MySTEMGrowth Survey Tool

Design Document

Sdmay 26-27

Dr.Rover

Dr. Rover, Yiqi Liang

Ethan Buenting: Backend/Cloud

Ethan Van Caster: Developer

Nina Gadelha: Cybersecurity Developer/Testing

Ryan Mamrot: Developer

Sam Craft: Developer/Testing

Caleb Hemmestad: Backend/Cloud

https://sdmay26-27.sd.ece.iastate.edu/

Revised: 09/30/2025

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	Protects student data and promotes well-being by giving learners clear feedback on their research experience and growth. Security controls help prevent misuse of personal information.	Encrypted auth and least-privilege roles. Clear progress + "resume later" to lower survey fatigue. Admin tooling that avoids accidental exposure of student results.
Global, cultural, and social	Respects diverse backgrounds and programs by using inclusive language, accessible design, and flexible scales/question types. Supports different academic cultures and course structures.	Mobile-first layouts. Wording review to avoid cultural bias. Role views for admins, coordinators, and students.
Environmental	Digital-first delivery reduces paper and travel overhead; efficient cloud choices lower compute waste. Development practices consider energy usage of hosting and CI/CD.	Online surveys instead of printed packets. Adjusting the database size to match what we actually need. Static asset caching/CDN. Deleting stale accounts/data to reduce storage footprint.
Economic	Keeps costs sustainable for universities and research programs. Streamlines coordinator/admin time. Supports student success and program improvement that can attract funding.	Predictable hosting. Automated CI/CD to cut maintenance. Batch export of results for grant reports. low training/onboarding time due to simpler ops.

4.1.2 Prior Work/Solutions

One of the biggest advantages our application offers is the ability to collect survey data virtually, allowing for easier exportation for research purposes, or simply to make it easier to allow the student to view how their personal outlook and abilities have changed throughout their research experience. Using digital processes to collect information allows data to be processed more time efficiently, and has been used in a variety of different applications to help assess data and provide analytics [1]. The survey itself also holds similarity to previous projects that involved assessing undergraduate research program effects on students, such as the SURE survey. This survey was created to help collect student responses and evaluate things like student retention and educational development based on the survey responses, similar to one of the potential benefits listed for our project [2]. One of the primary actions they performed to solve issues was to ensure data comparability and standardization to ensure that results collected would be effective if program coordinators or researchers wanted to use the data later on. The previous group, team sdmay25-24, has also had a lot of documentation and a solid code basis for us to work on and help us determine fixes and improvements we wanted to implement, as well as what worked well [3]. One of the biggest shortcomings we found was with the security of the application, as there was not much

documentation, and there were notable security concerns with the application when we received it. However, there were many benefits to be had with having previous documentation as it made understanding the structure of the application very easy to understand and begin working on.

[1] M. M. Hanbury, E. L. Cox, J. M. Culley, and C. M. Pruinelli, "A web-based management application to improve data collection in an emergency care research network," *Journal of the American Medical Informatics Association*, vol. 26, no. 12, pp. 1547–1554, Dec. 2019. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC6351988/

[2] D. Lopatto, "Survey of Undergraduate Research Experiences (SURE): First findings," *CBE—Life Sciences Education*, vol. 3, no. 4, pp. 270–277, Dec. 2004. [Online]. Available: https://www.lifescied.org/doi/10.1187/cbe.04-07-0045

[3] Iowa State University, "MyStemGrowth – Senior Design Team SDMAY25-24," *Department of Electrical and Computer Engineering, Iowa State University*, 2025. [Online]. Available: https://sdmay25-24.sd.ece.iastate.edu/

4.1.3 Technical Complexity

1. The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles

Our application consists of many different subsystems, all of which I will list and describe here:

1.1. Frontend: This component uses React and Javascript to implement the user interfaces that our undergraduate users and other roles will interact with. It is the primary tool that is used to collect data and display it in a clear and readable manner, whether that be for program coordinators to view how their program is influencing undergraduate students, or to allow students to see have they have grown according to their personal experience.

1.2. Backend: The backend is what allows our system to communicate with our databases and hosting environments, allowing for authentication and IO handling to create a seamless experience for users regardless of their role.

1.3. Database: The MySQL database management system allows for the application to store and sort data such as account information as well as survey questions and their response data.

1.4. Hosting: With the current use of AWS and soon to be Digital Ocean, these systems allow us to host our application as well as offer many industry standard tools for secure data transmission and role based authentication.

2. The problem scope contains multiple challenging requirements that match or exceed current solutions or industry standards.

    2.1. Scalable Design: Our application requires the use of industry level design for the purpose of scalability. Our application needs to be able to grow with its usage, as it has the opportunity to be used in a number of research programs.

    2.2. Data Privacy/Security: Our application will implement industry standard security features, such as hashing and the use of cookies, to ensure that user data such as passwords will be safe.

    2.3. Cross-platform accessibility: While the current application works on a browser, our team is working on UI improvements to help make the application easier to use, as well as offer a more readable view for admins or Program Coordinators to look over all responses for the respective programs that they oversee. In addition to that, we are also developing mobile support to allow users to take the survey or view their data on their mobile devices.

