Vessel Vanguard 360

MADELABS TECHNOLOGY

11/12/2025 - Findings and Recommendations

AGENDA

01 Review the primary **OBJECTIVES**

02 Review the process we went through **TOGETHER**

03 Discuss findings and **RECOMMENDATIONS**

04 Review proposed **NEXT-STEP** projects

MADELABS
TECHNOLOGY

SOW Primary Objectives

Analyze current state of Vessel Vanguard 360 architecture and cloud configuration.

Discuss with Vessel Vanguard leadership and staff to better understand what is working well and what existing pain points or opportunities need to be addressed.

Propose improvements to software architecture including API, web, and mobile.

MADELABS
TECHNOLOGY

Vessel Vanguard Engagement Schedule

MadeLabs is spending four weeks with Vessel Vanguard to learn and evaluate the current state of the technology performance and reliability of the VV 360 application stack.

The next several slides will describe what we uncovered as well as our recommendations.

Discovery & Analysis

Review

Pre-Kick off	Week 1	Week 2	Week 3	Week 5
Initial overview and discovery conversations. Contract sign-off and schedule start date. Access set up.	Document core entities, relationships, architecture, data storage, cloud configuration, and existing integrations.	Identify performance issues in the admin. Identify performance issues for customers. Identify data loss issues.	Solution changes needed for performance and reliability improvements. Estimate high level effort and risk of changes.	Recommendation Write-up Review Recommendations Build Action Plan Schedule final review.

ARTIFACTS

01 VV-web Source Code Repository

02 VV-ios Source Code Repository

03 AWS Account

04 Current Application Architecture

05 Backend Lambda Function Summary

MADELABS
TECHNOLOGY

Who We Talked To

Executives:
Brian Hagan
Dan Roberts

Sales:
Rhiannon Silvashy

Operations and Administration:
Stephanie Post

MADELABS
TECHNOLOGY

Backup Strategy

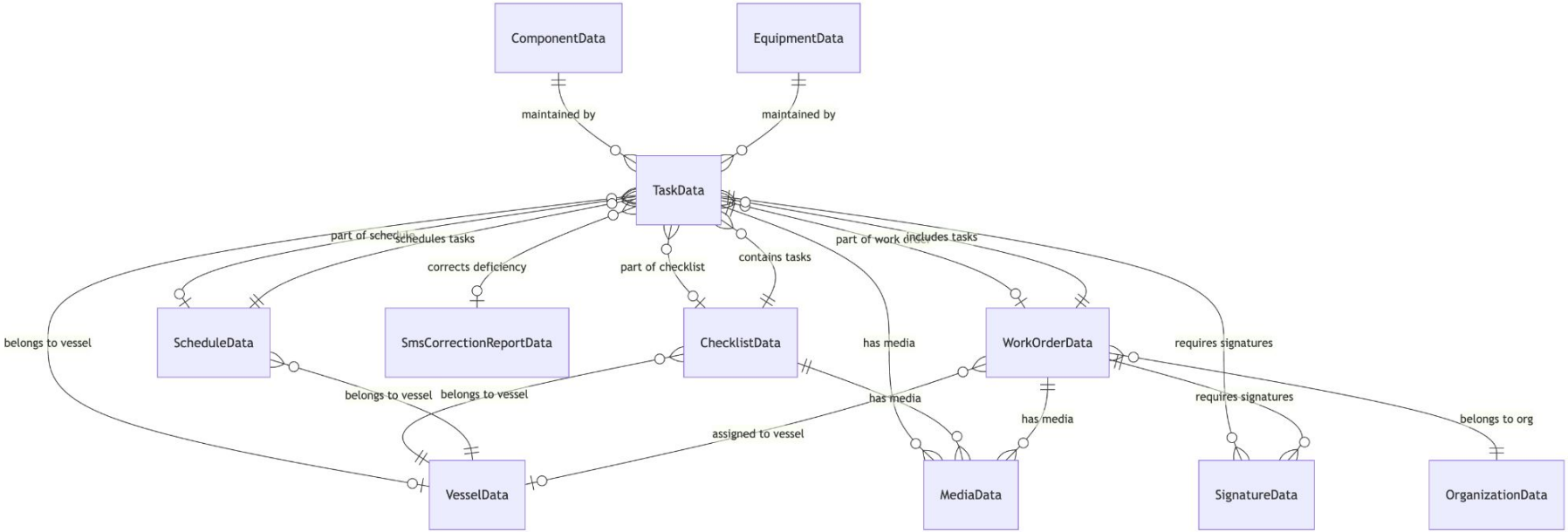
AWSBackup is set to run daily backups on the DynamoDB tables.
The backups are kept around for a 5 week retention period, and are deleted after 35 days.

