Introduction:

Preeclampsia is a pregnancy complication that a woman might develop during her pregnancy. If left untreated, preeclampsia can lead to eclampsia, which can put a mother and her baby at risk [1]. It is one of the leading causes of maternal death in developing countries. The Congo Red Dot (CRD) Quantkit is an effective tool for diagnosing this condition [2]. One symptom of preeclampsia is protein misfolding. When Congo Red dye is mixed with a preeclamptic woman’s urine, it binds to misfolded proteins [2]. Professor Hemant Tagare’s Lab has been developing both an iPhone and Android application to accurately assess the level of misfolding in preeclamptic women with the hopes that the applications would help provide these mothers with the healthcare they need [2].

A problem arises from the fact that researchers are working on this problem from remote locations. As these researchers are developing image processing algorithms for the Congo Red Dot testing mobile applications, it is becoming increasingly necessary that there be one central location for their images, image processing algorithms, and image testing results. In addition, people working in this field should be able to run tests on the images uniformly. Today, each lab working on this problem is using its own algorithms. However, it is important that researchers be able to run each other’s algorithms to ensure accuracy across different variables and measures. The mobile applications developed earlier fall short in that all images, algorithms, and results are stored to the phone itself.
As such, I propose the use of a web application to consolidate research done in this area and to enable seamless, collaborative work in the future. This web application requires that a user log in to access any data. It is connected to a database that holds all results and images. It has an interface that allows for easy upload and testing of CRD kit images and allows for easy search and manipulation of all data pertaining to Congo Red Dot testing.

**Architecture:**

In order to build the website, I used KeystoneJS; a NodeJS based content management system that implements the Model-View-Controller framework. KeystoneJS uses the MEAN JavaScript framework. As such, I use MongoDB for storing my data, NodeJS for the server side, and AngularJS for the front end. Together, these tools helped me develop an application that is written in JavaScript from front to back.

![Mean Stack Diagram](image)

*Figure 1: Mean Stack [3]*
Implementation:

Database design:

In KeystoneJS, “Lists” control data schema and models and documents in my database are “Items”[4]. In addition to the types MongoDB carries, KeystoneJS has a library of its own “types,” some of which were particularly useful for this website. I only use three Lists in my application: Gallery, Results, and User.

The Gallery list has three main fields: name, publishedDate, and Cloudinary_Images. Cloudinary_Images is a field type specific to KeystoneJS with its own types, one of which is a URL to where it is stored on the Cloudinary clouds. Cloudinary is a cloud image service that allows for smooth image saving. When a user wants to upload images for testing, they first create a gallery and can then upload images to the gallery. To the right is an example of a JSON file generated for an album with two images. You can see that Cloudinary_Images is a list of JSON files for Cloudinary Images with fields that amongst other fields include their original size, format, and URL.

```
{
  "id": {
    "$oid": "572e71c1d1e197fc1e621b3"
  },
  "key": "default_album",
  "name": "DefaultAlbum",
  "images": [
    {
      "public_id": "535",
      "version": 1462460074,
      "signature": "c5d16037e44e0a781d9cdd0df57d759e",
      "width": 256,
      "height": 256,
      "format": "png",
      "resource_type": "image",
      "secure_url": "https://res.cloudinary.com/congoreddot/image/upload/v1462460074/535.png",
      "_id": {
        "$oid": "572e71d1d1e197fc1e621b4"
      }
    },
    {
      "public_id": "592",
      "version": 1462460074,
      "signature": "51089241d1373cb6eafa770e8de34b8aa62438",
      "width": 256,
      "height": 256,
      "format": "png",
      "resource_type": "image",
      "url": "http://res.cloudinary.com/congoreddot/image/upload/v1462460074/592.png",
      "secure_url": "https://res.cloudinary.com/congoreddot/image/upload/v1462460074/592.png",
      "_id": {
        "$oid": "572e71d1d1e197fc1e621b5"
      }
    }
  ],
  "publishedDate": {
    "$date": "2016-05-07T14:52:49.000Z"
  },
  "__v": 1
}
```

