WebMap (ugrad_SD_7)

Team: Aren Ashlock, Braeden Hegarty, Michael Gelineau, Seth Watgen, Subham Bhattacharya

Client: Nitin Gadia

Course: COM S 4020C

Instructor: Simanta Mitra

TAs: Mohammed Rahman, Yonas Sium

Requirements

We began the semester with a few different requirements or criteria goals from our client. There were some major issues that needed to be addressed following the completion of development from the previous semester's group. Other requirements that were requested included new features such as layers outside of groups, and the ability to split the website into multiple instances, each of which could be specifically tailored to a certain need.

We began by working on requirement number one. This took the brunt of our time developing, with all developers working through a list of notable bugs. Originally, we had hoped to get these completed within 2-3 weeks, but some of them proved to be rather messy. Overall, it took from week 2 (around when development actually began) up until spring break (week 8) to complete an acceptable number of the bugs. We say acceptable because we did not end up getting through the entire list, focusing on higher priority bugs, and fixing regression bugs as they arose with further development.

Once we had the website to a point that our client found it suitable, he requested we change our focus to helping him learn about the structure, and specifically, splitting the instances of the website and implementing documented steps for how he could do so as well. There was significant research focus and development done to take the existing structure of the website and host it on a separate index page, with access to a new database connection. Through completing this, we had to figure out how to split the Prisma schemas and how to deal with the API routes being shared among the components on different pages.

At the same time, the development was being done to complete the above, there was an effort put into adding the ability for users to draw layers. One of the requirements from our client was that the users of the website have existing GIS functionality from multiple other software, in one hosted site. The previous group had already implemented the ability to link layer tilesets from existing GIS data, and he wanted us to add the ability to draw and save our own tilesets, which could display for any authenticated user of the website.

Finally, after everything above had been completed, we were asked to form detailed documentation for the website, including the old code which was handed off without any documents, and for the new features and abilities we researched and implemented. We have since formed a library of documents laying out specific steps to recreate any of the platform hosted development we completed such as splitting instances, a video detailing all website features and including a code-walk of where each of those features or routes are, and also increased inline documentation for future developers, so they can have easier logical references to the logic in methods as they are working on them.

Design

The Webmap project is architected as a modern, modular web application built using Next.js with TypeScript and integrated with Mapbox GL JS for mapping, Prisma for database access, and MongoDB as the database backend. Our goal was to create a scalable, extensible platform that allows for dynamic map interaction, drawing tools, and instance-based deployment—all without sacrificing performance or user experience

Application Architecture

At the core, the application is structured as a monorepo-style Next.js app, with dynamic routing and server-side API routes for interacting with the database. The frontend and backend share the same codebase, and the app is deployed using Vercel, which simplifies builds and supports scalable hosting.

Key technologies include:

- **Next.js + TypeScript**: Offers server-side rendering and static site generation for optimal performance.
- **Mapbox GL JS**: Handles all map rendering, interactivity, and layer control.
- **Prisma**: Interfaces with MongoDB to manage persistent data for maps, layers, and drawn features.
- **Tailwind CSS**: Used throughout the application for responsive, utility-first styling.

Layer and Drawing System

The layer system is designed to accommodate both predefined GIS tile layers and user-drawn features. Tile layers are configured using a combination of JSON schema and API-managed documents. Users can toggle layers on and off, zoom to an extent, and interact with map features like lot boundaries and historical overlays.

We extended this design by implementing a custom drawing tool using *@mapbox/mapbox-gl-draw*, enabling users to:

- Draw polygons, lines, and markers.
- Edit or delete drawn features.
- Save drawings to persistent storage with metadata like source labels and layer names.
- View and toggle these new layers after submission and refresh.

This functionality was integrated cleanly into the existing map UI and supports both group-based and standalone layers.

API Design

The API is built using Next.js API routes and Prisma's ORM abstraction. Core endpoints manage:

- **User-authenticated layer creation** and saving of drawings.
- **Layer retrieval and toggling** via RESTful GET and POST endpoints.
- **Instance separation**, allowing each subdomain to point to a different Prisma schema and MongoDB collection.

This modular API design allows clients to configure multiple maps from a single codebase, while still supporting full CRUD operations on layer data.

Instance Separation

To support different historical or geographical datasets, we implemented an instance-splitting architecture. Each instance:

- Runs from a unique entry point/index page.
- Connects to a dedicated MongoDB collection and Prisma schema.
- Can be configured independently using *.env* files and schema-specific deployments.

This flexibility empowers the client to generate new maps for other regions (e.g., North America vs. Manhattan) without engineering support.

Work Done

Aren:
New technologies - Next.js, TypeScript, Prisma, MapBox API, Vercel, Package.json scripting.
Bug Fixes - Changed icons from links to globes, Hardcoded the "Zoom to World" button, Helped figure out layers not loading, Optimized data loading to reduce crashing of the site, Worked on the hash parameters to get them to update and auto zoom to a specified level, Reduced the lag of the timeline slider, Fixed the styling of the info modals, Made it so certain sections opened by default.
Separate Instances - Restructured the project to support multiple instances on the same domain, Made an instructions document, Modified the prisma schema, Recorded a video showing how to make a blank instance, Assisted the client with their first attempt to make a blank instance.