    2.4. Automatic Deployment: We are implementing an automated CI/CD pipeline to allow our team, or any team following, to continue to develop the application and seamlessly integrate these changes to the web application without any interruption to service.

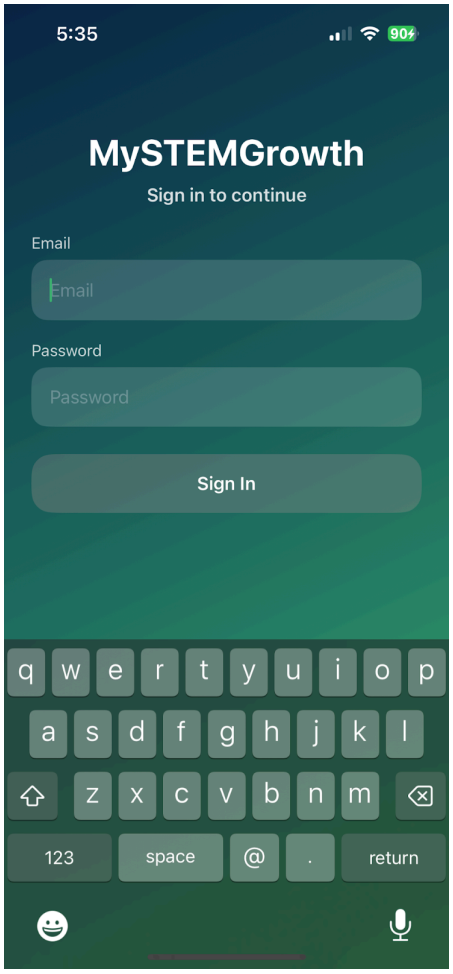
4.2 Design Exploration

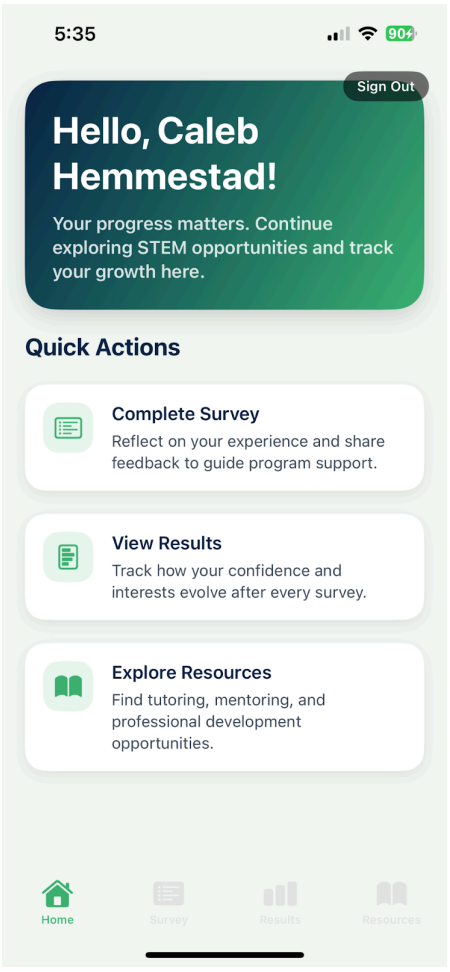
4.2.1 Design Decisions

Switching from AWS to Digital Ocean:

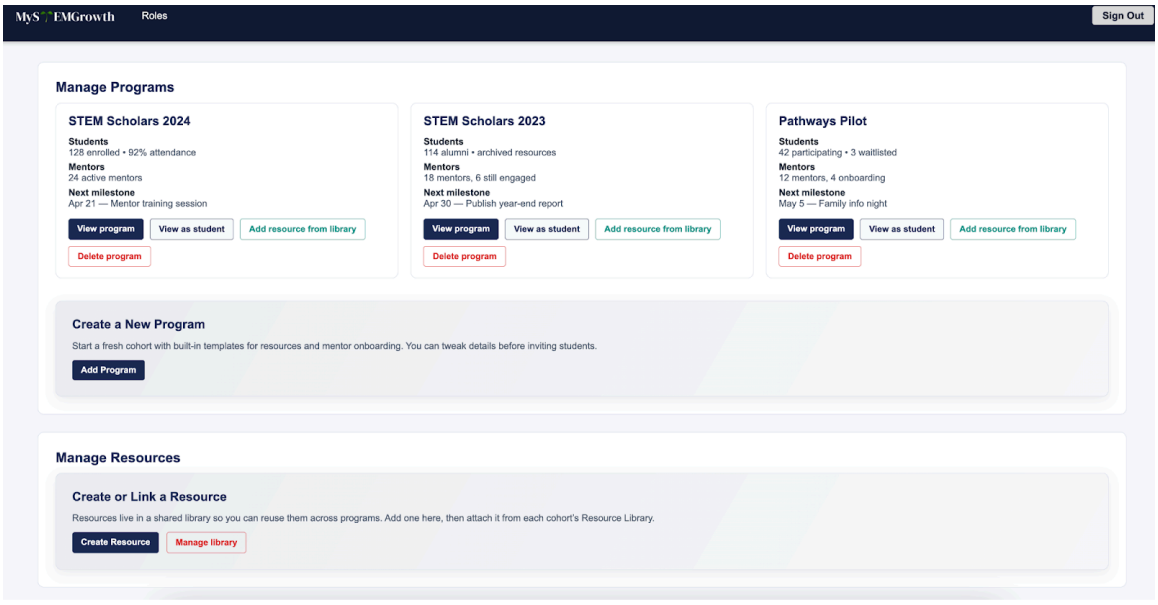
Our team chose to migrate the MySTEMGrowth Survey's infrastructure from AWS to Digital Ocean after coming to an agreement that AWS was overly complex and costly for our project needs. Digital Ocean's App Platform and Managed Database services offered a simpler setup, predictable pricing, and an easy way to integrate with our existing Node.js backend and MySQL database. This transition will enable easier onboarding for future developers with less of a learning curve to get started. It also improves security through encrypted environment variables and managed backups while maintaining scalability and reliability. Overall, the move to Digital Ocean supports our goals of affordability, maintainability, and long-term sustainability for the MySTEMGrowth platform.