Figure 2: Gallery JSON file
The Results list has four field types: img_name, img_url, result, publishedDate. These are all required values that a user is not able to modify. As a KeystoneJS List, the default option allows for users to fill out forms to create their own results, but this is overridden by the noedit option. Therefore, all results populating the database have to come from the automatic generation of result items that occurs when a test is run. In addition, this list has some default settings for which columns to show when viewing the results page and how to organize the results. These are attributes of a list that a user can customize in the results page.

The User list has four field types, three of which are KeystoneJS types. One of the most important field types is “isAdmin,” which allows a user to access the full website. In addition, I would like to note that the password is encrypted, which you can get a sense of from looking at the JSON document for our default user. This is a List that is included in the KeystoneJS package. I made no modifications.

```
{  "_id": {    "$oid": "572e71a1d1e197fcc1e621b2"  },  "password": "$2a$10$HbW0N6JXJz8.awv6SwDr6O7zxx0SOpx4jwpz0Z8oD7XY.VqldPcK",  "email": "test@test.com",  "name": {   "first": "Test"  },  "_v": 0,  "isAdmin": true}
```

Figure 3: Result JSON file

```
{  "_id": {    "$oid": "572e9bc104cc47d2c9c11d11"  },  "sortOrder": 1,  "publishedDate": {   "$date": "2016-05-08T01:52:01.978Z"  },  "result": "0.17",  "img_url": "/res.cloudinary.com/congoreddot/image/upload/v1462670706/test163.png",  "img_name": "test163.png",  "_v": 0}
```

Figure 4: User JSON file
**User Interface:**

There are 6 main pages that an admin user has access to: “home,” “upload,” “test,” “results,” “howto,” and “keystone.”

![Howto web page](image1)

**Figure 5:** Howto web page

The “home” and “howto” pages are the only pages that are completely public. The “home” page has information about the Congo Red Dot kit and testing and the “howto” page explains to a user how to use the web application. The home page links to the “signin” and the “howto” page in order to prompt users into starting their activity on the page.

![Results web page](image2)

**Figure 6:** Results web page
The “keystone/upload” and “keystone/results” pages are automatically generated private pages that come from the idea of Keystone lists. These pages allow for users to sort, filter, and download the data of each list. They also allow users to customize what attributes the page shows. Users are also able to delete content. The "keystone" page comes included with the KeystoneJS package and lists all Keystone Lists.

![Test web page](image)

**Figure 7**: Test web page

The “test” page is the most interesting of the pages. It is also a private page. On the left of the page, you have the list of albums that you can click on so that you can reach those albums on the page without having to scroll so long. The right box also gives users the option to select all images and run the test on selected images. Once a user submits images for testing, this sends a get request that in turn sends all images to a java file for image analysis. These results are the stored in the database. Once all images are analyzed, the page redirects a user to the “results” page.
In addition to these main pages, there is also the “signin” and “signout” page. If a user that is not logged in tries to access any page other than the “howto” and “home” pages, they are redirected to the “signin” page. Each gallery also has its own page that allows you to upload images to the gallery.

Deployment:
To deploy the application, I used four different services. Cloud9 was my development environment that I used in conjunction with Github. With Cloud9, I was able to run a server and reach my website. In addition, I host my images on Cloudinary and Mongolabs provides a simple interface for managing my database.

Usability:
Methods:
Some of the factors that I tested for were: self-evidence, relative ease with which users performed certain tasks important to the application, and understandability of instructions.

