



Computer Science Clinic

Final Report for
Cradlepoint

Cradlepoint 2021-2022 Final Report

May 3, 2022

Team Members

Jessica Kwok (Fall Project Manager)
Dylan McGarvey (Spring Project Manager)
Matt Tran
Natasha Wong

Advisor

Melissa O'Neil

Liaisons

Lynn Gates
Aaron Ebling
Alex Terrell
James Hawk
Chirs LaPeters

Contents

1	Introduction	5
1.1	Company Background	5
1.2	Development Process	5
2	Problem Statement and Solution	7
3	Technology	9
3.1	Description of Stack	9
3.2	Frontend Libraries	9
3.3	Documentations	10
4	Backend	11
4.1	Connecting to MongoDB	11
4.2	API Routes	11
4.3	Data Schema	12
5	Hosting	19
5.1	Set up	19
5.2	Mongodb	22
6	Frontend Architecture	25
6.1	Pages	25
6.2	Components	29
6.3	Modals	30
6.3.1	Create New Modal Flow	33
6.3.2	Edit Modal Flow	33
6.3.3	Delete Modal Flow	34
6.3.4	BOM-related Modal Flows	34

7 Future Work	37
7.1 Known Bugs	37
7.2 Additional Data Validation and Testing	38
7.3 Recommended API Changes	38
7.4 Recommended Schema Changes	39
7.5 Feature Extensions	40
7.5.1 Review and Iteration Process Extensions	40
7.5.2 Quality of Life Extensions	41

Chapter 1

Introduction

1.1 Company Background

Cradlepoint is one of the leading solutions provider for LTE and 5G wireless network problems.¹ It provides a wide range of physical products including cellular and Wi-Fi antennas, private LTE and 5G solutions for businesses, and other network accessories. In addition, Cradlepoint also offers a cloud-based management and control platform, networking technologies, routing and security, and other functionalities to support its network equipment. Its customers ranges from private businesses to the public sector.

Of particular significance, Cradlepoint's Sales Engineering Center of Excellence (COE) team, formed by a group of specialists, architects, corporate system engineers (SE), and lab designers, consults with customers to provide solutions based on their specific situations and use cases. The COE does so by executing customized proof-of-concept (POC) test plans and uses the results of these tests to provide workable insights and compelling proof points for the technical architects and operators to design and implement network solutions.

1.2 Development Process

In the Fall Semester, we mainly focused on creating a Minimum Viable Product (MVP) for our web application, which involved creating most of the user interfaces, API endpoints, and integration between them. At the

¹As noted by Cradlepoint in their project description

start of the Spring Semester, we focused on bug-elimination, reducing of technical debts from the MVP, and started works towards the Refined Final Product (RFP). Our original focus for the RFP was to be on the Review and Iteration Process, however, this was pushed towards being an area of future work (see section 7.5.1). Instead our focus was put towards hosting our application through AWS, refactoring APIs to be more RESTful, and creating functionality for cloning from libraries and data deletion.

Deliverable	Date
Minimum Viable Product (MVP)	December 7, 2021
Midyear Update	December 7, 2021
MVP Documentation and Technical Debts	February 1, 2022
Refined Final Product (RFP)	April 1, 2022
Software Feature Freeze	April 8 2022
RFP Documentation and Tests	April 14, 2022
Coding Freeze	April 22, 2022
Project Poster	April 22, 2022
Final Report	May 6, 2022

Table 1.1 Original Schedule of Deliverables

Chapter 2

Problem Statement and Solution

Due to the valuable insights that these POC test plans can provide to both Cradlepoint and its customers, Cradlepoint has seen a dramatic growth in demand for POC testing. Currently, much of the work the COE team has done to create, collect, manage, and present its POC test plans occurs manually. This complicated testing process involves multiple stakeholders, including customers, partners, SEs, and POC engineers. To complete a test cycle, different stakeholders need to frequently exchange documents, ask for and provide approval and feedback, and present results. Thus, the current process is time consuming and resource intensive for the company. To ameliorate this problem, Cradlepoint wishes to automate its current manual process via a web based portal that will manage the workflow of the POC testing process from start to finish.

Chapter 3

Technology

3.1 Description of Stack

Our technology stack consists primarily of React built through the NextJS framework that provides middleware and backend routing capabilities in addition to the conventional frontend features. Further, we rely on MongoDB to serve as database and a number of Amazon Web Services such as S3 and Amplify to support our hosting needs for images and files and the website itself respectively.

3.2 Frontend Libraries

There are a number of frontend libraries we have leveraged to ease our building of the web application. Below is a list of these libraries:

- Material UI: provides many of the simple UI components we customize or directly use.
- React Modal: provides the modal component we use throughout the app.
- Toastify: provides error pop ups.
- React Table: provides the table UI we use, which supports robust functionality such as sorting and organizing data within the table.

3.3 Documentations

This document serves to be the source of truth for all documentation pertaining to the development of this web app. Additionally, we have taken steps to comment the codebase of our app to ensure it is easily readable to future developers.

Chapter 4

Backend

4.1 Connecting to MongoDB

We have created a file in the utility folder which will create a client utilizing the Mongodb package for Nodejs, and return that as a result. This object is then used for any actions involving the database.