This enables Point-in-Time Recovery for all **production** DynamoDB tables, so we can restore any table to any moment within that 35 day period.

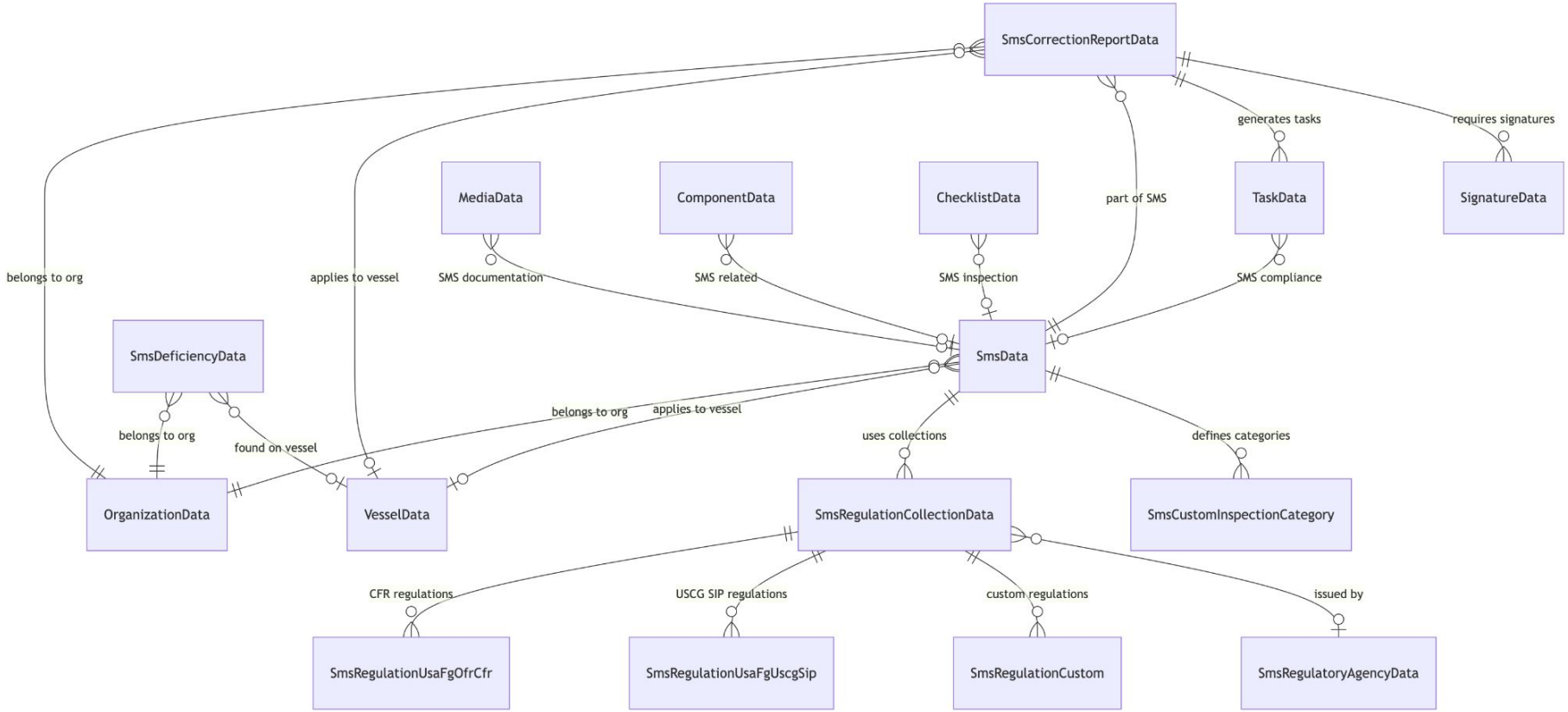
Recovery needs to be performed one table at a time, restoring to a new table so no data is overwritten.
Then the recovered data is manually moved from the recovered table to the production table.

