Visualizing eTextbook Study Sessions

Group 15
Chiang, Lily
Vellanki, Arjun
Lukiyanov, Egor
Chau, Tuan
Polina, Kavya

Course: CS 4624 Multimedia, Hypertext and Information Access
Instructor: Mohamed Farag
Client: Dr. Mohammed Farghally

Virginia Tech,
Blacksburg, VA 24061
11/31/2023

Executive Summary / Abstract

OpenDSA is an online platform that allows professors to create e-Textbooks with fundamental CS courses. Our project seeks to enhance OpenDSA by providing instructors with a user-friendly web interface and visualization tool. This tool allows them to understand student interactions during study sessions, in the areas of: Reading, Visualizations, and Exercises. The tool could lead to improvements in a student's learning process. OpenDSA is heavily used at Virginia Tech and other universities for CS courses. It records student interactions with learning materials but doesn't have an efficient way for instructors to understand these interactions.

Our project tackles this issue by developing a web interface that visualizes student interactions. We expand upon past research by sorting interactions into Reading, Visualizations, and Exercises, displaying detailed study session data. These visualizations will give insight into whether students are active learners or credit-seekers.

We utilize docker by creating two containers, one for backend and another for frontend. The visualization tool consists of a dashboard page with a student list containing students, their ID's, and overall grade. When wanting to find more details about a student, simply clicking on them would show different graphs linked to their individual data including time spent in each state and exercise behavior analysis.

Our expected impact is to allow for instructors to increase student engagement with the OpenDSA content. The tool would enhance their ability to monitor students to improve their learning experience.

The plan involves encapsulating interactive data in a meaningful graphical format. We have already achieved successful file uploads, established a student list, and completed most of the graphs. Furthermore, we plan to include detailed insights on student grades past their final grades in other areas such as: projects, midterm, etc. Our project is structured into three major planning phases: data synthesis, frontend development, and visualization enhancement. Throughout these phases, we will complete thorough testing to guarantee reliability. Also, to accommodate for potential future development, we will maintain adaptability and include a developer's manual.

When starting the project, we encountered three challenges: data transformation, code refactoring, and frontend-backend communication. With data transformation, we encountered issues trying to translate raw data into node graphs. For code refactoring, working of Master's code made it difficult due to previous hardcoded paths and other inconsistencies. The last challenge was setting up a seamless connection between the frontend and backend with issues

related to CORS policies. With our commitment to delivering an efficient final product, we were able to overcome these challenges over time.

Introduction

Problem

OpenDSA is a technology infrastructure which allows CS instructors to create online eTextbooks for core CS courses like data structures and algorithms, formal languages, data and algorithm analysis, and more. OpenDSA is currently being used at Virginia Tech and a lot of other institutions across the globe. OpenDSA has a data collection tool that collects and records all student interactions with the learning content. During a study session, a student can read textual material, interact with algorithm visualizations, and solve exercises. Currently, we don't have a straightforward mechanism for instructors to know how their students learn from and interact with different OpenDSA artifacts.

Motivation

In this project, our goal is to implement a web interface to visualize interactions generated from students' study sessions using OpenDSA content, so that instructors can conveniently understand how their students learn from OpenDSA and identify possible improvements to their learning process.

Previous Research

This project expands on previous research done on categorizing student textbook interaction trends. Previously, research was done by determining the total amount of time spent before engaging with exercises and comparing that with time spent after exercises were started. We elaborate on this model by introducing more defined states for student interaction (Reading, Visualizations, and Exercises). This is assisted by new features in OpenDSA that track scrolling events in deeper detail. We will also zoom into cyclical patterns in textbook usage and look to give further conviction that the student is an Active Learner or Credit Seeker.

Requirements

During our discussions with our client Dr. Mohammed Farghally preceding project start, we established that our top-level goal was to create a tool used by instructors to visualize student interactions between three states (Reading, Visualizations, and Exercises). The input data containing information about students for a specific class per semester is pulled from the OpenDSA database by an instructor and uploaded into the tool. The client and team agreed on a web interface and visualization tool delivered through two Docker containers (frontend and backend) as the project deliverables.

Over time, the requirements evolved to reflect the technical and temporal boundaries of our project. As a summary, we considered these high-level criteria to be imperative to the usefulness, effectiveness, and satisfaction of our system.

Data: The primary data source is OpenDSA, providing anonymized student interaction logs from previous semesters in CSV format. We needed to build our OpenDSA visualizer to be robust and able to accommodate data from different students of different semesters, requested by the user.

Visualization Framework: The project requires an easily understood visualization framework capable of rendering individual student graphs, highlighting the most prominent cycles of student interaction journeys, and displaying time spent in Reading, Visualizations, and Exercises states.

Web Interface: The final deployment should include a website interface accessible to instructors. This interface should facilitate the exploration of student graphs, allowing for easy navigation and analysis.

Scalability: The project should be structured to accommodate future expansions, allowing for the integration of additional visualizations as needed. Code modularization is crucial in order for developers to work in parallel, which maximizes productivity and ability to push out early prototypes for feedback. This includes separation of data processing, graphical interface, and data visualization components.

Meetings and Communication: We set up bi-weekly virtual meetings with our client to circulate project progress and feedback. Our team preferred to use Zoom and Google Calendar invites for scheduling and hosting the meetings.

Privacy and Ethics: We established early in the project that we would not have direct access to the OpenDSA database due to privacy concerns and the need to maintain anonymity of student data. Any dataset used for testing or demonstrations is supplied by the instructor who has access and student identifying information is abstracted away from us.

Deployment: The project should be deployable via a Linux server. The backend and frontend are encapsulated in two Docker containers that standardize the environment between developers and users.

Documentation: Our project should include comprehensive documentation such as a user manual and developer manual to ensure the project can be handed off smoothly to future developers and users have a holistic understanding of the system features and tools.

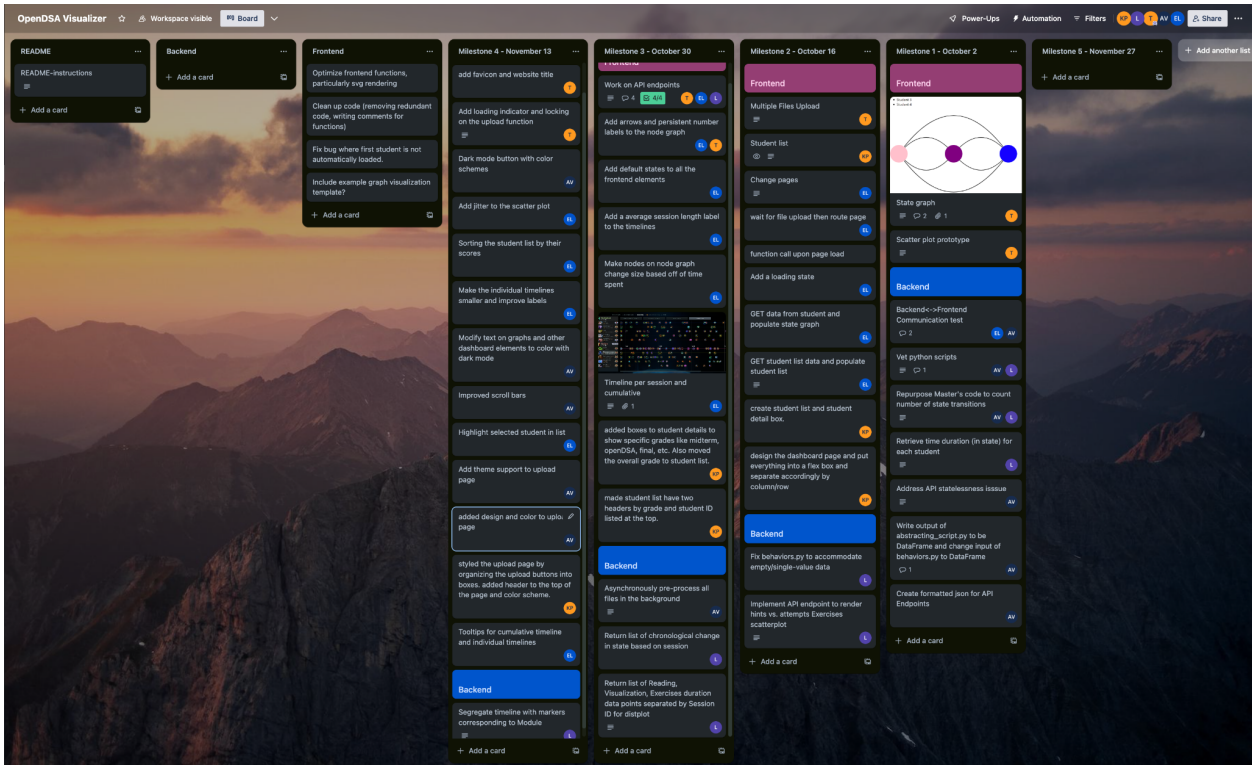
Approach

Following a thorough immersion into the project and a productive meeting with our client, our team seamlessly transitioned into deliberating on individual roles.

Below lists the role of each team member.

- Chiang, Lily, Backend developer
- Vellanki, Arjun, Backend developer (Meeting notes taker)
- Lukiyanov, Egor, Integration and Front-end developer
- Chau, Tuan, Front-end developer
- Polina, Kavya, Front-end developer

We established consistent team meetings every Friday from 11:00 to 12:00 a.m., exclusively for internal collaboration. Additionally, we scheduled bi-weekly Monday sessions starting at 10:30 a.m. dedicated to client discussions. Simultaneously, we initiated the development of a comprehensive project timeline, outlining key milestones occurring at least twice per month. These milestones encompassed critical tasks such as finalizing the conceptual structure of the program, vetting Python scripts from previous research, creating prototype wireframes for the website layout, implementing Python to JSON output for visualizations, translating JSON to visualizations in Javascript, conducting integration and testing, and concluding with the final deployment through Docker containerization. To facilitate efficient progress tracking, we implemented a Trello page that allows us to monitor and manage individual contributions and responsibilities.