4.2 API Routes

Each of the API Routes can be found in the ./pages/api folder. In Next.JS, each of these files will create an API route which can be called with [host]/api/[pathToFile]. Additionally, many of these routes, if they are a GET route, contains two instances of the same function. The default exported function is utilized when the API route is called using an http request. The other instance serves to be directly imported into a page when rendering. This makes the application run faster by not having to await an API call from its own backend. Additionally, it allows the application to run without needing to know the name of the site in which it is being hosted. Many of these routes also import a function from within the util folder for allowing connections to the database, which is done in the way described in the previous section.

We have detailed the various endpoints needed to allow our frontend to communicate with our backend, in this supplemental [document](#)¹.

¹Endpoints: <https://bit.ly/3oeUADE>

4.3 Data Schema

First, we will attempt to include a file which contains our dummy data in json form which can then be imported into another Mongo server. In the event that this is not possible, or far too complicated of a solution, the database should still be able to be recreated with the tables below within this section. In this section, you can find the schema that was used for each collection. The top of the table outlines the name of the collection in the following tables. The collections include:

- engagements
- statusCodes
- testPlan
- testPlanLibrary
- testCases
- testCaseLibrary
- tests
- testLibrary
- results
- user (created but not implemented in the application yet)
- userTypes (created but not implemented in the application yet)
- device
- BOM (not a collection, a customized object used in other collections)

Table 4.1 engagements

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
createdOn	[Date]	
testPlanId	[Object ID]	ID of the active test plan
statusCode	Int	Relates to Status Codes in the collection
SFDC	String	Should be a url
customer	String	
description	String	
POC_Engineer	String	POC Engineer Responsible
SE	String	Site Engineer Responsible

Table 4.2 statusCodes

Field	Data Type	Notes
_id	Int	Each status is represented by an integer
status	String	Status to be displayed
description	String	More detailed description of the status

Table 4.3 testPlan

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
createdOn	[Date]	
summaryBOM	Array[Object]	Array of objects containing BOM Objects (shown later)
testCases	Array	Array of Ids of the associated test cases
engagementId	[Object Id]	Id of the associated engagement
deviceConfig	String	would be replaced with filename later, and be a download link
isActive	Boolean	Whether or not this is the active test plan
version	String	Version of the test plan
authors	Array	Array containing all of the authors of this test plan
customerFeedback	String	
description	String	
name	String	

Table 4.4 testPlanLibrary

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
createdOn	[Date]	
summaryBOM	Array[Object]	Array of objects containing BOM Objects (shown later)
testCases	Array	Array of Ids of the associated test cases
engagementId	[Object Id]	Id of the associated engagement
deviceConfig	String	would be replaced with filename later, and be a download link
version	String	Version of the test plan
customerFeedback	String	
description	String	
name	String	

Table 4.5 testCases

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
name	String	
description	String	
config	String	would be replaced with filename later, and be a download link
topology	String	filename in the S3 Bucket
testPlanId	[Object ID]	Id of the associated Test Plan
tests	Array[Object ID]	Array of IDs all associated tests
BOM	Array[Object]	Array of objects containing BOM Objects (shown later)

Table 4.6 testCaseLibrary

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
name	String	
description	String	
config	String	would be replaced with filename later, and be a download link
topology	String	filename in the S3 Bucket
testPlanId	[Object ID]	Id of the associated Test Plan
tests	Array[Object ID]	Array of IDs of all associated tests
BOM	Array[Object]	Array of objects containing BOM Objects (shown later)

Table 4.7 tests

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
name	String	
description	String	
testCaseId	[Object ID]	Id of the associated Test Case
resultStatus	String	Status of the results
results	Array	Array of Ids of the associated results

Table 4.8 testLibrary

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
name	String	
description	String	
testCaseId	[Object ID]	Id of the associated Test Case

Table 4.9 result

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
createdOn	[Date]	
testId	[Object ID]	ID of the associated test
resultStatus	String	Enum of: Pass, Unknown, or Fail
evidence	String	Unique filename in the S3 bucket
description	String	

Table 4.10 user

Field	Data Type	Notes
firstName	String	
lastName	String	
userType	int	Referenced from userTypes collection, specifically the code
email	String	
userId	String	Unique identifier for a user

Table 4.11 userTypes

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
createdOn	[Date]	
code	string	code for different usertypes, used for lookup
userType	String	Actual type of the user

Table 4.12 device

Field	Data Type	Notes
_id	[Object ID]	Mongo datatype for IDs
deviceName	String	
codeVersion	String	
SKU	String	
deviceType	String	Should be physical or software

Table 4.13 BOM

Field	Data Type	Notes
isOptional	Boolean	Whether or not the device is optional
_id	[Object ID]	Identifier for specific BOM entry
quantity	int	amount of that device needed
deviceId	[Object Id]	Id of the device from the devices collection

Chapter 5

Hosting

5.1 Set up

First, note that because of compatibility issues with AWS Amplify at the time that we attempted hosting, we are running the latest version of Next.JS 11.