No other backups have been found.

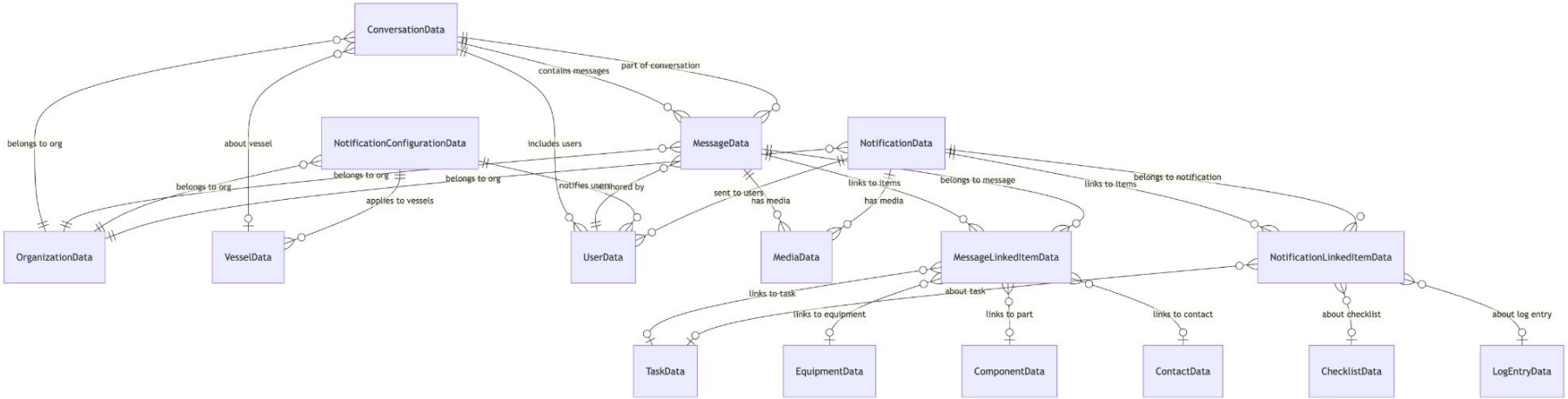
Maintenance & Work Management



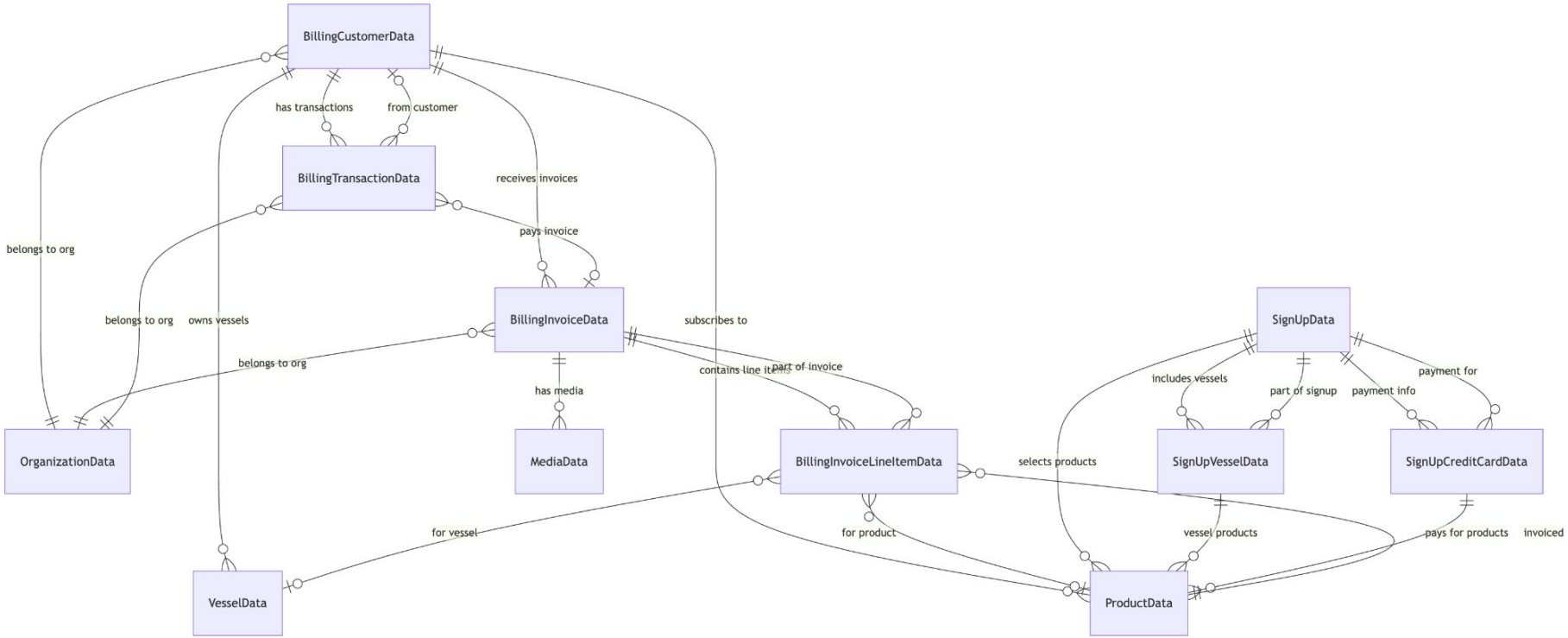
Safety Management System (SMS)



