Associated Engineering
Team Discovery Channel
Internal Design Document

Thursday, April 5, 2018
Version 2.0

Document Information

Revision History

Date	Version	Status	Prepared by	Comments
2018/02/02	1.0	Submitted	Ryan Koon, Andrew Chang, Shih-Chun (Joyce) Huang, Nicky Chan, Qiushan Zhao, Sebastian Wels-Lopez, Alan Wong	Initial Design
2018/02/13	1.1	Submitted	Ryan Koon, Andrew Chang, Alan Wong	Updated programming environments and APIs Deployment process for client and links to hosted sites
2018/04/05	2.0	Submitted	Ryan Koon, Andrew Chang, Shih-Chun (Joyce) Huang, Nicky Chan, Qiushan Zhao, Sebastian Wels-Lopez, Alan Wong	Update APIs to match implemented APIs in Server project. Added architecture tree for Client

Stakeholders

Name	**Role**	**Contact Details**
Shawn Goulet	Sponsor (Main Contact)	(reachable via TAs)
Steve Robinson	Sponsor	(reachable via TAs)
Julin Song	TA	(reachable on Piazza)
Iris Ren	Supporting TA	(reachable on Piazza)
Anna Scholtz	Supporting TA	(reachable on Piazza)
Ryan Koon	Project Manager / Front-end Developer	ryankoon@alumni.ubc.ca
Andrew Chang	Front-end Lead / Developer	[redacted]
Shih-Chun (Joyce) Huang	Back-end Lead / Developer	[redacted]
Nicky Chan	Front-end Developer	(reachable via Front-end Lead)
Qiushan Zhao	Back-end Developer	(reachable via Back-end Lead)
Sebastian Wels-Lopez	Back-end Developer	(reachable via Back-end Lead)
Alan Wong	Back-end Developer	(reachable via Back-end Lead)

Table of Contents

Overview

- A records management system that can be integrated with other systems through the API
- Client-server setup
- All database access, CRUD operations will go through the server APIs
- Client-side requests are passed to the server to complete operations

Programming environment

Operating Systems

- Windows 10 x64
- macOS High Sierra
- Linux

IDE

- IntelliJ IDEA Ultimate 2018.1: https://www.jetbrains.com/idea/
- Webstorm 2017.3.4: https://www.jetbrains.com/webstorm/

Source Control

- Git : https://git-scm.com/
- GitLab : https://about.gitlab.com/

Languages/Frameworks/Libraries

Server

- Java 8: http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html
- Tomcat 7.0.84: https://tomcat.apache.org/tomcat-7.0-doc/index.html
- Javax Servlet API 3.1: https://docs.oracle.com/javaee/6/api/javax/servlet/package-summary.html
- Spring Boot 1.5.10.RELEASE
- MySQL Server 5.7.21 : https://www.mysql.com/ (5.7.19 on AWS RDS)
- Maven

Client

- JSX: https://jsx.github.io/
- Node: https://nodejs.org/en/
- Yarn: https://yarnpkg.com/en/
- Bootstrap: https://getbootstrap.com/
- React: https://reactjs.org/

Project Tracking

- Slack - For team communication: https://slack.com/
- GitLab : https://about.gitlab.com/