Our approach involves encapsulating the outcomes of student interaction analysis within an intricate node graph, wherein the three states—Reading, Visualizations, and Exercises—are meticulously represented as nodes. Each node graph corresponds to an individual student, identified by their unique user ID. The directed edges interconnecting these nodes signify transitions between the states, with visual cues such as labels and line-weight conveying the relative frequency of these transitions.

However, we acknowledge that the node graph alone may not fully convey all the valuable insights within the input data. To enrich the representation, we complement the node graph with additional visualizations integrated into a comprehensive dashboard. These supplementary visualizations include details such as cumulative time spent in each of the three states, the average duration dedicated to Reading and Visualizations versus Exercises, and an in-depth analysis of student hints and attempted behaviors during Exercises. This holistic approach ensures a more nuanced and comprehensive understanding of the analyzed data.

We've incorporated an essential feature known as the student list, offering a comprehensive display of each student's ID along with their corresponding grade. The list is thoughtfully arranged by grade, ensuring that students with higher grades occupy a prominent position. Upon selecting a user from this list, the graphs dynamically adjust to reflect the chosen student's performance. Responding to our client's request for a more detailed grade breakdown per student, we've introduced a student detail box. This box provides a nuanced breakdown, encompassing the final grade, midterm assessments, project scores, and more.

Our existing features encompass file upload, student list, student detail, visualizations, and node graphs. After meticulous refinement, we have applied the finishing touches to both our front-end and back-end, achieving a finalized and polished design.

Upon concluding our efforts for the semester, we conducted a live demo for our client to elicit valuable feedback. Subsequently, we adeptly incorporated the client's input, refining our graphs with precision and finesse to ensure alignment with their preferences. This iterative process allowed us to seamlessly implement touch-ups, resulting in a more tailored and polished outcome.

Design

The layout of our visualization tool follows three main features:

1. Upload csv

This feature is a screen that allows the user to upload their own data files from OpenDSA. This includes an Exercises, Interactions, and Scores file of all students in a given semester.

2. List of students

This feature is a component of the dashboard and allows the user to select a student to zoom into. This list is generated directly after the user uploads data files and clicks the Go to Dashboard button. The list of students also sorts the students by grade, so that instructors can compare class performance with eTextbook behavior more easily.

3. Visualizations dashboard

I. State transition node graph

A. Research question: How did an individual student transition between the three states, and what are the relative frequencies of the transitions?

B. Description: Each node graph represents a single student in the data, identified by their user ID. Directed edges connecting the node states represent transitions between the states. The edges communicate relative frequency of transitions through visual indicators such as labels and line-weight.

II. Cumulative state duration distplot

A. Research question: How much time did the student spend in each state overall for each module?

B. Description: Show a distplot (distribution plot) with the 3 states being overlapping histogram traces. Each histogram represents a state, and each point represents a module. Depending on the shape of each distribution the user can gain information about the student's behavior, such as skewed left = student tends to spend comparative long durations in that state with some rare instances being short duration.

III. State transition timeline

A. Research question: How does the student allocate interaction time with the eTextbook across study sessions, and what is the chronological ordering of state transitions per session?

B. Description: Show a horizontal stacked bar chart where each bar represents a significant study session of an individual student. The x-axis portrays time in minutes, while the y-axis represents the session ID of each bar. Sections of the bar are labeled by the module and color coded by the state in which the interaction is classified. Multiple modules can be accessed in a session.

IV. Hints vs. attempts scatterplot

A. Research question: Is the student using the Hints feature to brute-force exercises?

B. Description: Show a scatter plot where x-axis is number of hints and y-axis is number of attempts. Each point represents a question on an exercise problem. Can also group points into different colors based on what module they are from, if not too visually distracting. Perhaps also lightly shade each quadrant of the scatterplot.

Quadrant analysis:

1. High Hints, High Attempts: If a data point is located in the upper right quadrant (high hints, high attempts), the student used a lot of hints and also made many attempts. The student might have struggled with the exercise and may have used both hints and multiple attempts to get the right answer.
2. High Hints, Low Attempts: If a data point is located in the upper left quadrant (high hints, low attempts), it means the student relied heavily on hints but made relatively few attempts. This might suggest that the student struggled and may have guessed/brute-forced the answer after using multiple hints.
3. Low Hints, Low Attempts: If a data point is located in the lower left quadrant (low hints, low attempts), it may suggest that the student successfully solved the exercise with minimal hints and attempts, indicating a good understanding of the material.
4. Low Hints, High Attempts: If a data point is located in the lower right quadrant (low hints, high attempts), it may suggest that the student made numerous attempts without using many hints. This could indicate that the student attempted to solve the problem through trial and error rather than seeking guidance from hints.

Implementation

Networking

- Use Docker to containerize the frontend and backend into separate containers to allow for a clear separation in roles between the two, and allows for easy and dynamic deployment of the services for the client

Frontend

- Upload page that uses POST requests to send data to the backend and then GET requests to retrieve processed data for frontend rendering
- Diagrams are generated using custom SVG and d3 graph rendering library
- Layout is done with React and MUI library
- Frontend is written in React to facilitate a responsive user interface

Backend

- Provide multiple API endpoints for the frontend to interface with the backend
- Process the input files and store them into files separated based on each student ID
- Calculate and provide data relevant to the graphs displayed on the frontend through JSON objects

Methodology

Phase 1: Project Initiation

Before meeting with our client for the first time, we listed questions for the client based on our available knowledge of the problem posted on the capstone Canvas. We took the transcript of observations and interviews conducted with our client during our briefing meeting and synthesized those meeting notes into a list of possible visualizations that would adequately present the information stored in the supplied csv files into digestible graphs that could be compared to determine student behavior patterns. From the client requests, we determined three overarching system features that should be included in our project: the upload file screen, list of students, and visualizations dashboard. We also began to research possible backend technologies that would be potentially useful to the implementation of our project, as well as Javascript data visualization frameworks for the frontend. We presented our first demo via a low to mid-fidelity prototyping engine, Figma (**Appendix Figure 3, 4**) to our client in our biweekly Zoom meeting.

Phase 2: Ideation

We vetted the preceding Master's student code to extract any useful data abstraction code we could use as a boilerplate. We also examined the raw dataset files we would use to test our system to brainstorm what visualizations would most effectively answer research questions on student eTextbook interaction behavior. We focused on visualizations that would concisely summarize the information while utilizing the full body of the available dataset. This led us to the conclusion that just the original node graph would not be sufficient in communicating the nuances of student behavior. We then returned to the ideation phase to brainstorm supplementary visualizations that would aid the presentation of data, and we proposed the list of visualization ideas to the client. With a plan for what the visualizations dashboard would contain, the second demo consisted of a static interface coded in raw HTML/CSS (**Appendix Figure 7**) to act as proof of concept.

Phase 3: API Endpoint Implementation

After obtaining approval from our client on the proposed design layout, we began to address each API endpoint for data visualizations. Using relevant code from the Master's student scripts, we streamlined the processing of the input Interactions file to maintain an intermediary Python dataframe representing an individual student's interaction logs. Passing around a Python dataframe reduced the risk of file path issues and decreased the number of parse_csv() calls or similar to read through the data. Furthermore, accessing a Python dataframe by column header name made our code more readable and robust. During this phase, we verified communication between frontend and backend and allowed the user-inputted Scores file to populate the list of students. Our front-end developers created custom SVGs and diagrams to display the requested visualizations. The resulting third demo initially presented static graphs (**Appendix Figure 8**) as placeholders for unimplemented API endpoints, but following demos featured dynamic visualizations (**Appendix Figure 11**) unique to each student when API endpoints were finished and tested.

Phase 4: Iterative Development

As the graph visualizations were becoming more polished and we gained familiarity with the OpenDSA data, we began to see other research questions that could be answered by the data. For example, the client wished to compare relative duration of study sessions over the course of the semester. This led to the timeline visualization being added to our features. The timeline was not originally in the list of conceived visualizations proposed in the Phase 1 and 2 ideation meetings. In order to implement this graph, we had to develop methods for parsing and segmenting the interaction data by session and module ID that the original Master's student code did not consider. Early versions of the timeline showed separate graph boxes for each session (**Appendix Figure 12**), but we later condensed the graphs into a single visualization for better visibility. We

presented a fourth demo to our client with fleshed out dynamic visualizations and all working endpoints (**Appendix Figure 14**).

Phase 5: Finalization

The final OpenDSA visualizer was visually polished. At this stage, we ensured that the interface design was consistent and responsive for different students. We incorporated a Light mode (**Appendix Figure 15**) and Dark mode (**Appendix Figure 16**). Additionally, we adjusted the interpretability of our graph visualizations. We decided to portray the density of data points on the Hints vs. Attempts scatterplot using opacity and size of dot as opposed to using a jitter to reduce clutter on the visualization. Opacity and size also reflect the discrete numerical nature of the data points. Another example is that we included directional arrows and labels with varying line thickness for the node graph to more clearly depict the relative frequency of a state transition within a given student's interaction behavior. We also made adjustments to the opacity of the histogram traces in the distplot and improved labels on the axes. The final state of our project was deployed to a Linux server and uploaded to the OpenDSA research repository.

Testing/Evaluation/Assessment

Our primary method of evaluating the effectiveness of our system was by gathering feedback directly from our client in biweekly Zoom meetings. We also received feedback from our professor Dr. Mohamed Farag, as well as our capstone class during progress updates and interim presentations.

User's Manual

Required Setup

You must have Docker installed and running on your device. No other package installations are necessary.