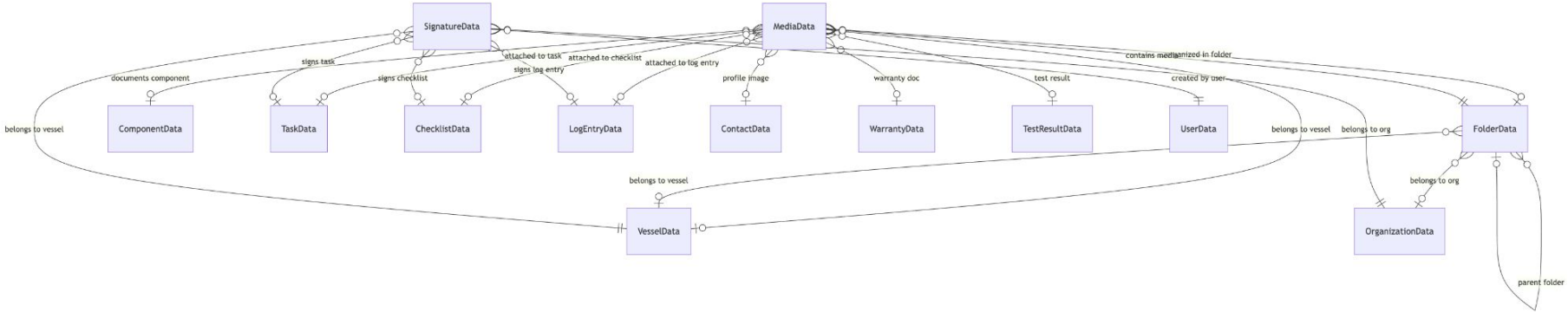
Communication & Notifications



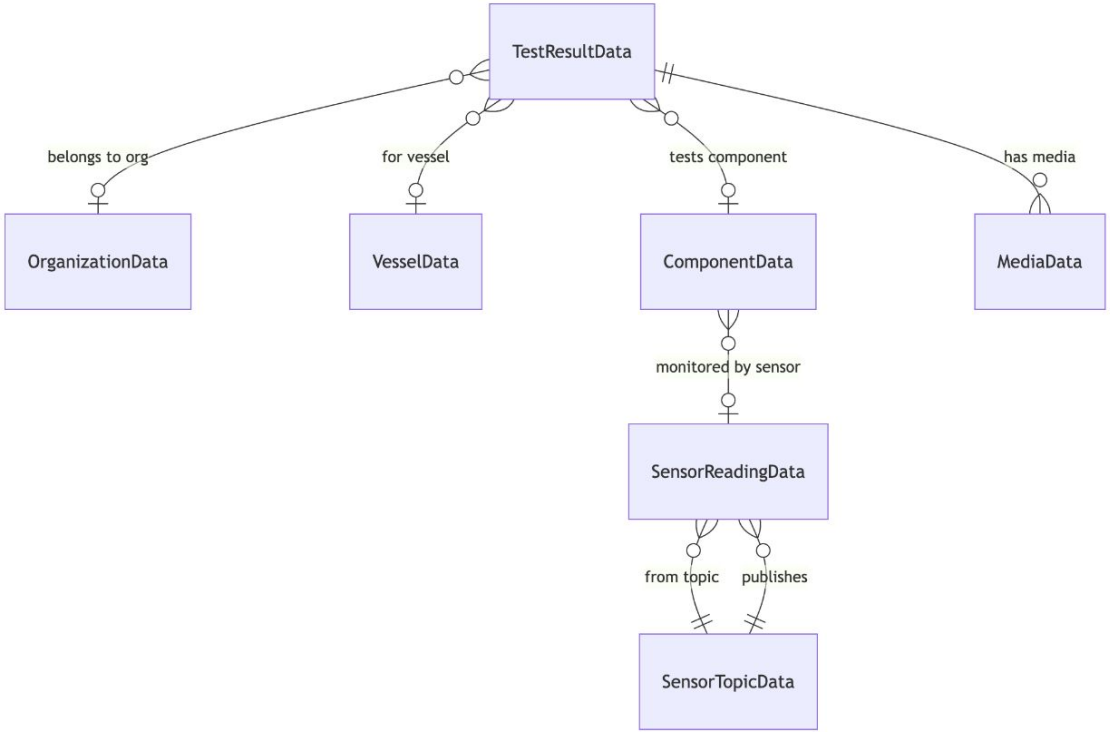
Billing & Sign Up



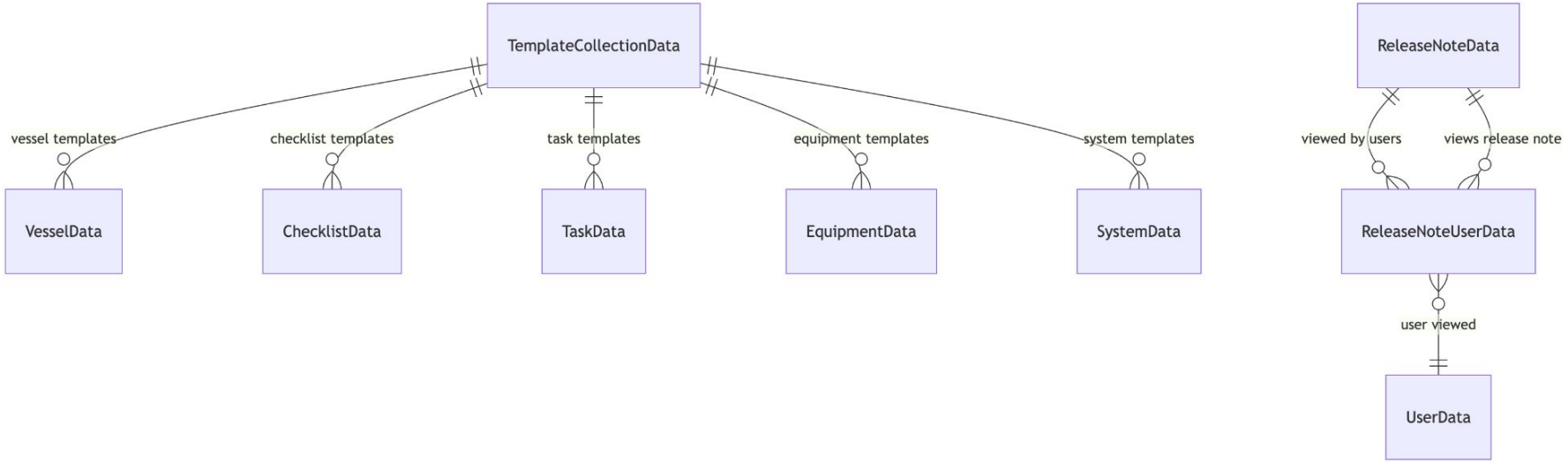
Media & Documents

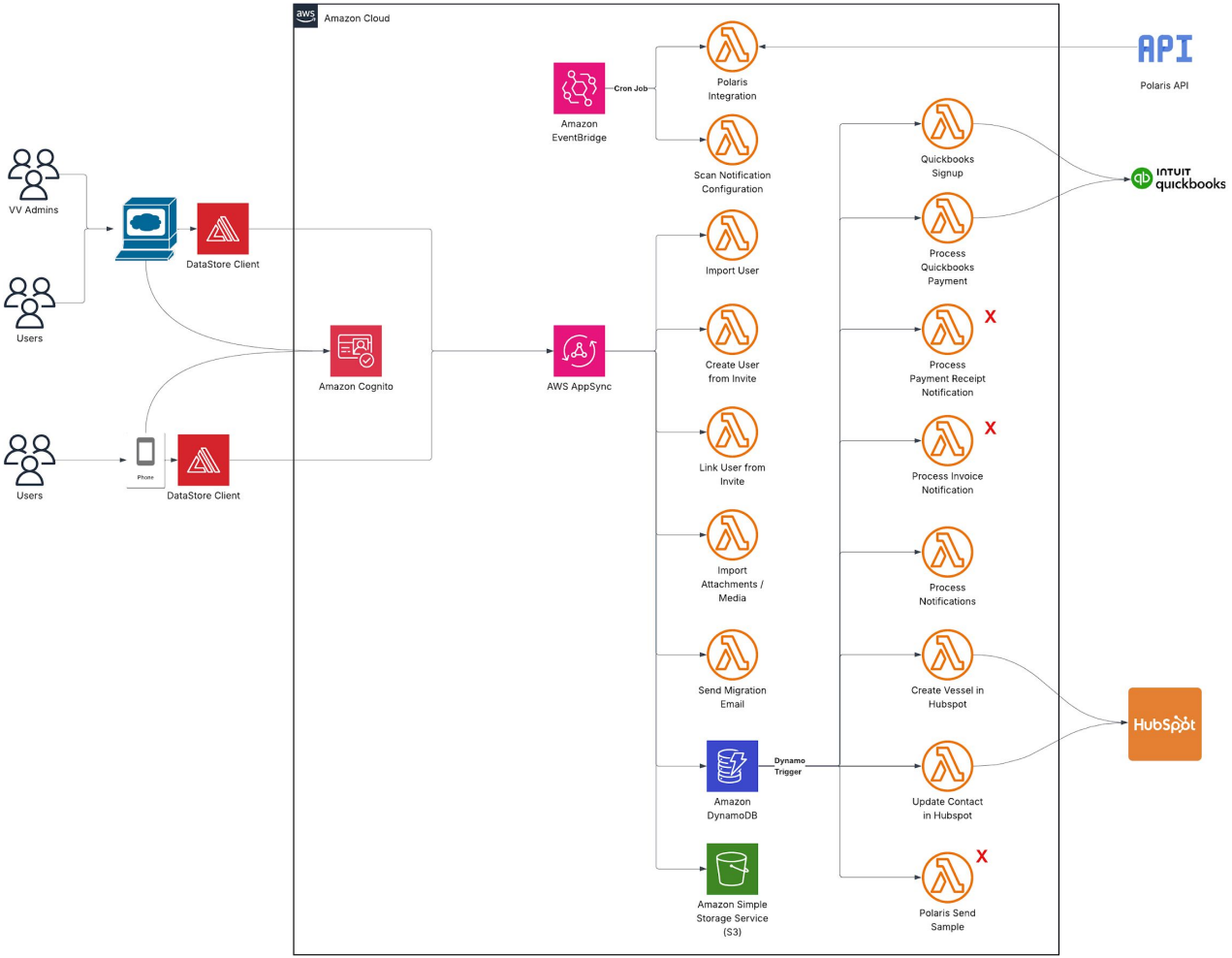


Testing & Sensors



Templates & Release Notes




Amazon Cloud
VV Admins
Users
DataStore Client
Users
Phone
DataStore Client
Amazon Cognito
AWS AppSync
Amazon EventBridge
Cron Job
Polaris Integration
Scan Notification Configuration
Import User
Create User from Invite
Link User from Invite
Import Attachments / Media
Send Migration Email
Amazon DynamoDB
Dynamo Trigger
Amazon Simple Storage Service (S3)
API
Polaris API
Quickbooks Signup
Process Quickbooks Payment
Process Payment Receipt Notification
Process Invoice Notification
Process Notifications
Create Vessel in Hubspot
Update Contact in Hubspot
Polaris Send Sample
INTUIT quickbooks
HubSpot

