# Cognitive Training Quiz Application

## Project Report

**Group 7**: Ran Liu, Nick Uebele
**Mentors**: Michael Cohen, Yuri Agrawal, Gorkem Sevinc

## Background

Many older adults suffer from visuospatial deficits, either following some kind of trauma or due to simple aging. Visuospatial deficits are characterized by many symptoms, all of which relate to the loss of the sense of "whereness" in the relation of oneself to one's environment or the relation of objects to each other. A pencil-and-paper test, consisting of five training and testing modules is currently used to assess and treat patients suffering from these deficits and is given with the help of a physician or test administrator.

## Problem

The already-existing pencil-and-paper test is cumbersome to use: the test consists of lots of paper materials and requires a physician to administer it. By creating this test in a web application format, we hope to allow patients to take this test by themselves—at home—as often as they would like in order to improve their visuospatial abilities. By saving their testing data in a database, physicians can easily view their patients' performance over time and examine valuable metrics such as accuracy and response time for each question and patient. Also, we could allow many other users who may not possess cognitive or visuospatial deficits to use this application as training to improve their general cognitive health.

## Approach

### Overview:

We built a responsive single-page web application. This consists of two components: a user-facing front-end, which users view in a web browser, either on a desktop, or on a tablet or similar mobile device, and a back-end, which serves a RESTful API which provides the necessary calls to retrieve, store, and modify data. As this is a software development project, we essentially have two types of deliverables: documentation, consisting of design documents and internal specifications, and the physical code itself. We use Stash, a private Git server hosted on the Johns Hopkins network. During development, we deploy our code to the harb.rad.jhmi.edu server.

### Front-end design:

Our front-end consists of a responsive web application written in AngularJS. As we lack an experienced front-end developer on the team, we use Bootstrap as a responsive design framework in order to enable us to rapidly build serviceable user interfaces. While development occurs in a desktop environment, we take a largely platform-agnostic approach, where the same interfaces are usable, regardless of the device and screen size. There are three types of users: patients, physicians, and administrators. Patients are able to access their own testing histories,

complete the training modules, and submit testing data via the front-end. Physicians are able to access their patients' data, as well as manage their associated patient accounts. Administrators are able to manage all users.

In order to translate the pen and paper exercises into web application format, we classify each component of the exercises into one of five categories. For each category, we build a standardized template, which is then populated with the data and images for each specific question. The responsibility of serving these assets- i.e. prompts and images for the training and testing modules, could be delegated to a content delivery network; however, because we are essentially running all our components on one server, anyhow, for the time being, these assets are served by the same HTTP server that serves the Angular application. Ultimately, the front-end of our application consists of static resources that can be served by any HTTP server. The most common choice would be Apache; we use both Apache and the NodeJS app http-server.

As part of the design process, we created a series of user interface concept sketches (Figure 1, 2). From these, we built static mockups, and from static mockups, constructed views and partials for the Angular single page application.