1. Host the code base for the application in a GitHub repository (Other services such as bit bucket will work as well, using just the files themselves may also work, but this guide will be written assuming that GitHub is the hosting repository in mind).
2. Obtain a GitHub account that has admin access to the repository, since this is required by AWS in order to host it on Amplify.
3. Log into AWS, and ensure that you are in the region where you would like the application to be hosted. We will now set up both the S3 bucket, and the Amplify application.
4. First, we set up the S3 bucket. Navigate to the S3 section of AWS. Create your bucket, and make sure to allow public access

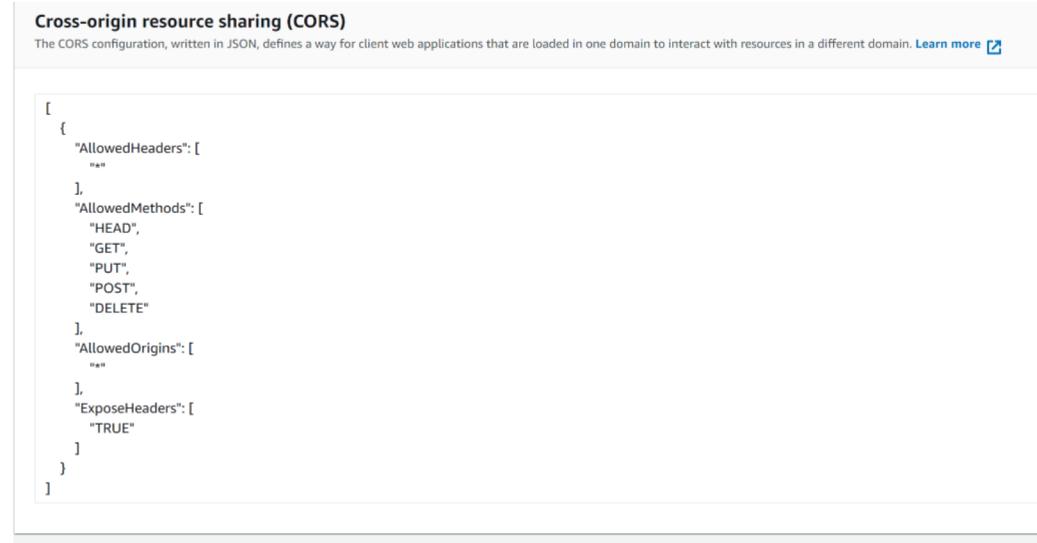


Figure 5.1 Set CORS Permissions

5. Set the CORS permission to be as follows:

```
[  
 {  
     "AllowedHeaders": [  
         "*"  
     ],  
     "AllowedMethods": [  
         "HEAD",  
         "GET",  
         "PUT",  
         "POST",  
         "DELETE"  
     ],  
     "AllowedOrigins": [  
         "*"  
     ],  
     "ExposeHeaders": [  
         "TRUE"  
     ]  
 }  
]
```

This can be found under the Permissions tab of the bucket. It should look like the above figure

6. Finally, set up a new user, to be used by the application, and give it read and write permissions to the bucket. Note down the Secret Key because this will not be accessible later.
7. Now, we will begin hosting on Amplify. Navigate to the Amplify application on AWS, and make sure that you are in your desired region.
8. Select new app, then host app, and choose to host from GitHub.
9. Connect your account which has admin access to the repository and select the application

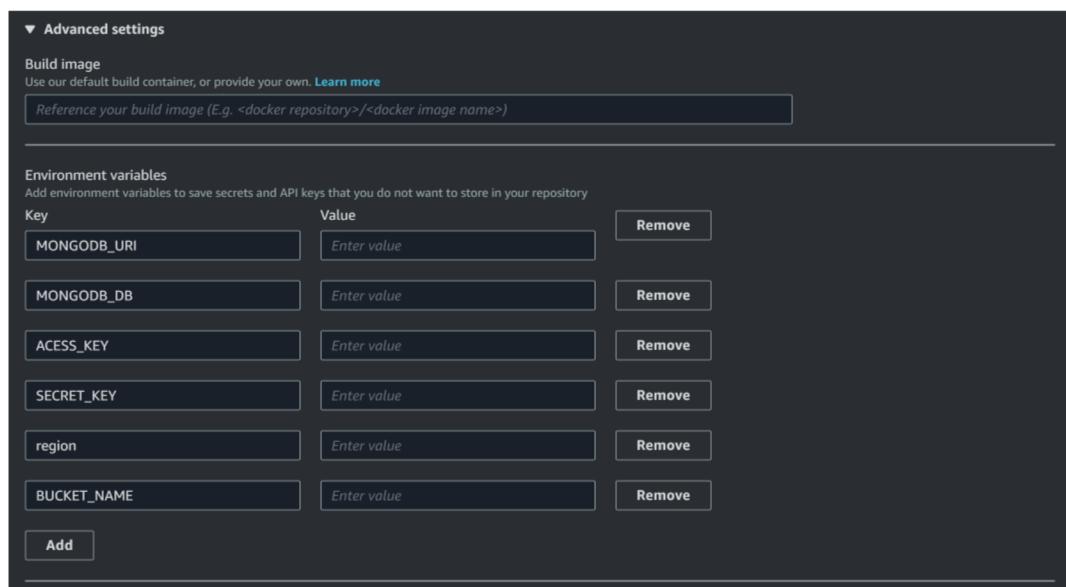


Figure 5.2 Set CORS Permissions

- Under advanced settings, add in the values for the following environment variables, depending on how they're hosted for you.