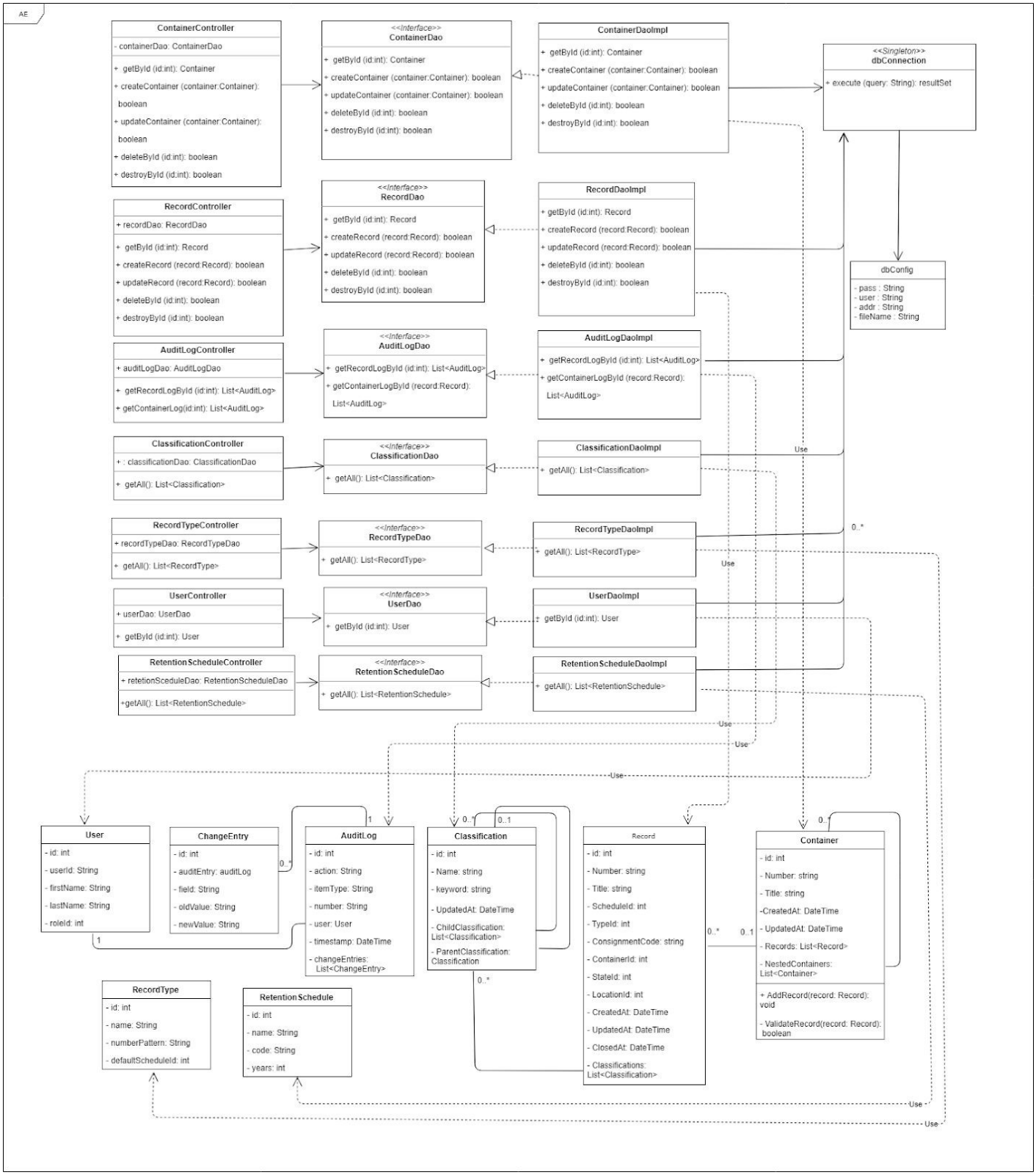
Cloud Hosting

- Azure App Service: https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/template/README.md#deployment
- Amazon Web Services EC2, and RDS: https://aws.amazon.com/rds/mysql/

