CSCI 370 Final Report

The Foundation

Tyler Collingridge
Mark Duffalo
Avie Burris
Ryan Jones
Revised May 23, 2023

CSCI 370 Summer 2025

Prof. Donna Bodeau

Table 1: Revision history

This will be updated with each submission

Revision	Date	Comments
New	May 15, 2025	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done I. Team Profile References Appendix A – Key Terms
Rev - 2	May 22, 2025	Updated Sections Completed: VI: System Architecture

Rev - 3	May 29, 2025	Updated Sections: Completed: VII. Software Testing and Quality (excluding test results) VIII. Ethical Consideration
Rev - 4	June 9, 2025	Updated Sections: VII. Software Testing and Quality (excluding test results) VIII. Ethical Consideration Appendix A: Key Terms Completed: VII. Software Testing and Quality (results) IX. Results X. Future Work XI. Lessons Learned
Rev - 5	June 10, 2025	Updated Sections (updated based on recommendations from other groups for paper swap): IV. Risks V. Definition of Done VI. System Architecture VII. Software Testing and Quality Appendix A

Rev - 6	June 13, 2025	Updated Sections: Completed: XII. Acknowledgements

Table of Contents

I. Introduction

The overall scope of this project is to design a custom web application for Infinity Concrete to replace the existing process for manually ordering, dispatching, and tracking concrete deliveries to job sites. Instead of relying on phone calls and edits to a spreadsheet in Excel, field workers and dispatch will be able to coordinate concrete pours through this web-based system and utilize a dynamic database of orders.

The client, Infinity Concrete, has a major focus on modernizing the concrete industry through use of technology and software to improve efficiency within the company. This web application will eliminate the process of calling in to dispatch from the field and the complex process of ordering concrete and verifying that orders have been placed, have been delivered to the site,

and satisfied the required amount of concrete for the project. Infinity Concrete is based in New Braunfels, Texas, and aims to become a leading member of the San Antonio concrete industry.

Currently, Infinity Concrete relies on Excel spreadsheets and phone calls between dispatch and foreman to coordinate concrete pours. This project aims to build a more efficient and structured process through a centralized web-based system via both mobile devices and desktop browsers.

The web application will use data from the existing Excel spreadsheets of concrete orders to populate the new database associated with the application. This spreadsheet will also be the main source used to format the web application to be able to handle orders.

Infinity Concrete will be the sole user of the software, using it internally to improve efficiency and to perform data analytics for the company's operations. The only other potential user would be Infinity Concrete's suppliers to receive and confirm concrete orders.

The person responsible for maintaining the software would be their software engineer (Jonas Edelstein.)

II. Functional Requirements

Table 2: Functional Requirements

Importance	Size (S, M, L, XL)	Requirement
2.	XL	**Projects Page**
2.	XL	Computer Display
2.	XL	Mobile Display
2.7.	L	Menu Button on Mobile Display
2.2.	L	Individual Project Display with Job Name and Time
2.5.	S	Expand Button for each project
2.6.	M	Expanded Project Display with all information
2.1.	XL	Get Data from DB
2.3.	M	Order projects by time and date
2.4.	S	Add New Button (Projects Page/Input Form)
3.	XL	**Input Form**
3.4	M	Edit/Update/Cancel Projects

3.5	M	Create format depending on project specifics
3.7	S	Add drop-downs and auto-fills
3.6	S	Submit, Update, Cancel order buttons
3.8	S	Add required project specifics so the form cannot be submitted without them
3.9	M	API for addresses
3.3	S	Create New Project Button
3.1	M	Create New Project Input Display
3.2	M	Connect Create new to DB
5	XL	**Calendar Page**
5.1	S	Display current month and days
5.4	S	Highlight today's date
5.2	M	Add projects to their corresponding day of occurrence
5.3	L	Clicking on a day takes the user to the Projects page, filtered to show only projects from that specific day.
1.	XL	**Database**
1.1.	L	Schema Design
1.2.	M	Second DB for dropdowns (suppliers, mixes, etc.)
1.3.	L	Roles (admin, superuser, user)
4.	XL	**Support Features**
4.3	M	Dark/light modes
4.1	S	Spanish Translations Spreadsheet
4.2	M	Apply Spanish Translations to app
4.5	S	Font Size

III. Non-Functional Requirements

Table 3: Non-Functional Requirements

Importance	Size (S,M,L,XL)	Requirement
1.	S	Cost Constraints
2.	L	ADA/Support Feature Compliance

IV. Risks

Table 4: Risks

Risk	Likelihood	Impact	Plan
Database Security	Unlikely	Major	We carefully created the database with security in mind. We vetted our design with Infinity's software engineer.
Database Formatting	Unlikely	Moderate	We carefully designed the schema with longevity in mind.
Language Translation	Unlikely	Major	Utilize Online Translators

V. Definition of Done

The minimal viable product for the project includes a “Projects” page, an input form for adding and updating project information, and a pour form for adding and updating pour information. The system will use a Cosmos NoSQL database containing documents for projects, suppliers, and other relevant data. An “Admin” page will be provided to allow for the editing and maintenance of these tables. A “Settings” page will be added to comply with ADA, translation into Spanish, the ability to change font size, and dark/light mode will be added. We manually tested the application using Swagger UI. Although we did not have time to create a testing suite, we set up all of the necessary functionality for Infinity to set up a testing suite for the backend. Upon completion, the project will be delivered through its GitHub repository, and Infinity Concrete will be notified of its availability via email.

VI. System Architecture

The fundamental foundation of the system architecture is built around a modular, client-server model that separates responsibilities using the model view controller design pattern to ensure scalability, maintainability and usability. The architecture is web-based with a React front-end for user interface (view), a .NET 8 backend for data handling (controller), and a NoSQL

(CosmosDB) database used to handle and store data (model). The NoSQL database is important because it is dynamic and is well suited for unstructured data, which Infinity predicts they will have because concrete Projects and Pours can be unpredictable. Figure 1 demonstrates the group's system architecture.

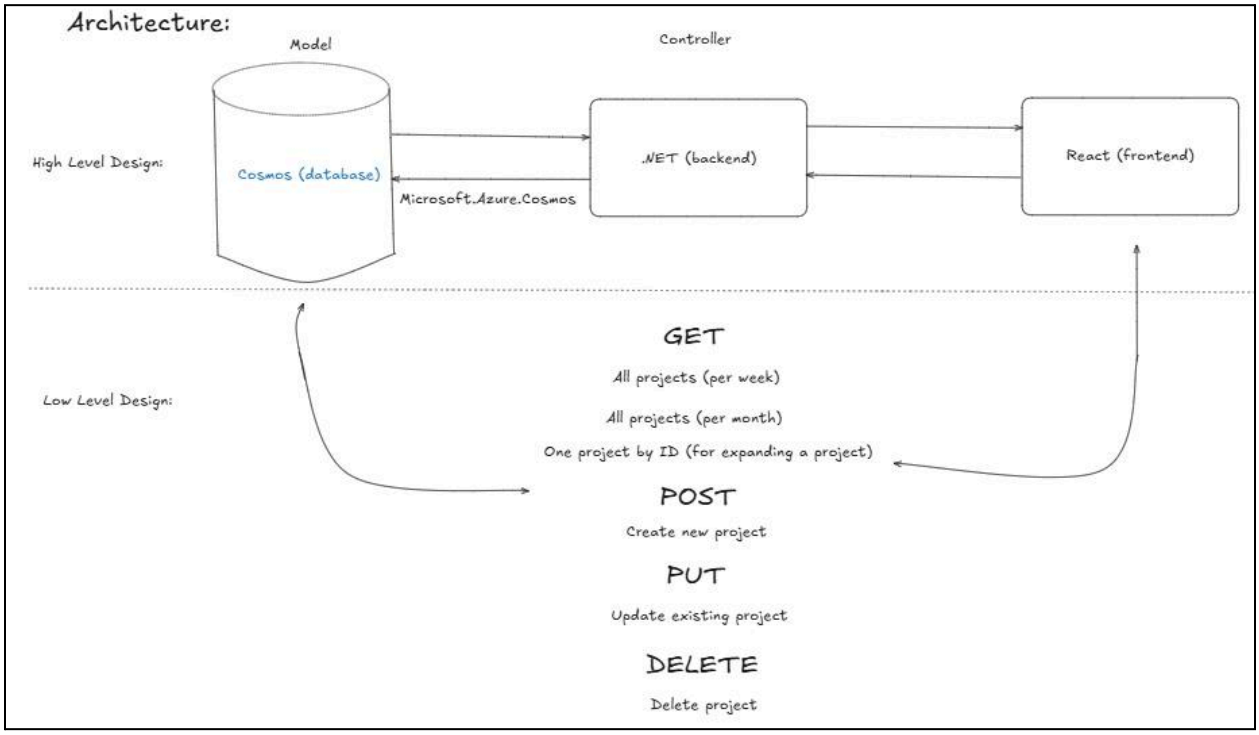

Figure 1: System Architecture

The frontend for the project uses TypeScript for maintainability and is structured using a component-driven architecture. The user will land on the Login page where they log into the web-app and from there get directed to the Projects page. From here they have the ability to switch between pages (Calendar, Settings, Admin). They will also be able to fill out project forms and pour forms from the Projects page. Material UI will be used to provide the design for the UI, this includes buttons, forms, and switches. Additionally the use of ThemeProvider supports dynamic light/dark mode through React. The calendar page utilizes FullCalendar to display project timelines in an interactive calendar. The events for the calendar will be populated by API response. When filling out a Project Form, Google Maps Address Autocomplete, a third party widget, will be used to ensure address accuracy by communicating with Google Maps' API. The addresses will then be stored within the database. Lastly, the use of multi-language support will be handled by i18next to inject translated strings into UI components. Figures 2 through 9 demonstrate the UI for a browser, and Figure 10 demonstrates admin abilities on desktop. Figures 11 through 15 represent the UI for mobile.

