It is unknown what browser the user will use, therefore we had to find a solution with consistent behavior across many browsers. The JavaScript library PDF.js provides the functionality needed for displaying PDF files in a secure manner. PDF.js displays each page of the provided PDF in separate canvas elements on the page, as shown in Figure 3, where one canvas is highlighted to better display the layout. It does this by loading each page one at a time into its own canvas and adding the canvas to the division element that is meant to hold all of the pages. This is demonstrated in Figure 4 where the division element that contains both pages is highlighted. The end result is a series of ordered canvas elements that each move fluidly when the user resizes the webpage. Once the PDF has been embedded, it is simple to remove the option for the user to download the file. Additionally, because all of the PDF data is drawn onto a canvas, it becomes much more difficult to access the raw PDF data.



Question 1

Write a "Hello World" program in a language of your choice.

Question 2

What is your favorite programming language?

Figure 3 - Dreamjob Deliverer Test-taker Page

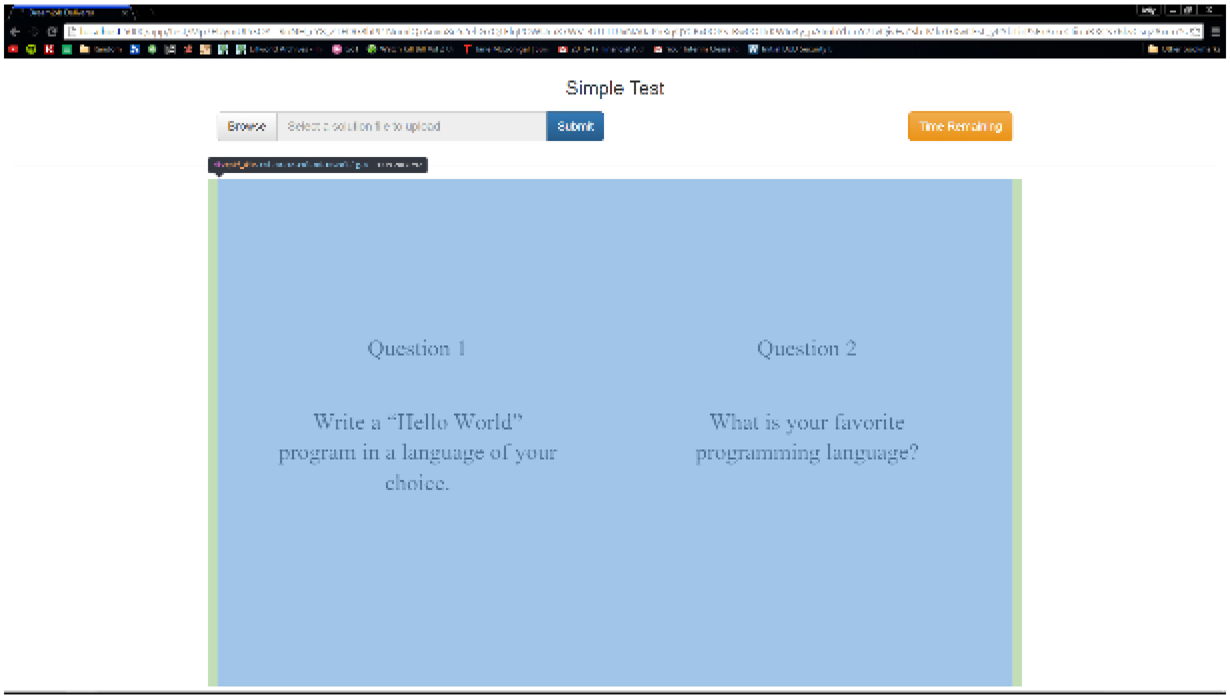

Simple Test
Browse
Select a solution file to upload
Submit
Time Remaining
Question 1
Write a "Hello World" program in a language of your choice.
Question 2
What is your favorite programming language?

