Team RMAD
Josh Dardis
Jake Lind
Shankar Sampath
Roya Shams

High and Low Design

High-Level Design Categories:
-Security
-Hardware
-User Interface
-Internal Interfaces
-External Interfaces
-Architecture
-Reports

Security

For our application, we will be using a database. The database security is of utmost importance (as well as user credentials). We want to make sure that we don't have any SQL/NoSQL Injection attacks, buffer overflow attacks, denial of service attacks, or potentiality to be exploited by malware (to the best of our ability).

To achieve security in our program in bare bones, we will be doing the following:

- Distributed Infrastructure
- Tight regulatory requirements
- Encrypt credentials and encrypt all files/information to the best of our best ability.
- Disable network access.
- Encourage them to regularly update the program as they release, regularly patch servers and regularly patch programs.
- Lock down accounts, lock down database, lockout logins, database role and privileges
- recommend any physical security dependent on what it's installed on.

We will tackle these security concerns by engineering/addressing solutions:

- The database has password requirements (12 characters, 1 uppercase, 1 lowercase, 1 number, 1 symbol)

- Audit trail - we will have a log file that will track and store each user who modifies and views specific records. This will be a history table that will record a user's name, a link to the record that was modified, and the date where this occurred. This will help keep track if someone might have tampered an entry.

- We will have encryption for our username and password - this will be taken care of by Trackhive, as the user will need to have a Trackhive account to utilize the app.
- Encryption of the email will be taken care of by the user's choice of email (i.e. using a gmail account, use Google's security. Using business account, under the umbrella of the business' security)

- Be sure that the database communicates locally rather than from a network or through the internet.

- We'll engineer a lockdown of a database account after 3 attempts of a login (to prevent brute force) with a downtime of 10 to 30 minutes Each lockdown will increase in time before being banned for the entire day.

- A reset password option in the instance there'll be a need for it.

- Roles for users that will have relative privileges (i.e. administrator, editor, reader, user, include race conditions, etc). Can create a role too, if need be.

Hardware

CPU: Intel Core i5-4690 3.5GHz / AMD A10-7800 APU 3.5 GHz or equivalent
CPU SPEED: Info
RAM: 4 GB
OS: Windows 8 and above

(Basically any kind of Windows machine from Windows 8 and above)

The user will also need to have a working email.

User Interface

For the user interface we will be creating a Windows Forms App using C#. This will allow for easy design options.

- The first screen will be a login screen. This will be used for connecting to an api we will be using for pulling package data.

- The next screen will be a homepage that will include basic tracking data for each package, a general setting button, and a more information button for each package.

- The general setting screen will hold settings for the majority of packages.

- The group screen will allow for users to see groups they are a part of. They will also be able to create groups form this screen

- Within the individual groups screen the user will be able to add/remove users from a group

- From the reports window the user will be able to send reports to their email. This page will allow options such as the amount of time the report will cover and which type of report to send.

- From the individual shipment window the user will be able to see an expanded history of the shipment. The user will also be able to change individual notifications on a package.

- The last screen is for an extended history of the package. It will also include options for changing the frequency of individual packages.

- For now past packages will be under the home page together with current shipments. In the future these will be separated between current and past shipments. In this future version the home screen will only hold shipments in transit and there will be a separate tab holding complete shipments.

- Refresh to be added to design

Josh will be working with the user interface.

Shankar will be developing the database.

Jake will be implementing the classes from the low-level design.

Roya will be in charge of making dummy data and possibly creating a docker.

All members will test the application.

Link to UI Design:
https://docs.google.com/presentation/d/1nQoaSpBjq-W9-Pg6A907eIFIBfDUUiwGAxbqO_izafs/edit?usp=sharing

Internal Interface

The components of this program will have:
-API from TrackHive
-SQLite local database
-User Interface
-Outside email.

-Outside calendar (potentially an invite for a Google Calendar or an Outlook Calendar)
-Configuration file
-Tracking number validation
-Database backup
-User roles

We will have a working email that is from an outside source. This is where the user will be receiving the shipping information (the ID, the carrier, the destination or date). A user will be ordering something and the confirmation of the package purchase and package information will be sent to the outside email.