Figure 2: Jobs Page Design

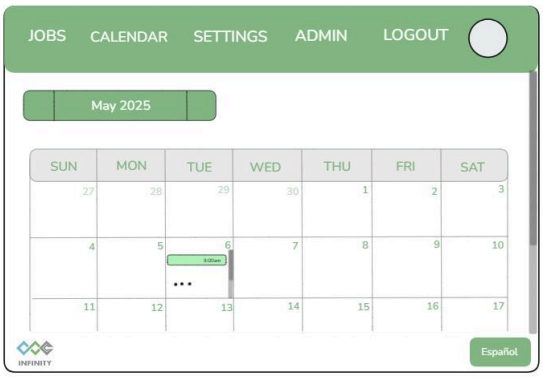

Figure 3: Calendar Page Design

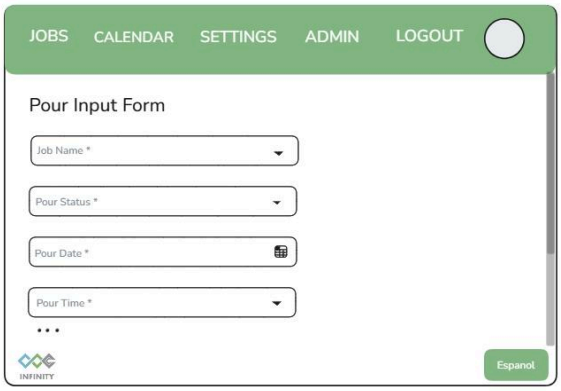

Figure 4: Pour Input Form Page Design

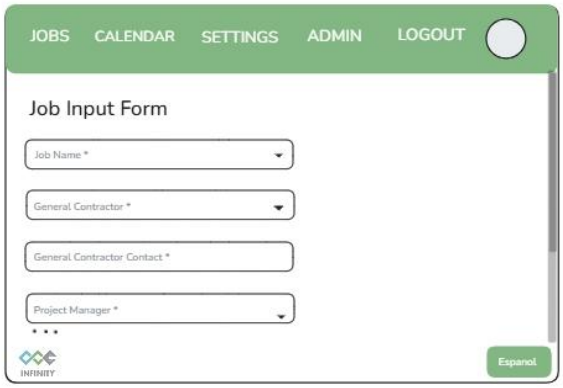

Figure 5: Job Input Form Page Design

Figure 6: Submission Pop-Up Design

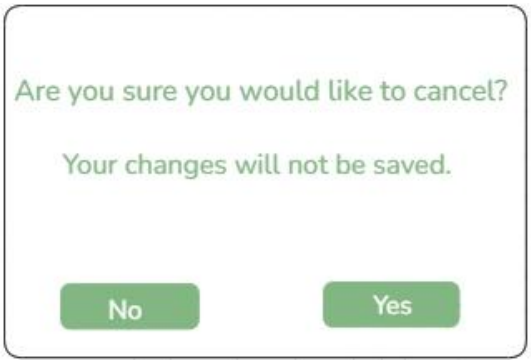

Figure 7: Cancellation Pop-Up Design

Figure 8: Admin Page Design

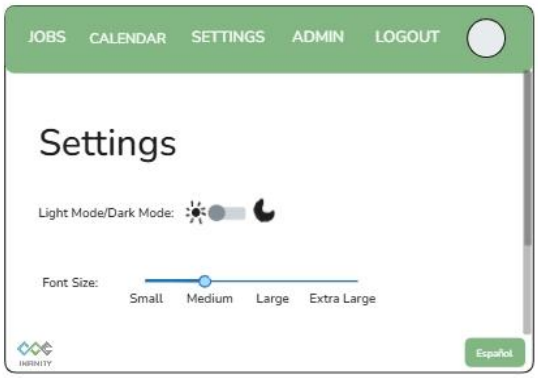

Figure 9: Settings Page Design

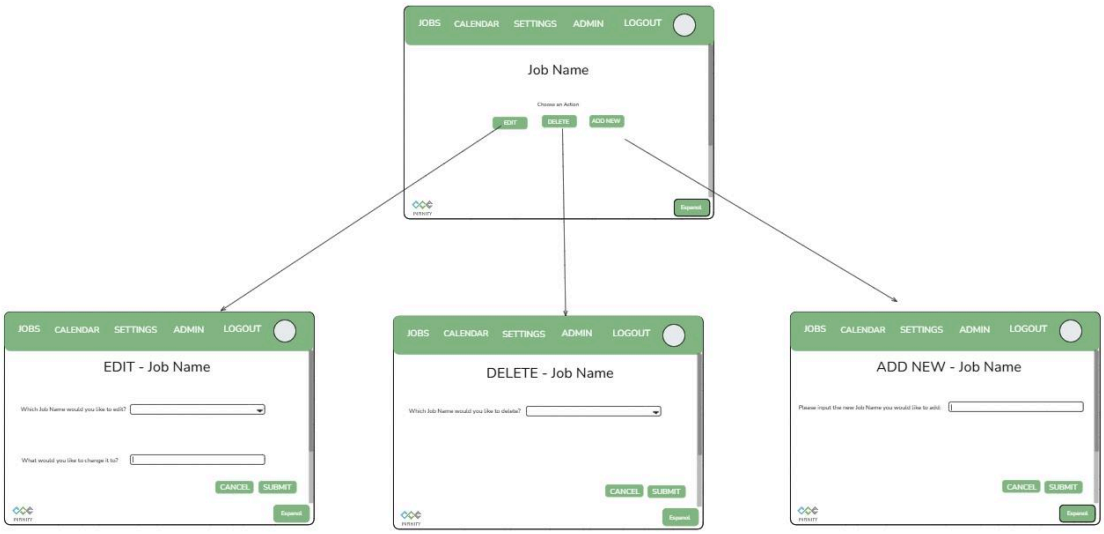

Figure 10: Different Admin Abilities in the browser UI

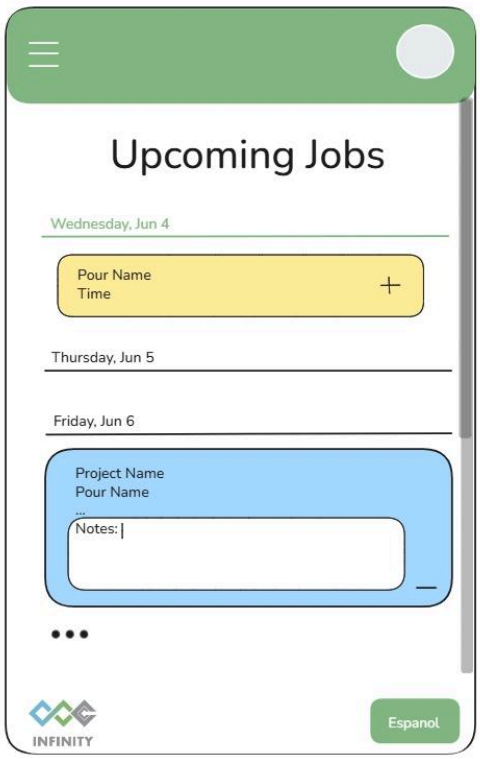

Figure 11: Jobs Page Mobile Design

Figure 12: Calendar Page Mobile Design

Figure 13: Mobile Navigation Bar Design

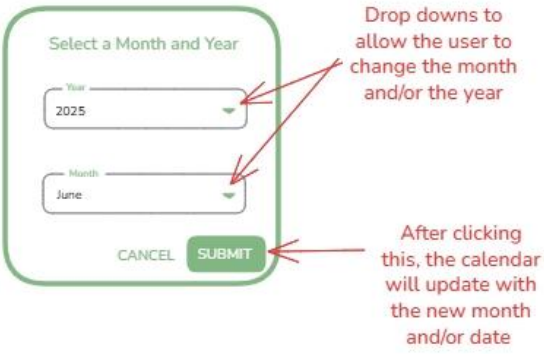

Figure 14: Mobile Month/Year Selector Design

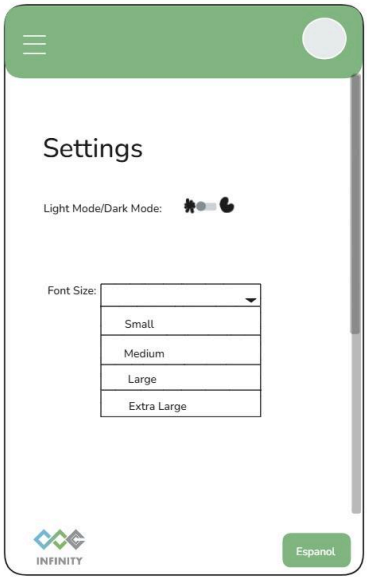

Figure 15: Settings Page on mobile

The backend enforces the data structure, or schema, so that the frontend and backend know how to communicate with each other. This also ensures data consistency and limits data redundancy. The general schema for our application are projects with names, IDs, creation dates, and other information, which are linked to pours which specify additional details. This way, the project data is not repeated for each pour, and the front end can display projects and then display all pours per project when the user needs it.

Following the MVC design pattern, the backend uses controllers to establish API endpoints to handle requests, such as GET requests for retrieving project data, and PUT requests for uploading new project data, and many more. These API endpoints are called by the frontend, the backend retrieves the data from the database, and returns it to the frontend in a pre-established schema so that the frontend knows how to handle it.

At this time, the team is unsure as to what information will be available in the calendar view on browser vs. mobile.

VII. Software Test and Quality

Frontend Testing

Most of the frontend testing is through human interaction and ensuring everything works visually and that there are no errors in the dev tools console.

1. Jobs Page Testing
    a. Purpose:
        i. To ensure all features on the jobs page function properly and align with the definition of done.
    b. Description:
        i. The jobs page needs to have a display of the jobs and pours of the current week at minimum. Jobs and pours need to have an expanded view with all of the proper information displayed. The current time and date should be shown for convenient use. The jobs page should also have access to the new job and new pour input forms.
    c. Tools Utilized:
        i. Human Interaction and Inspect Element Dev Tools
    d. Threshold:
        i. The current date is highlighted and the time bar follows the current time throughout the day. Tiles are properly aligned and formatted on the correct date and at the correct time. Tiles have expanding functionality that displays panels, which should display all of the proper information and involve draggable functionality. Navigation buttons for the new job form and new pour form are located at the top of the page and navigate to the correct pages.
    e. Edge Cases:
        i. Draggable panels register relocation when using the scroll bar on the panel.
        ii. Draggable panels remain on the page when the weeks are navigated.
        iii. Tiles overlap, causing issues with viewing content/expanding content.
    f. Results:
        i. The current date and time are correctly displayed/highlighted.
        ii. Tiles are properly aligned with the correct date and time, but do overlap when multiple occur at the same time.
        iii. Tiles properly expand and the panels function properly with dragging function, correctly displayed information, and an editable notes field.

iv. The panels don't drag when using the scroll bar and automatically collapse when the week is changed so that they are only visible when necessary.
v. The new job and new pour form buttons properly navigate to the proper pages.

2. Calendar Page Testing
    a. Purpose:
        i. To ensure all features on the calendar page function properly and align with the definition of done.
    b. Description:
        i. The full month should be displayed in a calendar format with month/year selection and the current date highlighted. Tiles for each job/pour should be displayed on the proper dates and be expandable in a similar manner to the panels on the jobs page (described in the previous test).
    c. Tools Utilized:
        i. Human Interaction and Inspect Element Dev Tools
    d. Threshold:
        i. The calendar displays properly, defaulting to the current month when loading the page. The current date is highlighted. The month and year are changeable through a selector. The tiles are displayed on the correct days and are expandable into panels. The panels should be draggable and display the full job/pour information.
    e. Edge Cases:
        i. Draggable panels register relocation when using the scroll bar on the panel.
        ii. Draggable panels remain on the page when the weeks are navigated.
        iii. Tiles overlap, causing issues with viewing content/expanding content.
    f. Results:
        i. The tiles and draggable panels do not display as time constraints did not allow for implementation. This should be an easy task and would reuse the components from the jobs page.
        ii. The calendar does display properly and the month/year can be selected.
