NorthStar LIS: A Simulated Laboratory Information System

Northern Michigan University, Senior Project in Computer Science, Fall 2025

Billy Block

Introduction

NorthStar LIS is a simulated laboratory information system created for the clinical science department to modernize their current workflow. I built it using Vue, Bootstrap, Express, and MySQL. The website allows specimen requisitions, workcards, and grading all in one area. In this paper, I will provide details on the development process, the issues I encountered, and the technical decisions I made.

This project was originally proposed to me and two fellow students during our CS495, Software Engineering class. The goal was to digitalize the Clinical Science's current workflow for simulating lab procedures, making the process easier for students and professors. They previously had a stack of specimen requisition papers that the professor would fill out and give to students. The students would then fill out a work card, documenting all their tests. Finally, the professor would gather all the papers, grade them, and give the students the results.

Given the time constraints on our Software Engineering class, it was difficult to gather all the requirements, analyze the information, and produce a good product, especially since it was our first time creating a software application. While we were creating the software, we encountered many issues, including slow load times, poor UX decisions, and overall, tkinter was the wrong choice for such a large system. Despite finishing the project and understanding the topics in the course, I contemplated what would have been a better solution for the Clinical Science Department.

Technologies

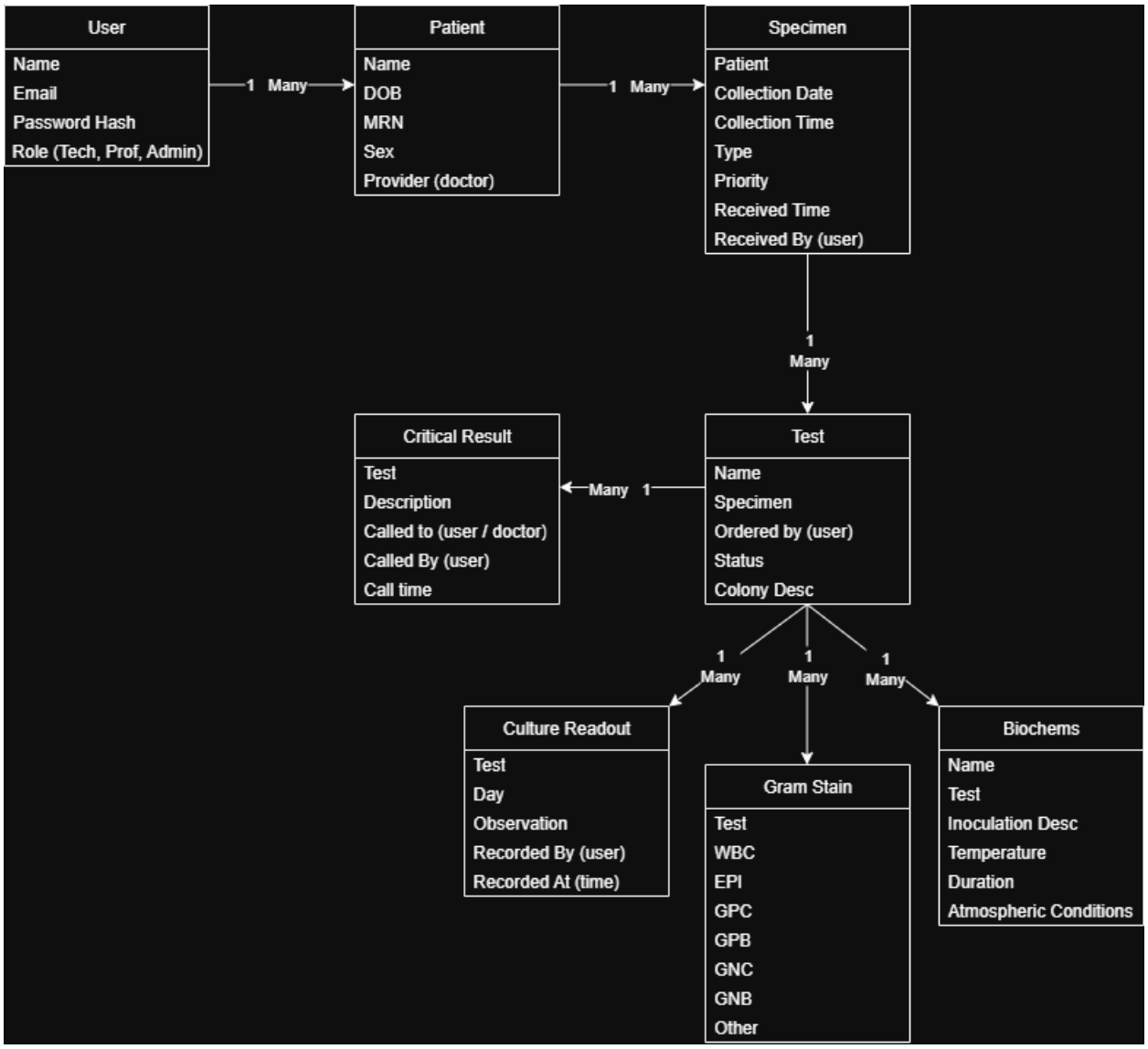
After researching what tech stack to choose, I decided on Vue and Bootstrap for my front-end development. In my sophomore year I took website construction, which gave me a solid foundation for HTML and CSS. CSS has always frustrated me, and I wanted to make styling as seamless as possible; therefore, I ended up choosing Bootstrap for my styling framework. Bootstrap has a cartoonish feel; however, both frameworks proved useful to keep development at a steady pace.