Braeden:
New technologies - Next.js, TypeScript, Prisma, MapBox API, GeoJSON
Bug Fixes - CSS Styling, layers not appearing, Screen crash, Layers not having zoom info, Zoom settings not being accessible for standalone layers

New Features - Standalone layer implementation, backend work for drawing functionality, implementing editability of drawn layers with existing api routes and code

Michael:
New technologies - Next.js, TypeScript, Prisma, MongoDB, MapBox API.
Bug Fixes - Enable by default function, Lot events displayed by default, Lot Event render issue, Layers not appearing, White screen crash, Zoom buttons for layers not appearing in groups, Enable by default not displaying as standalone layer.
New Features - CSS changes and functionality identical to original site (nahc-mapping), Initial Drawing research, New layer dialog, Drawing features on new layer front-end development.

Seth: Worked on various bugs and assisted some with the drawing features.

Subham:
New technologies - Next.js, TypeScript, Prisma, MapBox API, MongoDB.
*Bug Fixe*s - Slider hold feature where I changed how the data loads, info box scrolling (made it like a list instead of a single box), made data load faster in the northamerica/landowner page (modified the api calls before data was loading at an extremely slow speed it was taking few minutes for data to load, I changed it so it loads in a few seconds), info box appearance, CSS styling adjustments.
New Features - CSS changes and added functionality to the MENY site, added markers and polygons to the initial drawing page as shown during demo 2. Also worked on some documentation.

Results

The initial expectation for this project was to fix any bugs left over from the previous group assigned to this site, and then add in drawing features for the New York Map. Our group accomplished this and much more.

The initial approach we took to handling bug fixes was an “all hands on deck” approach, in which all devs would grab a bug to work on and, when completed, grab a new one or assist a developer with their bug. In order to manage this, we used an issue tracking software called JIRA, which allowed us to assign point values, determining how long we expect this bug will take to be completed. Using this method, we were able to fix recurring crashes, implement standalone layers, fix Lot Events appearing by default, as well as many more.

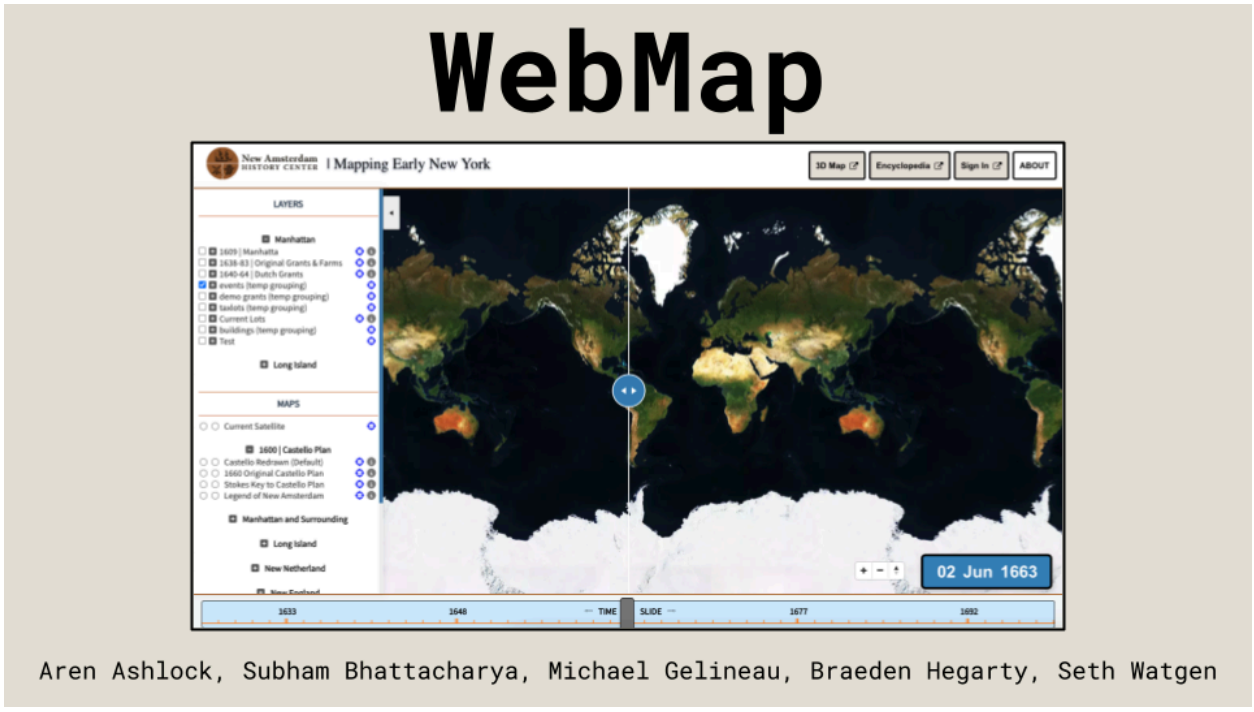
After this initial bug fix development phase, our client wanted us to shift to integrating additional maps and subdomains into the site. This required lots of research into our project in determining which components were needed to replicate this map and allow new data to be loaded onto a