        iii. The current date is highlighted and the next/previous month buttons properly function.
3. Settings Testing
    a. Purpose:
        i. To ensure all features on the settings page function properly and align with the definition of done. This also includes ensuring that all settings apply to all pages and aspects of the entire application.
    b. Description:
        i. The settings page includes a dark mode/light mode toggle switch and a font size slider. There is also a language button for toggling between English and Spanish that is constant across all pages.
    c. Tools Utilized:

        i. Human Interaction and Inspect Element Dev Tools
    d. Threshold:
        i. The selected dark mode/light mode should apply to all pages across the site and changes in the settings option should be reflected on all pages. The font size slider should apply to all applicable text on all pages and reflect the currently selected option. The language button should apply to all text across the site and maintain the selected option when switching pages.
    e. Edge Cases:
        i. Different font sizes mess up formatting with components on other pages. (Ex: large font size causes overflow in containers)
        ii. The currently selected dark mode/light mode option gets reset when the page is reloaded or the page is changed.
        iii. The font size option gets reset when the page changes or the page reloads.
    f. Results:
        i. The dark/light mode option maintains the selected option when switching pages and refreshing the page, and every page reflects that options theme.
        ii. The font size is saved when switching pages and refreshing the page, and is applied to all text on the site.
        iii. All font size changes are properly formatted and do not cause any overflow issues.
        iv. Almost all text is supported for both languages aside from some minor sections on the Admin page that we did not get to.
        v. The language selector is visible on all pages and properly changes the language across all pages and applies to the calendar objects as well.
4. New Job Input Form Testing
    a. Purpose:
        i. To ensure all features on the job input form function properly and align with the definition of done.
    b. Description:
        i. The job input form is where new jobs are created. This form includes dropdown menus, text fields, date pickers, and address selectors. The form also includes a submit button and cancel button with popups to confirm the submission/cancellation.
    c. Tools Utilized:
        i. Human Interaction and Inspect Element Dev Tools
    d. Threshold:
        i. The various input fields properly assign the values and include the expected options. The address selector should suggest valid inputs and only allow actual addresses to be inputted. The submission should not be permitted unless all required fields have inputs and should display errors for any that do not. If all required fields have inputs, then the submit

button should display a pop up to confirm submission. The cancel button should display a pop up to warn the user that information will not be saved and to confirm the cancellation. The user should not be able to navigate off of the page without confirmation, submission, or cancellation.

e. Edge Cases:
    i. The back button allows the user to exit the form without submitting/cancelling.
    ii. The address selector has limited usage due to the API key and format.
    iii. The user wants to input an address that has not been added to Google's database yet and cannot be found through the API.
    iv. Reloading the page resets all fields without warning the user.
f. Results:
    i. All input fields assign the values to the corresponding fields accurately and the options match the expected values.
    ii. The address selector properly suggests and only allows for valid addresses.
    iii. The address selector allows for 10,000 usages per month which will not likely be exceeded at this time.
    iv. The error messages for unselected, but required, fields properly display and prevent submission.
    v. The submit and cancel buttons properly display a pop up to confirm the action and allow the user to divert from such action.
    vi. Pressing the back button or the refresh page button displays a confirmation message to alert the user that information will not be saved.
    vii. Navigation off page is properly prevented through the removal of the navigation bar.

5. New Pour Input Form Testing
    a. Purpose:
        i. To ensure all features on the pour input form function properly and align with the definition of done.
    b. Description:
        i. The pour input form is where new pours are created. This form has the same cancellation and submission functionality as the job input form. The fields on this page include dropdowns, text fields, integer only text fields, date pickers, and time pickers.
    c. Tools Utilized:
        i. Human Interaction and Inspect Element Dev Tools
    d. Threshold:
        i. All input fields should display the expected options and assign the correct values. The submission should not be permitted unless all required fields have inputs and should display errors for any that do not. If all required fields have inputs, then the submit button should display a pop up to confirm submission. The cancel button should display a pop up to warn the user that information will not be saved and to confirm the cancellation.