For the backend, I decided on Express and MySQL. With my history of SQL databases and SQLite constraints, considering its lack of booleans and enums, I went with MySQL. Express seemed to be the only JavaScript backend framework that was worth my time researching. Express has a very straightforward approach for HTTP requests, and Express's middleware solves many problems, like authentication. I am more than satisfied with these technologies' performance in the NorthStar LIS project.

Project Approach: Database, MySQL

I started the project by developing the Entity-Relationship Diagram for my database. I initially thought this was going to be a good idea since everything is based on how each entity interacts with the others. However, I found myself later configuring other parts of my code to force these relationships. If I were to approach this project again, I would have developed it from

the front-end.



After creating the diagram, I developed scripts to make the database and tables. Later, I produced more scripts to seed the database with patient information that we used in our previous project. This was more than helpful during development, and creating the backend and frontend

was much easier with these seeds. However, the seeds became obsolete as I got closer to production and were later replaced by an entirely new patient system.

By far the most subtle but powerful part of the database was the ability to cascade delete tables whenever deleting a primary table. Originally, when deleting a table, it would fail on a foreign key constraint. My delete methods ended up being incredibly long because I had to delete every table that depended on the primary table. After learning about “on delete cascade”, it made the methods essentially one line long and much faster.

Project Approach: Express

After configuring the database, I started developing the backend using Express. For every request the database might have, I bound a function to its corresponding HTTP request. For example, if the user wanted to get all of the patients in the database, there would need to be a “patients” route that handled the GET request for all patients. When I initially set this up, I made a one-to-one correspondence from HTTP requests to database queries. This proved unhelpful because there are many tables you wouldn’t make HTTP requests for. After some refactoring of the backend, I instead handled the HTTP requests from a front-end point of view, only handling those that are necessary.

While most of the backend seemed simple, there were definitely some more complex

portions of the system. For example, authentication and authorization were two of the most important goals of this project, and I approached these two points carefully. The main issue I needed to address was students being able to access professor or administrative-level pages or queries on the database.

The time in which the students would use this system only lasts about 2 weeks, so having an entire day for them to set up accounts individually would take too much time. On NorthStar LIS, the administrator or professor can create users, and there is no other way to access the system unless an account is created for you. I will go into more detail on the front-end solution later; however, on the backend, middleware was by far the most powerful tool to solve this issue. Every time a user makes an HTTP request on a protected resource, the backend checks their role to see if they have access to the resource. This solution only required about 20 lines of code to nearly solve the issue entirely. The only other thing to consider is high-level roles leaving their computer running, allowing someone to perform administrative tasks on their behalf. Therefore, on every login, you are issued a token that expires after an hour. This solved the only other problem with roles and also doesn't require passwords to be ping-ponged for verification after login.