From there,email will then be exported or the data will be transferred into the SQLite database. We will have an entry created within the database - categorizing and arranging the information by either carrier, date, or priority of parcel.

Tracking number validation will be a specific algorithm to help confirm that it is a working and proper tracking number and that it was properly placed within the entry.

Application will call the TrackHive API with a token generated for each tracking ID and get the real time status of the package and update in the dashboard and database.

The user can log into the user interface made by us using TrackHive username and password. Here, the user can set up parcel tracking and notifications, sending tracking information to the user or other users (tracking number, location, carrier, extra information).

Customizable notification system - should be able to customize the frequency of the email updates.

User roles will determine whether or not the user who has logged in will be able to read, write, edit entries. User roles will determine what they can view and what they can do.

Configuration files will be the overhead of these processes, with the email server credentials and information, database location, type and credentials, deliver Time Check intervals per tracking time left, "'dwell" times for database entries and carrier information.

Database backup will be a process that runs daily in the background as all of these processes are happening for data storage and data backup, should the database be corrupted or lost.

External Interfaces

Trackhive API system.

Email (business email or personal email).

A program that opens .csv..

Calendaring.

Architecture

Monolithic, component based, event-driven, data-centric, role-based
Database Options: SQLite, RethinkDB, MariaDB, MySQL

We will be going with SQLite because it is lightweight and easy to use. It also has the ability to be used with C#, which we have decided to use for the program.

Reports

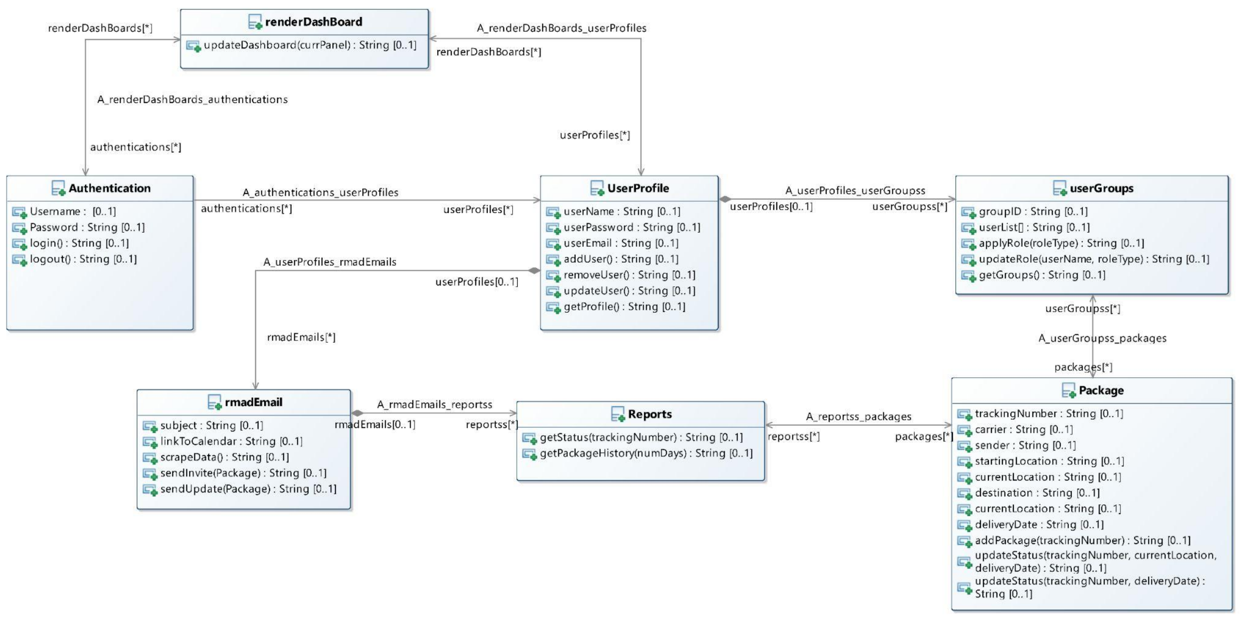
For this project, we will be building the local database in SQLite. We will be using relational databases to organize the data that we retrieve (tracking number, location, carrier, status, extra information) .The entry will be created after this information is retrieved.