Mobile layout support:

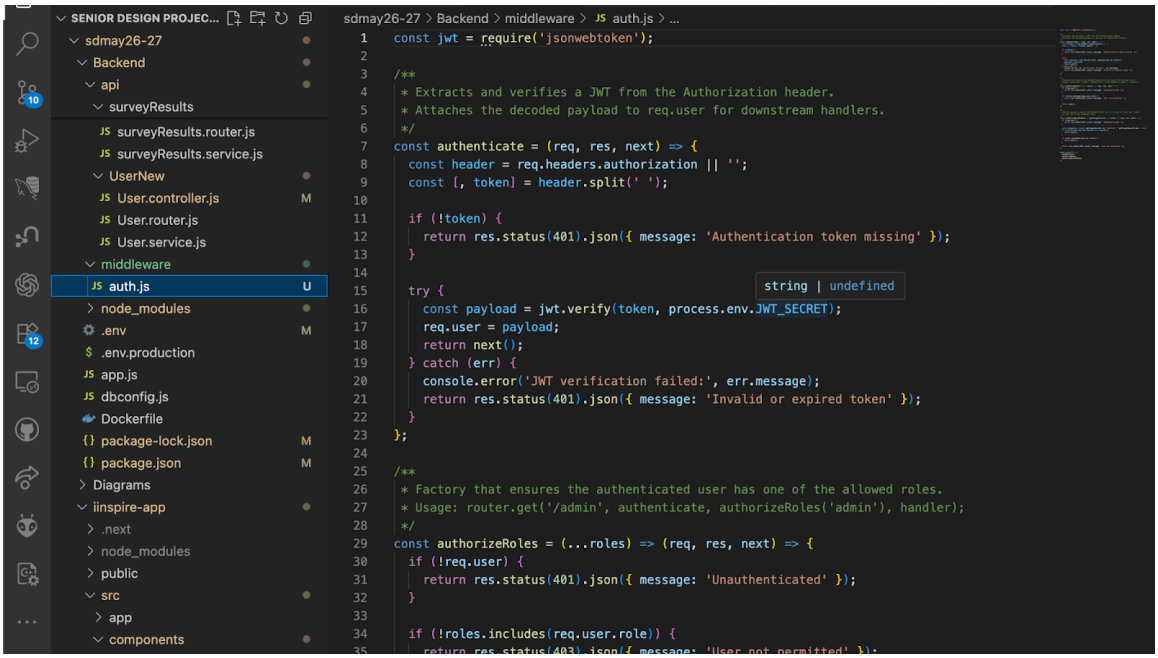




Program Coordinator layout changes:



Improved security with tokens and hashing:



4.2.2 Ideation

For the design decision of switching from AWS to Digital Ocean, our team used a collaborative brainstorming and comparison session to explore multiple hosting and cloud service options. We used a lotus blossom–style ideation approach, starting with the goal of finding a cloud platform that meets our needs while reducing the cost and keeping the app easy to maintain and scalable in the future, then expanding outward with potential solutions and sub-ideas.