Another important aspect of logins is user passwords. Luckily, JavaScript has extensive libraries for hashing passwords. The internet seemed to agree on Bcrypt being the most robust hasher, so I used their library for password hashing.

I initially followed BCrypts hashing documentation that uses lambda's and promises:

```
bcrypt.genSalt(saltRounds, function(err, salt) {
        bcrypt.hash(myPlaintextPassword, salt, function(err, hash) {
         // Store hash in your password DB.
        });
});"
```

However, I later simplified it using async/await:

```
password_hash = await bcrypt.hash(user.password, saltRounds);
```

Project Approach: Frontend, Authentication and Authorization

Previously, we mentioned the backend's solution to signing up and logging in, given time constraints and potential malicious users. The front end also takes a similar approach, guarding specific pages based on the user's role. In my frontend router, I define public pages such as the login page and "page not found." If the user has not signed in yet, they're always redirected to login. Furthermore, if users do not have a valid token, they get redirected to the login page. Finally, when a student tries to access anything other than patient information, it redirects them to "page not found." With all of these protocols in place, students and non-users cannot access unauthorized pages.

Project Approach: Frontend, Patients page

The patient's structure underwent a major rework in the final weeks of my project. Originally, the patient table showed every patient in the system, with a future of only showing patients that were assigned to a specific user. In the real world, this is how Laboratory Information Systems work. The professor I was working with noted that it would be much easier to grade if every student was given the same 28 patients. The generation of a patient is also an option if the professor finds it useful. This refactoring took quite a bit of time; however, I eventually seeded each new user with a copy of the same 28 patients.

Project Approach: Frontend, Specimen Requisition

Generating a patient was by far the most rewarding feature up to this point. Previously, manually creating a patient record took quite a bit of time. You would have to fill out around 20 text boxes of information, modeling all the questions you are asked whenever you visit the doctor. JavaScript has a library for generating fake data, faker js, that I used to generate fake data. Generation of names, dates, genders, etc, was simple. The only information the faker does not provide is hospital system-related data. For this situation, they provide a function that randomly selects from an array. I originally thought patient generation was going to be used constantly; however, it seems like professors won't use it much. It was still useful for development and was my favorite part of the project up to this point.

Project Approach: Frontend, Work Card

The work card was by far the most important portion of this project. However, I have never taken a biology class, nor have I taken a clinical science course. This made it extremely difficult for me to understand the layout or the purpose of functions inside the work card. I frequently showed my progress to the professor I was working with, Professor Matthys, and they would help me understand what was necessary. I, however, have other responsibilities outside of this project, making it very difficult to get in touch with Professor Matthys promptly.

The biochemical tests were also quite confusing to me. I believe the idea behind it is as follows: each specimen can have multiple colonies, requiring different tests for each colony. The clinical science department does not provide every test, as that would be unreasonable budget-wise; therefore, they only perform certain tests and simulate the other tests. The process of performing the tests is done on every colony for the specimen. On the paper work cards, this process was quite easy as they had an entire page with blank spaces for the student to write, which is not practical on a website. I eventually settled on collapsible sections for the tests that make it much less cluttered.

Project Approach: Frontend, Admin Tools

Admins and professors can create new users in the system manually. However, an idea Professor Kowalczyk had for our previous project was the automatic generation of users. His idea was to drop a file of users, provided by NMU, into a dropbox, and the system would

automatically create all the users. I had to modify this idea slightly for the new project, forcing the professor to also fill out a default password. While this seems unsafe to generate every user with the same password, this can only be performed on student roles, and students have no administrative privileges. This section of code was quickly another favorite of mine as it utilizes JavaScript's functional programming quite well.

```
// Parse CSV on file upload
function loadFile(event) {
	let file = event.target.files[0];
	if (!file) return;
	let reader = new FileReader();
	// When the file is read,
	reader.onload = (event) => {
		// Split on new lines, filter out all the falsey values like null, undefined from empty lines.
		let lines = event.target.result.split(/\r\n/g).filter(Boolean);
		// for each line, break it into parts sperated by a comma, removing quotes as well
		preview.value = lines.map((line) => {
			// format: LastName,"FirstName",username,nmuin
			let parts = line.match(/([^,]+)/g).map((part) => part.replace(/"/g, ""));
			if (parts && parts.length >= 4) {
				return { lastName: parts[0], firstName: parts[1], username: parts[2], nmuin:
			parts[3] };
		}
		return null;
		}).filter(Boolean); // filters out falsey values if a user was input incorrectly in the file
	 };
	reader.readAsText(file);
}
```