MONGODB_URI = URI string from Mongo to connect to the db instance
MONGODB_DB = database name in Mongo that stores all the data
ACCESS_KEY = Access key of the access account
SECRET_KEY = Secret key of that same account
region = region that the S3 bucket is hosted on
BUCKET_NAME = name of the S3 bucket

- Click next, and then deploy and the application should start building

5.2 Mongodb

- Set up a Mongo instance, and make sure to set permissions such that it can be written to and read from.
- Create a database for use by the application, this will be utilized in our environment variables when setting up the Amplify instance for hosting.

3. An outline of the schema can be found in the data schema section.
Create a Mongo instance with these parameters

Chapter 6

Frontend Architecture

6.1 Pages

The following is an exhaustive list of pages that are currently supported:

- Home: The home page shows all the existing engagements of the users.

Supported user actions on page:

- view Engagement details
- edit Engagement
- delete Engagement

- Engagement Details (Figure 6.1): After the user selects a specific engagement in the home page, they will see this page, which displays all the details and descriptions of the engagement. In addition, they will also see the active test plan, the archived test plans, and the summary BOM of the active test plan under this engagement.

Supported user actions on page:

- edit the Engagement details
- set archived Test Plan as active
- add new Test Plan
- view Test Plan Details
- delete (non-active) Test Plan

Engagement Details

Test Plans

Archived test plans:

Summary of Bill of Materials Elements (of active test plan)

ID	Device Name	Optional	Quantity	Physical/Software	Code Version	SKU	Actions
1	Router	False	50	Physical	none	323123	EDIT
2	Router	False	50	Physical	none	323123	EDIT
3	Router	False	50	Physical	none	323123	EDIT
4	Router	False	50	Physical	none	323123	EDIT
5	Router	False	50	Physical	none	323123	EDIT

Figure 6.1 Engagement Details

- Test Plan Details (Figure 6.2): This page displays all the details of the test plan. In addition, users will see all the test cases under the test plan and its summary BOM.

Supported user actions on page:

- edit the Test Plan details
- add new Test Case
- view Test Case Details
- delete Test Case

- Test Case Details (Figure 6.3): This page displays all the details of the test case. In addition, users will see all the tests under the test case and its BOM. Users can edit the test case details, add new tests, add new devices to the BOM, or click into a test to view further details.

Supported user actions on page:

- edit the Test Case details
- add new Test
- view Test Details
- delete Test

Test Plan Details

Details

Subject: *"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."*

Active: (Boolean)

Device Config:

Coverage:

Version:

Date Created:

Authors:

Customer Feedback

Detailed Description

Test Cases of Current Plan

ID	Subject	Description	Percent Passed	Topology	Device Config	Actions
1	Test case 1	This is a detail description of the test case	10%			<button>VIEW</button> <button>DELETE</button>
2	Test case 2	This is a detail description of the test case	20%			<button>VIEW</button> <button>DELETE</button>
3	Test case 3	This is a detail description of the test case	30%			<button>VIEW</button> <button>DELETE</button>
4	Test case 4	This is a detail description of the test case	40%			<button>VIEW</button> <button>DELETE</button>
5	Test case 5	This is a detail description of the test case	50%			<button>VIEW</button> <button>DELETE</button>

1-5 of 7 < >

Summary of Bill of Materials

ID	Device Name	Optional	Quantity	Physical/Software	Code Version	SKU	Actions
1	Router	False	50	Physical	none	323123	<button>VIEW</button> <button>DELETE</button>
2	Router	False	50	Physical	none	323123	<button>VIEW</button> <button>DELETE</button>
3	Router	False	50	Physical	none	323123	<button>VIEW</button> <button>DELETE</button>
4	Router	False	50	Physical	none	323123	<button>VIEW</button> <button>DELETE</button>
5	Router	False	50	Physical	none	323123	<button>VIEW</button> <button>DELETE</button>

1-5 of 9 < >

Figure 6.2 Test Plan Details

- add device to BOM
 - edit BOM device quantity and setting
 - delete device from BOM
- Test Details (Figure 6.4): This page displays all the details of the test. In addition, users will see all the results recorded under the test. Users can edit the test details and add new results.
- Supported user actions on page:
- edit the Test details
 - add new Result
 - view Result details
 - delete Result
- Result Details: This page displays all the details of a Result.
- Supported user actions on page:
- edit Result details
- Test Plan Library (Figure 6.5): Displays a table of all existing test plans.

Test Case Details

Subject: [EDIT](#)

Percent of Tests Passed:

Detailed Description

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Tests

ID	Subject	Description	Result Status	Actions
1	Test 1	This is a detail description of the test	Pass	DETAILS DELETE
2	Test 2	This is a detail description of the test	Unknown	DETAILS DELETE
3	Test 3	This is a detail description of the test	Fail	DETAILS DELETE

1-3 of 3 < >

Bill of Materials

ADD NEW

ID	Device Name	Optional	Quantity	Physical/Sw	Code Version	SKU	Actions
1	Router	False	50	Physical	none	323122	EDIT DELETE
2	Router	False	50	Physical	none	323123	EDIT DELETE
3	Router	False	50	Physical	none	323123	EDIT DELETE
4	Router	False	50	Physical	none	323123	EDIT DELETE
5	Router	False	50	Physical	none	323123	EDIT DELETE