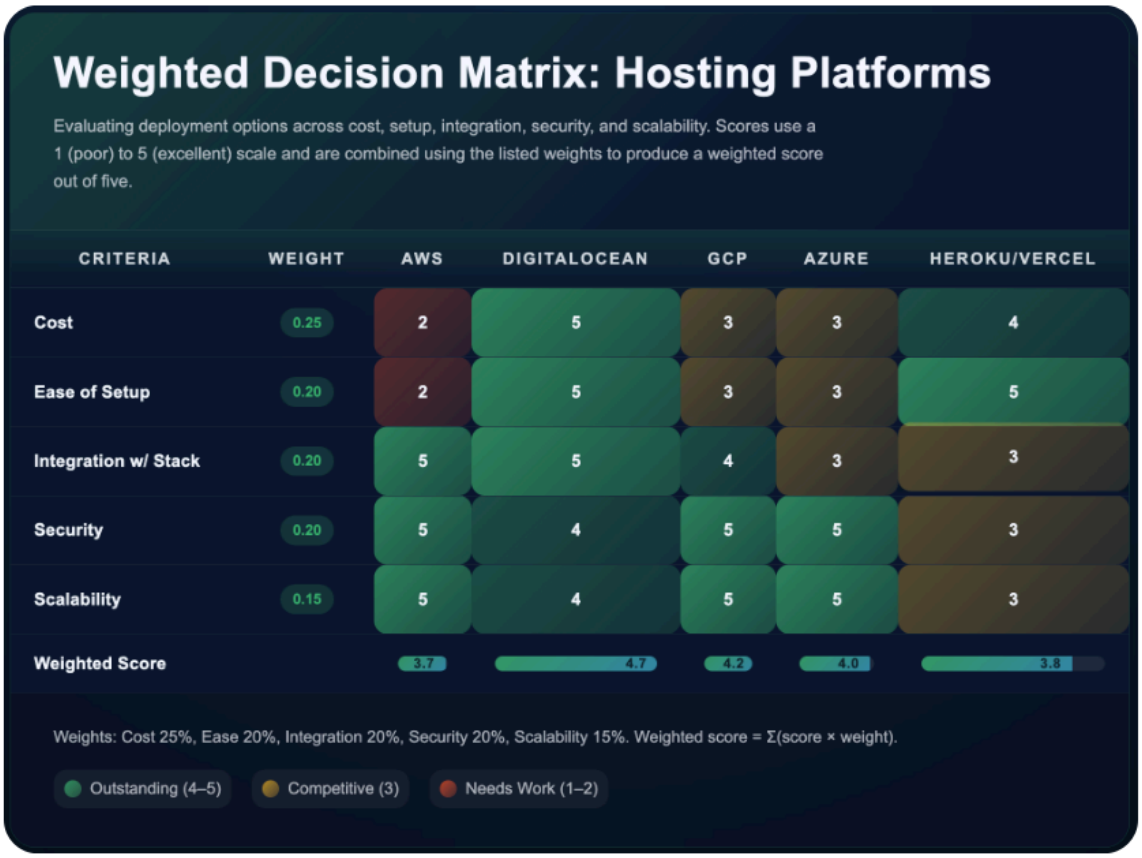
We initially identified five good options:

1. **Amazon Web Services (AWS)** – Continue using the existing infrastructure.
2. **Digital Ocean** – Migrate to a more developer-friendly and cost-effective environment.
3. **Google Cloud Platform (GCP)** – Leverage built-in integrations with CI/CD and analytics.
4. **Microsoft Azure** – Utilize advanced security and enterprise hosting capabilities.
5. **Heroku or Vercel** – Use a simplified app hosting service with auto-deploy pipelines.

After going through all the options, we eventually decided to go ahead with the switch to Digital Ocean which we will start working with as soon as we get access after being helped with ETG.

4.2.3 Decision-Making and Trade-Off

After looking at all five possible hosting options, our team evaluated each using a weighted-decision matrix based on five criteria: cost (25%), ease of setup (20%), integration with our stack (20%), security (20%), and scalability (15%). Each option was scored from 1 (poor) to 5 (excellent).



Digital Ocean achieved the highest overall score by meeting most of the criteria that we were looking for. While AWS and Azure offered stronger enterprise-grade scalability, their pricing and configuration complexity made them less practical. Heroku and Vercel were intuitive but lacked control over databases and back-end customization.

The trade-off in choosing Digital Ocean was accepting slightly fewer enterprise integrations in exchange lower cost and less of a learning curve for future developers.. We think this decision ultimately supports MySTEMGrowth's goals of maintainability, reliability, scalability, cost efficiency, and long-term sustainability.

4.3 Proposed Design

4.3.1 Overview

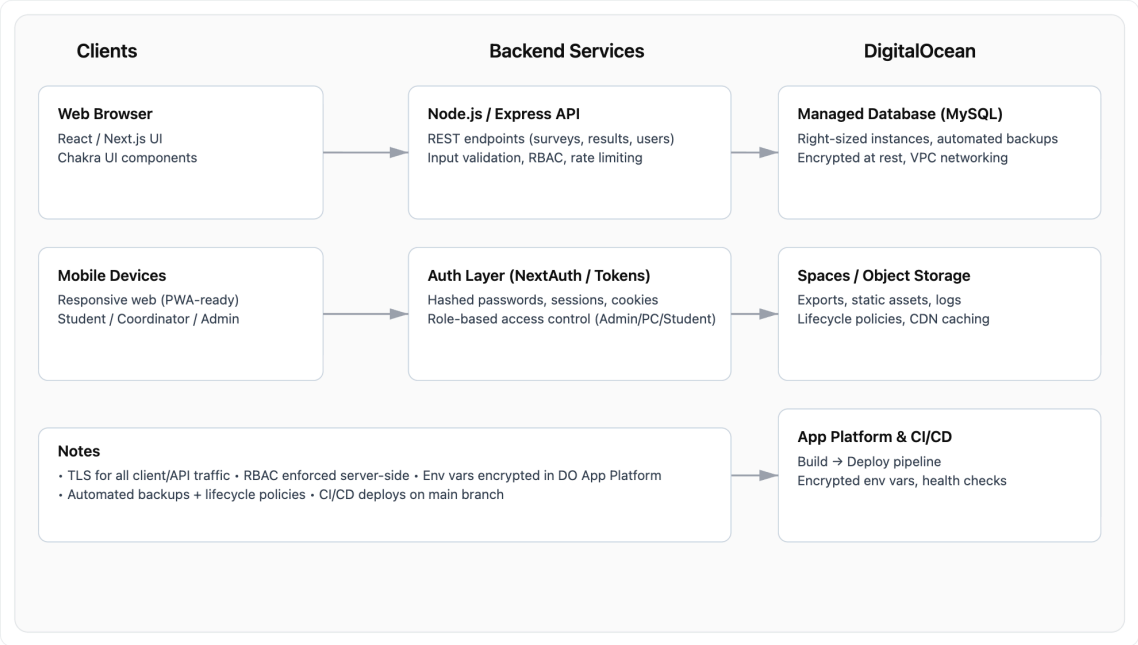
At a high level, the MySTEMGrowth tool consists of two main parts, the front end and the back end. The front end is the user interface of the program. This handles everything a user will need to interact with and use the tool. The other major component is the back end. The back end handles all data manipulation and storage. Inside the back end, the survey results are compiled and calculated to give the proper graph to the user. The front end and the back end are very entwined. For everything the user does in the front end, something in the back end will be run to facilitate that. .

