WheresMyShip / Team 8

Purdue CS307 Project Design Document

Purpose

As we move forward into the future, more and more consumers are making their purchases online rather than in stores. Why drive 40 miles to buy that rare video game when you can get it shipped with Amazon Prime overnight? Online shopping is fantastic in that it allows for customers to buy and receive whatever they may need from the comfort of their own home. Online shopping, however, requires the inconvenience of getting an item shipped. Shipping is not as fast as buying an item in store and can be quite expensive. It is also difficult to keep track of a package without having to constantly look up a tracking number.

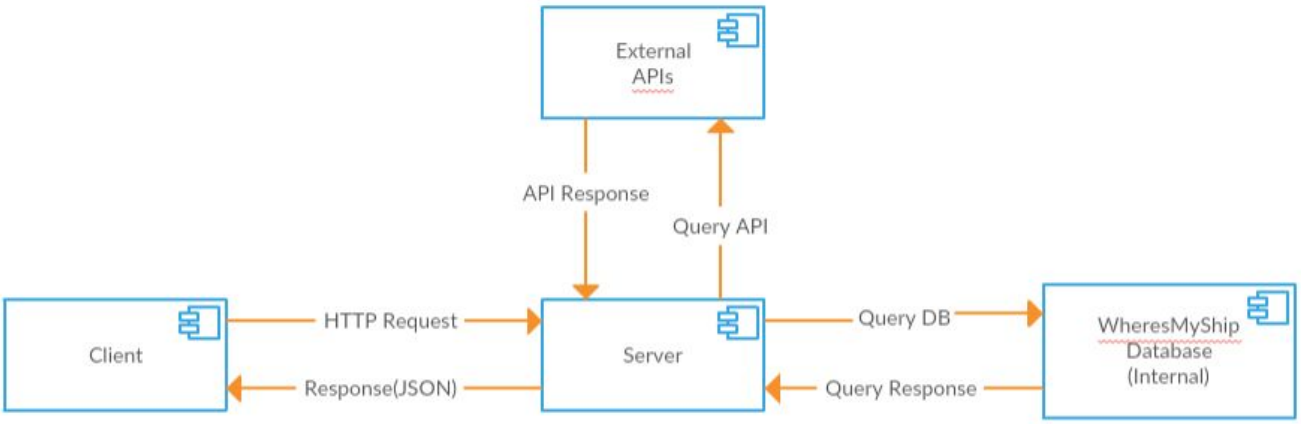
This is where WheresMyShip comes in. WheresMyShip will help online shoppers to keep track of all of their package shipments without the fuss of manually entering a tracking number.

What makes WheresMyShip different from other package tracking apps you ask? WheresMyShip will have a feature that automatically adds a package to the app to be tracked when a shipment confirmation is received, so that the app is completely hands free after installation and login.