The user should not be able to navigate off of the page without confirmation, submission, or cancellation.

e. Edge Cases:
    i. The back button allows the user to exit the form without submitting/cancelling.
    ii. Reloading the page resets all fields without warning the user.
f. Results:
    i. All input fields properly assign values to the corresponding fields and display the correct menu options.
    ii. Submission is properly prevented when required fields are missing data and the proper error messages are displayed.
    iii. Submission and cancel buttons properly display a pop up to confirm the action and allows the user to divert from that action.
    iv. The refresh page button and back button presses display a message alerting the user that information will not be saved and confirms the action.
    v. Navigation off page is prevented by removal of the navigation bar.

6. Mobile Display
    a. Purpose:
        i. Ensure that the phone view is readable and accessible
    b. Description:
        i. When starting up the phone view of the application, all words should be easily readable and all menu buttons should direct the user to their specific pages. The refresh button should work as intended as well as the logout button. There should be no input pages and the Jobs page calendar should go 2 weeks out and be displayed in a vertical manner by day.
    c. Tools Utilized:
        i. Human Interaction
    d. Threshold:
        i. All buttons work 100%, text is readable in all font sizes.
    e. Edge Cases:
        i. The user clicks logout but then is able to use the back arrow to get into the app, causing a security concern.
    f. Results:
        i. All other pages like the calendar and settings work as intended just like the desktop version, no edits are needed, no testing is needed as this has been tested for the desktop.
        ii. The admin page is not accessible on mobile, just as intended, no testing is necessary as the admin button does not display.
        iii. The jobs page is the only one that needs testing. The dates are displayed accurately and change to Spanish as intended within 1 second.
        iv. We did not complete a login page or authentication so we cannot test the edge case, the client has been made aware and is okay.

v. No projects have been pulled from the database to appear on the jobs page so no testing is needed as the team cannot complete it due to the time constraint, the client has been notified and is okay with it.
vi. Manual projects display correctly with an expand button, only one can be expanded at a time, just as intended.

7. Roles (admin, superuser, user)
    a. Purpose:
        i. Ensure certain actions are only accessible for certain roles/users
    b. Description:
        i. The base user should only be able to view jobs and pours, and to add notes to jobs and pours. Super Users should be able to add new jobs and pours and edit them in addition to base user permissions. Admin should be able to modify the forms for adding new pours and jobs, and should be able to delete jobs and pours. The admin has all permissions of the base user and the superuser.
    c. Tools Utilized:
        i. Human Interaction and Inspect Element Dev Tools
    d. Threshold:
        i. The roles should only be able to perform the actions described above, and the admin page should only be visible to admin users.
    e. Edge Cases:
        i. If an admin deletes a job while a superuser is editing that job and then submits the form, the job may incorrectly be added/other unexpected behavior may occur.
    f. Results:
        i. Unfortunately, due to time constraints, the team could not connect the login page to the Microsoft Access authentication database. This means that the official admin, super user, and user privileges are hard coded.
        ii. The Admin privilege accurately shows the Admin button on the app bar located at the top of the page. Additionally, it is clickable and functional.
        iii. The Super User privilege gets rid of the admin button located on the app bar and the user can no longer access the admin page.
        iv. The User privilege eliminates both the admin button and functionality along with the pour and job input forms. The user is not able to access all three of these items as intended.
        v. These have not been tested simultaneously as the app is not deployed and we have no real user authentication.
8. Fetch Testing
    a. Purpose:
        i. To make sure that the connection between the frontend and the backend is reliable and functions as expected.
    b. Description:
        i. When opening the app, all of the jobs from the database should be visible on the jobs page and the calendar page with the correct information. Additionally, updates should be visible with little to no delay after they are made.
    c. Tools Utilized:
        i. Human Interaction

d. Threshold:
    i. All data is correctly displayed and updates are visible to all users within 1 second of the update being submitted.
e. Edge Cases:
    i. The details of a pour update, including the time, but the pour itself does not get moved to the correct time slot on the jobs or calendar page.
f. Results:
    i. Results are shown below (Figure 5).

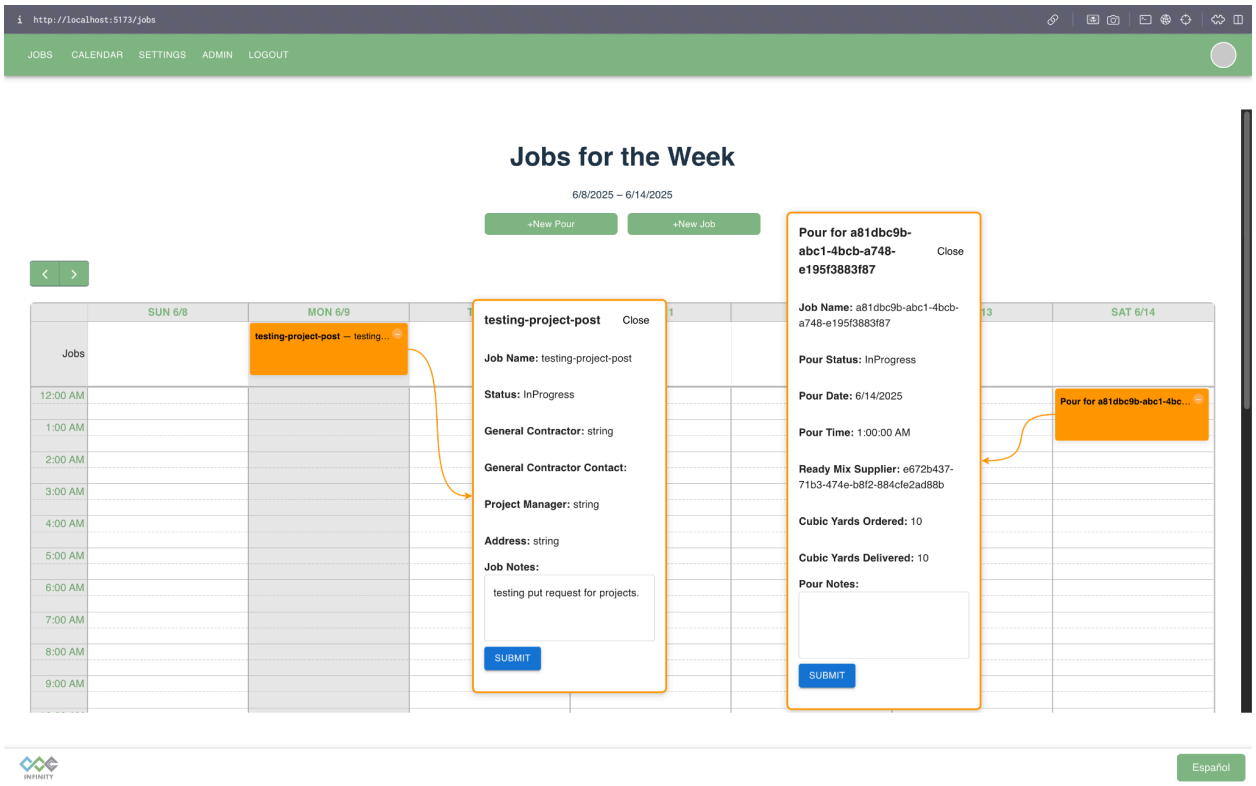

Figure 16: Screenshot of Job Page from Concrete Dispatch App displaying a Job/Project (on the left) and a Pour (on the right) using Fetch GET requests at the corresponding backend .NET endpoints

Backend Testing

Original Testing Plan: For the backend, we will only be focusing on integration testing because we believe that this will be the most effective way to test given our time constraints.

Updated Testing plan: Originally, the team had planned to do integration testing, however an error relating to the Docker Cosmos database emulator (regarding partition key versions) occurred which we could not find any documentation on. Due to the short time frame, the team decided to continue doing manual testing. We made the client aware of this and they said that it

was ok. For manual testing, the team used Swagger UI for the backend, which is the built-in tool for testing endpoints in a .NET application.

1. GET Request for Projects, Pours, and Suppliers
    a. Purpose:
        i. The purpose of this test is to ensure that Cosmos DB not only works, but is accessible through C#
    b. Description:
        i. We will have a temporary DB set up in Cosmos that stores small data.
    c. Tools Utilized:
        i. Swagger UI
    d. Threshold:
        i. The backend can freely access data from any given database.
    e. Edge Case:
        i. If we send multiple Cosmos requests at the same time, the system will be able to manage all of them sequentially.
    f. Results:

Using Swagger UI, a built-in application for testing .NET routes, we ensured that the GET routes for Pours: “api/Pours,” Projects: “api/Projects”, and Suppliers: “api/Suppliers,” was returning all of the respective data for Pours, Projects, and Suppliers stored in the Cosmos database.