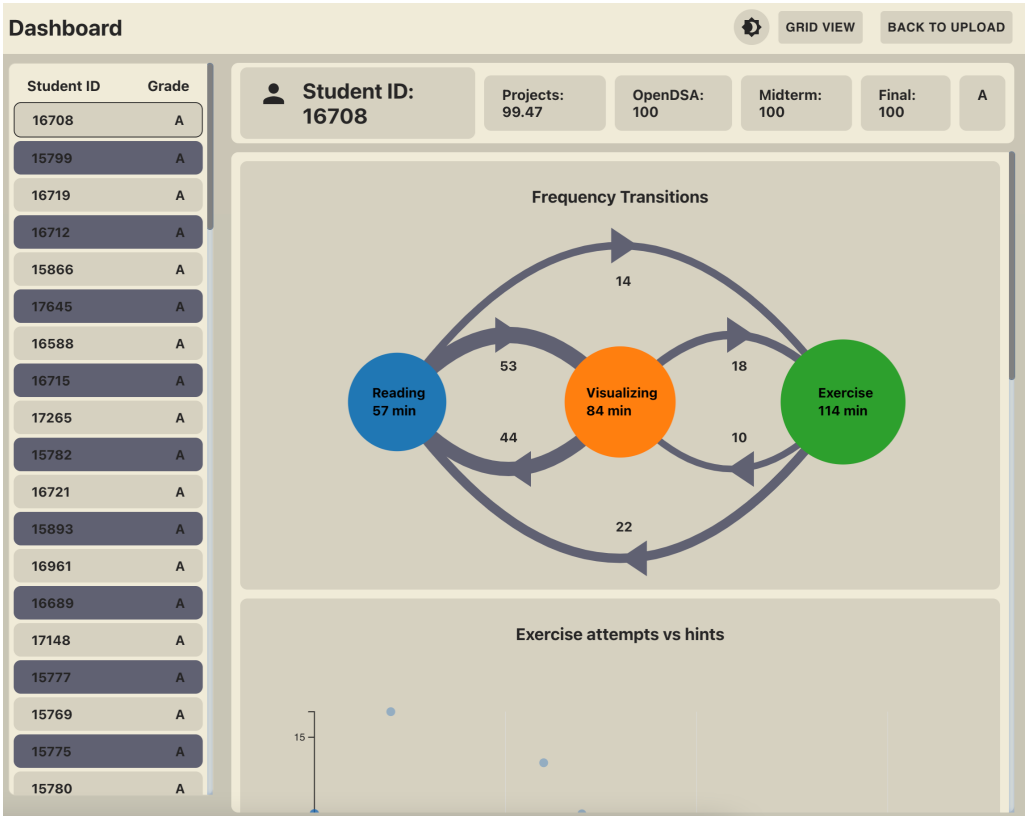
1. Expected SQL data for the Interactions file:

```
SELECT oui.id, oui.user_id, oui.inst_book_id, oui.name,
oui.description, oui.action_time,
oui.inst_chapter_module_id, exercise.inst_section_id,
exercise.inst_exercise_id, ex.short_name, ex.ex_type

FROM opendsa.odsa_user_interactions oui

LEFT JOIN opendsa.inst_book_section_exercises exercise ON
oui.inst_book_section_exercise_id = exercise.id

LEFT JOIN opendsa.inst_exercises ex ON
exercise.inst_exercise_id = ex.id

WHERE oui.inst_book_id = 852 Order by action_time ASC;
```

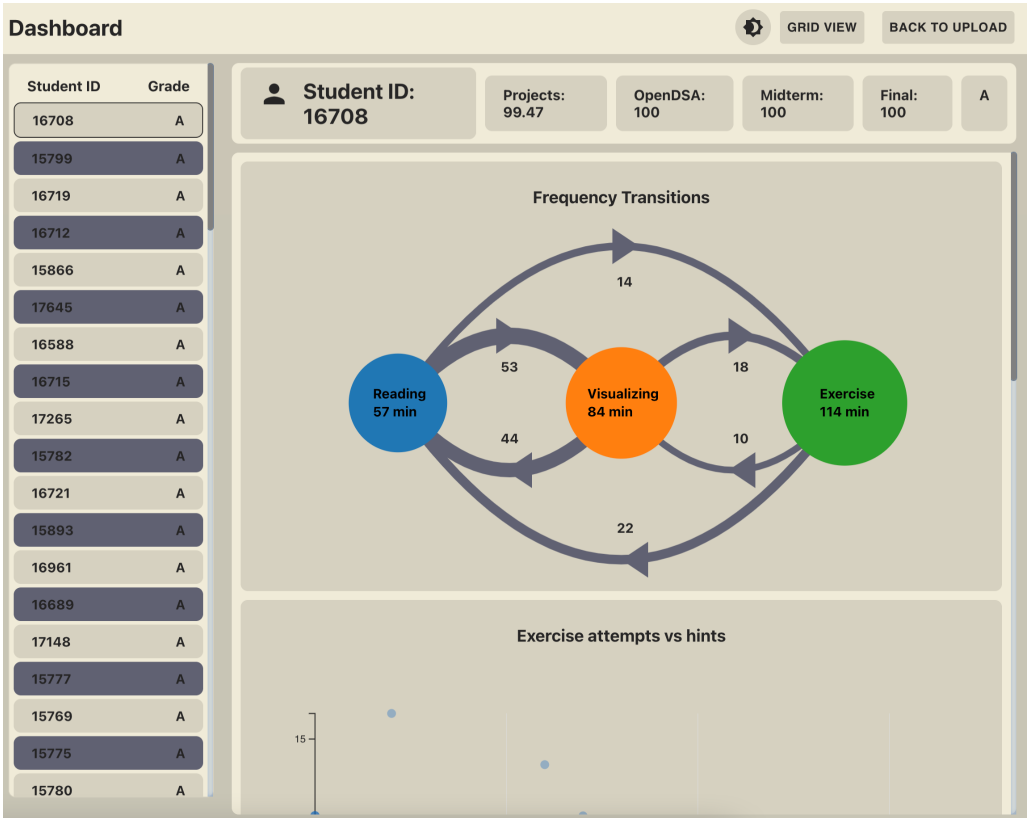
2. Navigate to http://localhost:80 to access the website. Upload the three required data files in the upload page (exercises, interactions, scores). Then hit upload.

   Important: The Interactions file must be of .tsv (tab-delimited) format. The Exercises and Scores file are of .csv (comma-separated) format.

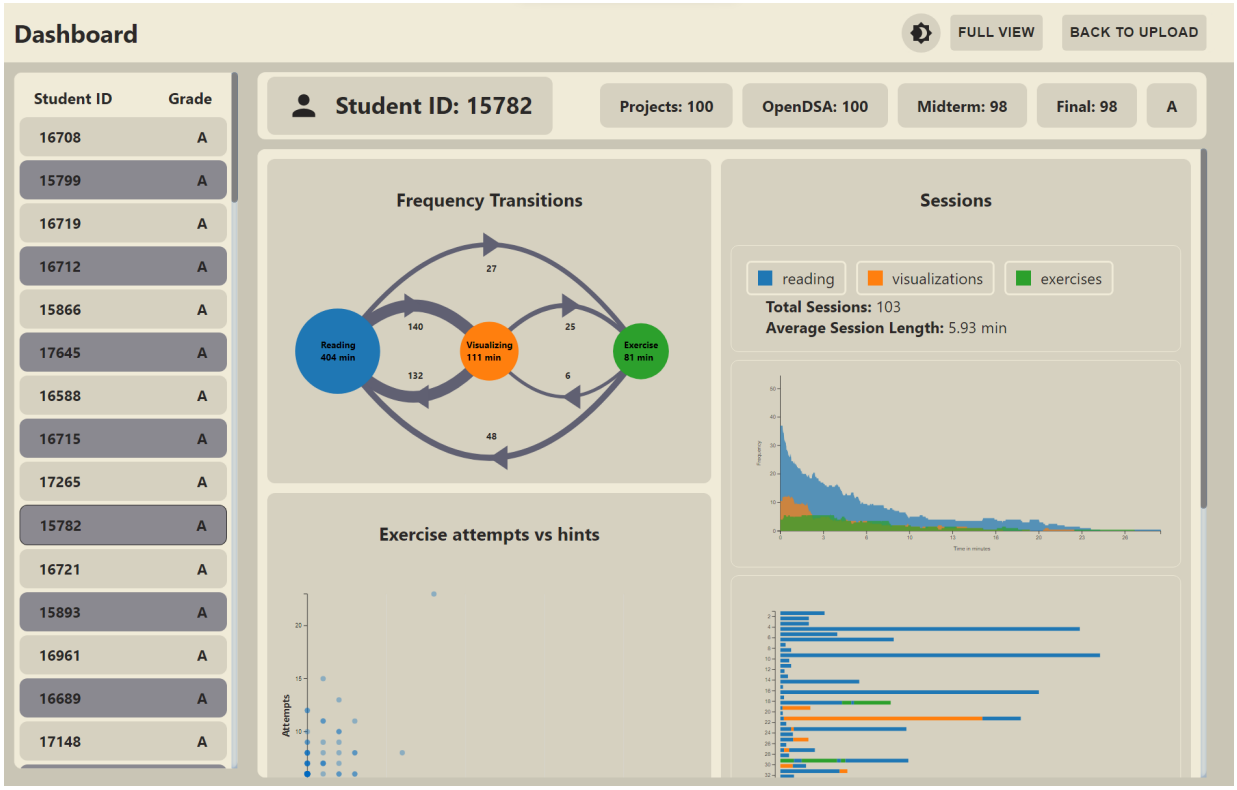
3. You will be navigated to the dashboard and a student list should be displayed on the left side containing user IDs and their associated final grade.



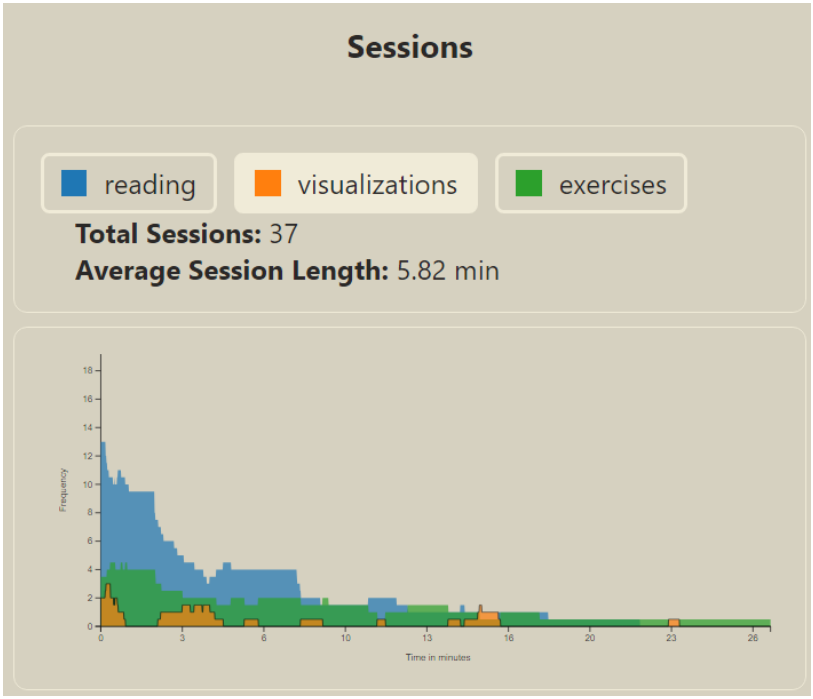
4. Click on any individual student in the list. This will render the right side of the dashboard containing graphs pertaining to their eTextbook interaction behavior.