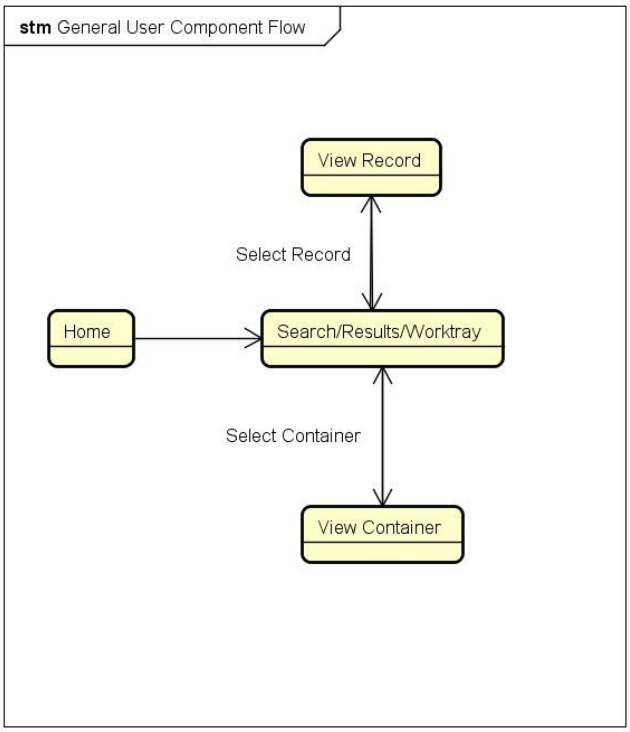
Production and Test environments

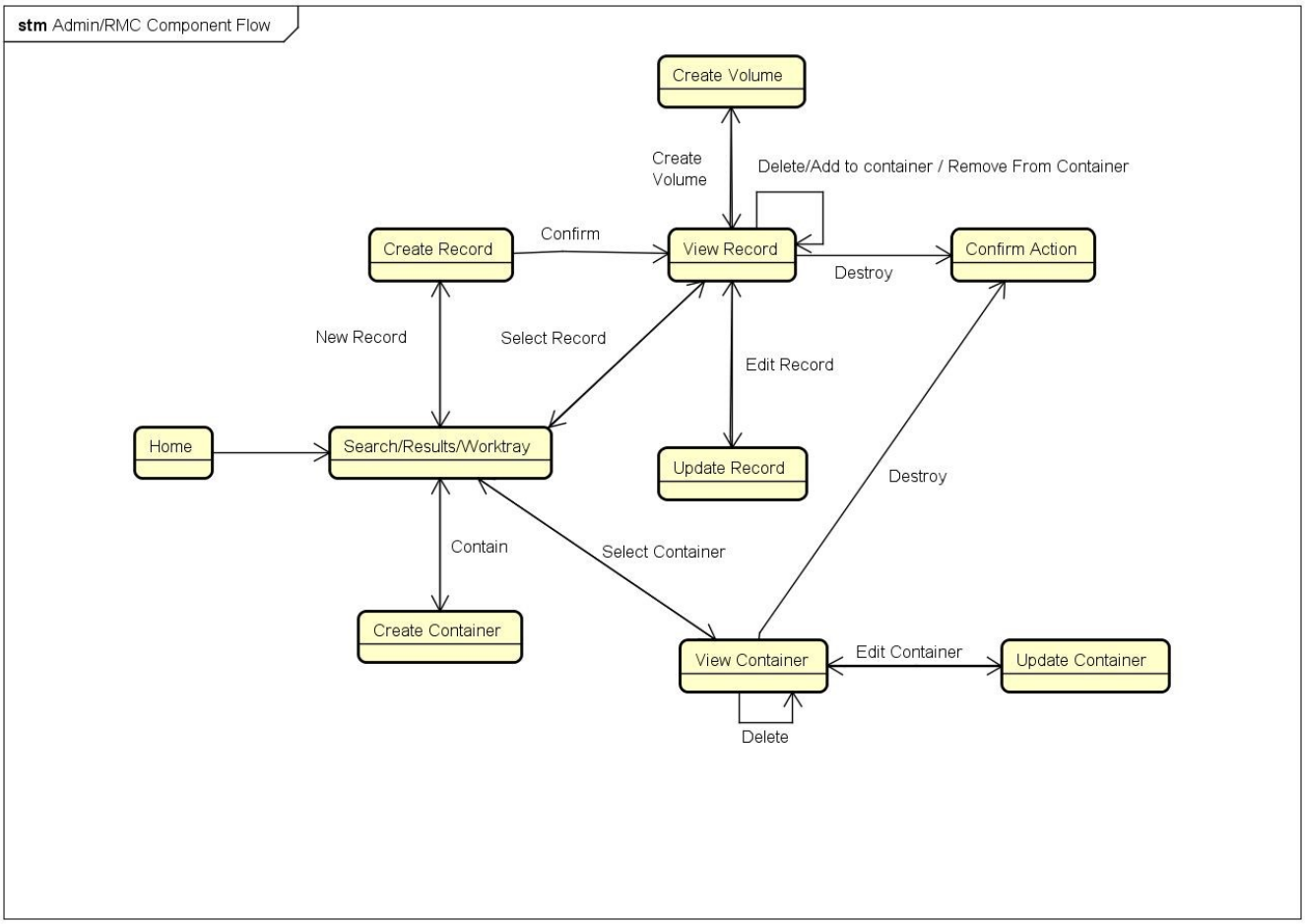
- Testing locally on development machines
- Production ready builds will be pushed to the cloud (AWS + Azure)
- Using JUnit testing framework for Java backend
- Automation testing with Cypress and Storybook for frontend
- Manual UAT for front-end development