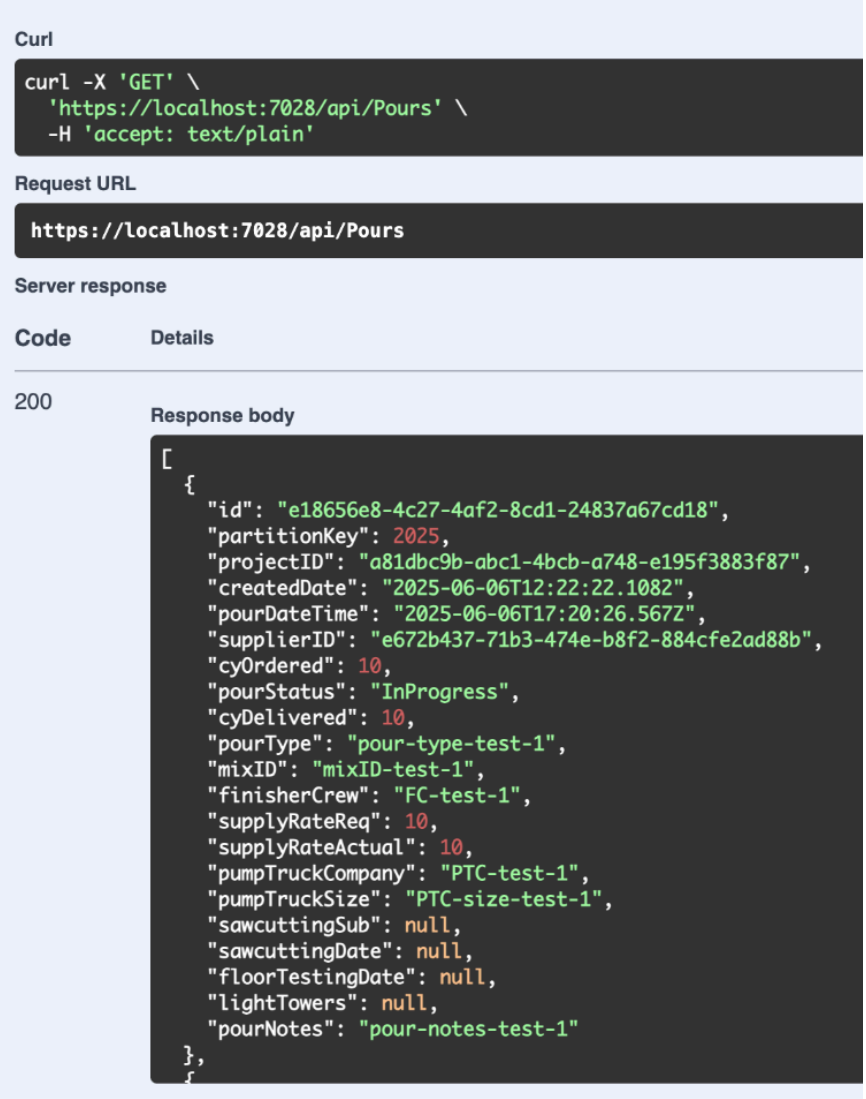

Figure 17: Screenshot of Swagger UI returning a 200 Code (Success), and the respective data for “api/Pours” route GET request

Curl

```
curl -X 'GET' \
  'https://localhost:7028/api/Projects' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7028/api/Projects
```

Server response

Code	Details
200	**Response body**


```
[
  {
    "id": "a81dbc9b-abc1-4bcb-a748-e195f3883f87",
    "projectName": "project-test-1",
    "createdDate": "2025-06-06T12:13:53.633794",
    "partitionKey": 2025,
    "projectStatus": "Incomplete",
    "generalContractorName": "GC-test-1",
    "projectManagerName": "PM-test-1",
    "address": "address-test-1",
    "notes": "notes-test-1"
  },
  {
    "id": "16ea9b4c-1c5f-4f3d-840b-bc7a6562631a",
    "projectName": "project-test-2",
    "createdDate": "2025-06-06T12:14:12.275878",
    "partitionKey": 2025,
    "projectStatus": "Incomplete",
    "generalContractorName": "GC-test-2",
    "projectManagerName": "PM-test-2",
    "address": "address-test-2",
    "notes": "notes-test-2"
  },
  {
    "id": "e009f56d-cec3-4f85-9a18-f4c4ce510ccc"
```


Figure 18: Screenshot of Swagger UI returning a 200 Code (Success) and the respective data for “api/Projects” route GET request

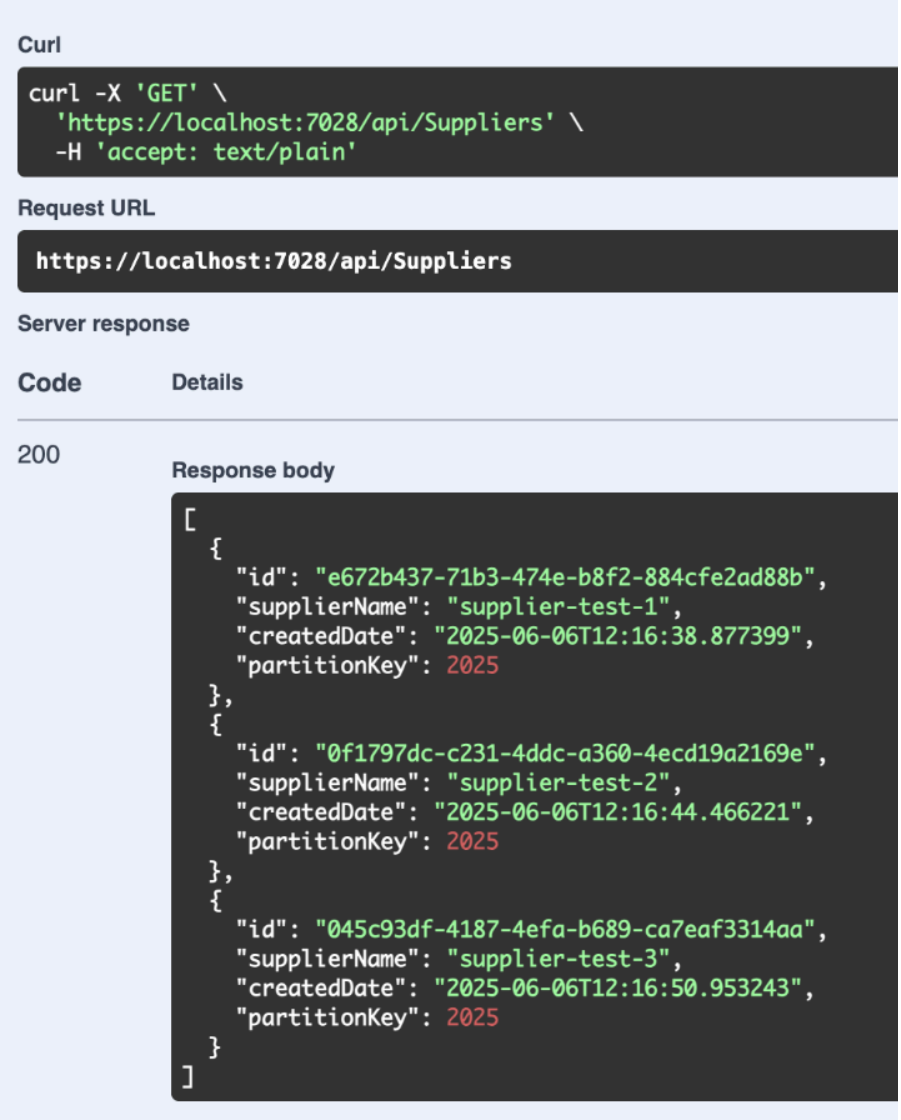

Figure 19: Screenshot of Swagger UI returning a 200 Code (Success) and the respective data for “api/Suppliers” route GET request

2. POST request for Projects, Pours, and Suppliers
    a. Purpose:
        i. Create multiple projects using POST requests that have different months (to test GET requests by month) and years (to test partition keys are working effectively).
    b. Description:
        i. Test that projects are being created on Cosmos, and test that the GET requests (GET all, GET by two week interval, GET by month) are reflecting the POST changes.
    c. Tools Utilized:
        i. Swagger UI
    d. Threshold:

        i. Ensure that all data is reflected by the GET requests and on Cosmos. Also ensure that GET all, which reads from various partitions, retrieves the data quickly.
    e. Edge Case:
        i. Get project data for a two week interval that crosses over a partition (weeks are in different years.)
    f. Results:

Using Swagger UI, a built-in application for testing .NET routes, we ensured that the POST routes for Pours: “api/Pours,” Projects: “api/Projects”, and Suppliers: “api/Suppliers,” creating new data for Pours, Projects, and Suppliers and uploading them successfully to the Cosmos database.