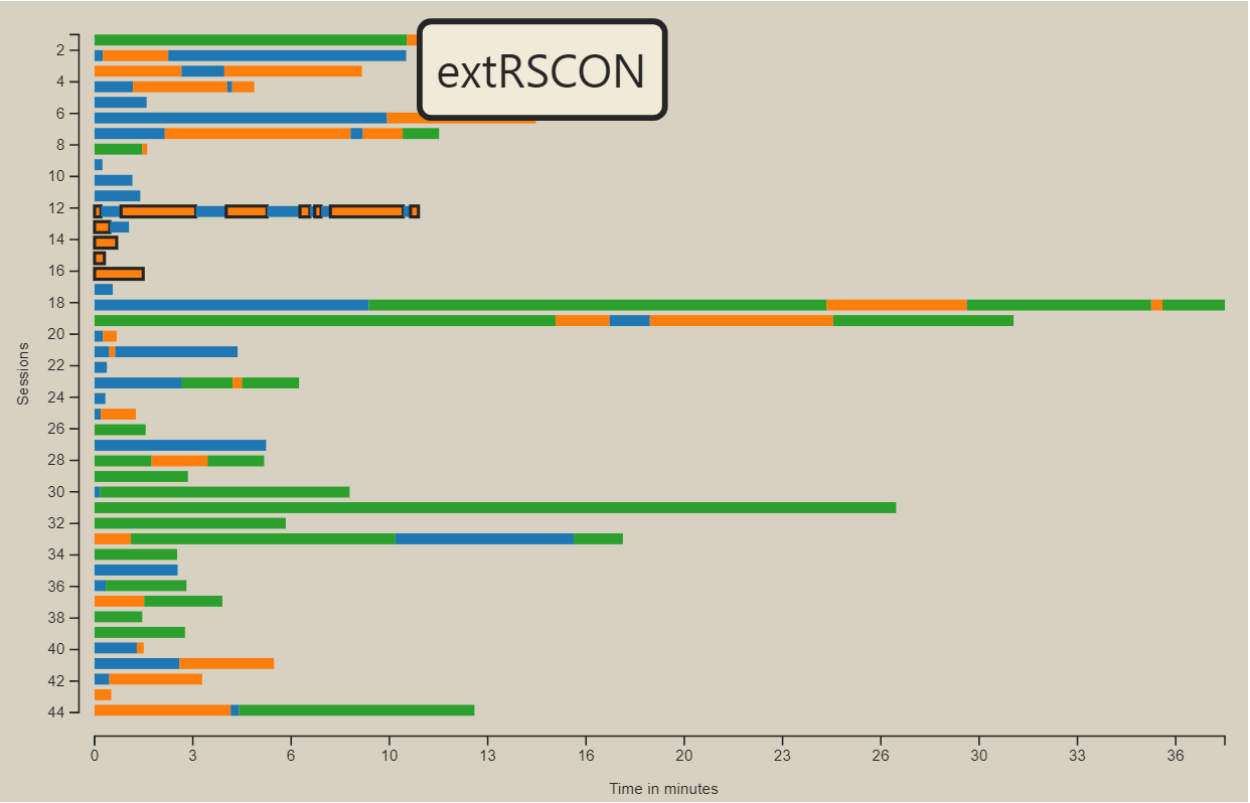
5. Click the Grid View / Full View button in the top right corner to toggle between grid and full-size views of the dashboard representations.



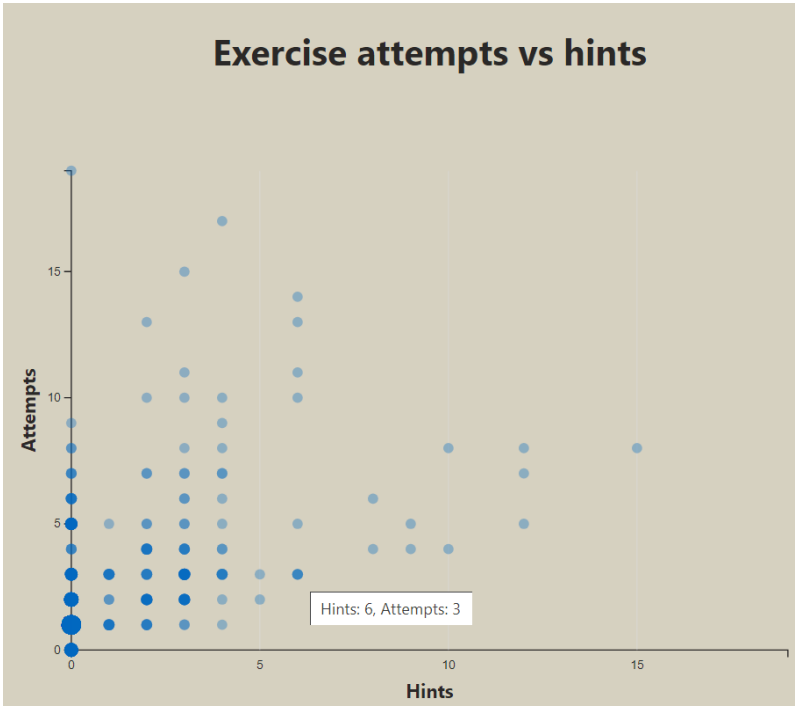
6. Hover over the Reading, Visualizations, and Exercises buttons in the legend of the distplot to outline the corresponding histogram trace and bring it to the front.



7. Hover over any section in the state transition timeline visualization to view a label with the module ID associated with the section. This also outlines other traces that have the same associated module ID.



8. Hover over any dot in the Hints vs. Attempts scatterplot to view a label displaying the number of hints and number of attempts for that data point.



9. Click the icon to toggle between Light mode and Dark mode (See **Appendix Figure 15, 16**).

10. If you need to change the uploaded files, simply hit the “Back To Upload” button to be taken back to the upload page.

Developer's Manual

1. Install and set up Docker🐳on the machine(s) that will run the frontend and backend services.
2. Clone the repo using `git clone https://github.com/egol2/openDSAVisualizer`.
3. If you are not hosting and using the service locally on your own machine: Modify the file in "openDSAVisualizer/frontend/src/components/Header.js" where the variable DOMAIN containing "localhost" to be changed to the IP or Domain Name of the device/network being used to host the backend container, and BACKEND_PORT to be changed to the port being used.
4. Run `docker compose up -d --build` with the code repository as the current working directory in a terminal.
5. By default without any modifications the backend will now be operating on http://localhost:8000 and the frontend will be operating on http://localhost:80, otherwise it will be running on the domain name and port as modified in the Header.js file.

Regarding the Docker containers:
Frontend

- runs JavaScript and React with Nginx running as the HTTP web server serving the web pages to clients

Backend

- runs Python with FastAPI, using uvicorn as the HTTP server accepting connections to clients

Structure guide

Frontend

Follows the standard React file structure with components including the graph visualizations, student lists, and student details. Other files exist just for styling each component in CSS. Pages include the dashboard page and upload page with the dashboard page implementing the majority of components. The Header.js file is used to import the global Styles.css file for uniform color styling and the global DOMAIN and BACKEND_PORT constants.

To add a custom visualization it would be necessary to create a new js file in components and add it to the grid view and full view of the dashboard html. Use the existing components in the dashboard as syntax reference to how the backend data is passed into them.

Backend

The backend was built around the Masters student's code which abstracts the OpenDSA interactions of students into data that can be easily analyzed. This is contained within the abstractor.py file. We split the student interactions file into separate files per student, allowing for faster data processing. We can then analyze this per-student data by feeding it into functions made in the behaviors.py file, and finally generating a JSON containing the behaviors and storing that locally as a .json file to allow faster retrieval for the next time the student is requested.

main.py

The main structure of the main.py file follows a RESTful API, but slightly breaking this format by allowing the frontend to make a request to process all of the students' data as a "background asynchronous" process. The file contains functions representing each endpoint and can be expanded on by following the same format, if additional endpoints are needed.

abstractor.py

The student_info() function handles processing a singular student's data based on its id, and only if the files were uploaded and processed, it will be able to proceed. It then calls the abstractor (code made by the masters student) to generate a dataframe and passes it onto different functions made within the behaviors.py file. These functions then return data that we can append into a JSON object and finally store it as a .json file and return this JSON to the client's browser to then be rendered. If a student's .json file already exists, it skips processing it and directly returns the file. This allows for faster processing.

behaviors.py

The behaviors.py file contains processing logic for taking an input Python dataframe representing the interaction logs of an individual student and transforms the data into JSON-formatted graph metadata to render in the visualizations dashboard.

The read_session_data() function in behaviors.py reads the input Python dataframe for an individual student's interactions generated from abstractor.py. It sets the global data variable to be the input data for use in the other API endpoints.

The getTransitionCounts() function corresponds to the API endpoint that generates the state transition node graph. It generates a dictionary of transition counts between the three states Reading, Visualizations, and Exercises by counting up the occurrence of state transition between two significant events.

The getReadingDuration(), getVisualizationDuration(), and getExercisesDuration() functions get the cumulative event count and time duration spent in the respective three states, used to implement the cumulative duration distplot and timeline visualization.