Software architecture (Server)

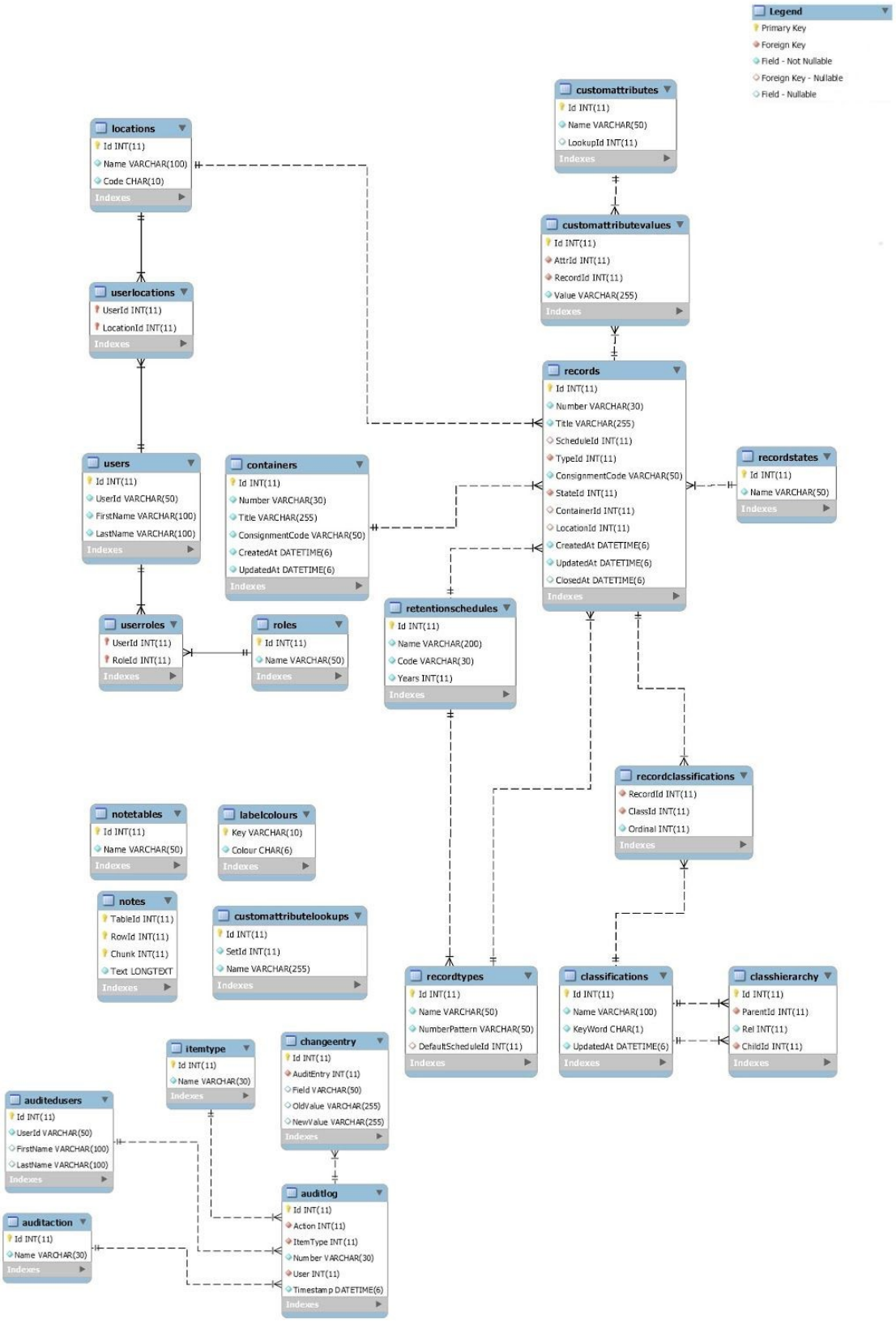


Software architecture (Client)




stm Admin/RMC Component Flow
Create Volume
Create Volume
Delete/Add to container / Remove From Container
Create Record
Confirm
View Record
Confirm Action
Destroy
New Record
Select Record
Edit Record
Home
Search/Results/Worktray
Update Record
Destroy
Contain
Select Container
Create Container
View Container
Edit Container
Update Container
Delete

Data design


Legend
Primary Key
Foreign Key
Field - Not Nullable
Foreign Key - Nullable
Field - Nullable
customattributes
Id INT(11)
Name VARCHAR(50)
LookupId INT(11)
Indexes
locations
Id INT(11)
Name VARCHAR(100)
Code CHAR(10)
Indexes
customattributevalues
Id INT(11)
AttrId INT(11)
RecordId INT(11)
Value VARCHAR(255)
Indexes
userlocations
UserId INT(11)
LocationId INT(11)
Indexes
records
Id INT(11)
Number VARCHAR(30)
Title VARCHAR(255)
ScheduleId INT(11)
TypeId INT(11)
ConsignmentCode VARCHAR(50)
StateId INT(11)
ContainerId INT(11)
LocationId INT(11)
CreatedAt DATETIME(6)
UpdatedAt DATETIME(6)
ClosedAt DATETIME(6)
Indexes
users
Id INT(11)
UserId VARCHAR(50)
FirstName VARCHAR(100)
LastName VARCHAR(100)
Indexes
containers
Id INT(11)
Number VARCHAR(30)
Title VARCHAR(255)
ConsignmentCode VARCHAR(50)