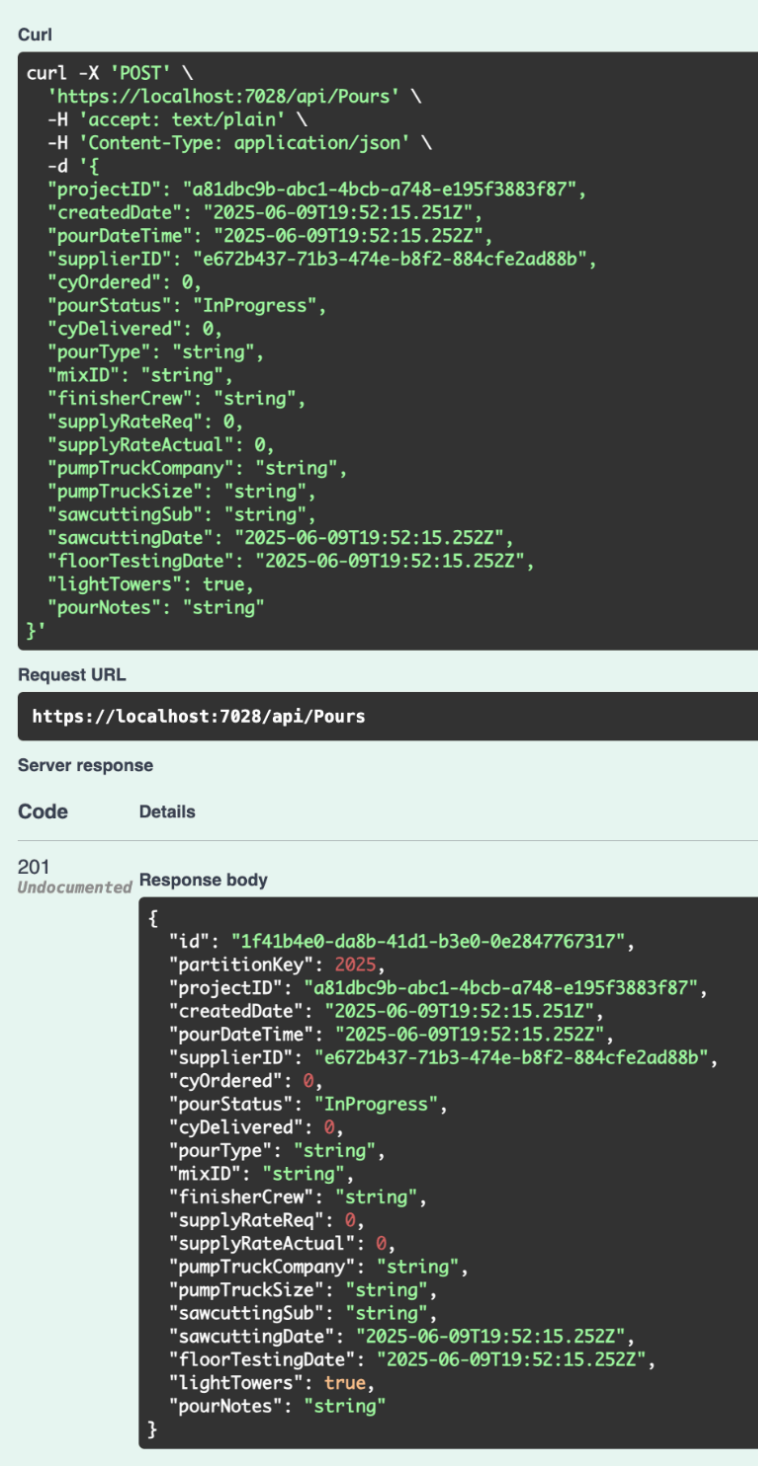

Figure 20: Screenshot of Swagger UI returning a 201 Code (Successful POST request) and the respective data for “api/Pours” route POST request

Curl

```
curl -X 'POST' \
  'https://localhost:7028/api/Projects' \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
  -d '{
  "projectName": "testing-project-post",
  "createdDate": "2025-06-09T19:55:06.155Z",
  "projectStatus": "InProgress",
  "generalContractorName": "string",
  "projectManagerName": "string",
  "address": "string",
  "notes": "string"
}'
```

Request URL

```
https://localhost:7028/api/Projects
```

Server response

Code	Details
201 *Undocumented*	Response body

```
{
  "id": "d9d98ecd-4d7a-48ea-af53-f43e3bd0ff88",
  "projectName": "testing-project-post",
  "createdDate": "2025-06-09T19:55:06.155Z",
  "partitionKey": 2025,
  "projectStatus": "InProgress",
  "generalContractorName": "string",
  "projectManagerName": "string",
  "address": "string",
  "notes": "string"
}
```

Figure 21: Screenshot of Swagger UI returning a 201 Code (Successful POST request) and the respective data for “api/Projects” route POST request

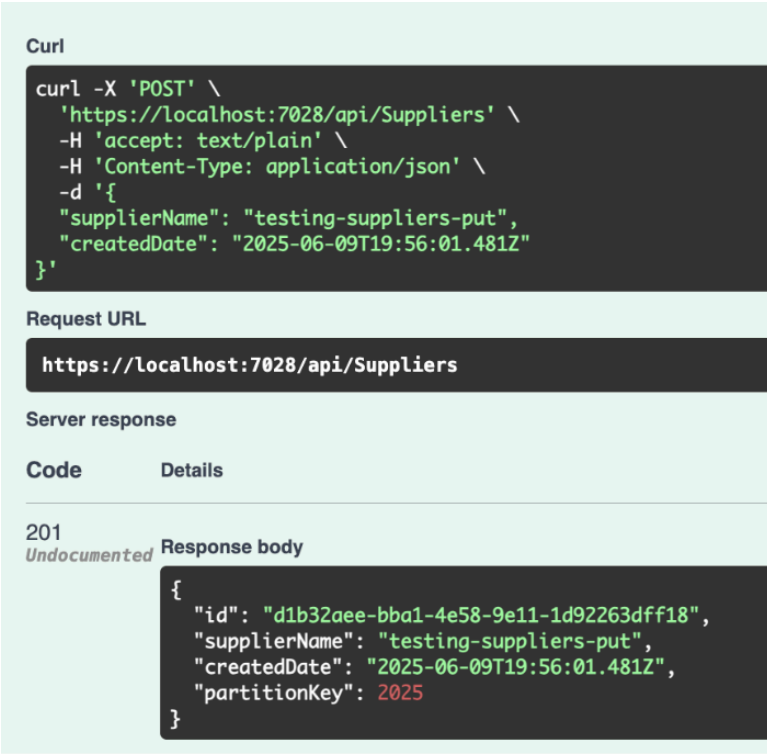

Figure 22: Screenshot of Swagger UI returning a 201 Code (Successful POST request) and the respective data for “api/Suppliers” route POST request

3. PUT request for Projects, Pours, and Suppliers
    a. Purpose:
        i. Update project information using PUT requests.
    b. Description:
        i. Change different fields for different projects, potentially including the project status.
    c. Tools Utilized:
        i. Swagger UI
    d. Threshold:
        i. Ensure that changes are reflected in Cosmos, and by the GET requests in a timely manner.
    e. Edge Case:
        i. Incorrectly update a field. For example, project status is an enum and must be one of three values: complete, incomplete, in progress.
    f. Results:

Using Swagger UI, a built-in application for testing .NET routes, we ensured that the PUT routes for Pours: “api/Pours/{pour-id},” Projects: “api/Projects/{project-id}”, and Suppliers: “api/Suppliers/{supplier-id},” were updating the existing Pour, Project, and Supplier data.