4.3.2 Detailed Design and Visual(s)

MySTEMGrowth is a role-based web application (Student, Program Coordinator, Admin) delivered as a responsive React/Next.js client that talks to a Node.js/Express API, backed by a managed MySQL instance on DigitalOcean. Persistent assets (exports/logs) live in object storage. Authentication uses hashed passwords and session cookies/tokens; authorization is enforced via RBAC on every API boundary. CI/CD builds and deploys the app platform components on push. See Figure 1 (System Architecture) and Figure 2 (Components and Data Paths).

Figure 1. System Architecture

High-level view of MySTEMGrowth showing client apps, backend services, and cloud resources on DigitalOcean.



Frontend (React/Next.js)

- **Survey UI**: renders question types (slider/likert/text), validates inputs, supports “save progress.”

- **Results Dashboard**: charts/filters for individuals and cohorts; CSV export.
- **Role Views**: conditional navigation and routes per Student/PC/Admin.
- **HTTP Client Layer**: fetch wrapper adds auth headers, retries, and uniform error handling; maps DTOs to view models.
- **Integration**: communicates with `/auth`, `/surveys`, `/results`, `/users` endpoints over HTTPS; receives JSON.

Backend (Node.js/Express)

- **Controllers**: route handlers for auth, surveys, results, users; input validation (e.g., zod/express-validator), pagination.
- **Service Layer**: business rules (e.g., PCs can only view their groups), audit logging.
- **Auth & Sessions**: password hashing (bcrypt/argon2), secure httpOnly cookies, optional JWT for API clients, token rotation.
- **Data Access Layer**: parameterized SQL; transaction boundaries for multi-write ops.
- **Integration**: uses a pooled MySQL client; emits structured logs/metrics for ops.

Data Stores (DigitalOcean)

- **Managed MySQL**: users, groups, surveys, questions, responses, response_items; automated backups and VPC isolation.
- **Spaces/Object Storage**: export files, static assets, optional logs with lifecycle policies.

Operations (CI/CD & App Platform)

- **Build/Deploy**: on main branch, build containers, run unit tests, deploy API and static site; health checks and rollbacks.

Figure 2. Components and Data Paths

Key subsystems with their roles and how data moves from UI to database.