CreatedAt DATETIME(6)
UpdatedAt DATETIME(6)
Indexes
recordstates
Id INT(11)
Name VARCHAR(50)
Indexes
userroles
UserId INT(11)
RoleId INT(11)
Indexes
roles
Id INT(11)
Name VARCHAR(50)
Indexes
retentionschedules
Id INT(11)
Name VARCHAR(200)
Code VARCHAR(30)
Years INT(11)
Indexes
recordclassifications
RecordId INT(11)
ClassId INT(11)
Ordinal INT(11)
Indexes
notetables
Id INT(11)
Name VARCHAR(50)
Indexes
labelcolours
Key VARCHAR(10)
Colour CHAR(6)
Indexes
notes
TableId INT(11)
RowId INT(11)
Chunk INT(11)
Text LONGTEXT
Indexes
customattributelookups
Id INT(11)
SetId INT(11)
Name VARCHAR(255)
Indexes
recordtypes
Id INT(11)
Name VARCHAR(50)
NumberPattern VARCHAR(50)
DefaultScheduleId INT(11)
Indexes
classifications
Id INT(11)
Name VARCHAR(100)
KeyWord CHAR(1)
UpdatedAt DATETIME(6)
Indexes
classhierarchy
Id INT(11)
ParentId INT(11)
Rel INT(11)
ChildId INT(11)
Indexes
itemtype
Id INT(11)
Name VARCHAR(30)
Indexes
changeentry
Id INT(11)
AuditEntry INT(11)
Field VARCHAR(50)
OldValue VARCHAR(255)
NewValue VARCHAR(255)
Indexes
auditedusers
Id INT(11)
UserId VARCHAR(50)
FirstName VARCHAR(100)
LastName VARCHAR(100)
Indexes
auditlog
Id INT(11)
Action INT(11)
ItemType INT(11)
Number VARCHAR(30)
User INT(11)
Timestamp DATETIME(6)
Indexes
auditaction
Id INT(11)
Name VARCHAR(30)
Indexes