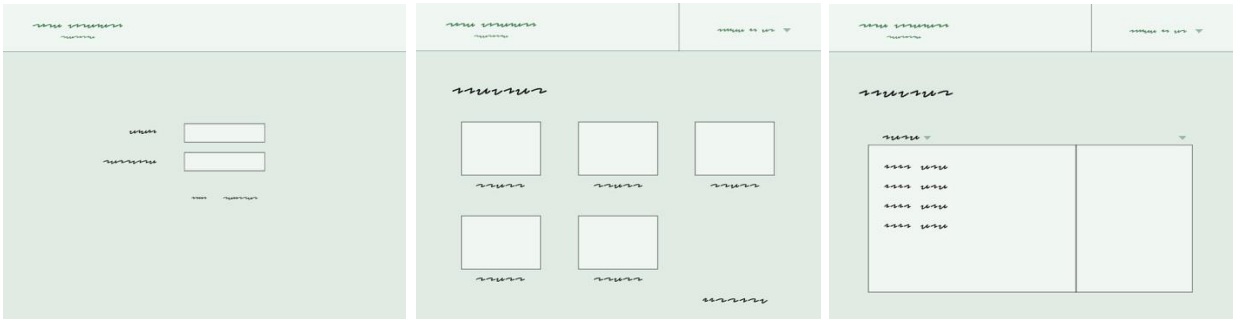

*Figure 1: UI Sketches made in Adobe Photoshop*

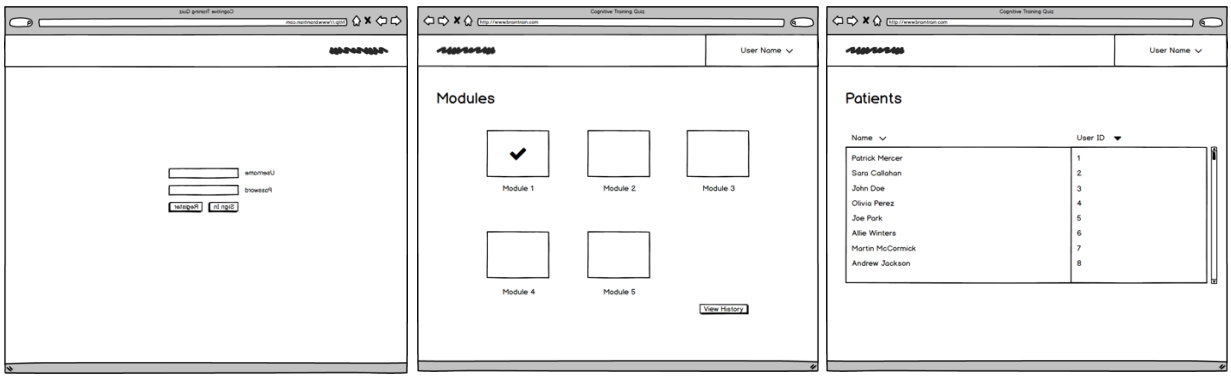


*Figure 2: UI Sketches made in Balsamiq*

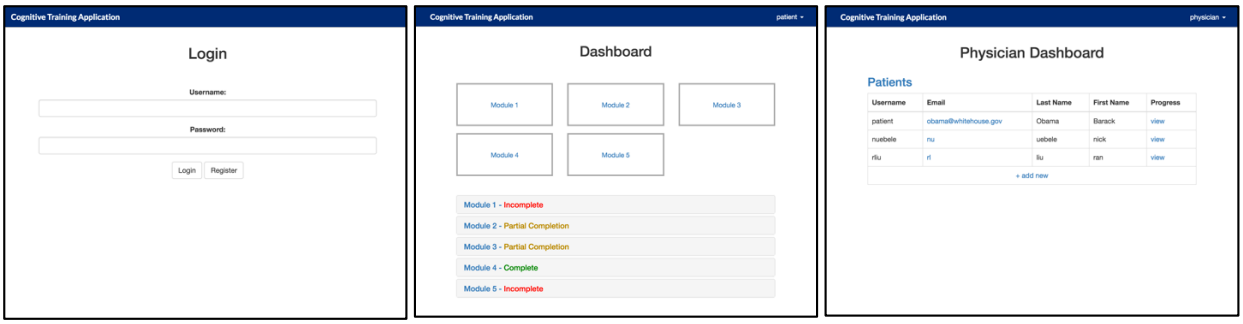


*Figure 3: Final Interface Designs*

### Back-end design:

Our back-end consists of a RESTful web service, served by a Ruby on Rails application. The RESTful API primarily serves CRUD (create, read, update, delete) functionality to the front-end, allowing for the registration and authentication of users, access to training/testing module data, and the submission of testing/training results. A RESTful API endpoint is accessed via HTTP requests (POST, GET, PUT, DELETE) send to a specific URL, which are then dispatched via a routing service to a controller method which is bound to that endpoint. Data sent between the front and back ends is stateless (i.e. it presumes no prior knowledge of application state), and serialized in JSON format. Certain privileged endpoints which require the method caller to identify/authenticate themselves also consume an authentication token, which are generated when the login endpoint is called, expire within two hours, and are single-use. These tokens are refreshed whenever they are used for a privileged API call. For a detailed overview of our API endpoints, consult the documentation provided in the appendix.

For development purposes, we used an embedded SQLite database; however, because our application is built on the Rails framework, it is a fairly trivial task to reconfigure the back-end server to interface with any of a myriad of database systems, from MongoDB to PostgreSQL.

The information contained within the pen and paper exercises, however, must be transcribed over into a digital format. We devised also a JSON format for this data, the specification for which can be found in the appendix. At a later date, this data can be migrated into our application database; however, in the meantime, as we work to transcribe the modules, the ability for humans to read and edit the module data is an important feature.

## Results

As of the latest build, we've managed to fully transcribe the five modules of pen and paper exercises, though the last two modules still lack new artwork (one of our dependencies). All core functionality has been implemented, along with a few convenience features. We also have had the opportunity to conduct a UX (user experience) review, and have applied some of the insights gained in the design of our user interfaces. As the target user-base for this application is patients whose visuospatial cognitive abilities may be impaired, the accessibility of the application is of no little importance.

## Significance

The significance of this project is that it successfully digitizes an already established clinical method for assessing and improving patients' visuospatial abilities, making it easier to use and more easily accessed by many who could benefit from this cognitive training. By automating the testing process, we have reduced the workload of physicians and made it easier to track their patients' performance and identify trends in performance data exported by the application. There already exists research that shows that cognitive training can improve cognitive abilities even in healthy individuals for up to five years after the training occurs, and we hope to see this application used for preventative purposes in addition to diagnostic or reactionary treatments.