1-5 of 5 < >

Figure 6.3 Test Case Details

Test Details

Subject: [EDIT](#)

Percent of Tests Passed:

Detailed Description

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

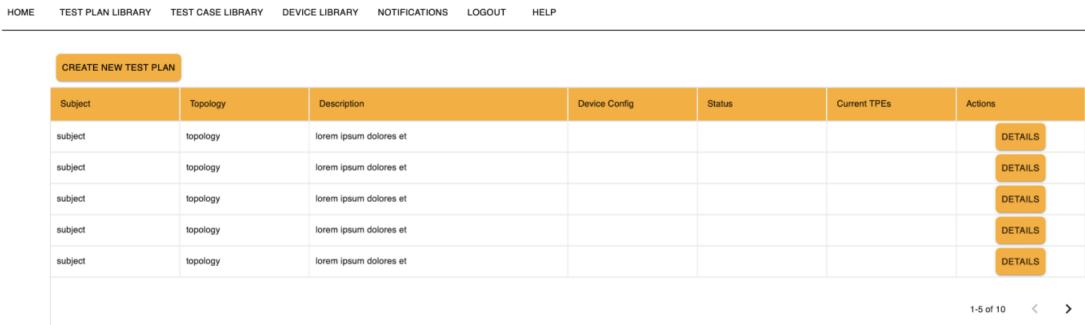
Results

ADD NEW

ID	Description	Result Status	Evidence	Actions
1	This is a detail description of the result	Pass		DETAILS DELETE
2	This is a detail description of the result	Unknown		DETAILS DELETE
3	This is a detail description of the result	Fail		DETAILS DELETE

1-3 of 3 < >

Figure 6.4 Test Details



The screenshot shows a web application interface. At the top, there is a navigation bar with links: HOME, TEST PLAN LIBRARY, TEST CASE LIBRARY, DEVICE LIBRARY, NOTIFICATIONS, LOGOUT, and HELP. Below the navigation bar is a table titled "TEST PLAN LIBRARY". The table has a header row with columns: Subject, Topology, Description, Device Config, Status, Current TPEs, and Actions. There are five rows of data, each representing a test case. Each row contains a "DETAILS" button in the Actions column. At the bottom of the table, there is a pagination indicator showing "1-5 of 10" and navigation arrows.

TEST PLAN LIBRARY						
Subject	Topology	Description	Device Config	Status	Current TPEs	Actions
subject	topology	lorem ipsum dolores et				<button>DETAILS</button>
subject	topology	lorem ipsum dolores et				<button>DETAILS</button>
subject	topology	lorem ipsum dolores et				<button>DETAILS</button>
subject	topology	lorem ipsum dolores et				<button>DETAILS</button>
subject	topology	lorem ipsum dolores et				<button>DETAILS</button>

1-5 of 10 < >

Figure 6.5 Test Plan Library

- Test Case Library: Displays a table of all existing test cases (similar to Figure 6.5).
- Device Library: Displays a table of all available devices (similar to Figure 6.5).

6.2 Components

Since there are components that can be reused in different pages and modals, we decided to first build out these common components. The components include

- menu bar
- plain screen
- split screen
- styled buttons
- drop-down
- text fields
- tables

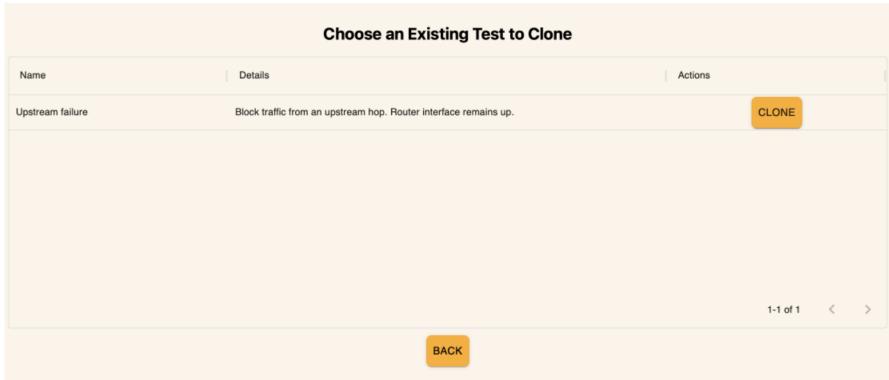
The screenshot shows a modal window titled "Fill in Engagement Info". It contains several input fields: "Name:" with a text input, "Customer:" with a text input, "SE:" with a text input, "POC Eng:" with a text input, "SFDC:" with a text input, and a large "Description:" field with a text input. At the bottom of the modal are two buttons: "BACK" and "DONE".

Figure 6.6 EngagementModalForm.jsx

6.3 Modals

The following is an exhaustive list of existing modals for reference. In the upcoming subsections, when and how the modals are used will be explained.

- Modal Forms: forms in which users can fill in/edit the information of the respective entity (e.g. Engagement Modal Form allows users fill in/edit the information of an Engagement, shown in Figure 6.6).
 - EngagementModalForm.jsx
 - TestPlanModalForm.jsx
 - TestCaseModalForm.jsx
 - TestModalForm.jsx
 - ResultModalForm.jsx
 - DeviceModalForm.jsx
- StartModal.jsx (See Figure 6.7)
Allows users to choose whether they want to create a new [Test Plan,

**Figure 6.7** StartModal.jsx**Figure 6.8** CloneModal.jsx

Test Case, or Test] from scratch or through cloning an existing one from the respective library.