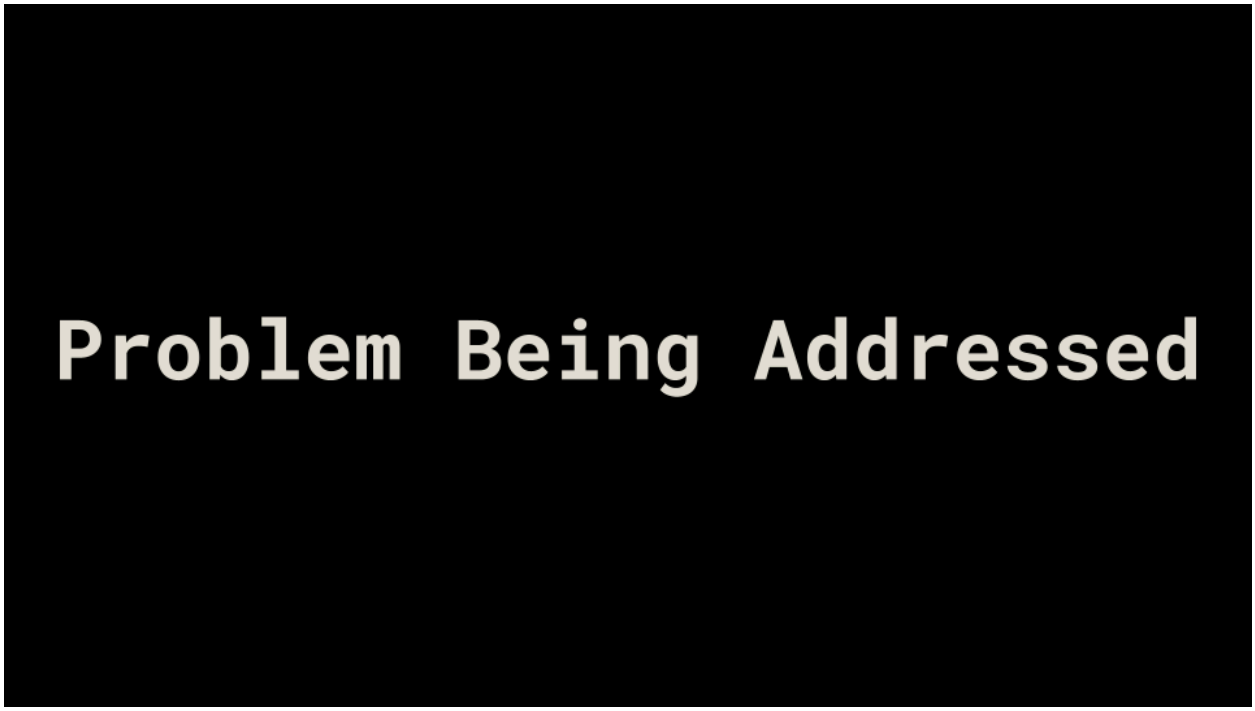
blank map for our client. We ended up creating lots of documentation for implementing these new sub-domains, as well as a video showcasing us creating a new sub-domain. Our client was then able to create new sub-domains without the help of developers.
Finally, we set out to implement the drawing features into the New York portion of the site. Using either the standalone layer or the new layer within a group, a user is able to draw polygons, lines, and markers and edit/delete them as they please. After choosing to submit, they will be prompted with a few more editing options for these drawings, and finally submit after adding in a label, sourceId, and source label. These new layers can then be seen after a refresh.

The initial goal for this project was to add drawing features to the site, however, we managed to accomplish this and much more. From reinforcing the logic behind the site, preventing crashes and layers from failing to load; to creating new sub-domains as well as documentation to describe how our client can repeat this process; finally to adding new drawing features to create new layers, we were able to make significant progress on this site and create an easier transition to any new developers that might take on this project.

Appendix

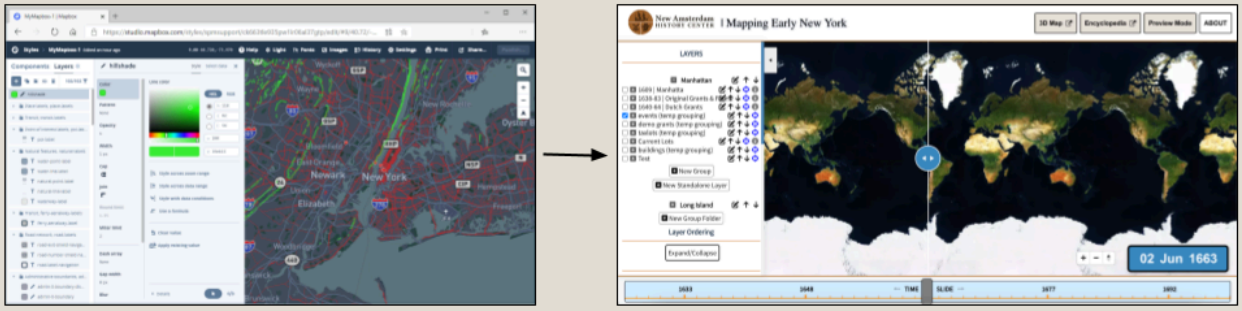
Demo 1 Slides





What is Our Project Solving?

- There aren't many options for creating interactive maps
    - Google My Maps, MapCreator.io
- MapStructor emphasizes customization
- Make map creation simpler



Basic Features

- Choose an area to start building
- Import or draw map layers
- Share map with others and collaborate

Development Challenges and Choices

Challenges

- Discovering wants of client
- Catching up to previous group
- Picking up an existing project
    - Understanding structure, code, and functionality
    - Utilizing their code to fix bugs quick
- Working with MapBox GIS
- Communicating limitations/capabilities of project with the client

Choices

- Spend time learning about the site and its functionality
    - Both new and old sites
    - Watching videos and talking with the client
- Begin working on bugs and assisting other developers
    - When done with one, pick up another
    - "All hands on deck" approach to identifying issues
- Keep an honest yet driven schedule with our client

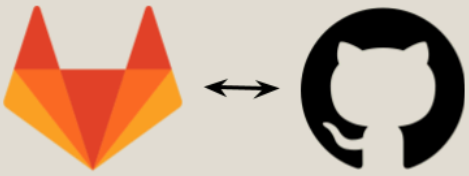
Software Development Practices and Tools