Curl

```
curl -X 'PUT' \
  'https://localhost:7028/api/Pours/1f41b4e0-da8b-41d1-b3e0-0e2847767317' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": "1f41b4e0-da8b-41d1-b3e0-0e2847767317",
    "partitionKey": 2025,
    "projectID": "a81dbc9b-abc1-4bcb-a748-e195f3883f87",
    "createdDate": "2025-06-09T19:52:15.251Z",
    "pourDateTime": "2025-06-09T19:52:15.252Z",
    "supplierID": "e672b437-71b3-474e-b8f2-884cfe2ad88b",
    "cyOrdered": 0,
    "pourStatus": "InProgress",
    "cyDelivered": 0,
    "pourType": "string",
    "mixID": "string",
    "finisherCrew": "string",
    "supplyRateReq": 0,
    "supplyRateActual": 0,
    "pumpTruckCompany": "string",
    "pumpTruckSize": "string",
    "sawcuttingSub": "string",
    "sawcuttingDate": "2025-06-09T19:52:15.252Z",
    "floorTestingDate": "2025-06-09T19:52:15.252Z",
    "lightTowers": true,
    "pourNotes": "Changing notes for pour put request testing."
  }'
```

Request URL

```
https://localhost:7028/api/Pours/1f41b4e0-da8b-41d1-b3e0-0e2847767317
```

Server response

Code	Details
204 *Undocumented*	Response headers

```
date: Mon,09 Jun 2025 20:01:43 GMT
server: Kestrel
```

Figure 23: Screenshot of Swagger UI returning a 204 Code (Successful PUT request) and the respective data for “api/Pours/{pour-d}” route PUT request

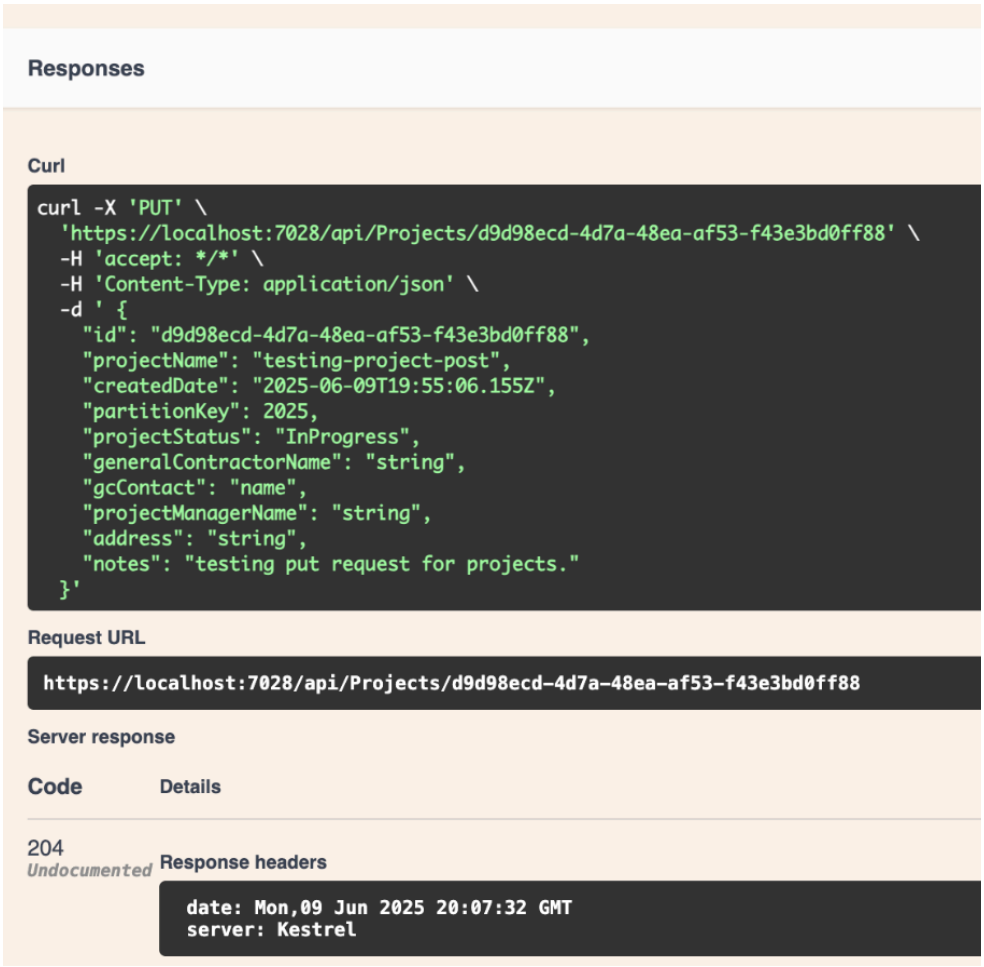

Figure 24: Screenshot of Swagger UI returning a 204 Code (Successful PUT request) and the respective data for “api/Projects/{project-id}” route PUT request

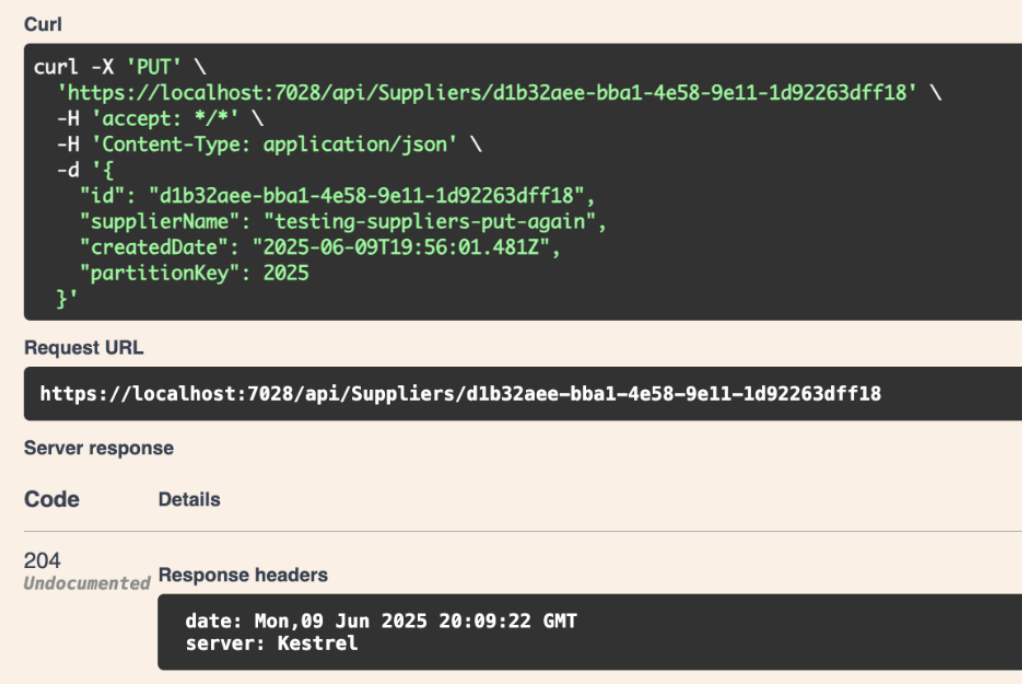

Figure 25: Screenshot of Swagger UI returning a 204 Code (Successful PUT request) and the respective data for “api/Suppliers/{supplier-id” route PUT request

VIII. Project Ethical Considerations

As with any project, the group as developers are required to uphold ethical standards outlined in the IEEE/ACM Software Engineering Code of Ethics. Some particular ethical concerns revolve around public interest, client obligations, product integrity, and long term impact

The first potential ethical consideration is the displacement of dispatch workers due to the implementation of the web application. This potentially interferes with Principle 1.03 in the IEEE Software Engineering Code of Ethics which states, *approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good*. This web application will effectively replace the dispatch roles , diminishing the quality of life for those employees if they are laid off. The client should make efforts to retain or transition affected employees.

The second potential ethical concern involves the continued maintenance of the software. The team will no longer be involved in the development of the web application, thus it is important that the group address any ethical concerns before handing off development. Principle 3.15 states, *treat all forms of software maintenance with the same professionalism as new development.* This is an important consideration for the client before moving on with production and maintenance. Before handing over the software, the group needs to ensure they are abiding by Principles 3.11 and 2.06. These ethical considerations include providing

adequate documentation to the client, in the form of readable code, user manuals, and maintenance manuals, as well as ensuring that all concerns and possible failures are reported to the client so they can be aware of potential problematic elements of the software. The team's goal is to be transparent and responsible with the software being handed to Infinity Concrete.

The third ethical consideration is imperative to the overall design of the app for future sustainability and accessibility. It is important that the group abide by Principle 1.07 stating, *consider issues of physical disabilities, allocation of resources, economic disadvantage and other factors that can diminish access to the benefits of software.* This ethical concern includes ensuring the web app can be accessed by users with limited technical literacy, as well as users that speak a different language or need to access larger fonts to read contents. The team must thoughtfully account for the diverse demographics of users to ensure the application's capabilities reflect the company's vision and meet the needs of all employees.

The last ethical concern relates to ensuring the clients are aware of project boundaries and limitations of the group. The team is composed of student developers, as such, we must act within our areas of competence as outlined in Principle 2.01 of the IEEE Software Engineering Code of Ethics. The group is uncertain about development going beyond their capabilities, it is imperative that they communicate that clearly to the client as soon as possible. Upholding transparency to the client is incredibly important as to avoid deception or incorrect claims about the software.

The team is committed to producing high quality software that adheres to the developmental process ethical considerations as outlined in IEEE/ACM Software Engineering Code of Ethics. We hope to serve the client's needs while respecting workers, maintaining professional integrity, and promoting public good.