- CloneModal.jsx (See Figure 6.8) Displays all the library [Test Plans, Test Cases, or Tests] such that users can choose one to clone from.
- selectDevice.jsx (See Figure 6.9) Displays all the devices from Device Library such that users can multi-choose device(s) they want to add to the corresponding BOM.
- selectQuantity.jsx (See Figure 6.10) Displays a selected list of devices and allows users to enter quantity desired(text-field) and whether it is optional (checkbox) for each entry.
- DeleteModalForm.jsx (See Figure 6.11) Allows users to confirm their decision of deleting an/a [Engagement, Test Plan, Test Case, Test, Result, or Device].

Add new device(s)

Device Name	Physical/Software	Code Version	SKU
5-yr NetCloud Branch Essentials Plan, Ad...	Physical	7.21.90	BAA5-2200120B-NN
5-yr NetCloud Enterprise Branch Essential...	Physical	7.21.10	BFA5-30005GB-GN
3-yr NetCloud Mobile Essentials Plan, Adv...	Physical	7.21.70	MAA3-0900600M-NA
Windows 10 Workstation	Software	Windows 10 Workstation	n/a
Ixia Test Client	software	1	n/a

1-5 of 5 < >

CANCEL **NEXT**

Figure 6.9 selectDevice.jsx

Double click to edit quantity and check box if optional

Device Name	Physical/Software	Code Version	SKU	Quantity	Optional?
5-yr NetCloud Branch Esse...	Physical	BAA5-2200120B-NN	1	<input type="checkbox"/>	
3-yr NetCloud Mobile Essen...	Physical	MAA3-0900600M-NA	1	<input type="checkbox"/>	

1-2 of 2 < >

BACK **ADD**

Figure 6.10 selectQuantity.jsx

Are you sure you want to delete this? This action is irreversible.

BACK **DELETE**

Figure 6.11 DeleteModalForm.jsx

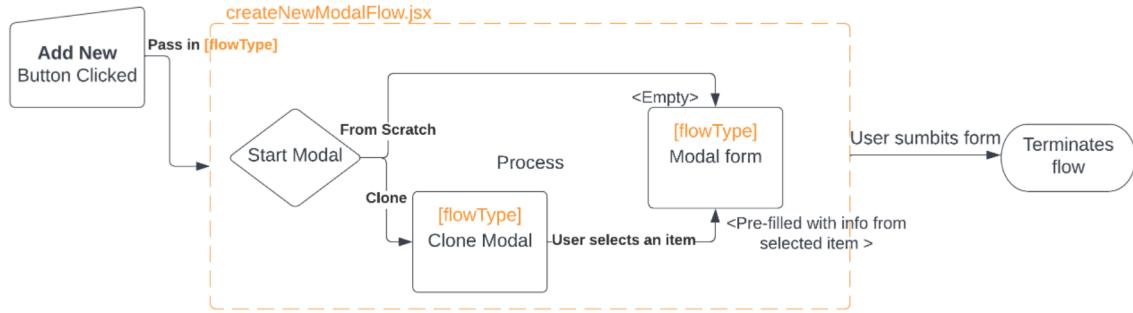


Figure 6.12 Create New Modal Flowchart

6.3.1 Create New Modal Flow

Recall that for Test Plan, Test Case, and Test, they each have a corresponding library that stores the pre-approved existing entries. Thus, when users want to create a new one, they have the option of creating from scratch or cloning an existing one from the library. Figure 6.12 shows the flowchart of the modal flow.

The corresponding details page contains a *CreateNewModalFlow* component which will be triggered once the user clicks on the "Add New" button. It will by default go to the Start Modal, passing in the *flowType* (e.g. Engagement), in which user can decide whether to create from scratch or clone. In the first case, an empty modal form of the corresponding entity will pop-up in which user can fill out the necessary information and submit. In the second case, a modal containing a table of all the library entries of that entity will show up. The user can choose one of them by clicking on the clone button and a modal form that is pre-filled with the information of the selected entry will show up. The user can edit the information and submit the form.

For entities like creating new Engagement, or Result, since there are no libraries, they will only have the create from scratch path.

6.3.2 Edit Modal Flow

The edit modal flow applies to Engagement, Test Plan, Test Case, Test, and Result. Figure 6.13 shows the flowchart of the modal flow.