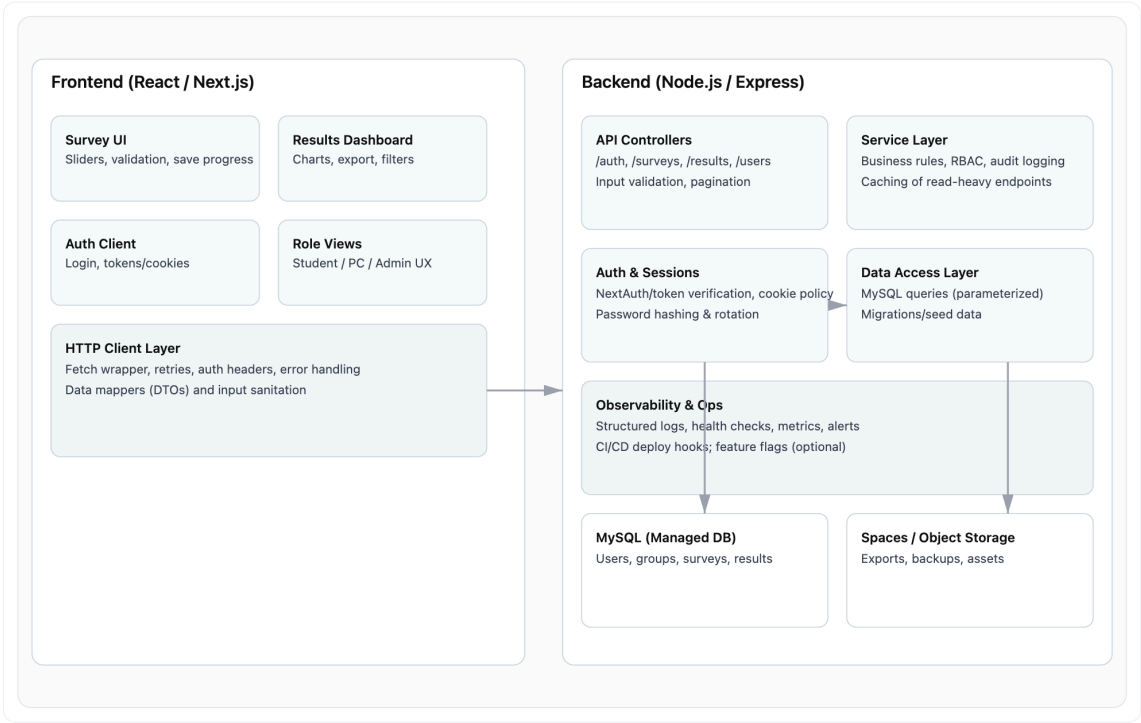


Figure 4 (Mini ER Data View) shows the core entities and relationships: **Users** (PK `user_id`) join **Groups** via **GroupMembers**; **Surveys** contain **SurveyQuestions**; groups are linked to surveys through **Assignments**; **Responses** capture a user's submission for a survey (with denormalized `group_id` at submit time); **ResponseItems** store per-question answers. This schema supports role-scoped reads, longitudinal pre/post comparisons, and efficient aggregation. (Crow's foot can be added if your class prefers that notation.)

Representative API surface (abbrev).

- `POST /auth/login` → set session cookie after hash verify; returns user/roles.
- `GET /users/me` → returns profile + role; used to gate client routes.
- `GET /surveys/:surveyId` → survey + questions (role-agnostic).
- `POST /responses` → start or submit responses (supports partial save, `is_complete` on items).
- `GET /results?groupId=…` (PC/Admin) → scoped aggregate + per-student results; Students use `GET /results/me`.

- `GET /exports/group/:id` (PC/Admin) → generates CSV to Spaces (pre-signed download).

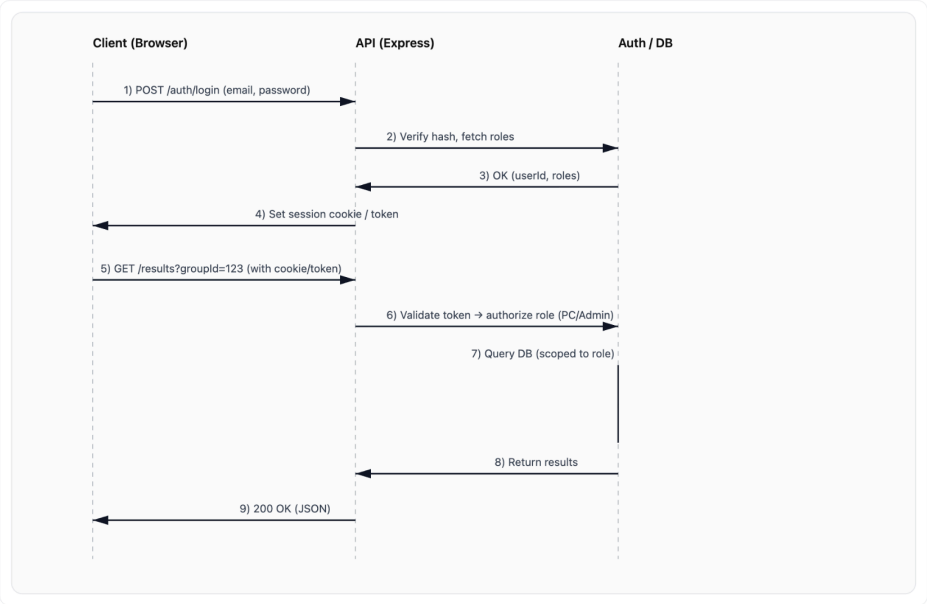
AuthN/AuthZ model.

- **Authentication**: email + password (hash stored), secure httpOnly cookie; optional short-lived JWT for programmatic access.

- **Authorization**: RBAC middleware on every protected route; Students only see their own data, PCs see assigned groups, Admins have elevated read/export scopes. See Figure 3 (Login & Session Flow).

Figure 3. Login and Session Validation Sequence

Sequence showing how the client authenticates and how the API validates role-based access.



Notes: Sessions use secure, httpOnly cookies. Role-based access ensures Students can only access their own records; Program Coordinators see only their assigned groups; Admins have elevated read scopes.

Validation, errors, and resilience.

- **Validation**: server-side schema validation; client mirrors constraints for better UX.

- **Errors**: uniform JSON shape `{code,message,details}`; 4xx for validation/permission, 5xx for server.