Programming language and framework

Database and APIs

Practices and Tools

- Agile development
- Require a minimum of 1 peer review before every merge
- Mirroring GitLab to GitHub
- JIRA Ticket Software
- MapBox API



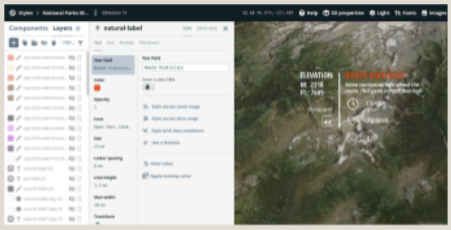
Current Progress and Challenges

Current Progress

- Understand the code better
    - New MENY
    - Basic Map Editor
- Initial bugs are mostly fixed
    - Keep discovering more bugs

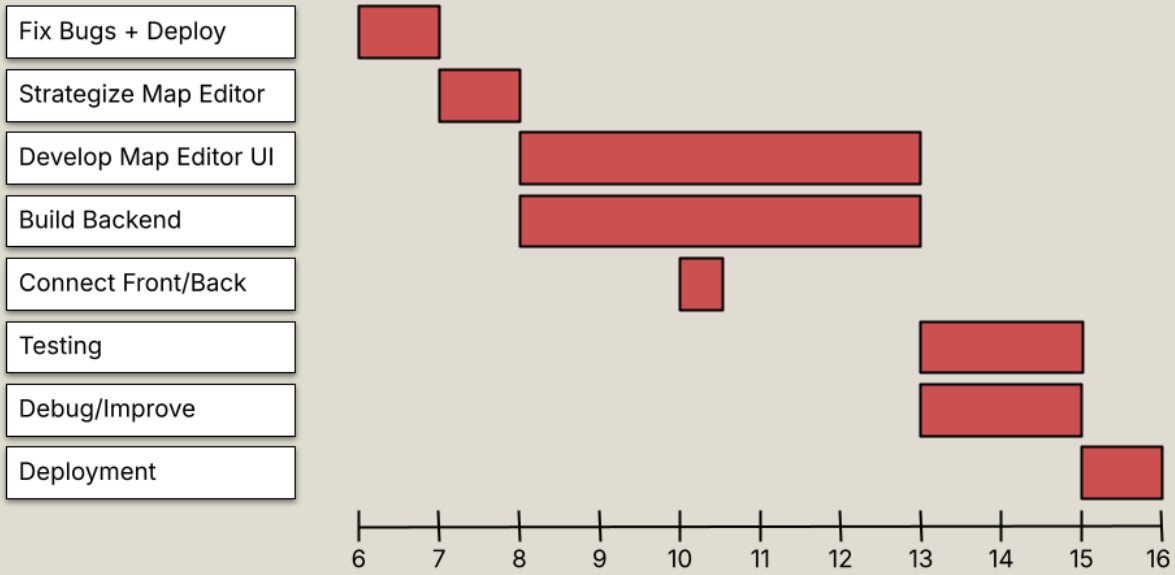
Challenges

- Learning a new language (TypeScript)
- Lack of documentation in code
- Understanding MapBox GIS
- Discovering more bugs impedes the development timeline
- Application crashes when launched with VPN



Remaining Work Timeline

Timeline



Demo!

Demo 2 Slides

WebMap

Aren Ashlock, Subham Bhattacharya, Michael Gelineau, Braeden Hegarty, Seth Watgen

Our Project

Recap

- Map Creation Site (maptructor.org) - Mapping New York
- Goal #1 - Build a stronger foundation for site
  - Fix bugs
  - Improve UI
  - Complete features
- Goal #2 - Replace old implementation of the site

(Original site)

(Development site)

Current Goals

- Fix remaining bugs
- Make cosmetics/interactions identical

← Up until 1.5 weeks ago

- Separate instances for different maps
- Create feature to draw maps
- Add accounts
- Allow users to select a map

← Past 1.5 weeks and on

Design and Implementation

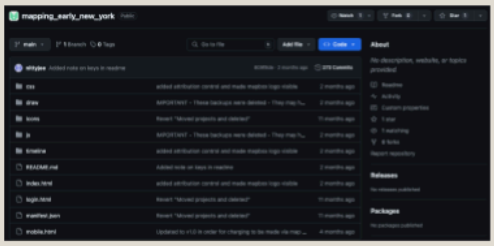
Fixing the Bugs

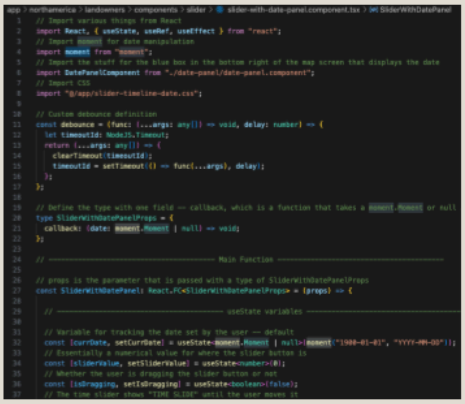
How we solved them:

- Work together to learn interactions
- Comment the old code (and refactor)
- Check the console log for "breakpoints"

Challenges:

- Understanding the API
- Past design choices make them more difficult





Same Interactivity

Resources to help resolve:

- Tab between sites to notice differences
- Using the old site's code as reference

Challenges:

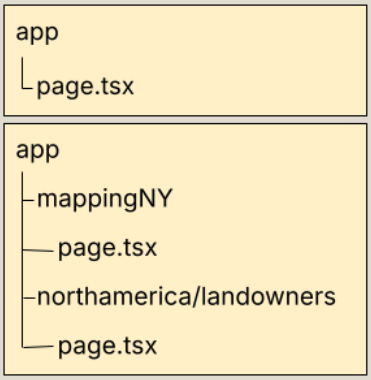
- Many layers of CSS building
- Timing, sizing, colors had to match
- Dynamic elements depending on window size