In each details page, it contains a *EditModalFlow* component. Once the edit

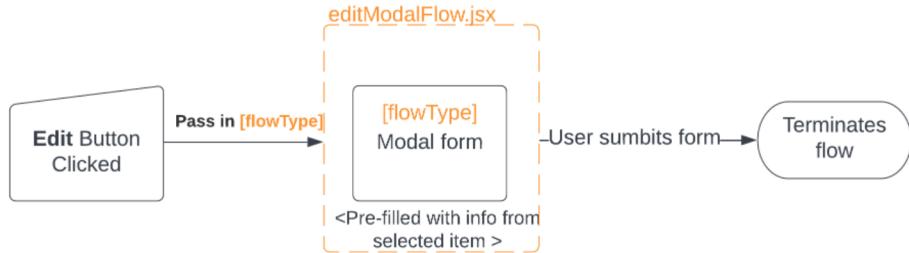


Figure 6.13 Edit Modal Flowchart

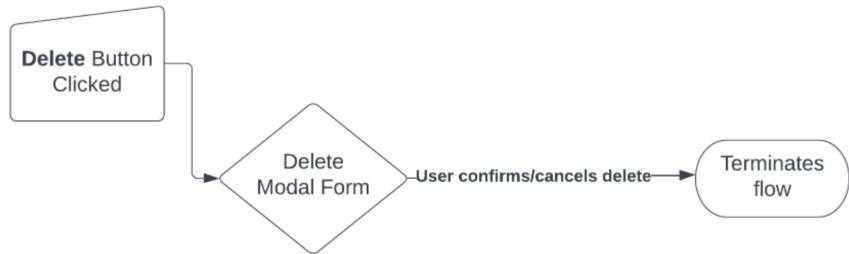


Figure 6.14 Delete Modal Flowchart

button is clicked, the corresponding modal form will pop-up with pre-filled information.

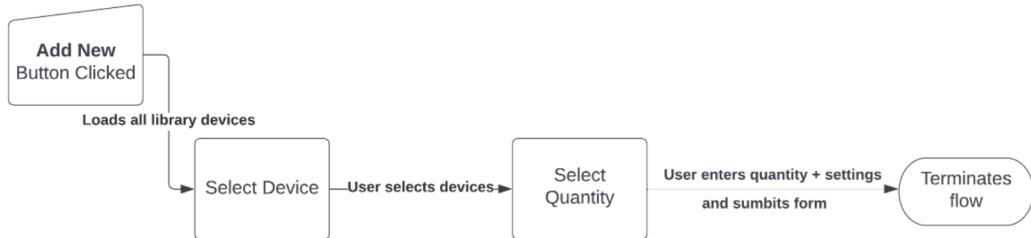
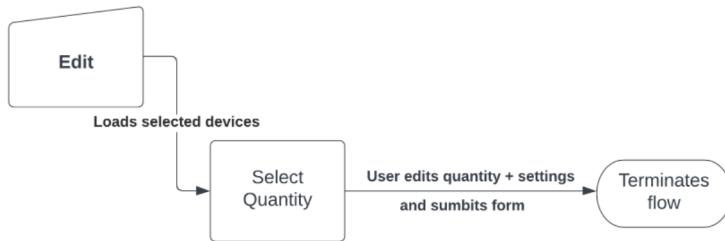
6.3.3 Delete Modal Flow

The delete modal flow applies to Engagement, Test Plan, Test Case, Test, Result, and BOM Device. Figure 6.14 shows the flowchart of the modal flow.

Clicking on the delete button will trigger a *DeleteModalForm* which asks the user to confirm the deletion.

6.3.4 BOM-related Modal Flows

Since the nature of device and BOM is very different from the other entities (i.e. Engagement, Test Plan etc.), they have their own create new and edit flow (though they have the same delete flow).

**Figure 6.15** Add Device to BOM Flowchart**Figure 6.16** Edit BOM Device Flowchart

Adding a device to BOM can only be done in the test case level. Thus, only the *TestCaseDetails* page contains the *SelectDeviceModal* and *SelectQuantityModal*. Figure 6.15 shows the flowchart of the modal flow.

Once user clicks on Add New button, the *SelectDeviceModal* is trigger and will display a table of all the library devices. Once user has selected all the devices they want to add to the BOM, clicking next will trigger the *SelectQuantityModal*. All the selected devices will be displayed in a table in this modal in which users can edit the quantity and whether the device(s) are optional. The flow is terminated once the user submits the form. Editing a type of device in BOM can also only be done in the test case level. Figure 6.16 shows the flowchart of the modal flow.

Once user clicks on the Edit button, the *SelectQuantityModal* is trigger and will display a table of the selected type of device, along with its current quantity and setting. The users can edit the quantity and whether the device(s) are optional. The flow is terminated once the user submits the form.

Chapter 7

Future Work

7.1 Known Bugs

To the best of our abilities we have attempted to fix bugs as we developed the web application. However, there were a few that escaped us in the midst of the time crunch. Below are some of the known bugs we have identified as well as some preliminary potential fixes that we have brainstormed:

1. **Bug:** Result status field for test not updating

Description: If a user edits the result status field of a result, the most recent result status filed of the parent test does not update automatically.

Steps to reproduce: Edit the result status of a result object.

Potential Fixes:

- Create an additional field under the test schema (in the database) to keep track of the most recent object
- Add a query that retrieves the most recent result object

2. **Bug:** Navigation directory is incorrect when back button is used or page is reloaded

Description: The navigation directory on the top left that tracks which layer the user is in will display the incorrect directory when the page is reloaded or when the back button is used.

Steps to reproduce: Enter an engagement and click into a couple nested layers to build the directory navigator. Then either reload the page or use the back button to see the bug.

Potential Fixes:

- Reconstruct the navigation directory upon every page load, and not in a React state (as it is currently doing now) which loses its state when the page reloads.
- Do a check before every page render to make sure we do not lose the state of the navigation, repopulate the state upon a reload to avoid losing the information.