The getExercisesInfo() function is a simple parsing of the Exercises file in order to generate a list of lists where each sublist is of the form [hints, attempts] for each unique exercise question. This function corresponds to the API endpoint that renders the Hints vs. Attempts scatterplot exercises.

The getDurationBySession() function corresponds to the timeline visualization and segments the data grouped by unique session ID. The module ID mapping is done using the `Event Name` column in the Interactions data.

Lessons Learned

Timeline/Schedule

Project management and organization was facilitated via Trello, following the general milestones listed below.

Sept 25	Setup environment ● Create a Docker container that standardizes development environment (packages, plugins, run builds, etc)
Oct 2	Working frontend prototype ● Using basic HTML/CSS/JS, provide a static prototype of Upload csv and Visualizations dashboard screens with either hardcoded placeholder data or dummy JSON data in order to show to the client as first iteration of design
Oct 16	Working backend prototype ● Given a raw csv file, clean and filter the data. Query for an individual user ID and pass to Python functions as Dataframe ● Using Python, implement functions to generate JSON information for each visualization and return to frontend as JSON
Oct 30	Frontend/backend communication ● Enable upload of file from client page to backend for processing into Dataframe ● Ensure JSON is passed and can be rendered dynamically by frontend
Nov 13	Finalize backend ● Testing and verification of Python functionality including edge cases
Nov 27	Finalize frontend ● Improve visual polish and intuitiveness of client interaction

Challenges

One of the challenges that we encountered at the start of our project was the conceptual grasp behind the transformation of raw data to the node graph, the main visualization in our tool. In our initial meetings with the client, the client had a solid grasp of what states they wanted (Reading, Visualizations, Exercises) and the general visual layout of the graph. However, we had to make decisions on how to translate this vision into a practical application. We decided that an aggregate of all students would not be a useful design because an aggregate does not provide

useful information about specific student behaviors. It would also be more difficult to track student transitions and detect cycles. Our solution was to use a lighter Scores file to process only the student IDs and final grades first after the user clicks the Go to Dashboard button in the first Upload csv screen. An API endpoint gets the list of students without rendering data from the bulkier Interactions file. Later, when a user selects a student from the student list, the JSONs representing graph metadata will be generated and cached to render in the visualizations dashboard.

Another challenge that we faced was using the boilerplate Master's code. Many paths in the code were hardcoded and header references were inconsistent, requiring us to do an intensive amount of refactoring in order to be able to run the code with given csv files. Additionally, the Master's code was designed to execute once for a longer period of time, iterating through all students in a research analysis context. Our context was different; we needed to implement a tool that was adequately fast in loading and processing time so that the user would not have a slow experience on the interface. In order to solve this problem, we decided not to use the Master's code as-is in its raw form. Instead, we handpicked snippets of logic from some of the Python files such as abstracting.py and behaviors.py in order to base our own handwritten functions off of. Moreover, we designed our functions to expect the data of a singular student as input instead of all the students at once, to reduce the processing load upon selection of a student in the student list.

Another challenge that we faced was having working communication with the frontend loaded on a client web browser and the backend docker container. We ran into issues with CORS policies as browsers require a certain type of header to be returned by the webserver hosting the resource we are requesting when an options response is sent. We were able to resolve this by adding the appropriate CORS headers to the backend's code to be included with every response.

Future Work

Looking ahead, we envision expanding the project further by introducing additional graph visualizations to enhance the richness of our dashboard. In response to our professor's suggestion, we plan to implement aggregated graphs that succinctly summarize the overall performance of the entire class. Additionally, we aim to enhance user experience by exploring UI customizability options, ensuring a more tailored and user-friendly interface for our stakeholders. We can assess and improve our project by surveying other CS professors' usage of the tool and gathering qualitative and quantitative feedback on their user experience. Our commitment to continuous improvement and innovation drives our aspirations for the future of this project.

Acknowledgments

We would like to acknowledge our client Dr. Mohammed Farghally (mfseddik@vt.edu) for his continued support and feedback during the development of the Visualizing eTextbook Student Interactions tool. We would also like to acknowledge Masters researchers Samnyeong Heo *et al*. for their boilerplate Python code that we used as a basis for cleaning, processing, and analyzing the raw student interaction files. Lastly, we would like to acknowledge Dr. Mohamed Farag (mmagdy@vt.edu) and our capstone class for invaluable questions and constructive criticism elicited during progress updates and interim presentations throughout the semester.

References

Heo, S. (2022). Analyzing Student Session Data in an eTextbook. *Masters Theses*. http://hdl.handle.net/10919/111286

Create a palette. Coolors. (n.d.). https://coolors.co/generate

Appendix

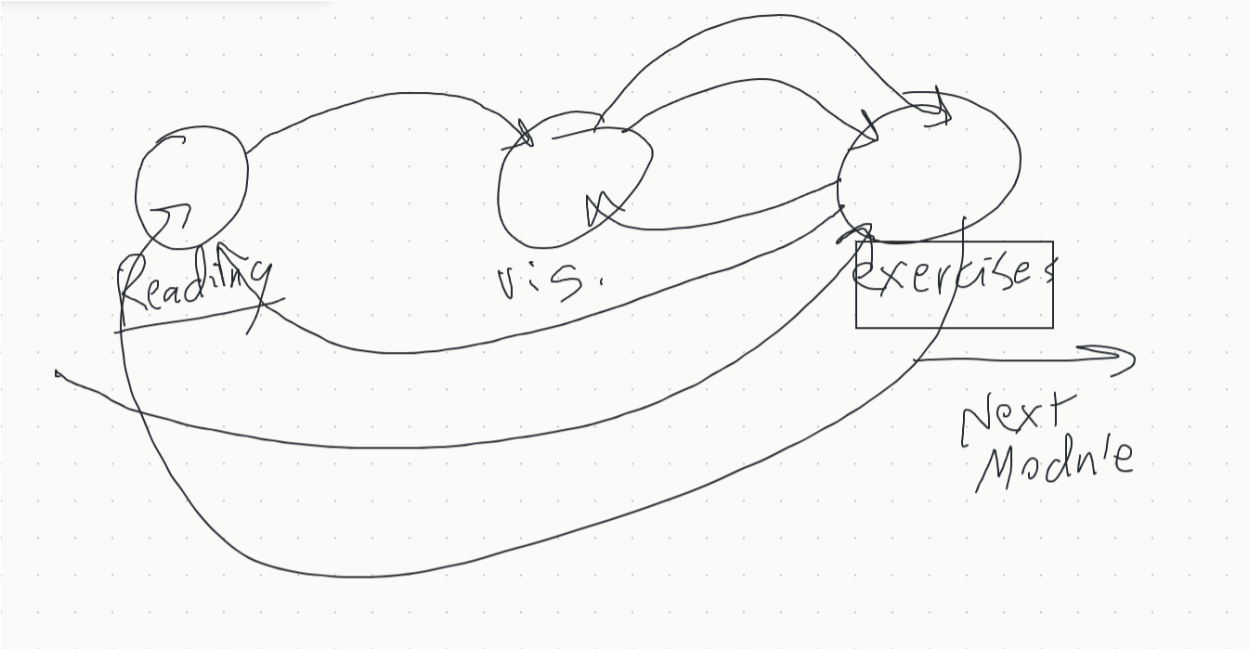


Figure 1. 09/04/2023: Dr. Farghally's initial sketch of his vision for the node graph