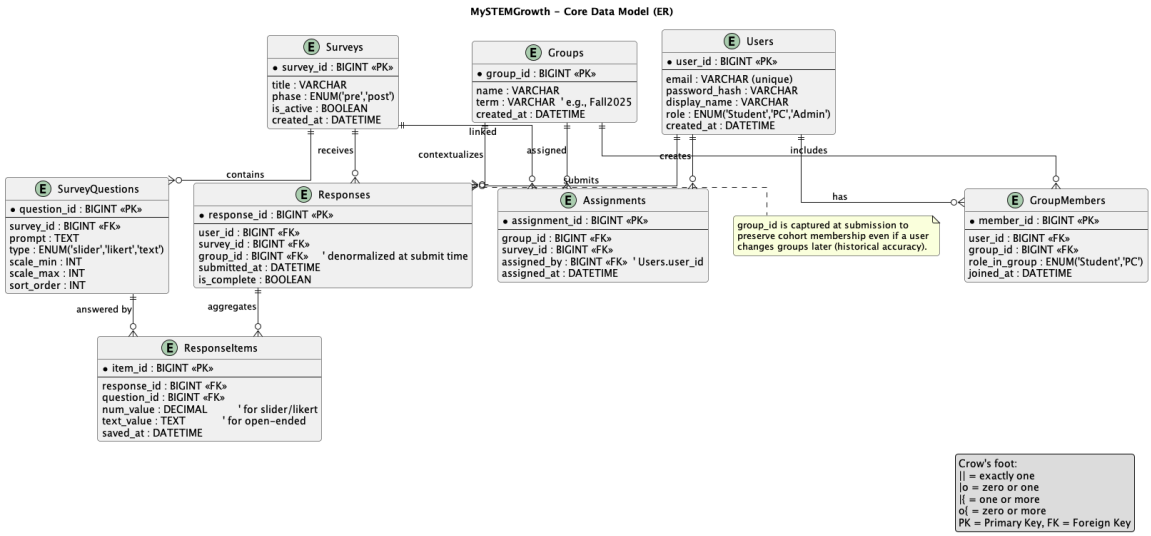
- **Resilience**: DB connection pooling; exponential backoff/retries for transient failures; idempotent POSTs where feasible.

Performance & scalability notes.

- Read-heavy endpoints (e.g., cohort dashboards) support **pagination and simple caching** at the service layer.
- **Right-sized DB instances** and query indexes (on `user_id`, `group_id`, `survey_id`) keep latency low.
- Static assets are cached; API uses gzip/br compression.
- Horizontal scaling path: add API replicas behind the App Platform; DB vertical scale as needed.

Security & privacy.

- Password hashing (argon2/bcrypt), TLS everywhere, least-privilege DB credentials, parameterized queries to prevent SQLi.
- Encrypted environment variables, role-scoped queries, audit logs for sensitive reads/exports.
- Backups tested via periodic restore drills; PII access limited to Admin/PC roles per policy.



4.3.3 Functionality

A standard user (student) would start by making an account with the code given to them by the program coordinator. After making the account, the student would have the potential to have extra

required demographic information, depending on the program coordinator's choice. After that, the next major step would be for the student to take the survey itself. Once the survey is complete, the student will be able to see the data from their survey. If the student were to come back and retake the survey at a later date, the graph at the end would show a comparison of the growth, rather than just an isolated result.

A program coordinator will create an account, and then create a program. Once the program is created, the code will become available for the program coordinator to give out to students. The program coordinator will also be able to demonstrate the survey to students from their program coordinator account. After students have taken the survey, the program coordinator will be able to go look at student's results to understand where they are at. This includes both the single data point graph and the comparison graph. Program coordinators will also be able to link resources to a program for students to access as necessary.

An administrator will be more removed from the survey itself. The focus of the administrator is to make sure that all programs are set up and running smoothly. They will be able to delete/modify accounts, change settings in specific programs, and look deeper into any program they desire. This role is the highest level of access of the three roles, and as such, the role given out to the least amount of people. Administrators will most likely not directly interact with students, but only with program coordinators.

4.3.4 Areas of Concern and Development

The current state of the application is operational, but there are 3 main areas of improvement that our group is focusing on in the immediate future. The application functions well for demonstration purposes, but lacks in some important areas that our team deems important before a full release. The first area that we want to improve is security. The primary security concern is that all variables are saved in local storage to allow for persistent login, but that is an easy vulnerability that allows a malicious user to change their role or view passwords in plain-text. Currently we have 2 people on our team working on hashing passwords and implementing cookies. The 2nd area we want to improve is mobile device usage. The application works well on a browser, but currently the mobile element is an awkward scrollable element that does not look very professional. The team is working within React to set pixel width settings to improve the mobile view. The final area would be simplifying the cloud hosting services. By switching to Digital Ocean, the hosting service will be more cost effective for a service, as well as provide a more simplistic service that will be easier for future hosts or developing teams to use.

4.4 Technology Considerations

Front-end:

React: JavaScript library used to create reusable components for interactive user interfaces

Pros:

- Reusable components
- Easy to integrate with other libraries
- Fast rendering

Cons:

- Steep learning curves for beginners
- Frequent updates can compromise feature compatibility

Trade-offs: Allows for fast, reusable UI development with complexity of frequent updates and steeper learning curves.