Design Outline

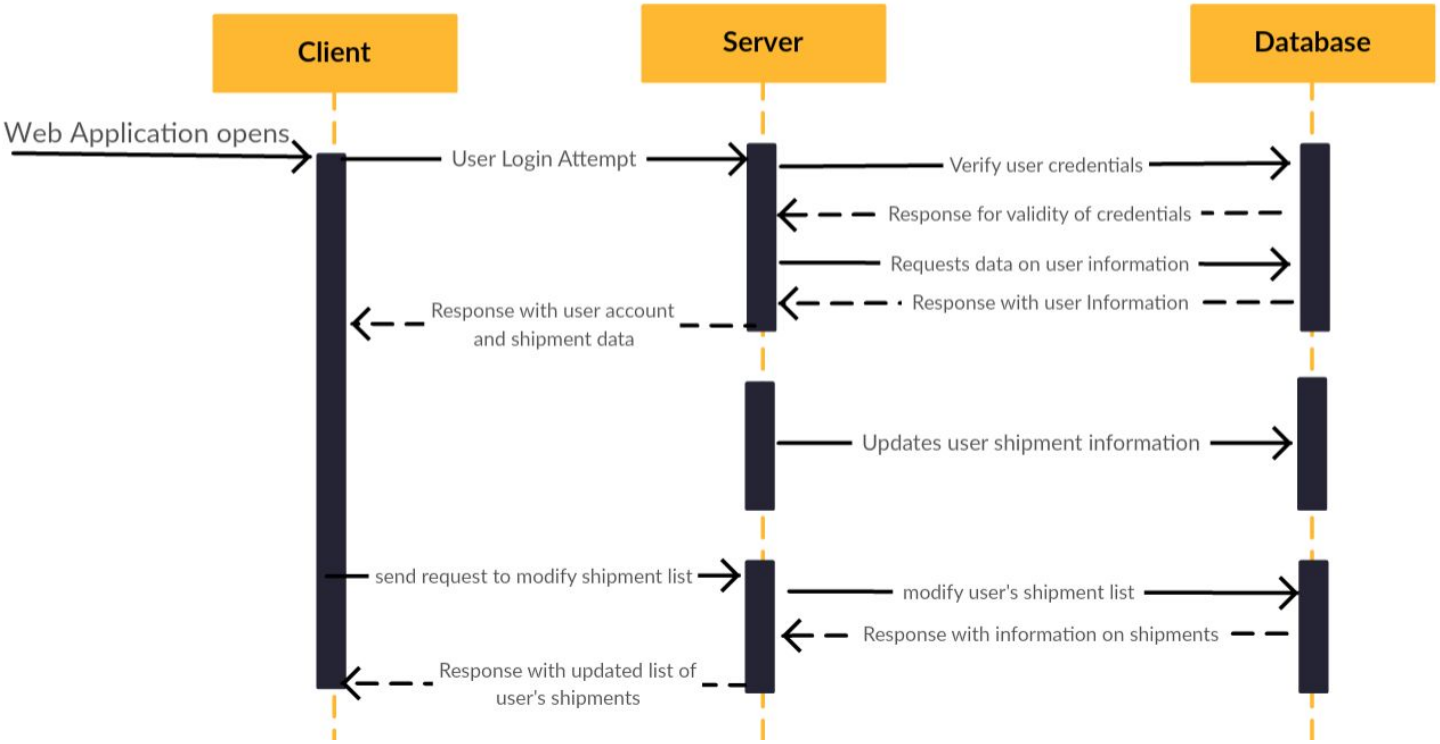
Our project is a service that facilitates shipment tracking for users by automating the process of indexing all the users tracking numbers and retrieving up-to-date shipment information. This service will be implemented using the client-server model, where multiple clients will be requesting resources from the server. The server will be using the Model-View-Controller(MVC) architectural pattern.

High-Level Overview:



- Client
  - Sends and requests information from the server based off of the user's interactions.
  - Displays shipment information affiliated with the user's WheresMyShip account.
  - The client view will be within the browser, and it will display the user interface where the user can view account and shipment information affiliated with the user's account.
- Server
  - Receives and validates requests from the client.
  - Processes the requests and send responses to the client with information from the database.
  - Selects appropriate view for the client
- Database
  - Keeps track of all service information such as user account(e.g user's name, email, shipments) and shipment information (e.g shipment's location, carrier, source)
  - SQL database to allow for efficient access of information via SQL queries

Sequence Diagram:



Design Issues

Functional Issues:

How will users sign in to the service?

Option 1: Login to the service using Google

Option 2: Login to the service using a WheresMyShip account

Option 3: Login to the service using either a Google account or a WheresMyShip account

Option 4: No login is required

Choice: We decided that it will be best for users to be able to sign in with either their WheresMyShip account or their Gmail. Our service will likely be scanning through user's Gmail accounts to find shipping confirmation, so it would be wise to allow people to simply sign in with their Gmail rather than force them to sign in with a WheresMyShip account and then have to give the app access to their Gmail account anyways.

How should the log of packages be displayed?

Option 1: Display a list that shows all of the package information for each package

Option 2: Display a list that shows some of the package information, and will display full information upon being clicked on

Choice: We decided that it would be best for the app to display a list with all of the packages along with some package information for each package, so that the app can display more packages on the screen at once and will allow users to navigate through their list of packages faster and easier, and still allows the user to grab more detailed info on the package if needed by just tapping on the package entry in the list.

How should the app notify users of a change in their shipment?

Option 1: Send push notifications to the user's browser or device

Option 2: Send email to the user's email account

Option 3: Send a text message to the user's phone

Choice: We decided that the app would benefit from only sending push notifications to the user. Sending an email for every notification would spam the user's inbox and we feel that users typically expect to use SMS for social communication only. A push notification is something that is very prominent and notifies the user effectively, but also goes away after a few seconds.

How will the user enter packages into the service?

Option 1: Only through scanning emails for tracking numbers

Option 2: Scan emails for tracking numbers and allow users to manually enter tracking numbers

Option 3: Only allow users to manually enter tracking numbers

Choice: We decided that it would be best for the service to allow for manually entering tracking numbers as well as scanning emails to find them. This is because the point of the app is to provide hands-free service to the user, so we must allow for email scanning, but the app must also respect a user's privacy wishes, so allowing for the user to manually enter their own tracking numbers would ensure that nothing scans through their emails. It also allows for the user to add a package in the case of failure to scan a tracking number from emails.