To measure these factors, I assigned participants tasks and recorded whether a user accomplished a task without hints from me and asked them questions about their experience. Furthermore, I took note of how many times they clicked on an incorrect element in the page before they told me they were done with the task. If they made more than 5 incorrect clicks on the page they were on total for a task, they were marked as having made a critical error. Otherwise, if participants made at least one but less than 5 incorrect clicks, they were marked as having made non-critical errors. My role was to provide hints only after the participant had spent more than 3 minutes trying to accomplish a task or gave up on the task. In addition, at the end of the 10-minute task completing session, to collect subjective measures of usability, I conducted a brief survey.
with each participant asking about his or her experiences with the web application. On a scale of 1 to 5, they rated how easy it was to find information on the page and how easy it was to navigate and understand the flow of the page. They were also asked to detail their frustrations, likes, dislikes, and suggestions.

The tasks were:
1. Find the “Howto” page and read it
2. Upload Images
   a. Create a gallery
   b. Upload 5 images to your gallery
3. Test Images
   a. Test 3 of your images
   b. Test all of the images
4. See your results
   a. Identify the image with the highest value result (if they tried eyeballing it, I explicitly told them to sort)
   b. Hide the image’s URL

For the self-evidence of the application, I gave the website to a volunteer user without a “howto” page and gave him the tasks to do on the website without having let him explore the interface. He was eventually able to accomplish all of the tasks, however, he made 3 incorrect clicks when uploading images (2b), had trouble knowing when images were done being processed (task 3), and failed to sort his results in order to accomplish task 4a. Therefore he made a few non-critical errors when uploading and made a critical error when sorting.

Five other volunteer users were recruited to interact with the page. They were all given access to the howto page. These users gave ratings that fell in the range of 3-4 with 4 being the mode and very dispersed ratings for how easy it was to navigate and understand the flow of the website. Of the 4 users, three saw exactly the same website that the “self-evidence” user saw. They were all able to accomplish tasks 1, 2a, and 4b without critical errors. One was not able to accomplish task 2b within the 5-click window. In addition,
though they all were able to accomplish tasks 3a and 3b, when completing 3a, all users waited for an indication that the application was done processing. This also seems to be a non-critical error as they all had to confirm with me whether they had completed the task, asking, “Did it work?”

In their interviews, these participants all claimed that they liked that the design was clean and that the “howto” page made it really easy to understand the web application. However, they all expressed frustration with the lack of indication that images were done being processed. They also articulated that sorting was not as intuitive as it was to filter and show attributes of results. One expressed that he wished the results showed the image itself, not just a link to it. One also added that the “howto” page might be more helpful if it included visual aid.

**Results:**

Overall, from the study, it seemed that the application was self-evident in how it should be used, as my self-evidence user was able to accomplish all of the tasks within 10 minutes. However, the relative ease for certain tasks was low. Users really struggled with sorting results and it was evident that testing images was unnecessarily confusing. I responded to the complaints that arose from task 3, and the page now redirects to the results page once the algorithm is done processing images. To address the issues with task 4b, I took the suggestion of modifying the “howto” page so that it now provides visuals for how to accomplish tasks on the application.

Though no volunteer user explicitly noted why they had trouble with uploading images or had any suggestions for how to make this task more intuitive to complete in their interview, I noted that there were two buttons on that page. “Upload Images” and “save.”
“Upload Images” actually only opens a window that allows you to select images for upload and “save” saves changes made to the album, including images selected for upload, so I changed the “Upload Images” button text so that it now says “Select Images for Upload.” My last two users saw this updated website. They were able to accomplish all of the tasks and gave ratings between 4 and 5 for how easy it was to find information and for how easy it was to navigate.

Conclusion:
This semester I was able to build a web application for uploading and testing Congo Red Dot kit images. This application will be useful in consolidating research involving the Congo Red Dot quantum kit and in allowing collaborative work. After having undergone a bit of usability testing, it now appears to be user-friendly.

Acknowledgments:
I’d like to thank Professor Hemant Tagare and Professor Richard Yang for their continued support, encouragement, and feedback in both the development of the android application and web application for the Congo Red Dot Test.
References:

   <http://www.units.miamioh.edu/mtsc/usabilitytestingrevisedFINAL.pdf>.