Issues and Solutions

Of course, I did run into some technical issues, the biggest of which was SQL errors. If I changed one table ever so slightly, it broke every backend route tied to it. It would also break all

of my seeds that I previously wrote for it. I never found a solution to this either; I just had to manually change every query on the database if I ever had a table change. When I worked on projects in the past, it was actually quite easy to rename every occurrence of a variable or class. However, with SQL, it was very difficult to rename a table.

I was also not knowledgeable about any of the technologies that I used for this project. I spent most of my time reading documentation, watching video implementations, or reading other people's code. When it came to writing my own code, it didn't come naturally to me, as it's not similar to Java or C++. I would consistently get stuck on small things for Vue. For example, emitting and watching changes in a component gave me many troubles. I eventually settled on parent components sending the data to the child component through what they call "props". The child component would then create a local version of that variable, then emit changes back to the parent. This seems unnecessary to me compared to the typical sending messages to objects. For example, having the child component send a message to the parent to update instead of the child component creating its own copy and emitting that copy to the parent.

Conclusion

NorthStar LIS filled in the biggest hole in my knowledge of computer science, that being web development. I previously got to the final interview of an interview process, but they didn't hire me because of my lack of knowledge on React or any front-end technologies in general. I am now confident that I have the skills for most entry-level positions in the computer science

field. Vue and Express are two of the most popular frameworks, and I am excited to apply my knowledge to different projects in the future.

William Block

Phone: 586-623-9910
Email: WJamesBlock@gmail.com
Portfolio: billyblock.github.io

Professional Summary

Passionate Computer Science student with a 3.96 GPA and hands-on experience in IT support, software development, and STEM education. Seeking an entry-level position to contribute technical expertise and grow within a collaborative team.

Education - Northern Michigan University - GPA: 3.96

Bachelor of Science, Computer Science | Minor in Mathematics | Expected: Spring 2026

- **Systems Programming:** Designed C++, Java, Python, and SQL programs, including a checkbook, chess engine, maze solver, shell, and a laboratory information system.
- **Embedded Systems:** Developed a pulse-width modulated temperature controller using AVR assembly, C, and an Arduino.
- **Networking:** Built basic servers on Linux, implementing graph search algorithms for routing.
- **Web Development:** Created full-stack projects using HTML/CSS/JavaScript, Angular, Vue, and Express.
- **Custom Programming Language:** Designed and implemented a compiler with my class for a custom programming language, including a grammar, parser, pretty-printer, and code generator in SmallTalk.

Work Experience

Software Support Assistant - NMU, Technology Support Services, Helpdesk
August 2024 - Present

- Imaged and deployed 100+ computers via PXE boot and NMU's custom imaging system.
- Delivered technical support in person, by phone, and remotely (via TeamViewer).
- Installed and maintained licensed software through NMU's internal repository.
- Performed maintenance on laptops, fully disassembling motherboards for cleaning and repair.
- Assisted in a campus-wide desktop replacement project, deploying and configuring 100+ new workstations for classrooms, labs, and offices.
- Maintained detailed documentation of IT procedures and troubleshooting steps using Confluence.

Instructor - iD Tech
March 2023 - March 2025

- Taught programming and game design to over 1,000 students worldwide in Java, C++, and Lua, using Godot, GDevelop, Roblox Studio, and text-based code editors.
- Led more than 100 projects ranging from beginner to advanced, encouraging creativity and first principles problem solving.
- Adapted teaching style to accommodate diverse learning backgrounds and skill levels.

Personal Projects

- **Internet Performance Tester (Python)**: Developed a diagnostic tool that measures packet loss, latency, bandwidth, and detects access point switching.
- **NBA Salary Web Scraper (Python)**: Built a web scraper to analyze and visualize relationships between player nationality and salary.