Next.js: React framework that enables server-side rendering and static site generation

Pros:

- Easy routing system
- Full-stack capabilities with API routes

Cons:

- Larger bundle size
- Steep learning curve
- Limited flexibility with custom configurations

Trade-offs: Offers enhanced performance and full-stack capabilities, but is more complex than using React by itself.

Chakra: React component library that provides customizable UI components for web applications

Pros:

- Prebuilt, responsive components
- Easy styling with consistent design system

Cons:

- Difficulties overriding when styling or complex layouts
- Limited design capabilities compared to competitors (CSS or Tailwind)

Trade-offs: Faster UI development with styled components, yet, can limit design flexibility and increase storage usage.

Front-end design solutions and alternatives: We inherited this design from two previous senior design teams. We have decided it would be in our best interest to continue working with the libraries/addons they have already implemented. We have spent a lot of time in CPRE 4910 reviewing the current code and learning more about the libraries used. This allows us to practice and ensure we understand the capabilities when the time comes to implement desired features, presented by our clients.

Back-end:

Node.js: Javascript runtime that allows developers to build applications using a non-blocking, event driven architecture

Pros:

- High performance
- Scales easily
- Large ecosystem with ‘npm’ packages

Cons:

- Frequent updates can compromise feature compatibility
- Callback based code can be complex

Trade-offs: Provides high performance, scalability for server side development but struggles with CPU intensive jobs/processes.

Postman: Collaboration platform for API development

Pros:

- Easy API testing and debugging
- Collaboration features for teams

Cons:

- Limited (free) testing/features
- GUI dependency

Trade-offs: Makes API testing and collaboration easy, but is resource heavy.

Back-end design solutions and alternatives: Many of our team members have experience with at least one of the two main backend technologies implemented. We could switch to alternatives, such as cURL or Django, however, since these technologies have already been implemented by previous teams, and we have experience with them, we decided it would be in our best interest to continue utilizing their features, already integrated with the site.

Security:

Bycrypt: Library that supports hashing and password verification

Pros:

- Easy to implement
- Hashing resistant to brute force attacks

Cons:

- Slower algorithm
- Adds computational overhead to authentication

Trade-offs: Requires careful/tedious configuration, but ensures very safe password security and provides easy hashing and salt implementation.

Tokens: Pieces of data used for authenticating/authorizing users (JWTs)

Pros:

- Carries user data securely
- Stateless authentication (no server-side needed)

Cons:

- Must be securely stored
- Revocation/expiration can be tricky to implement

Trade-offs: Require careful storage and tedious implementation but enables stateless, cross-service authentication.

Security design solutions and alternatives: Bcrypt and tokens are something new we are adding to the project, since security of the website was previously lacking. We could use alternatives such as multi-factor authentication or API keys, but our cybersecurity developers have already begun learning how to implement our chosen security features.

Cloud:

GitHub/Git: Platform used for version control and collaborative software development

Pros:

- Easy collaboration and code sharing
- Integrates with CI/CD
- Tracks changes/maintains history

Cons:

- Limited free usage
- Complex for advanced workflows

Trade-offs: Simplifies version control and collaboration but has limited free usage and can become complex for larger projects.

Digital Ocean: Cloud infrastructure provider for managing applications

Pros:

- Simple and user friendly interface
- Affordable pricing
- Fast deployment

Cons:

- Fewer advanced features than competitors (AWS)
- No team experience using this platform

Trade-offs: Offers simple, affordable cloud management solutions while lacking some advanced features and scalability.

Cloud design solutions and alternatives: One of our biggest decisions in the early stages of this project was switching cloud providers, from the previously implemented AWS. This switch was one of the first changes we started on, due to the time, complexity and learning curve increases. We would have stuck with AWS, but we believed that switching to Digital Ocean would reduce cost and decrease complexity for maintaining the site.

4.5 Design Analysis

Our cloud developers have begun transitioning from the old cloud service, AWS, to our new alternative, Digital Ocean. We are currently in the processes of moving databases, updating the CI/CD pipeline and implementing new features/protocols that Digital Ocean provides. Digital Ocean has all the capabilities as AWS (needed for the MySTEMGrowth Survey Tool), while making the process more user friendly. While researching this alternative, we also estimated we would be able to reduce the cost by switching clouds, as AWS charges more, due to its extensive feature options, most of which would not be utilized by the survey tool.

Another early focus has been security. Our cybersecurity developers have been testing misconfigurations in the current state of the survey tool. We have discovered major security vulnerabilities and have already started to implement fixes for them. We are working on adding tokens for user authentication and implementing hashing/salting user passwords. One of the main improvements we intend to complete is securing user data/information. Not only should sensitive account information (email, passwords, etc.) be protected, but also the results of students after completing the survey.

Not much has yet been physically changed and deployed within the website files itself (frontend and backend features), due to the extensive planning we have been conducting. We have been discussing improvements with our clients, to ensure we understand their vision for the product. We have presented a demo UI and a list of features we intend to implement into the current state of the app. Once we have clarification and confirmation, we intend to start implementing the changes and start making changes to the website. We also have been hindered to start writing/implementing code, due to the processes of changing clouds. We wanted to ensure we have the cloud setup before changing the functionality, to ensure the compatibility will work with Digital Ocean.