7.2 Additional Data Validation and Testing

Within the docs directory of our project, there is a validation folder containing several json-like files that define MongoDB serverside validation rules, and instructions¹ for how to update them. Currently, these rules are set to only produce warnings instead of errors. MongoDB has documentation² which gives more information on what can be done with their schema validation.

While most of our testing focused on API's and the backend, we originally aimed to do automated Front-End testing with Selenium's Webdriver³. This would give us a way to test that each page has its required html elements working, such as buttons, modals, fields, and tables. We quickly realized that we would not be able to fully implement this, however, if desired, Selenium could be utilized and implemented in the future to continue testing of this application.

7.3 Recommended API Changes

Towards the end of our project, we made several improvements to the various API calls to make them more RESTful, have better error handling, and allow for greater code reuse. These improvements have largely been made only to the edit and add API's, and need to be extended to the get, clone, and delete API's. It may be helpful referring to section 4.2 for some of the API implementation details before reading the following recommended changes.

¹[Instructions](#) for validation

²[Documentation](#) for validation

³[Selenium](#)

For the delete API's, we recommend creating a delete directory, and renaming each relevant API, i.e. "pages/deleteTest.js" to "pages/delete/Test.js". We then recommend refactoring the delete functions in "utils/deleteEntry.js", such that each await call has try/catch handling of errors, and returns the proper http response code. Currently, each of these functions is placed within one large try/catch block, and returns the same 500 error message when any of the awaits causes an error.

The clone API's are in a similar state, and we recommend changes identical to those for the delete API's, particularly, changing the names of files and refactoring util functions for better error handling. The get API's, again, require similar changes, however there is currently no util for these API's.

For many of the get, clone, and delete API's, they currently only display a generic 500 status error when something goes wrong, making them very difficult to debug. These changes make it significant easier to debug any problems. There is also significant code-reuse among the API's and having most of their logic all in one util file makes it easier to share common functionality.

7.4 Recommended Schema Changes

This section details our recommend changes to the object schemas, that we did not have time to implement. Somewhat in order of increasing implementation difficulty, on the whole, these changes would requires significant refactoring to multiple files in the project on both the FrontEnd and the BackEnd.

1. **Change:** Remove redundant "createdOn" fields.

Description: Most object schemas currently contain a field named "createdOn". According to the mongoDB documentation⁴, each ObjectId can use a method to get the timestamp from when it was created, making this field somewhat redundant, since the "_id" field essentially store this already.

Refactors: Remove the "createdOn" field from each schema in which it is located. Then in each relevant add API, use the ObjectId's getTimestamp method to add a field to the returned json response.

⁴[Documentation](#) for ObjectId's getTimestamp

2. **Change:** Add a new "lastModified" fields.

Description: This change proposes instantiating the lastModified field with this method, and updating it with a new Date() timestamp on each edit.

Refactors: This would require an addition to each schema of a "lastModified" field. Each relevant edit API's would be refactored to update the "lastModified" field of the returned json response. To display this field, changes would need to be made to each details page to ensure that it is properly displayed.

3. **Change:** Replace "comments" fields with "feedback" field.

Description: Change the schemas with a String typed "comment" field to instead have an Object typed "feedback", consisting of entries of users, comments, and any nested replies.

Refactors: This is not likely to require any changes to current API's, but would require significant changing of each details page. These changes would likely include buttons (and potentially a new modal) to add, edit, and view comments.

7.5 Feature Extensions

7.5.1 Review and Iteration Process Extensions

It is intended that our application will be used to support the review and iteration process for creating and managing POC Engagements. This process involves having Cradlepoint's POC and SE engineers having access to edit and comment on each level of data, and be notified of any major approvals or changes to that data. To fully support this functionality, we recommend the following features to be implemented in the future:

- Creation of user accounts with differing level of permissions.
- User sign-in and logging of changes.
- Assigning tasks to different users along with a notification system to keep track of these tasks.
 - Creation of new API's to allow for email alerts.
 - Integration with Engagement and Testplan status, to send alerts for change status approvals.

<i>Future Work</i>	<i>Feature Extensions</i>
--------------------	---------------------------

- Flesh out success criteria of Test Plans and Test Cases based on the result of tests.
- Creation of more extensive comments, see section 7.4 for more details.

7.5.2 Quality of Life Extensions

Moving forward, there are a number of additional features we believe would significantly add to the user experience and useability of the web application. Below is a list of these features, in no particular order:

- File Upload and Download buttons for specific object fields such as topologies and configs.
- Export as PDF Button for Engagements to produce a client-friendly summary document.
- Additional Export Json Buttons for TestPlans, TestCases, and other levels.
- View and Edit Buttons for entries in each of the library pages.
 - Extend Edit modals to libraries and potentially create a view button and page to see the lower hierarchy objects attached to a library object.
- A general search feature, to find objects by Name or other fields.
 - Add a specific "tags" field to objects that can be used to find them with the search feature.
- API integration with SalesForce to allow for scheduling and tracking of information through the platform.
- Implement Versioning and Archiving to allow for storage, reuse, retrieval, and backing up of old data.

Acknowledgements

The Cradlepoint Clinic Team would like to thank each of the Cradlepoint liaisons, Lynn Gates, Aaron Ebling, Alex Terrell, James Hawk, and Chirs LaPeters, as well as our project advisor Professor Melissa O'Neill for their guidance and support over the course of the clinic program.