## Management

In terms of the task assignments, we remained fairly true to our management plan. Ran worked on development of both the back-end and front-end. Nick contributed to front-end development, module transcription, and worked on UI design. Both Ran and Nick worked to iterate and polish the UI, gathering feedback from both mentors and fellow students on how to improve the design.

Our main goal at the outset of the project was to produce a refined, single-page web application to serve all five modules of the cognitive training quiz. In order to do so, our task list included development tasks (such as writing the backend server, testing and deploying the backend, and development of Angular SPA front-end) and refinement tasks, including iterating and refining our design documents, result storage and reporting, and user interface. Up until very recently, when we discovered that David Rini from the Arts as Applied to Medicine Department would be unable to produce all the needed images on time, we believed that we would be able to fully complete all of our expected deliverables, chief among which were serving all five modules of the quiz and allowing for result reporting. However, we did already transcribe all of the test questions for the remaining modules, making it very easy to complete them once we have the digital images.

The main area where our accomplishments differ from our plans is our maximum deliverables. At the onset of the project, we set the following as our maximum deliverables: capabilities for data analytics on stored data, capabilities for advanced queries on data, and usability studies/pilot studies with actual patients. We were able to implement the capabilities for data analytics on stored data, but did not quite complete the other two. While considerations have been made in the design for allowing for advanced queries, we were unable to implement it as of yet, choosing to spend more time on UI refinement than extra features. The infrastructure for this feature already exists in the application, but it would require more time to implement. Lastly, though we were unable to conduct pilot studies, we consulted with the Technology Innovation Center's resident UI/UX expert, and gained insights, some of which we were able to immediate apply, as to how we could improve our interface. The incomplete features, as well as the incorporation of the artwork for the $4^{th}$ and $5^{th}$ modules, would comprise the majority of the future work on this project.

Though neither of our group members were particularly skilled in front-end development at the beginning of the semester, our group gained a great deal of insight into web development, particularly with AngularJS. We also had the opportunity to employ agile methodology in our software development process, and gained some familiarity with agile techniques. A more complete knowledge of development higher on the stack also better informs our design decisions when developing a RESTful service; in addition to the experience gained working with Ruby on Rails, we've also gained a more holistic understanding of the web development process in general.

# Appendix

**API Specification:**

| Method | URL | Description |
|---|---|---|
| POST | /api/user/login | User login. |
| POST | /api/user/verify | Auth token refresh/verification. |
| POST | /api/user/logout | User logout; deactivates an auth token. |
| POST | /api/user/register | Patient registration. |
| GET | /api/user/physicians | Lists physicians. |
| GET | /api/module | Lists training/testing modules. |
| GET | /api/module/:moduleid | Retrieves info for module |
| GET | /api/module/:moduleid/training/:componentid | Retrieves component data for training module component. |
| GET | /api/module/:moduleid/testing/:componentid | Retrieves component data for testing module component. |
| POST | /api/module/submit | Result submission endpoint. |
| POST | /api/module/results | Retrieves results for patient. |
| POST | /api/admin/list | Lists all users. |
| POST | /api/admin/register | Registers a user. |
| POST | /api/physician/list | Lists all patients for a physician. |
| POST | /api/physician/register | Registers a patient to a physician. |

**Module Data Specification:**

[Type] denotes an array of objects of Type
Type = {} defines the format of a Type object
("field") denotes that "field" is optional
"value1"|"value2"|"value3" indicates that the value is either "value1", "value2", or "value3"

File format:

[Module]

Module = {
"training": [Component],
"testing": [Component]
}

Component = {
"type": ("instruction"|"single-select"|"multi-select"|"numbers"|"timed-section"),
"prompt"|"promptUrl": "Prompt html"|"reference to prompt html file",

If type = "single-select"|"multi-select":
"promptImg": "reference to prompt image file",
"options"|"optionUrls": [],
"answer": index of answer or [answer indices]
("angle"): integer angle in degrees - rotates the prompt image.
("img_block"): true|false - Whether or not to display prompt image on side, or as new block.

If type = "numbers":
"sets": [Span]

If type = "timed-section":
"duration": time limit in seconds
"category": "single-select"|"multi-select"
"components": [Component (either single-select or multi-select)]
}

Span = [Sequence]

Sequence = [Number], e.g. [0, 1, 2, 4]

## Deployment Instructions

Back-end**:**

1. Clone from Git repository
2. Create, migrate, and seed database
3. Initialize Rails server.

Front-end**:**

1. Clone from Git repository
2. Start HTTP server