Findings and Recommendations

MADELABS
TECHNOLOGY

Certain Themes Emerged

- **Offline Support** is only partially implemented and needs a more robust approach.
- **Performance issues** are widespread and impacting both users and admins.
- **Data access and storage** does not match the needs of the application currently.
-

MADELABS
TECHNOLOGY

Offline-First Strategy

Finding:

The current application on web and mobile are significantly based off an “offline-first” strategy. This means all data is loaded into the browser on initial login. This results in extremely large load times that increase over time as more vessels are added to an account and more data related to those vessels are added.

This is likely an unsustainable approach as you scale and customer usage increases.

Recommendation:

For web: challenge why we need an offline-first approach. It is typical for web to be connected to the internet and be able to progressively (“lazy-load”) data as you need it.

For mobile: challenge what is needed in an offline scenario; likely it is only critical information to check SOP’s, checklists, access manuals, and maintenance requirements. However, it may not include all vessels for the account (only the active one you are out to sea on) or perhaps remove other data that is more historical in nature.

Next step:

Further conversation about true business needs of offline functionality and based on that, a revision to what data is loaded initially.

N+1 Data Retrieval Performance Problem

Finding:

The current querying of data (and technology choices used) result in a N+1 query pattern. For example, getting high level entities like “Vessel”, will result in the following conceptual flow:

1 query for vessel
10 queries for equipment related to that vessel
100 queries for components related to each equipment
200 queries for tasks associated with that vessel
etc.

Recommendation:

A relational database with proper indexed queries would in most cases cause this to be a single query which is typically significantly faster.

Next step:

Depends on if we are going to change database. If we do move to a relational database we need to rewrite the queries and indexes. If we do not, we need to discuss true offline needs and see what can be moved from eager-loading to lazy-loading techniques as changes to the graphql schema and resolvers.

DynamoDB: Your Database

Finding:

The database used is a technology called DynamoDB, which is what is called an access-pattern first database. This means it excels when you know exactly what you want to read in a very repeatable and high volume way. It is not a good database choice for deeply relational data that involves a lot of exploratory querying such as searching, applying filters, fetching related child records which is the primary value proposition of VV 360.

Recommendation:

Begin transitioning to AWS Aurora Postgres as a relational database.

Next step:

We recognize this is a **significant** change to the application stack. We have diagrammed the current relationships to demonstrate how relational the application currently is. We can move each major cluster of relationships into Postgres at a time and adjust the graphql resolvers or API to use the new data-persistence layer.

Admin: Initial Load is for All Vessels

Finding:

When logging in as an admin, all vessels are loaded initially with all of their related data (tasks, components, equipment, etc.).

Recommendation:

Do not load all data, instead load total counts if necessary and paged data on-demand as needed. Move away from Datastore for this use case and instead make direct API calls with pagination and filtering.

Next step:

Rewrite the current admin queries and functionality fixing the performance issues and also attempting to address some of the existing bugs and incomplete work.

Historical Data Archiving

Finding:

We have not found any strategies yet for archiving historical data or limiting the amount of data retrieved by time or volume in the current codebase. As systems are new or small, this usually does not surface as a problem. However, as they grow, there needs to be a thoughtful way to archive data or reduce the amount of data retrieved by business use case which still allows users to access what they need.

Recommendation:

Prioritize data retrieval with recency bias and discuss the business needs for data retention. Implement filters on data retrieval based on reasonable dates where high record counts exist. Provide a means of retrieving archived data upon request.

Next step:

Discuss the business needs further.

Web: Data Loss on Logout

Finding:

If a user is offline and has made changes which are stored locally, but selects to logout, all data in the datastore is cleared out before syncing can occur.

Recommendation:

Ensure that users know when logging out if the system is currently offline and that data loss may occur.

Next step:

Change code to not allow logout until sync has either completed or has been declined by user.