API design

Records:

Get record by record id	`GET serviceRoot/records/{id}`
	
	`Parameters:`
	`UserId - Integer<Required>`

Get multiple records	`GET serviceRoot/records`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`Ids - List of Integers`

Get most recently updated records (limit 20)	`GET serviceRoot/records`
	
	`Parameters:`
	`UserId - Integer<Required>`

Search records and containers with pagination	`GET serviceRoot/search`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`Num - String<Required>`
	`Record - Boolean`
	`Container - Boolean`
	`Page - Integer`
	`PageSize - Ingeger`

Create a new record	`POST serviceRoot/record`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`Number - String <Required>`
	`Title - String <Required>`
	`ScheduleId - Integer <Required>`
	`TypeId - Integer <Required>`
	`ConsignmentCode - String`
	`StateId - Integer`
	`ContainerId - Integer`
	`LocationId - Integer <Required>`
	`ClassIds - List of Integer`
	`Notes - String`

Get root classifications	`GET serviceRoot/classifications`

Get child classifications	`GET serviceRoot/classifications`
Get all record types	`GET serviceRoot/recordtypes`
Get all record states	`GET serviceRoot/recordstates`

Delete record(s)	`DELETE serviceRoot/record`
	
	`Parameters:`
	`UserId - Integer<Required>`
	
	`Request Body:`
	`RecordIds - List of integers`

Update a record	`PUT serviceRoot/record/{id}`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`Title - String`
	`ScheduleId - Integer`
	`ConsignmentCode - String`
	`StateId - Integer`
	`ContainerId - Integer`
	`LocationId - Integer`
	`Notes - String`

Destroy record(s)	`PUT serviceRoot/destroyRecords`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`RecordIds - List of Integers`

Create a volume	`POST serviceRoot/volume{id}  ← record id`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`CopyNotes - Boolean`

Get volumes by number	`GET serviceRoot/volume`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`Num - String`

Example serviceRoot/Requests (for	`{` `"Id":51,`

GET/POST/PUT requests) response body	"Number":"EDM-2003/001", "Title":"Sample Record", "ScheduleId":26, "ScheduleName":BUSINESS, "TypeId":3, "ConsignmentCode":"DESTRUCTION CERTIFICATE 2009-01", "StateId":6, "State":6, "ContainerId":24365, "ContainerNumber":2007/014-EDM, "LocationId":5, "LocationName":Edmonton, "CreatedAt":"1063677156", "UpdatedAt":"1063677156", "ClosedAt":"1063677156" "ClassificationIds":[3,4,5,6] "Notes":"This is a note!" }

Containers:

Get container by container Id	GET serviceRoot/containers/{id} Parameters: UserId - Integer<Required>
Get multiple containers	GET serviceRoot/containers Parameters Ids - array of ids UserId - user id Example request: http://localhost:8080/containers?ids=24608,24613,28413&userId=500 Example response: [{ "containerNumber": "2007/020-KEL", "title": "Esse placeat sunt fugiat ducimus doloribus",