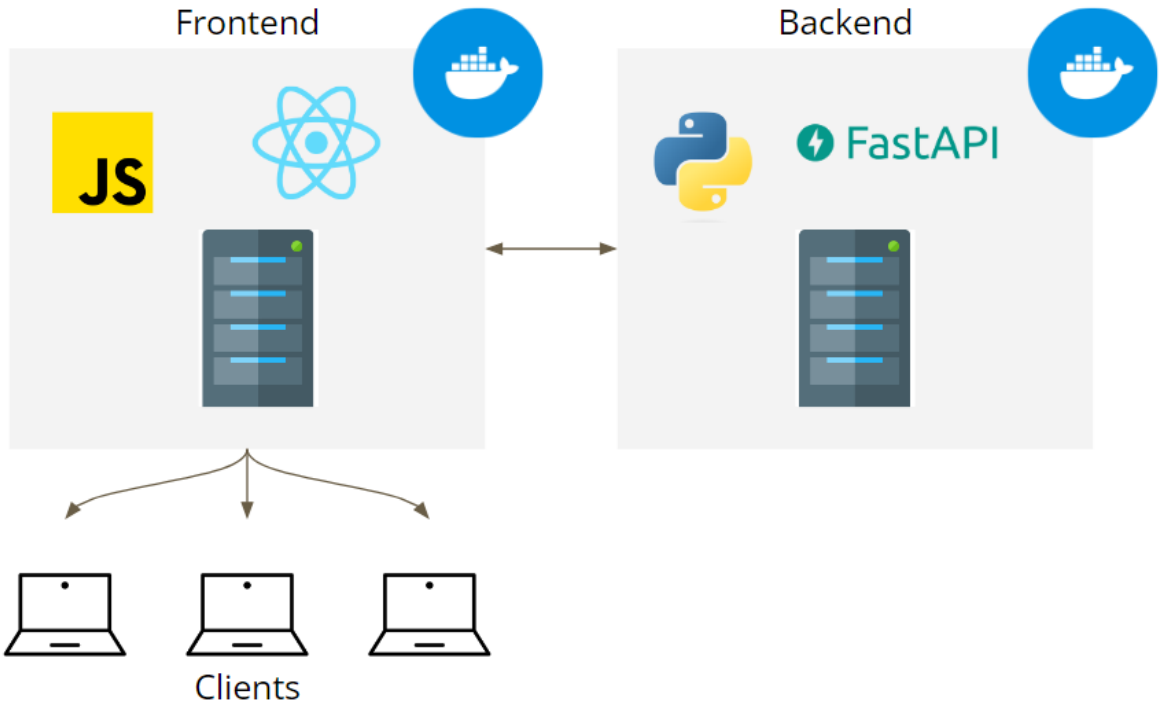


Figure 2. 09/13/2023: System architecture of backend

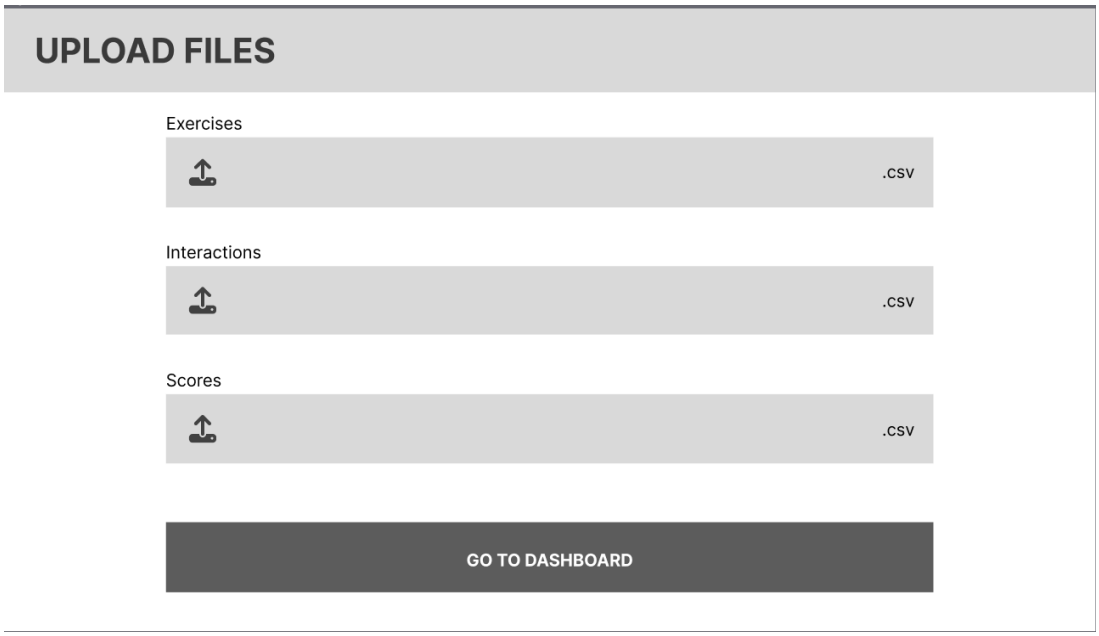


Figure 3. 09/13/2023: First iteration of Upload csv screen from prototype

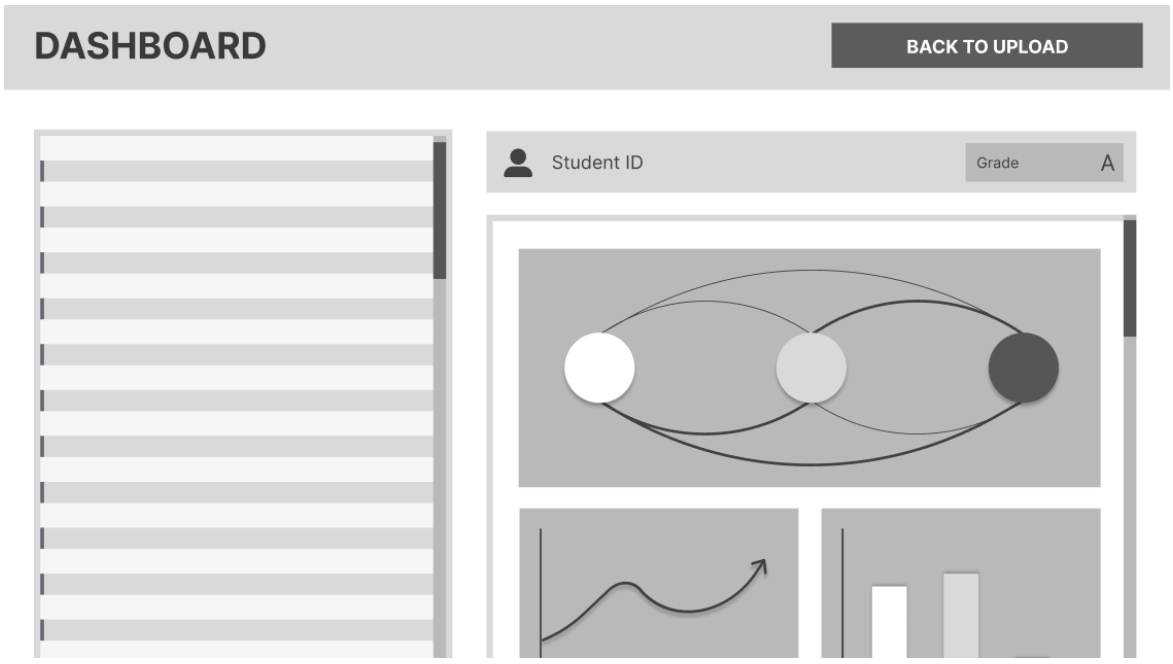


Figure 4. 09/13/2023: First iteration of Visualizations dashboard screen from prototype