Web: Data Loss on Change of Vessel

Finding:

If a user is offline and has made changes which are stored locally, but selects to change vessel, all data in the datastore is cleared out before syncing can occur.

Recommendation:

Implement a selection of what vessels you need to run offline for a session (with a max number of vessels based on system limitation) in the frontend of the app.

Next step:

Design the frontend for vessel offline selection and ensure the user knows when syncing is required or that data loss may occur.

Admin: Data Loss on Switching Organization

Finding:

If an admin makes changes and the datastore has not yet had a chance to sync with the server but switches organization, the datastore immediately clears and may lose data.

Recommendation:

Ensure that admins know when logging out if the data is not yet synced and that data loss may occur if they continue.

Next step:

Replace administration functionality to not use datastore or at least to handle datastore events that signify syncing is in progress or complete.

Discuss whether admin should be its own completely different app for security reasons (or at least hide behind a WAF rule to prevent public internet access).

No Resolution to Conflicts

Finding:

There are instances via the datastore implementation that can cause conflicts in the state of the data based on changes to web or mobile while online or offline that rely on the datastore syncing. If the data is ever in conflict, the server wins silently and changes from the client may be replaced silently.

Recommendation:

We need to implement a conflict resolution handler; ideally one that is visual to be able to show the conflict to the user and allow them to choose what is right.

Note: AWS Amplify has built in conflict resolution strategies, however, we recommend taking this a step further so it is clear to the user what is happening.

Next step:

Mockup visual discrepancy resolution by user.

No Resolution to Conflicts (Examples)

Scenario A: Multi-Device Editing

1. User edits task on mobile app (offline)
2. User edits same task on web app (offline)
3. Mobile comes online first, syncs changes
4. Web comes online second
5. **Conflict detected**: Server has mobile version, client has web version
6. **Web changes are silently dropped** (AUTO_MERGE uses server version)

Scenario B: Simultaneous Edits

1. User A edits equipment description on desktop
2. User B edits same equipment notes on mobile (both online but race condition)
3. User A's save reaches server first
4. User B's save creates conflict
5. **User B's changes are lost**, User A's version wins
6. No warning to either user

Data Loss: Silently Failing Data Changes

Finding:

Throughout the application, a common pattern is used to prepare the record to be saved (regardless of type of record: vessel, contact, equipment, task, etc.) and then call the save() from the datastore. However, there is no error handling in case the save fails, resulting in a potential silent failure that the user may not be aware of.

Recommendation:

Add error handling and user notification when things go wrong and assume that always something will go wrong due to network errors, data validation, or bugs inadvertently introduced into the system.

Next step:

Mass update of error handling throughout the API and web application.

Directional Requirement Change: Onboard Local Server

Finding:

Much of the offline functionality is predicated on crews continuing to work while at sea and offline. However, the current storage is for one device only and state can not be shared across multiple crew members. The sync reconciliation is therefore accomplished by single device only to the server. Is the real need to have all crew members working together in a shared system while being offline, and then collective updates synced to the web server?

This is what Helm offers and past Vessel Vanguard customers have asked for.

This decision, changes the direction of performance, load testing, data handling, and several other key architectural decisions.

Note: Confirmed with Dan that we will not be tackling this at this stage in the product. Saved for future project needs with large customers.

Next Steps Action Plan

1. Resolve the issues via a new SOW covering:
    a. The minimally viable data for offline (with input from VV).
    b. Adjust the offline load functionality to be either selective sync or a single API load for offline endpoint by vessel(s).
    c. Rewrite admin to call data as needed instead of load entire datastore.
    d. Add reconciliation strategy and UI.
    e. Replace Dynamodb with Postgres relational database with rewrite of GraphQL resolvers and Data API.
    f. Create intentional offline strategy UI for selecting vessels or included information.
    g. Create intentional offline/online reconciliation UI for conflict resolution.
    h. Create server-side lambda functions for conflict detection to work in conjunction with conflict resolution.
    i. Implement error handling through application (visual UI and logic).
    j. Migrate data from DynamoDB to Postgres and setup new database backups.

MADELABS TECHNOLOGY

Open Questions or Blockers

1. Discuss offline requirements for the web.
2. Discuss offline requirements for mobile (and eventually ipad).

MADELABS
TECHNOLOGY