We will audit changes .There will be a log file that will record each user who modifies and views specific records. This will be a history table that will record a user's name, a link to the record that was modified, and the date where this occurred.

For database maintenance we will hold shipment data up to 6 months. This will be live data within the application and accessible. Past 6 months, data will be exported and appended to a .csv file to make sure we do not end up with a bloated/too big of a database to run. This .csv file will be used for creating reports, processing information, searching information, and tracking history.

We would like to have a search function and organizational options (organize by carrier, date, priority of parcel) within the application, and is considered a very big and high priority, but we do not consider it necessary for now. If we have extra time, we will develop this..

UML


renderDashBoard
updateDashboard(currPanel) : String [0..1]
renderDashBoards[*]
A_renderDashBoards_userProfiles
renderDashBoards[*]
A_renderDashBoards_authentications
authentications[*]
userProfiles[*]
Authentication
Username : [0..1]
Password : String [0..1]
login() : String [0..1]
logout() : String [0..1]
A_authentications_userProfiles
authentications[*]
userProfiles[*]
UserProfile
userName : String [0..1]
userPassword : String [0..1]
userEmail : String [0..1]
addUser() : String [0..1]
removeUser() : String [0..1]
updateUser() : String [0..1]
getProfile() : String [0..1]
A_userProfiles_userGroupss
userProfiles[0..1]
userGroupss[*]
userGroups
groupID : String [0..1]
userList[] : String [0..1]
applyRole(roleType) : String [0..1]
updateRole(userName, roleType) : String [0..1]
getGroups() : String [0..1]
A_userProfiles_rmadEmails
userProfiles[0..1]
rmadEmails[*]
userGroupss[*]
A_userGroupss_packages
packages[*]
rmadEmail
subject : String [0..1]
linkToCalendar : String [0..1]
scrapeData() : String [0..1]
sendInvite(Package) : String [0..1]
sendUpdate(Package) : String [0..1]
A_rmadEmails_reportss
rmadEmails[0..1]
reportss[*]
Reports
getStatus(trackingNumber) : String [0..1]
getPackageHistory(numDays) : String [0..1]
A_reportss_packages
reportss[*]
packages[*]
Package
trackingNumber : String [0..1]
carrier : String [0..1]
sender : String [0..1]
startingLocation : String [0..1]
currentLocation : String [0..1]
destination : String [0..1]
currentLocation : String [0..1]
deliveryDate : String [0..1]
addPackage(trackingNumber) : String [0..1]
updateStatus(trackingNumber, currentLocation, deliveryDate) : String [0..1]
updateStatus(trackingNumber, deliveryDate) : String [0..1]

Low-Level Design Categories

-OO Design
-Identifying Classes
-Building inheritance hierarchies
-Refinement of our classes
-Generalization of our classes
-Object composition
-Database design
-Relational databases
-First Normal Form

OO Design

We will be using the C# programming language, so we will be creating classes that define the general properties and behaviors for the set of objects throughout this program. We will use instances for processes.

Classes

Authentication

- Username
- Password
- login()

- logout()

UserProfile
- userName
- userPassword
- userEmail
- addUser()
- removerUser()
- updateUser()
- getProfile()

userGroups
- groupID
- userList[]
- applyRole(roleType)
- updateRole(userName, roleType)
- getGroups()