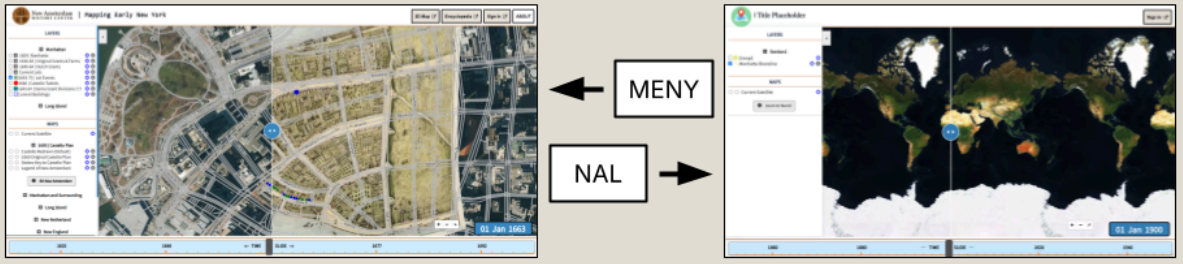
Separate Instances

Client: Wanted different domain paths

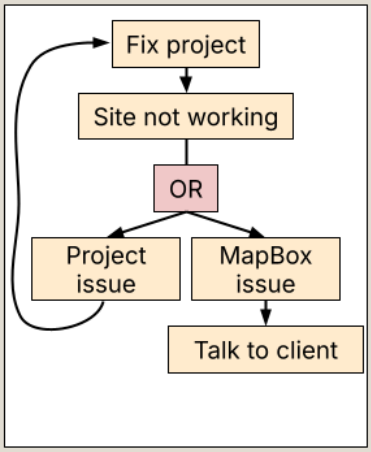
Solution: Restructure the code

- Determine what code needed to be separate
- Build new backend (and generate prisma)
- Document process for future implementations





Separate Instances



Challenges:

- Next.js API fetches are specific
- Incorporating multiple environment variables
- Building different prisma schema
- Ensuring project is functional on Vercel
- Testing locally vs deployment
- Understanding if issues are MapBox or code
- Lots of little changes needed

Prisma Issues

Drawing

Client: Ability to draw maps using tools

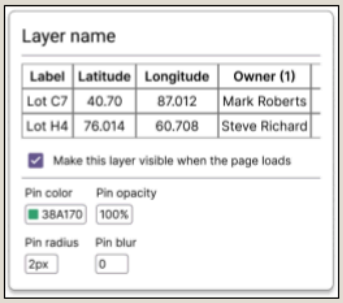
What we have learned so far:

- Development page for research/experimenting with new drawing features
- Ability to add markers, lines, polygons

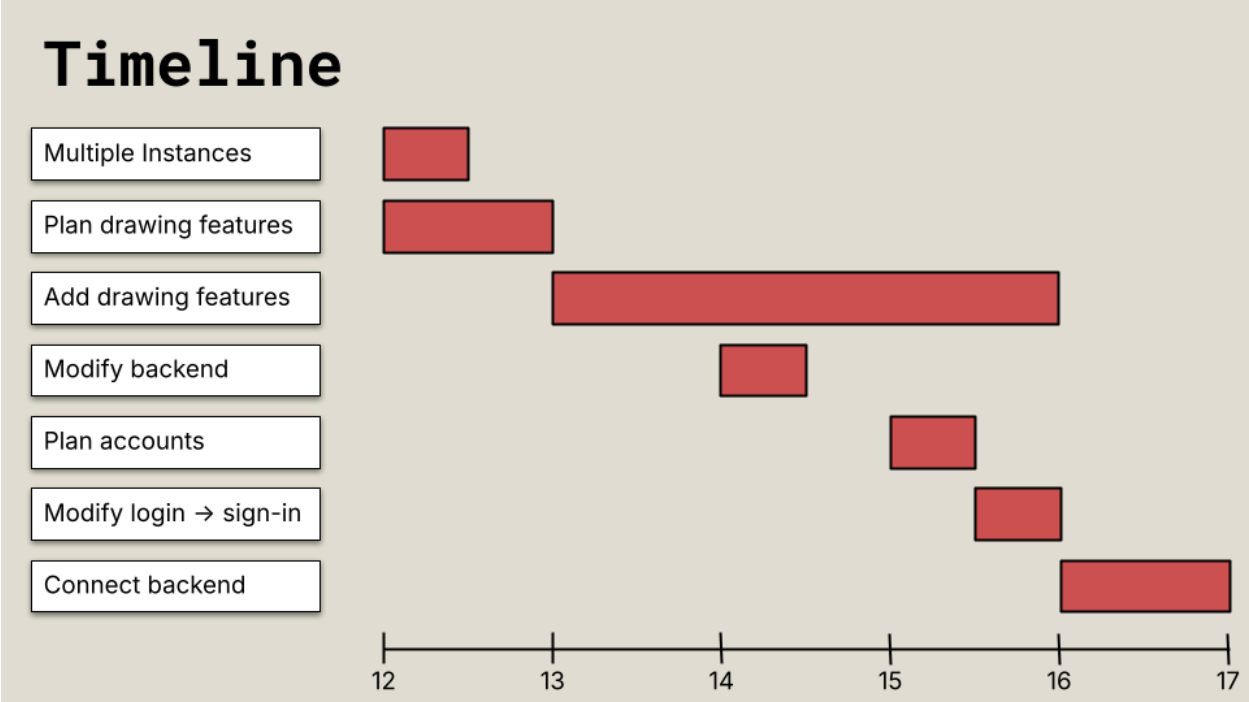
Drawing

Early thoughts:

- Create a new collection
  - Just stores drawing information
  - GeoJSON
- Connect user accounts to drawing
  - Already implemented, just expand
  - Include registration
  - Only let creators see data they make

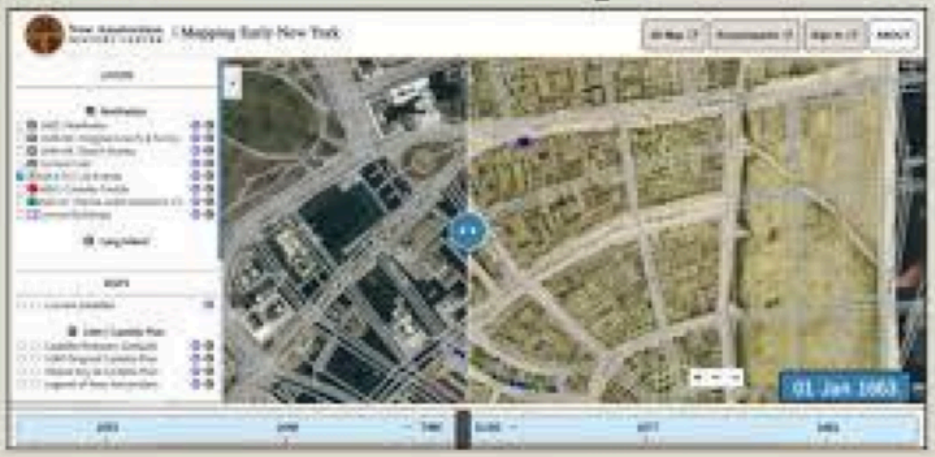


Remaining Work Timeline


Timeline
Multiple Instances
Plan drawing features
Add drawing features
Modify backend
Plan accounts
Modify login → sign-in
Connect backend
12
13
14
15
16
17

Demo!

WebMap



Aren Ashlock, Subham Bhattacharya, Michael Gelineau, Braeden Hegarty, Seth Watgen

https://youtu.be/P3KrNbE0vMc

Demo 3 Slides

WebMap

Aren Ashlock, Subham Bhattacharya, Michael Gelineau, Braeden Hegarty, Seth Watgen

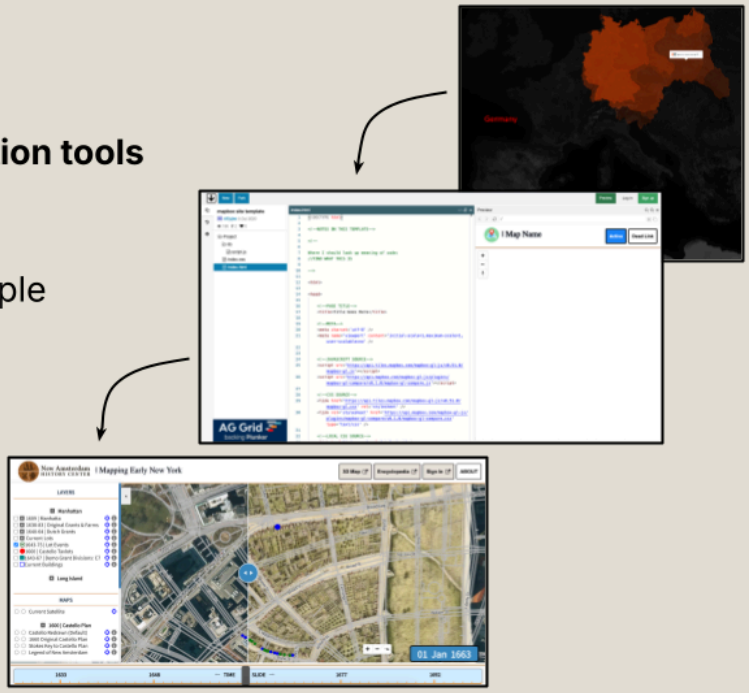
Problem Being Addressed

Problem

Lack of existing map creation tools

Current process:

1. Create map using multiple software
2. Start a website
3. Upload data via code
4. Make data look good



Our Solution

Streamline the Process

Goal:

1. Have our existing site
2. Users can add data
3. Users can draw directly on map
4. Share/work on maps with others

Design Challenges/Choices

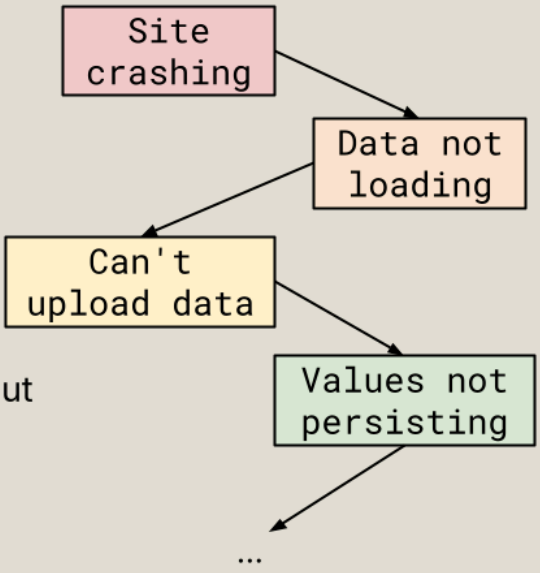
Challenge - Buggy Code

Issues:

- Existing bugs affected development
- Solving certain bugs would reveal others

Solution:

- "All hands on deck" approach
- Consistent feedback from client about functionality



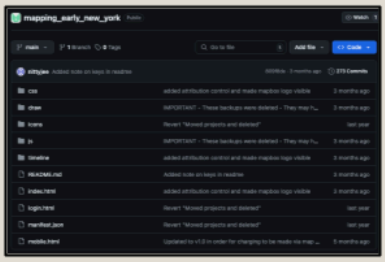
Challenge - MapBox API

Issues:

- Unsure what properties of the API mean
- Documentation is difficult to understand

Solution:

- Refer back to old code to understand the API
- Incorporate code written by client's colleagues



Existing code

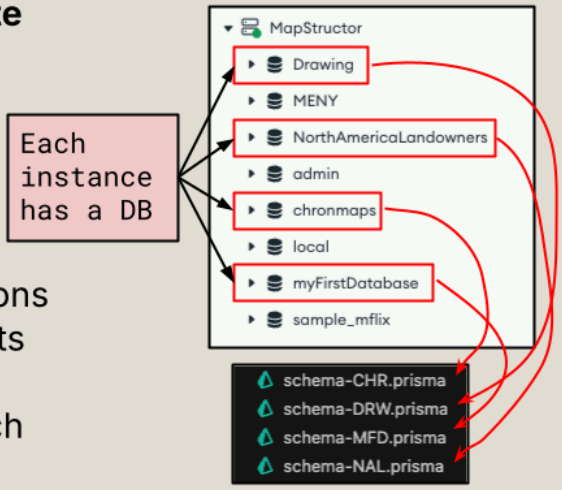
Choice - Multiple DBs

Choice: Separate instances = separate databases

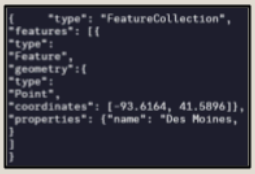
- Environment variables include database connection URIs

Solution: Split the prisma schema

- Allows multiple database connections
- Flexibility to change schema objects per page
- Modify build script to generate each time

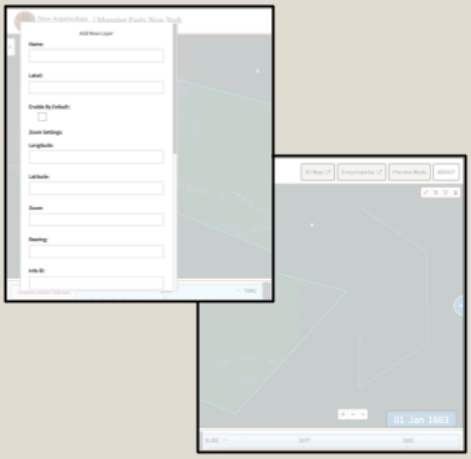


Choice - Draw Feature



Choice: How to implement and get data

- Data type is GeoJSON rather than vector



Solution: Utilize existing UI

- Minimizes the collections in database
- Keep data similar
- Info box is already functional
- Able to tie drawing info to existing properties

Software Development Practices and Tools

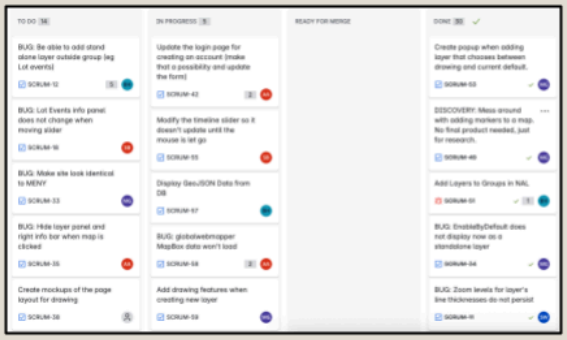
Programming language and framework

Database and APIs

Tools

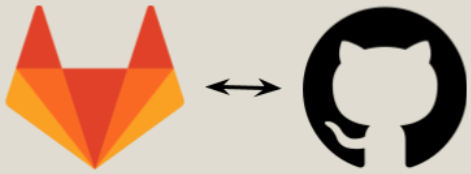
Jira:

- To-Do → In Progress → Ready to Merge → Done
- Point estimates



GitLab → GitHub:

- Mirror repositories
- Client had a public repository beforehand



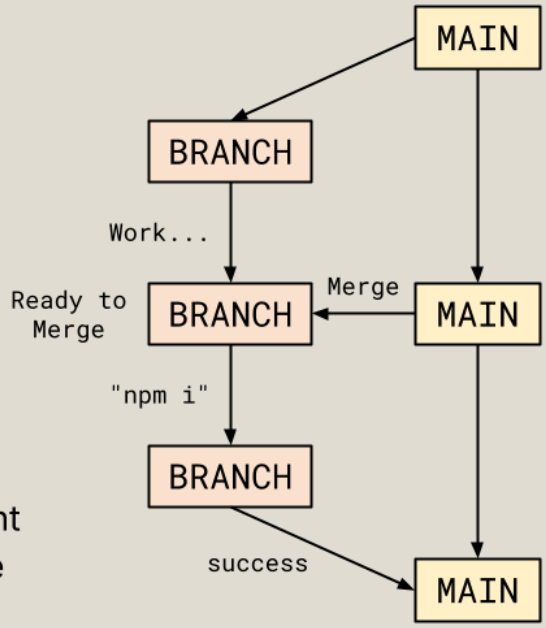
Practices

SCRUM:

- Daily standups
- Demo = sprint

Git:

- MASTER branch as archive
- Branch per feature/bug
- Merge main into branch first
- Run "npm i" to ensure deployment
- Minimum of 1 reviewer per merge



All We Did

Bugs/Work From The Last Group

Bugs:

- Site doesn't crash
- Data loading/manipulation is functional and more reliable
- Styling and interactions are more similar

Standalone Layers:

- New feature left from last group
- Layers can exist outside of groups

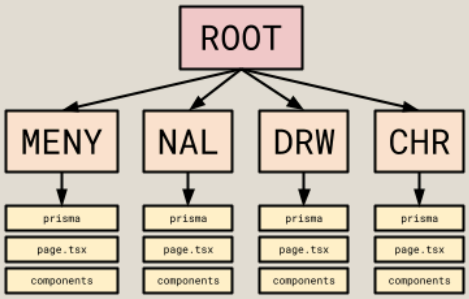
Original site

Our site

Making The Project Cloneable

Project Restructuring:

- Determined what files/folders need separated
- Modified prisma loading and build scripts



Documentation:

- Steps to follow
- Video demonstration
- Assisting client with their attempt

YouTube link for a follow-along tutorial: https://youtu.be/9ur1xzv-6Zc

Making a New Database

Creating a Database in MongoDB:

1. Log into MongoDB
2. Connect to the MapDirector connection
3. On the connection, click the "plus" button
4. Give your database a meaningful name and set it as both the Database Name and Collection Name:
5. Create collections (by clicking on the "plus" button on the new database you just created) so that the database has the following collections
   - BaseLink
   - LayerData
   - LayerGroup
   - LayerSection
   - Map
   - MapFloorGroup
   - MapFloorItem
   - MapGroup
   - User
   - ViewOrderCounter
   - ZoomLabel

Connecting the Database to the Project (.env file):

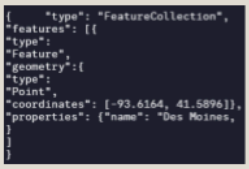
1. If you have a .env file in your project, add the following to it
   DBNAME DATABASE_URL="..."
   (replace with the name of the database from **Creating a Database in MongoDB** - Step 4)

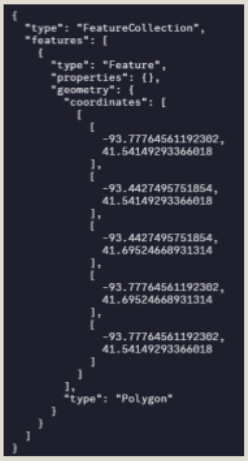
Drawing Functionality (Pt 1)

- Initial experimentation
  - Separate page dedicated to learning how to implement drawing features
- Transition to main site
  - Adjust current map instance so it can be shared across React components
    - Shared MapContext file allows us to share the Mapbox instances
    - Wrap HTML element of the page.tsx file with <MapContext.Provider> tag to allow any component nested in this tag to access the map
  - Access MapContext through layer components to allow drawings
  - Turn drawings into GeoJSON data for submission

Drawing Functionality (Pt 2)

- Experimenting with MapBox Adding GeoJSON as Source
  - Discovered existing API methods worked for JSON in correct format
- Research GeoJSON format to ensure proper data read
  - GeoJSON Files have numerous possible data types and structures
  - Built out data format files and passed them to the frontend devs once I was able to display from the database
- Integrate GeoJSON style layers into our existing edit form
  - Used existing API routes
  - Once the layer is drawn, the type and GeoJSON data are auto-set in form
  - Code parses the layer 'link' as a JSON object to display the layer on the frontend





What We Are Most Proud Of...

1. Fixed some VERY MAJOR issues with the website

---

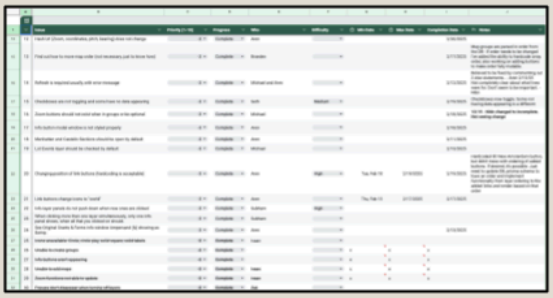
2. Added the drawing features as a starting point

Challenges Encountered

Challenges Encountered

Lack of handoff from last group:

- Hard to understand what they did
- Bugs were more difficult → took longer
- Features were left unimplemented (standalone layers)



Technical unfamiliarity:

- Next.js project structure
- Mapbox (API & map creating)

Contributions

Aren: Bugs - Site crashing, Layers not loading, Hash parameters (url, initial zoom), Timeline slider lag, Page styling (buttons, modals, sections open by default); Project Restructuring - Whole thing (restructure, documentation)

Braeden: Bugs - Standalone layer implementation, CSS adjustments (page styling), Documentation, Layers not appearing; Drawing Features - Research backend adjustments, GeoJSON formatting, integrate drawn layers into existing API methods

Michael: Bugs - Enable by default function, Lot events displayed by default, Lot Event render issue, Layers not appearing, White screen crash, Zoom buttons for layers not appearing in groups, Enable by default not displaying as standalone layer

New Features - CSS changes and functionality identical to original site (nahc-mapping), Initial Drawing research New layer dialog, Drawing features on new layer front end development.

Seth: Bugs - various tasks; Drawing Features - various tasks

Subham: Bugs - Slider hold feature, info box scroll, made data load faster in the northamerica/landowner page, info box appearnce, CSS adjustments, Documentation; Drawing Features - Research, Added markers and polygons.