Figure 5. 09/18/2023: Sketch of timeline visualization idea

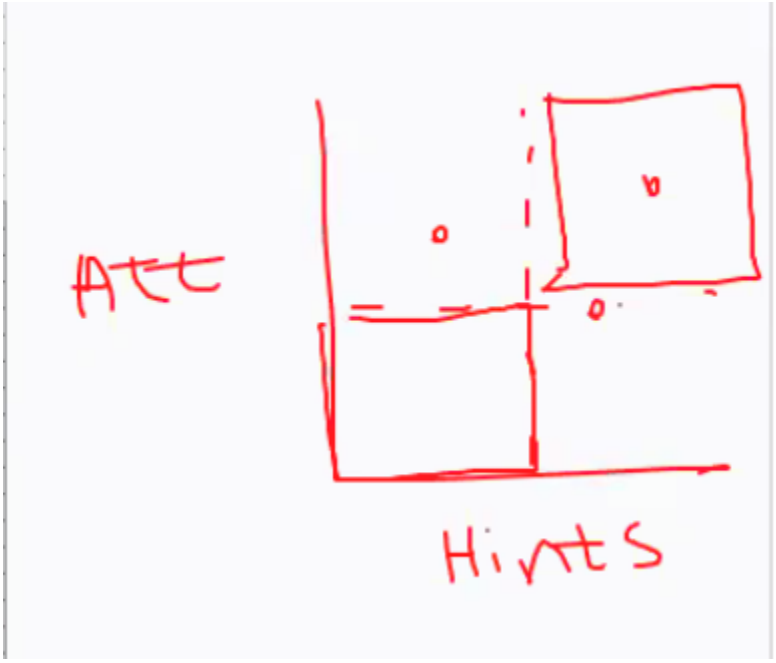


Figure 6. 09/18/2023: Sketch of hints vs. attempts scatterplot visualization

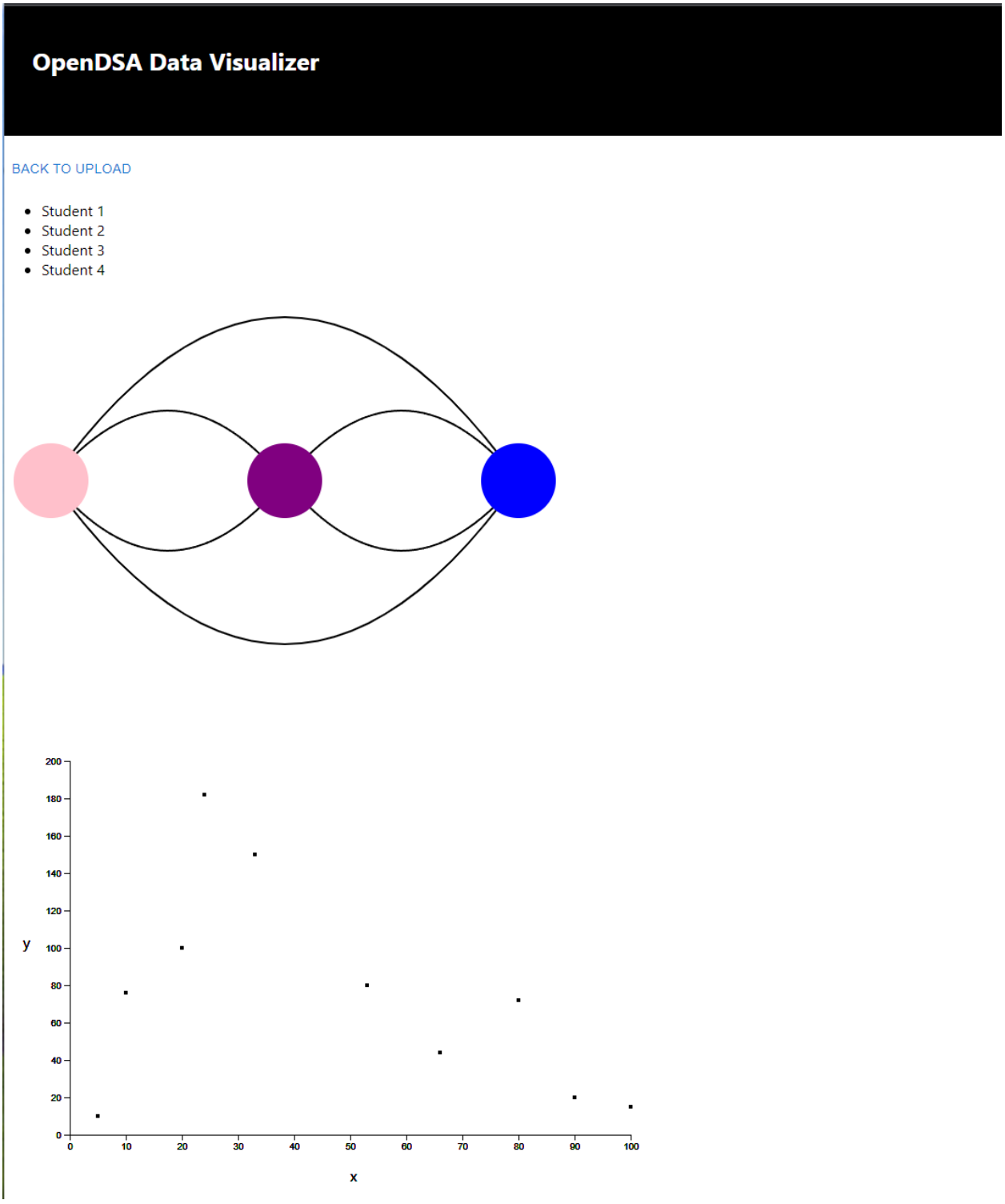


Figure 7. 09/21/2023: Static HTML/CSS skeleton web interface

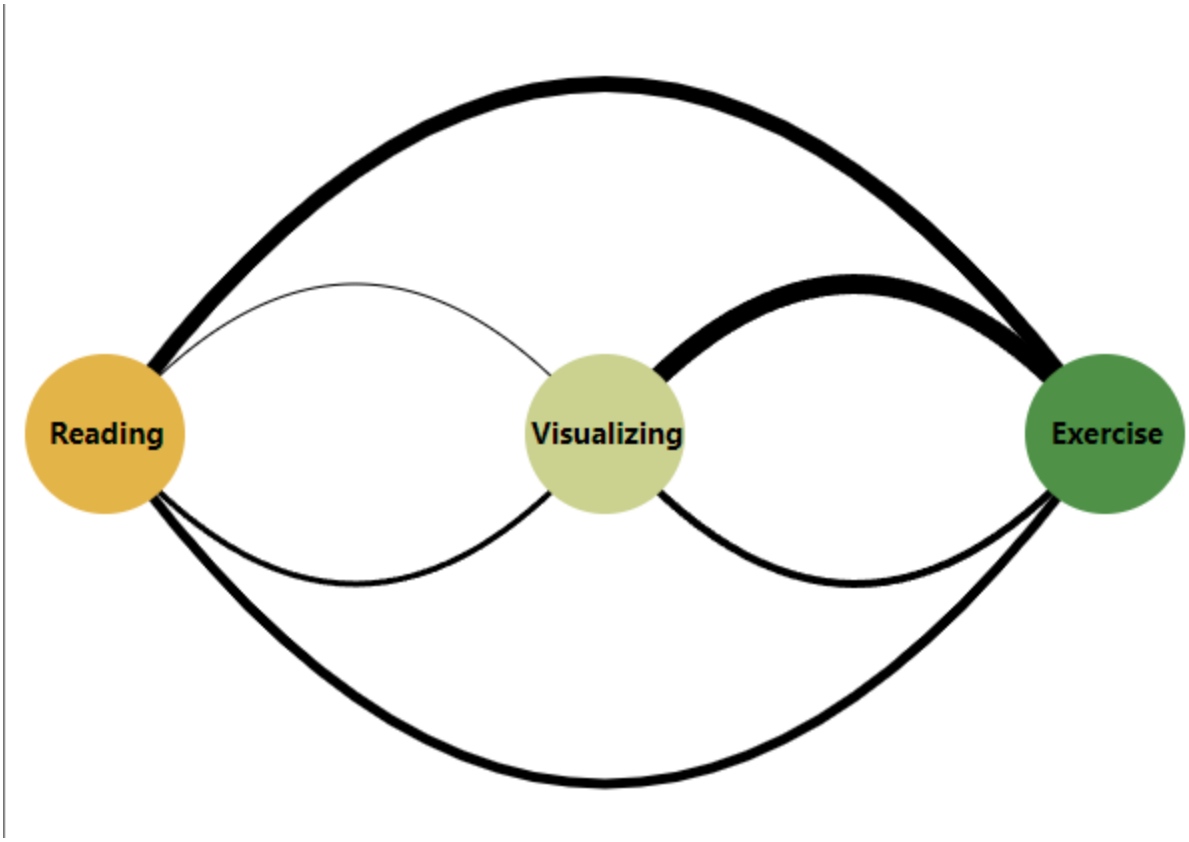


Figure 8. 09/26/2023: Unstyled prototype node graph constructed from brainstorm sketches with the client

Figure 9. 10/18/2023: Original inspiration for timeline visualization with state duration transitions

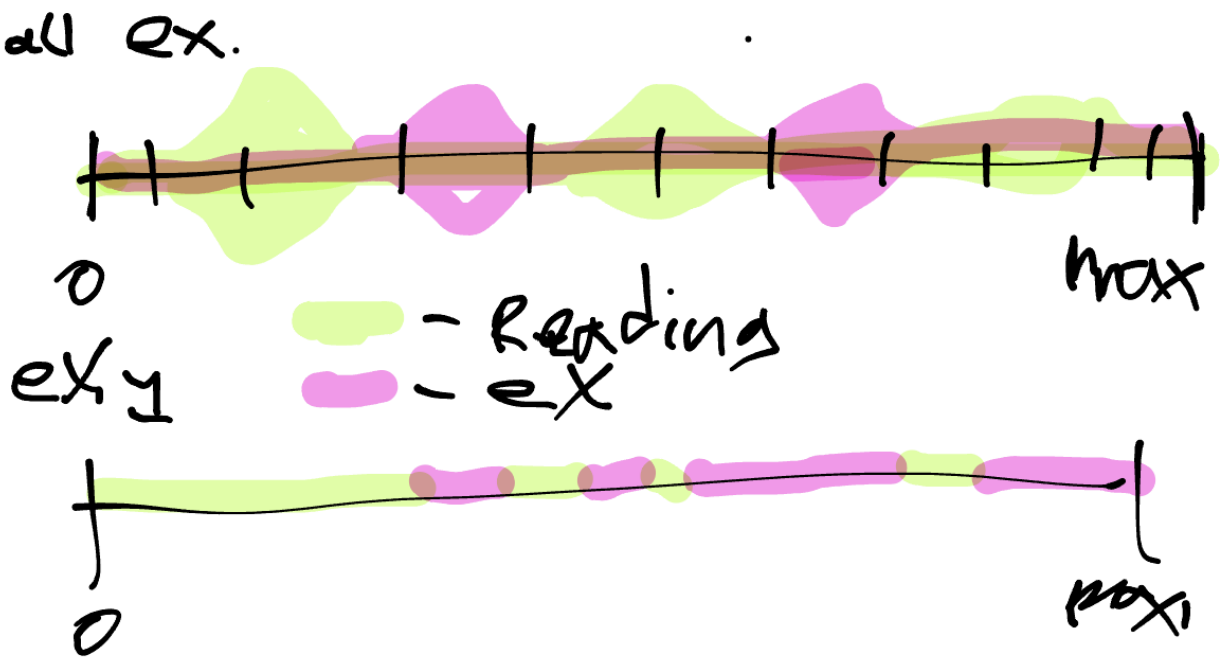


Figure 10. 10/24/2023: Low-fidelity sketch of timeline visualization with state duration transitions

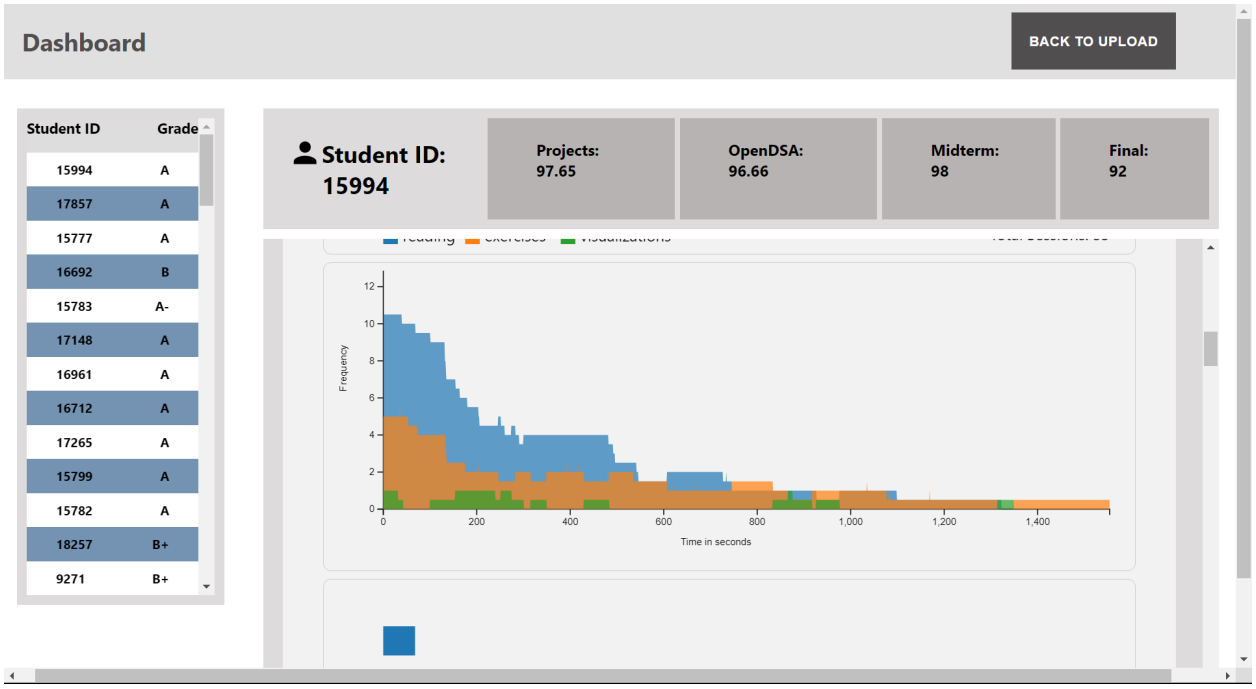


Figure 11. 10/29/2023: Prototype draft of state duration distributions visualization