Non-Functional Issues:

What language should the app use to power the server side of the application?

Option 1: Node.js

Option 2: PHP

The application will be powered by PHP. Two of our software engineers have previous experience with PHP and believe that it would be the most powerful language to handle our server-side needs. It is also easy to integrate PHP with a MySQL database.

What will the app use to catalog its user and package info?

Option 1: MySQL

Option 2: Local Storage

The application will use MySQL because it is a powerful service used by many applications and services and a couple of the people on our team have experience with using it. It is also more efficient to use for an application with many users and will allow for the program to expand easier.

What will the app use as a hosting service?

Option 1: External Hosting

Option 2: Local Hosting

External hosting is the app's best option as we do not have the resources to run a server 24/7, so we must host the entirety of the app remotely on another server to ensure that the app will run continuously. For the duration of development, however, the app may be hosted locally, or on Purdue servers for easy iteration, testing, load testing, and debugging.

Design Details

Class Overview

User

The user class is the base class for the system. All shipment information is stored within the user class. The server accesses the user class when a specific user logs into the WheresMyShip web app.

User •
Name EmailAccount • Password + int getNumShipments() + setPassword(String newPass) + setUsername(String newName)

EmailAccount

Every user has zero or more Email Accounts tied to their WheresMyShip account. Email accounts contain Emails which are filtered down to shipment email messages. A shipment email message contains the entire message that was flagged as containing a shipment. From these, the shipments array is automatically populated with Shipment objects from the shipment email messages array. In the event of the program not being able to retrieve the tracking number for a particular shipment email message, the message is flagged and the user notified that they may have to manually enter the tracking number for that shipment. AdditionalEmail is a linked list to additional email addresses the user may have.

EmailAccount •
Shipments • ShipmentEmailMessages • AdditionalEmail •

EmailMessage

An EmailMessage is a simple class that is merely a container for an email file. Every EmailAccount has zero or more EmailMessages.

EmailMessage •
MessageFile

Shipment

The shipment class contains the source email message for the shipment, and the tracking number as a string. From the tracking number, the shipment class calls the corresponding shipping company's API and populates the Origin, Destination, Carrier, CurrentLocation, ETA, Waypoints, and Status fields. These fields are all accessed by the client side web app and parsed by the UI code to display to the user.

Shipment •
SourceEmailMessage • TrackingNumber Origin Destination Waypoints Carrier • Seller/Sender • CurrentLocation ETA Status •

CarrierInfo

The CarrierInfo class is used in conjunction with the Shipment class to determine the specifics of a shipment. It stores data about the specific format for the carrier's tracking numbers. It also abstracts API access so that the Shipment class is less complicated.