Figure 4 - Division Element Holding the Canvases

The PDF.js library can only read PDF data from a file location or from a byte array. Due to the fact that the files are not stored in a file system, we have to retrieve the file data from the database and send it to PDF.js as a byte array. This presents a challenge because of the degree of separation between the database and PDF.js. Figure 5 illustrates the data flow of the PDF files from the database to being displayed on the page. PDF files begin as data in the database which is retrieved by Node.js as a bytea buffer. This type of buffer is impractical to send through an HTTP response so Node.js must first convert it into a base64 string. After being sent to the web page through the HTTP response, the web page converts the string into a byte array which PDF.js uses to create the canvas elements and display the PDFs on the page.

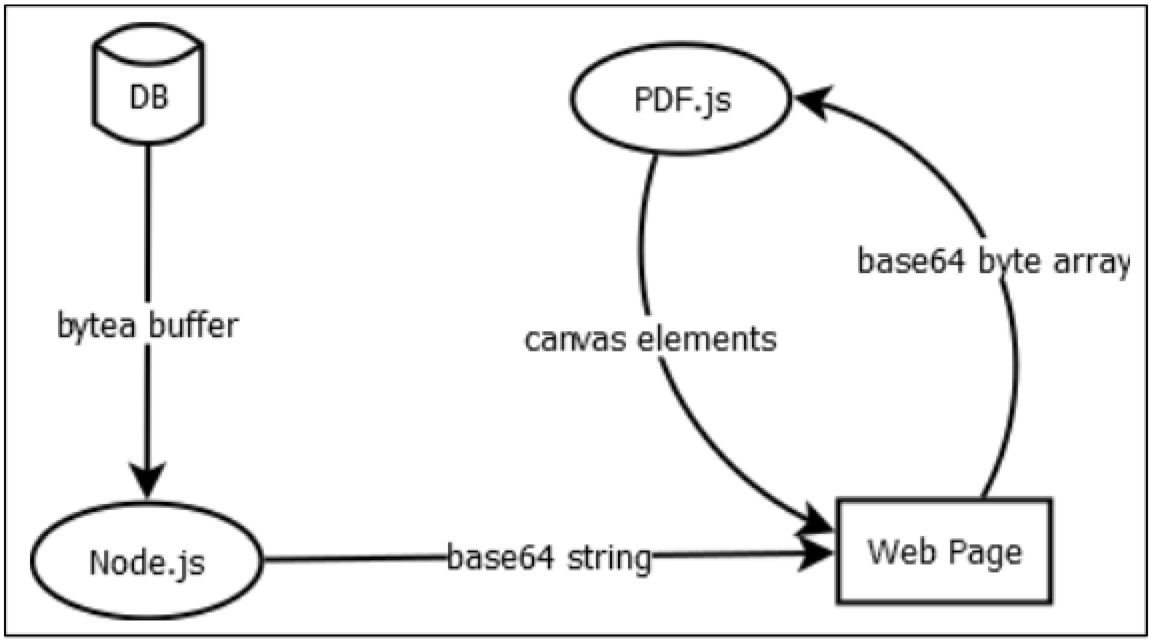


Figure 5 - PDF Data Flowchart

5. Design and Implementation Decisions

5.1 Languages

Dreamjob Deliverer was written using a combination of Node.js, HTML, CSS and JavaScript. As per the client's request, the application was required to be deployed using Heroku (a platform owned by Salesforce and used to build and deploy web applications on the cloud). Of the many options, the best choice for developing a web application in a Windows environment was Node.js. In addition; HTML, CSS, and JavaScript were selected due to browser standards and team member experience. More specifically, CSS3 was used in conjunction with JavaScript libraries to create page layouts and formatting.

5.2 Data Management

Data for Dreamjob Deliverer is stored in a PostgreSQL database on Heroku. PostgreSQL was chosen for the database language because it is simple to use with Node.js through Heroku and one team member had previous experience using the language. The application requires storing various file types, dates, times, and text. Storing files in a database rather than in a file system allows for better version management, replication, and portability. Naturally, the rest of the data is also stored in the database.