	"notes": "", "containerId": 24608, "createdAt": 1176361200000, "updatedAt": 1488528000000, "destructionDate": null, "consignmentCode": "517402708", "stateId": 4, "state": "Archived - Interim", "locationId": 21, "locationName": "Kelowna", "scheduleId": 639, "scheduleName": "PROJECT RECORDS WITH 30 YEAR RETENTION", "typeId": 83, "type": "Project", "childRecordIds": [4107, 4112, 4114, 4116, 4315, 4361, 5138] }]
Create new container	POST serviceRoot/container Parameters: UserId - Integer<Required> Number - String <Required> Title - String <Required> LocationId - String <Required> Records - List of Integers Notes - String
Get destruction date (To preview destruction date before creating container and to validate selection)	GET serviceRoot/container/destructiondate Parameters: RecordIds - Comma separated list of integers (one or more) <Required>

Update container	`PUT serviceRoot/container{id}`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`Title - String`
	`LocationId - Integer`
	`StateId - Integer`
	`ConsignmentCode - String`
	`Notes - String`

Delete containers	`DELETE serviceRoot/containers`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`ContainerIds List of Integers`

Response body example	`Container:`
	`{`
	`"containerId":10749,`
	`"containerNumber":"2006.001-TES",`
	`"title":"Sample Container",`
	`"consignmentCode":"362817350",`
	`"createdAt":1063677156,`
	`"updatedAt":1063677156,`
	`"notes":"This is a note!",`
	`“Destruction Date”: 1063677156`
	`"ChildRecordIds":[2,3,4]`
	
	`}`

Get destruction date	`GET serviceRoot/containers`
	
	`Parameters:`
	`UserId - Integer<Required>`
	`RecordIds - List of Integers`

Get most recent close at date	`GET serviceRoot/container/{id}/closedAt`
	
	`Parameters:`
	`UserId - Integer<Required>`

Users:

Get user	`GET serviceRoot/users{id}`
Response body example	`{`

	"Id": 42, "userId": "swels-lopez", "firstName": "Sebastian", "firstName": "Wels-Lopez", "roleId": 1 }

Record Types:

Get record type	`GET serviceRoot/recordTypes`

Response body example	[{
	"id": "10",
	"name": "CASE RECORDS",
	"numberPattern": "XXX-ZZZ/NN",
	"defaultScheduleId": "322"
	}, {
	"id": "3",
	"name": "subject",
	"numberPattern": "KKK-yyyy/ggg",
	"defaultScheduleId": ""
	}, {
	...
	}]

Retention Schedules

Get retention schedules	`GET serviceRoot/retentionSchedules`

Response body example	[{
	"id": "2",
	"name": "INFORMATION MANAGEMENT - ACQUISITION",
	"code": "I1.A1.01",
	"years": "3"
	}, {
	"id": "4",
	"name": "BUSINESS DEVELOPMENT - COMMITTEES",
	"code": "B1.C2.00",
	"years": "1"
	}, {
	...
	}]

Classifications

Get classifications	`GET serviceRoot/classifications`

Response body example	<pre>[{
	"id": "2",
	"name": "COMPLIANCE",
	"keyword": "F",
	"updatedAt": "2003-10-10 19:00:48.000000",
	"parent": "",
	"children": [3,4,5]
	}, {
	"id": "3",
	"name": "SAMPLE",
	"keyword": "G",
	"updatedAt": "2003-10-10 19:00:52.000000",
	"parent": 2,
	"children": ""
	}, {
	...
	}]</pre>

Audit logs:

Get audit logs for record	`GET serviceRoot/auditLog/record('record id')`
Get audit logs for container	`GET serviceRoot/auditLog/container('container id')`