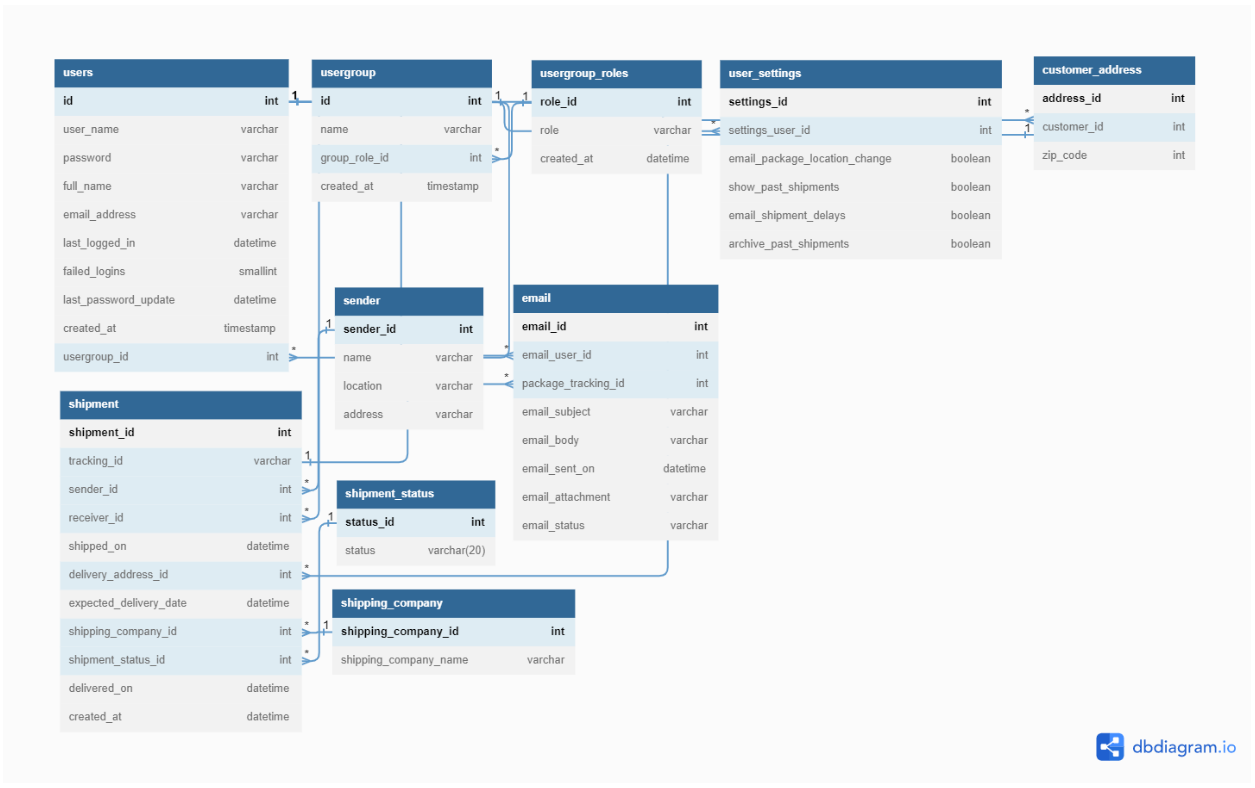
Package
- trackingNumber
- Carrier
- sender
- startingLocation
- Destination
- currentLocation
- deliveryDate
- addPackage(trackingNumber)
- updateStatus(trackingNumber, currentLocation, deliveryDate)
- updateStatus(trackingNumber, deliveryDate)

Reports
- getStatus(trackingNumber)
- getPackageHistory(numDays)

renderDashboard
- updateDashboard(currPanel)

rmadEmail
- Subject
- linkToCalendar
- scrapeData()
- sendInvite(Package)
- sendUpdate(Package)

Relational Database


users
id int
user_name varchar
password varchar
full_name varchar
email_address varchar
last_logged_in datetime
failed_logins smallint
last_password_update datetime
created_at timestamp
usergroup_id int
usergroup
id int
name varchar
group_role_id int
created_at timestamp
usergroup_roles
role_id int
role varchar
created_at datetime
user_settings
settings_id int
settings_user_id int
email_package_location_change boolean
show_past_shipments boolean
email_shipment_delays boolean
archive_past_shipments boolean
customer_address
address_id int
customer_id int
zip_code int
sender
sender_id int
name varchar
location varchar
address varchar
email
email_id int
email_user_id int
package_tracking_id int
email_subject varchar
email_body varchar
email_sent_on datetime
email_attachment varchar
email_status varchar
shipment
shipment_id int
tracking_id varchar
sender_id int
receiver_id int
shipped_on datetime
delivery_address_id int
expected_delivery_date datetime
shipping_company_id int
shipment_status_id int
delivered_on datetime
created_at datetime
shipment_status
status_id int
status varchar(20)
shipping_company
shipping_company_id int
shipping_company_name varchar
dbdiagram.io