5.3 Libraries

Libraries used by Dreamjob Deliverer include Bootstrap, jQuery, Moment.js, and PDF.js. Bootstrap was used in CSS and Javascript to make the page look more professional through aesthetics and page formatting. It also makes writing simple front end visual functionality much simpler. Bootstrap requires jQuery, but jQuery also simplifies writing basic front end functionality. Moment.js is also required for some Bootstrap functionality that is used in the application. Moment.js is primarily used for manipulating and displaying date information. PDF.js was used to embed a PDF into our page in such a way that the user has a difficult time downloading the file. It also provides additional PDF support for the future if needed.

5.4 Modules

Modules used within Node.js as part of the Dreamjob Deliverer application include express, pg, crypto, multer, body parser, and fs. The express module is used for page routing and HTTP helpers. This module is the backbone for the entire application. The pg module is used to interface with the PostgreSQL database through queries and error handling. The crypto module is used for creating a unique URL path for each test instance that is scheduled. The body parser module is used to get values from elements in the DOM during a POST request in Node.js. In Dreamjob Deliverer it is implemented along with multer and fs to allow uploading files from the front end and storing them in the back end. Multer works closely with body parser to get the file data from the upload and save it to a temporary location on the file system. Afterwards, fs reads the data into the database and removes the temporary file. The use of all three modules together simplified the process of allowing users to upload files.

5.5 Testing

Dreamjob Deliverer was tested using a combination of automated testing and directed exploratory testing. The Selenium webdriver was used to simulate a user's actions within the application in various browsers. The assertion library Chai was used to verify expectations about various characteristics of a page.The tests themselves were written in coffee-script in an effort to streamline the test creation process. Mocha was used to tie all of these libraries together into a contiguous testing environment that would interact seamlessly with the application.

5.6 Alternatives

Throughout the development of Dreamjob Deliverer there were alternatives to several of our design decisions. Several methods were considered when displaying a difficult to download PDF. Yet, due to the way Firefox embeds files it was decided that none of the other options were viable. There were also several options for implementing date and time pickers in a web page, but the simplest, HTML5 datetime input fields, are not supported in Firefox. The best method for date and time pickers in Dreamjob Deliverer was determined to be the Bootstrap datetimepicker. PostgreSQL had several alternatives, though the only experience with databases on our team involved PostgreSQL so every other option provided an unnecessary learning curve. AngularJS was an alternative to Node.js, but had a steeper learning curve than desired. Although there were many alternative options, the best options were selected for the context of this project.

6. Results

6.1 Project Results

The functionality of Dreamjob Deliverer has been verified using automated webdriver testing as well as directed exploratory testing. The Selenium webdriver was used to test functionality in both Google Chrome and Firefox. Directed exploratory testing was used primarily to test functionality in Internet Explorer and Microsoft Edge. Our project will likely not function as expected in older browsers that do not support CSS3. All functional and nonfunctional requirements were met apart from the administrator authentication. Using OAuth 2.0 for administrator authentication could not be completed due to time constraints. As stated by the client this requirement had the lowest priority. Any features that involved uploading a file were implemented in a way that limits its functionality due to language limitations and time constraints. While the user can upload files, any file that is too big cannot be held in the datatypes of the libraries we used. As a result files to be uploaded cannot be larger than 100 megabytes in size. However, due to the scope of the project, this should not consistently cause problems.

6.2 Lessons Learned

- Libraries such as PDF.js and fs, while useful, still have their own limitations. Whether or not to use a library becomes a compromise between simplicity of using a library and the restricted functionality they offer.
- Handling file uploads can be difficult. While there is some support for doing this, if there are any complications it can become an extremely challenging task.
- Automated testing of web applications is a powerful tool but is difficult to use. Automated testing allows for speed, repeatability and reliability but locating all of the elements necessary through a webdriver is much more difficult than locating them manually. All in all, automated testing will save time in the long run.
- Browser differences force programmers to think creatively. An initial approach may not work in all of the major browsers so many alternatives must be considered and explored.