Response body example	<pre>[{
	"AuditId":1,
	"Action":"UPDATE",
	"ItemType":"Record",
	"Number":"EDM-2003/001",
	"User": {
	"UserId":1,
	"FirstName":"Sebastian",
	"LastName":"Wels-Lopez"
	},
	"Timestamp":"2003-09-15 18:52:36.000000"
	"ChangeEntries":{
	[{
	"Field":"Title"
	"OldValue":"initial title"
	"NewValue":"updated title"
	},{
	"Field":"LocationId"</pre>

	<pre>"OldValue":"3"
	"NewValue":"2"
	}]
	}
	},{
	...
	}]</pre>

Algorithms

- There are no complex algorithms in this system
- Mainly basic CRUD operations

Addressing Technical Requirements

System Availability

- Out of scope for our deliverable in terms of providing ways to maintain uptime during patches
- Our design will be modified and integrated into AE's system
- We will not be addressing server downtimes for upgrades and bug fixes.

Capacity – including average and peak concurrent usage

- The provided schema and data is manageable locally in our development machines so we will not be making any specific optimizations for this
- It is more likely an infrastructure or cost issue to host a performant database in the cloud
- Estimated database size of 133,000 records (legacy records)
- ~1000 users, 20-30 peak concurrent
- Capacity is largely up to how much server space AE is willing to provide for the application

Performance – end user timings a key

- We will look into ways we can optimize database but with the given data and minimal changes to the database for auditing, we do not anticipate changes required to meet a response time of about 2 seconds in the UI.
- Operations that involve the database will be requested asynchronously from the client to avoid blocking the UI
- For large number of record information, we will paginate response from search to limit load time

Scalability

- Anticipated annual growth of ~2000 records

- Given adequate space for storage on AE servers, we do not anticipate annual growth of 2000 records to be a problem for the first couple of years
    - If adequate space is not provided, we need to discuss if it is possible and how to remove records from the DB (ie FIFO basis) if space is an issue

Security

- Authentication: out of scope, we will check the given user id with our user table
- Authorization: we will check the user's roles and locations to control access to records
- Security will be integrated with AE existing authentication system and will only be mimicked by checking user's role in query string in this application

Maintenance

- Given consultation with sponsor, we can implement Matomo on landing page to track visitor metrics and annotations
    - Similarly, we can implement a custom alert banner that can be only be create by admins that can display messages to alert for server downtime, new patches, etc

- The client will use yarn to install package dependencies
- We will deploy our node application using EC2, Elastic Beanstalk, and S3
    - This will connect to an AWS RDS instance hosting the MySQL database

Client deployment to Azure

- Commits on master are automatically deployed to staging site on Azure from GitLab
- After manual verification on staging, trigger deployment to production site on Azure

Staging: https://staging-aeclient.azurewebsites.net
Production: https://aeclient.azurewebsites.net

UI design and screen mockups

Wireframe and UI Design - Please access via https://ex27e9.axshare.com

Notable trade-offs

- Keeping consignment code in the Container table will remove the need to access its containing records for the code.
- Since backup is out of scope we will keep all the audit entries.
- After handoff, offloading old entries to another database for storage is recommended.
- Tables related to the auditing will have identical fields to other tables such as user fields and record numbers. We intentionally avoid foreign key references to modifiable tables (non-auditing) because entries could be deleted. Instead we take snapshots of the user information at the time of change and which will not be affected by deletion or modification.

Notable risks

- As indicated in the API section, some APIs need to follow the ODATA 4 specification in order to be able to integration with other systems
- Unable to test authentication system with AE: assuming that inserting uid a query string will integrate with existing system
- Unable to test with AE's internal systems to guarantee integration compatibility
- Unable to test for peak usage for application with many concurrent users (~20 people)
- Unable to handle any problems that may arise after course finishes (no maintenance phase after production launch)
- Not familiar with deployment to cloud with many moving parts (NodeJS for frontend, Java for backend, mySQL for database)