IX. Project Completion Status

Throughout the course of the project, the team has created a foundational version of the Infinity Concrete web application. The team aimed to improve submission of concrete orders for specific job sites, view upcoming scheduled pours, and comply with ADA requirements. Many features and structural components were completed, but there are many more that remain uncompleted.

Among the various implemented features were most notably the Jobs Page and the Calendar page. The Jobs page contained the Job and Pour Input forms, these include required field validation, red error highlighting, and inline messages to support accurate and user-friendly data entry. These are connected to the database on the web application so all forms submitted are stored and are visible on the weekly calendar which appears on the Jobs page. Role-based access control was established, this is currently hard-coded and tested. This allows access to certain pages depending on the user's assignment role. A mobile-responsive calendar UI framework was developed to display jobs day by day in a vertical manner.

Several of the features for the web application are currently partially implemented or non-functional. First, the Calendar page UI is in place and the user is able to click on the date to select a year and month they would like to view. The calendar currently does not display any of the projects that are in the database, therefore the user cannot click on any to see their expanded view. Another partially implemented feature is the drop downs on the forms. The dropdowns are not dynamically populated from the backend, instead values have been hard coded in for testing. Additionally, the admin page is currently non-functional. The admin page is intended to allow the administrator to add items to the drop downs for users. The team was unable to complete the tables for the different drop downs in time and thus the admin page only contains the menu for all of the required drop down elements, but with no functionality. Furthermore, the frontend-backend syncing works on the desktop version, but not for mobile devices.

There are several features that the team was unable to get to and are therefore not implemented. The web app currently does not support editing or updated submitted jobs or pours, there is no reporting dashboard for the admin to review job performance, and email notifications for job submission and status changes were not added along with the ability to directly submit orders to ready-mix suppliers for pour orders. Lastly, the app additionally lacks a login page or secure user authentication using Microsoft Access.

X. Future Work

Considering much of the current project is non-functional, the team has devised a plan for future work to aid the client in the completion of the project.

1. **Backend Integration for Dropdown Menus and Admin Page**

The goal of backend integration for dropdown menus is to connect fields to the backend and allow the admin to change what information is inside the dropdowns. The dynamic population of the values is crucial for accurate pour and job order forms. This requires access to the NoSQL database, CosmosDB, along with the hosting service Azure. This would require backend and database skills as well as front end skills for the creation of the dropdowns and the added functionality for the admin page. The estimated time for this would be 1-2 weeks for a team of 2 people.

2. **Calendar Job Display and Edit Functionality**

The calendar display is an integral part of the overall functionality of the web app, allowing the user to view jobs each month. This feature involves pulling pours from the backend and displaying them on the calendar. This requires use of a NoSQL database and extensive

knowledge in full-stack development as there are components for this that involve the front and backend. The time for this would be about 5-7 days for a team of 2 people.

3. **User Authentication and Login System**

Securing the web application is crucial for the overall functionality of the different roles. This would involve access and usage of a current Microsoft database with company users are their assigned roles as well as access to Microsoft Access for authentication. The skills required include frontend knowledge which will be useful for building authentication flows, editing the current hard-coded version of role definitions, securing data handling, and managing sessions and tokens. The time for this would be 1-2 weeks for a team of 2 people.

4. **Admin Dashboard with Reporting Tools**

The client informed the team from the beginning that they would like the web app to support operational oversight. This includes adding a reports section to the admin page that will allow the user to display metrics such as job history, pour history, order timeliness, and other performance indicators. This will involve using charting libraries, of which the team is not familiar with at this time. This would need to be integrated with the existing NoSQL database, Vite frontend and .Net backend. It would be recommended that the developer has knowledge in full stack development along with knowledge in data analysis. This would take about 2 weeks for a team of 2 to complete.

5. **Automated Email Notifications**

The client told the team that they wanted to eventually integrate an email notification system into the web app. This would allow the user to get an email notification when a form was submitted as well as an email notification when the status of a project was changed. This would involve access to Microsoft to access user emails, at this time the team is unaware of other requirements as this is outside of their scope and competence. It is recommended that the developer have knowledge of full stack development as well as knowledge with Microsoft. The time to complete this would be about 5-7 days for a group of 2.

6. **Integration with Concrete Suppliers (Optional/Future Scope)**

This feature was mentioned by the client as an optional feature that would be added farther down the line. This would allow users to input forms that would then be directly sent to ready-mix suppliers that would then approve or deny the order. This would then be sent back and processed by the web application. This would require the use of Microsoft as well as full stack development knowledge to integrate this into the frontend and backend to access the database. This would take 1-2 weeks for a 2 person team.

XI. Lessons Learned

The development of the web application was tough! The team gained valuable technical experience as well as insight into real-world software development and engineering challenges. One of the key takeaways from this experience was the importance of early and clear communication with the client. It was incredibly important that the team knew the exact requirements, scope, and system priorities for the company's vision. Miscommunication in expectations could cost the group valuable development time and the team made the utmost effort to communicate effectively, honestly, and transparently with the client.

One of the most time-consuming aspects of the project was the design and backend planning phase. While the frontend development progressed smoothly, many of the more advanced and interactive features relied heavily on backend integration, which remained incomplete. As a result, several of the team's more ambitious features, such as dynamic dropdowns, job editing, and calendar updates could not be implemented to their full potential due to missing API endpoints and limited database connectivity.

This experience underscored the critical importance of backend infrastructure and the need for strong coordination between frontend and backend development from the start. It also highlighted the necessity of designing for scalability and maintainability. Features like role-based access and dynamic data inputs must be built with future growth in mind, requiring flexible structures and well-planned data models. The admin page, in particular, was envisioned as a central tool to manage evolving data needs, but its full functionality was limited by time constraints and gaps in backend integration. Investing more time into the backend would have helped this issue and it is nice that the team is aware of this now as they can use this lesson in future software development projects.

Overall, this project provided the team with valuable hands-on experience and meaningful lessons in software development. Through the process, team members strengthened their skills in React, TypeScript, NoSQL, and modern design patterns, while gaining a deeper understanding of full-stack application development. Beyond technical growth, the project also fostered collaboration and improved the team's ability to work effectively in a shared codebase. The team is grateful for the knowledge gained and the opportunity to grow both individually and collectively. This experience has sparked excitement for future challenges and opportunities in the field of software engineering.

XII. Acknowledgments

The team would like to sincerely thank our client at Infinity Concrete for the opportunity to collaborate on this project. Their clear vision and considerate, consistent feedback were instrumental in our development of the overall completed web application. The team would also like to thank our technical advisor, Professor Donna Bodeau, for her valuable support, encouragement, and feedback throughout the entire process. We are proud of what we have

accomplished and our growth as software engineers. Thank you for your trust and mentorship throughout the course.

XIII. Team Profile

Tyler Collingridge
Data Science Focus Area
Colorado Springs, CO
CSM Bookstore
Snowboarding

Avie Burris
Data Science Track
Roswell, New Mexico
CSCI 128 TA
Running, hiking, swimming

Ryan Jones
AI Track
Tulsa, Oklahoma
Aberdeen Dynamics, Data Analysis
Walking, skatting

Mark Duffalo
Robotics and Intelligent Systems
Bel Air, Maryland
Food Delivery Driver
Skiing, rock climbing, whitewater kayaking

References

Appendix A

Table 5: Key Terms

Term	Definition
Foreman (user)	*Person giving directions for pours in the field (working at the actual site.) They will be able to view, but not edit, pours and projects in our application. Also, they will be viewing the*

	application on a mobile device (phone, tablet, etc.)
Project Manager (PM, superuser)	*Person creating and managing pours and projects. They will be able to view, edit, and create new projects and pours. They will typically be viewing the application in the browser from their computer.*
Admin	*This is anyone with higher permissions than a PM. They have the permissions of a PM, while also being able to edit dropdowns for inputting a project or pour. In the future, they will have even more permissions, but those have not been implemented yet. These users will typically be viewing from a computer.*
Dispatcher	*These individuals work at Infinity Concrete and at the Ready Mix Supplier. When a PM wants to create an order, they tell the Infinity dispatcher, who calls the supplier dispatcher to confirm the concrete order. Then, the Infinity dispatcher notifies the PM that the order has been confirmed. Our application hopes to eliminate this role entirely.*
Concrete	*[Provide definition of the term used in this document.]*
Job/Project	*These two terms are used interchangeably throughout our project. A job/project refers to any and all work for a residential or commercial contract for a customer. This typically consists of one or more pours.*
Pour	*An individual pour of concrete for a customer.*
GET Request	*Request data from the database.*
POST Request	*Create new data to be added to the database.*
PUT Request	*Update existing data within the database.*

Figure 26: further describing permission levels (as defined above in Foreman (user), Project Manager (superuser), Admin definitions)