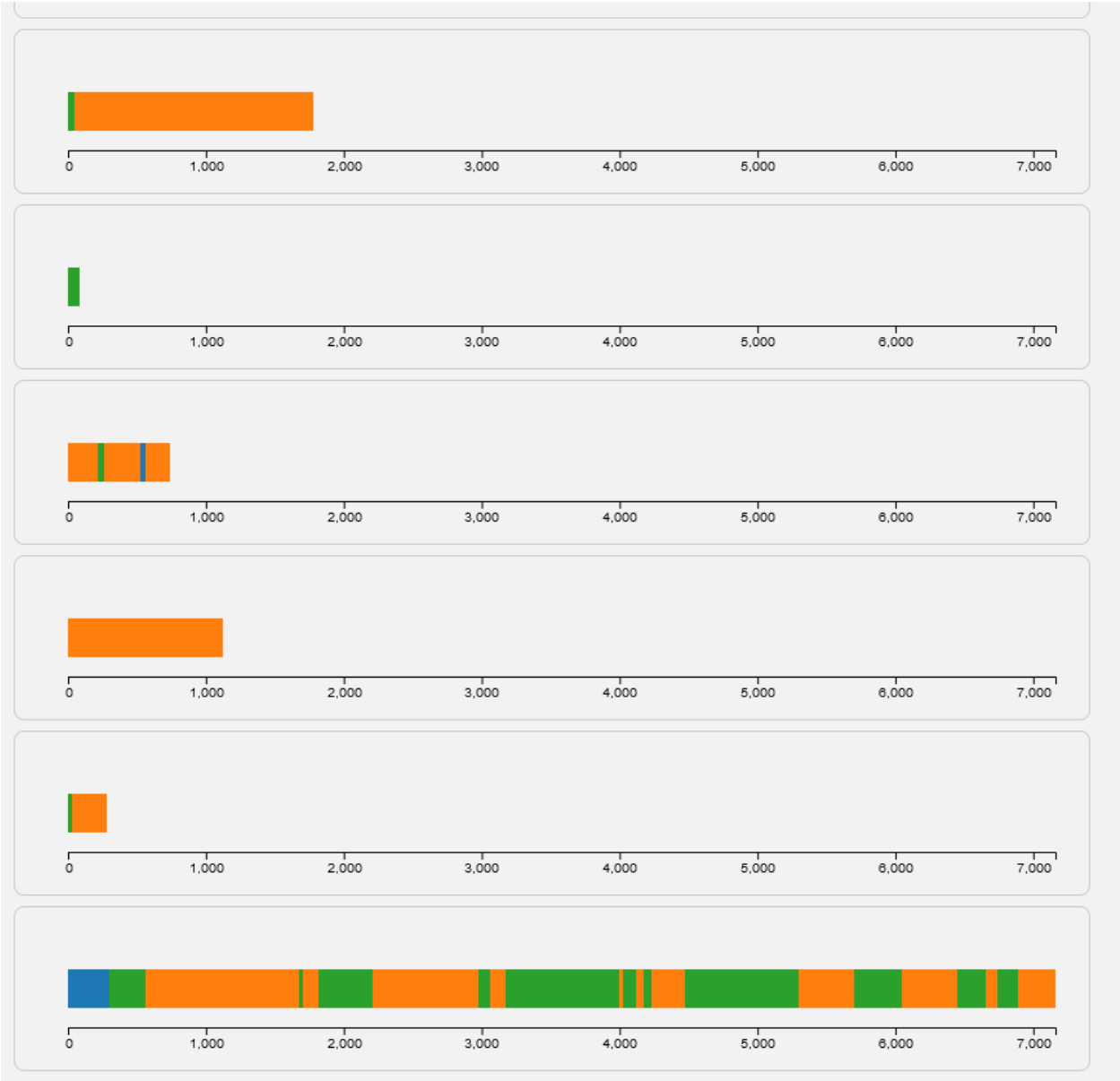


Figure 12. 10/29/2023: Breakdown of an individual student's cumulative state durations per session as predecessor to timeline visualization

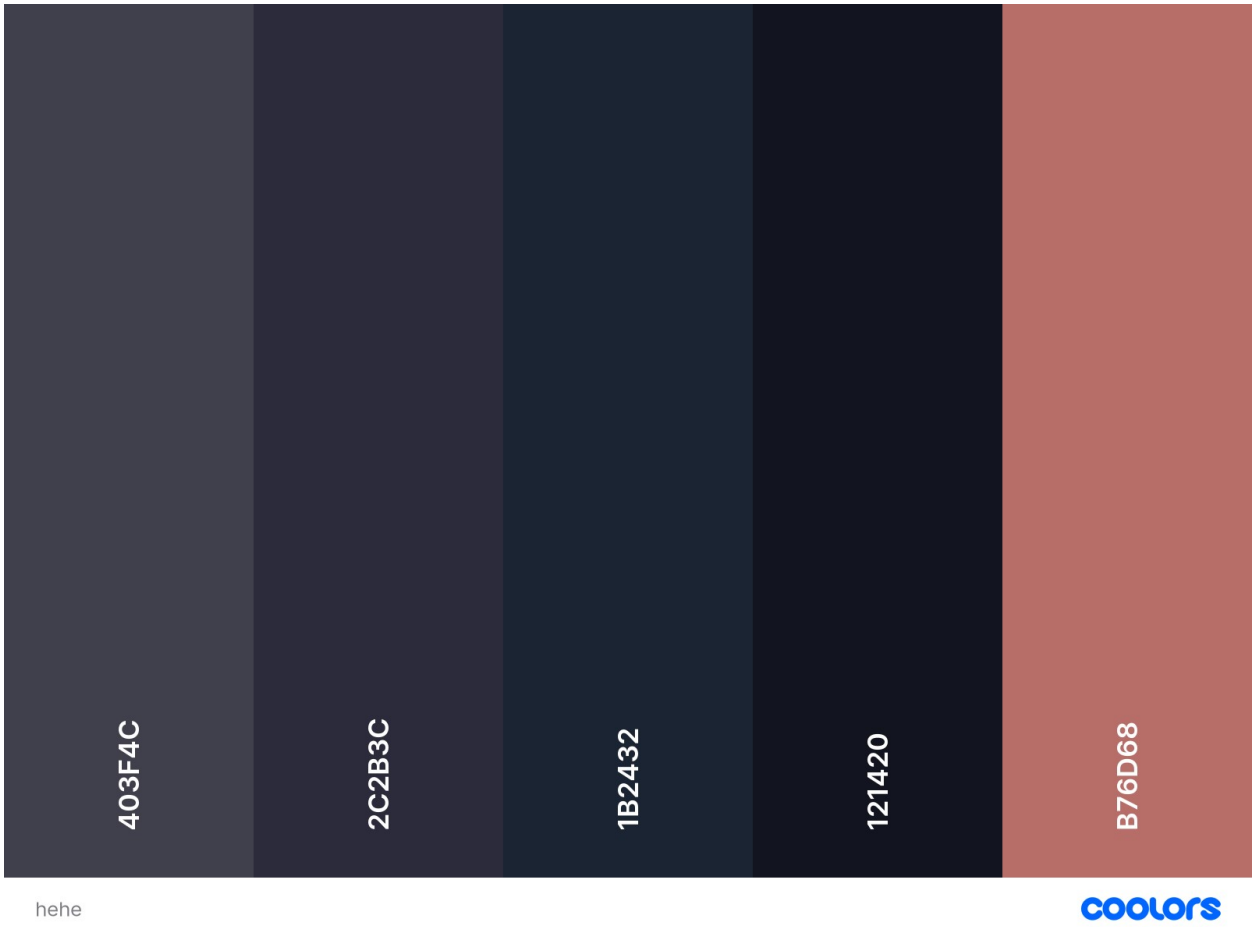


Figure 13. 11/02/2023: Dark mode color palette reference generated by Coolors tool

Figure 14. 11/10/2023: Early stages of grid view dashboard layout

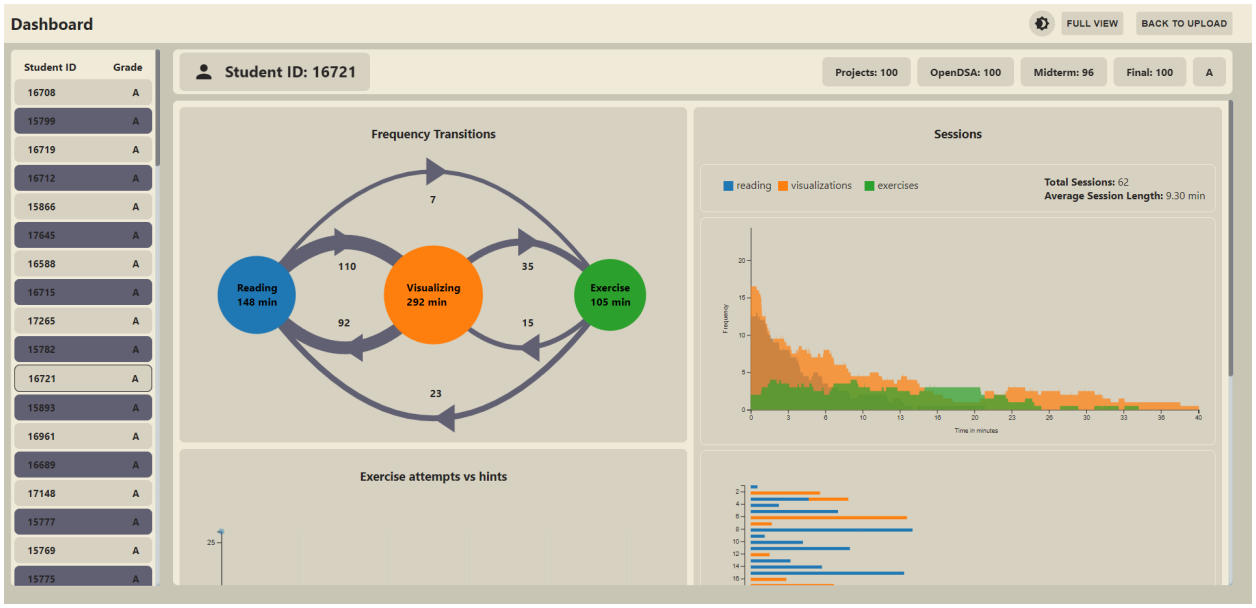


Figure 15. 11/16/2023: Finished visualizations dashboard in light mode

Figure 16. 11/16/2023: Finished visualizations dashboard in dark mode