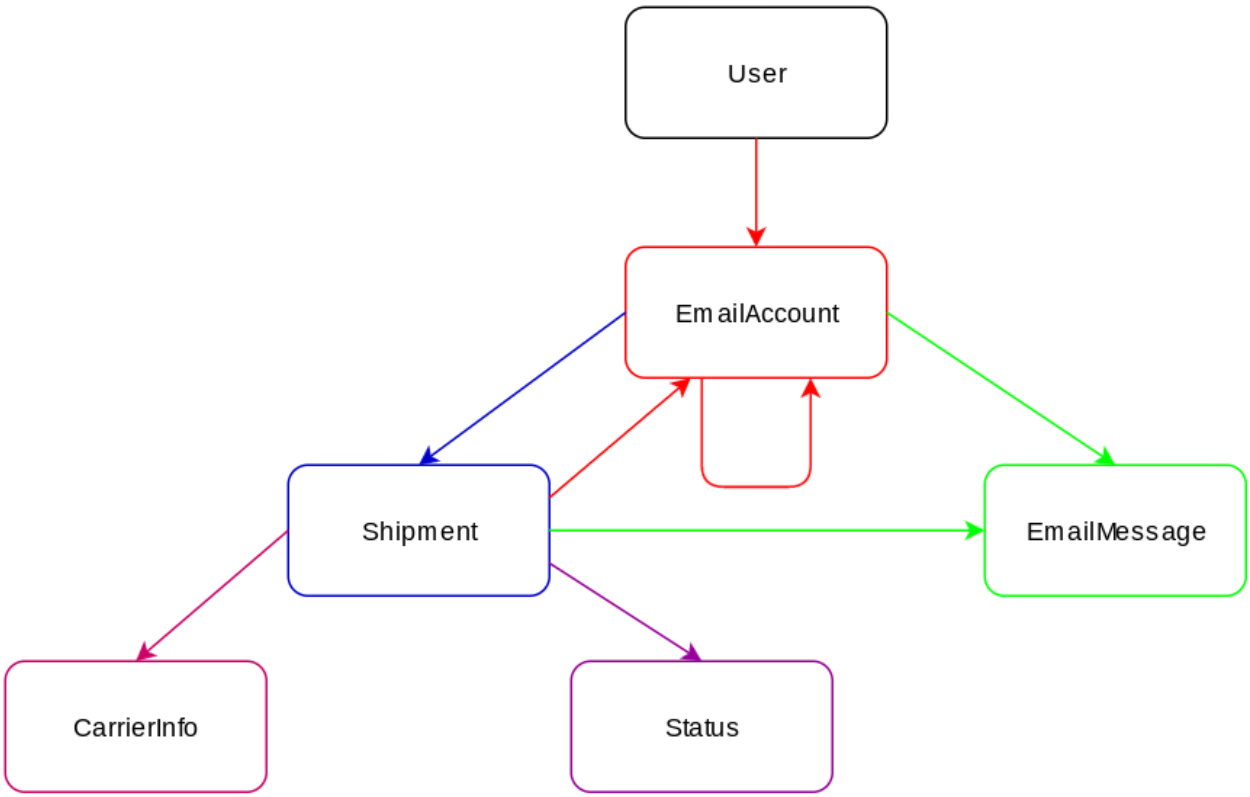
CarrierInfo •
Name TrackingFormat API

Status

An important message is only used if there is a delay or other problem with a shipment. In the case that the shipping API returns a delay, this class will be used to store the specifics of the problem. This class is then accessed by the client side UI and displayed appropriately.

Status •
Attention (On/Off) Message

Class Association Diagrams



User Event Sequences

User Links Their Email

Web-App	Server	Database
User opens the WheresMyShip website	Server sends the website files	
User clicks “Create Account”	Server sends the corresponding page files	
User enters their information and clicks “Proceed to link Email”	Stores user information as object and sends to database Server sends the corresponding page files	Stores user information
User is taken to the correct authorization page for their email provider	Server marks user’s email as authorized	Updates user’s information
User is take to a page notifying them that their email was successfully added and they should begin to see shipments auto populate within a few minutes	Server begins to scan the user’s email account for shipment messages	The database stores emails flagged as shipments by the server

User Logs Into the Web-App

Web-App	Server	Database
User opens the WheresMyShip website	Server sends the website files	
User enters their username and password and click “Login”	Server checks the user’s credentials against those stored in the database	Database recalls the credentials for the specified username and sends to the server
User is taken to their home page	Server grabs the user’s shipments and sends them to the Web-App	Database retrieves the specified user’s shipments and sends them to the server

User Tracks a Shipment

Web-App	Server	Database
User is already at their homepage User can look through a list of their shipments with basic information (ship date, ETA, seller) User taps on one of the shipments and the list entry expands to reveal detailed information about the shipment (tracking history, and map representation of progress) User can click refresh to retrieve most recent information from Shipment API	Server receives request to update the shipment Server retrieves shipment info from database Server sends API requests for the shipment service Server updates the shipment and sends new data to database and web-app	Database retrieves shipment info and sends to server Database updates shipment information
New data is displayed to the user		

User Receives Notification of Delayed Shipment

Web-App	Server	Database

	Server periodically (about every half hour, TBD) makes shipment service API calls on a user's in progress shipments The server receives a notification that one of the user's shipments is delayed Server sends push notification to user through TBD notification service	Every call with updated information is stored in database

User Manually Adds a Tracking Number

Web-App	Server	Database
User is already logged into the web app and on their home page		
User clicks the "Manually add a shipment" link	Server sends the corresponding page files	
User enters the shipment's tracking number and any additional information and clicks "Add"	Server receives the shipment information, adds it to the user's shipments and sends updated information to the database	Database stores updated user information
The web-app receives the new shipment information and shows the new shipment in the list of shipments	Server makes API requests for the new shipment and sends the data to the Web-App and database The server begins updating the status of this package just as it does with automatically added packages	Database stores updated shipment information

User Is Notified of Possible Shipment Email and Manually Adds Tracking Number

Web-App	Server	Database
	Server scans through the user's emails and flags an email as containing a shipment The server is unable to retrieve the tracking number from the email	
The user receives the push notification and clicks on it, opening the web-app	The server sends the user a push notification through a TBD service telling them that their "DATE purchase from SELLER could not be automatically added to shipments"	
The user is prompted by a pop-up asking them to manually add the tracking number for the shipment Adding the shipment proceeds as described in "User Manually Adds a Tracking Number" above		