Differentiation:

Intro

In this files, we will mention the differentiations between our application and the general requirement and other potential teams' applications. The new feature we tried to add is mainly from the ***functionality*** in both backend server and front end web, as well as the specific designed protocol.

Protocol:

1. To make the communication between world and amazon **consistent,** and **reuse and abstract** the codes out, we keep the gpb protocol for the interaction between ups and amazon.
2. To meet the minimum requirement, we designed 7 different sub-message which will wrapped by the two command message. Each of them has a necassary functionailty to exchange information between ups and amazon. Below we will introduced some points we figured out which may be different with others.
3. For the request pickup, we require the amazon to send all the information of the packages to the ups, including the **optional ups_account** which is required in "Actually Useful" section. In addition, we also include the destination of the packages. Although, this will not be used at this moment, but we ask the ups to **store the message in advance** which will be convenient for another requirement: "**Allow user to change** the destination on UPS website".
4. Followed by the above feature, we also require the UPS **pass the actual destination** to the amazon when the worlds send the package to deliver. We include the information in the UTAOutDelivery message.
5. We also make an agreement on the package pickup logic. As the truck could carry as many packages as possible, so to make the pickup more efficent, we as UPS will try to allocate as many packages as possible on the truck, and try to reduce the time wiaiting. The implementation will be elaborated in the next section.

Server end:

RAII

When we want to insert some messages into a google protocol buffer, it internally allocates some memory on heap(which might fail). We achieved **higher exception safety level** by using **unique pointer** on the google protocol buffer pointer. So the previous allocated memory will be deallocated when exception happens.

Database consistency

As we need to store data into dababase, and our sever is running on multiple threads, in addition, so we have to make sure the database consistency. We maintain our isolation_level as the **default of Postgres** instead of *SEARLIZABLE.* Sometimes, we may need to firstly query and then modify the value of it, if we write these into two sql, which could have some problems when the queried result being modified before we actullay update it. Instead, we coose to use **UPDATE … IN (SELECT … FOR UPDATE) RETURNING..**, which is one sql that could do the query and **lock the row**, and do the update, returning the columns that you wan to return after update.

This could greatly reduce the use of locks, and still maintain the correct modification to the database.

Truck selection:

As mentioned in the previous section, our UPS will try to select truck and assign to pacakge with some unique logic. To utilize the truck effciently, when a new package came, and if we detected there is **already a truck heading** to the same wharehouse, we will not request a new truck from the world, instead we will make the package associate with the existing traveling truck. When the truck arrive the warehouse, we will **select all the packages** that applied and notify the Amazon, the packages we are expecting to load.

Ack and Seq

We used a set to record the sequence number that is sent by world or amazon, later we realized that if our server runs for a long time, these numbers will not be deleted and the memory might be insufficient. We make our server robust by create a **new data structure,** which **combines a set and a priority queue**. We **pop** the smallest number in the set and the priority when the numbers ***exceeds*** *our pre-defined capacity.*

Front end:

Bind order:

As there is a choice, when placing order at amazon, the user could provide its user account name, at that time. However, there will be some orders are **not taken**. As a result, we provide the option for the logged in users, if they could provide the **package id** and the correct **corresponding name,** it could bind the package with themselves, and see the orders in their packages list, but we still set the constraint that the order shouldn't be already taken by other users.

Email notification:

We will send email notification, when the order destination has been changed, if there is a user associated with the order, and when they create their account, they will be asked to input their email where they want to receive the message.

Location map view:

As the package has two address, one is the the current location, and the other is destination, we will provide a thumbnail view of the map to indicate the package is in transit or has already arrived.

Annual Report:

We have the functionality to show the total number of our users and the shipments. If the user is logged in, we will show the **users stats** for the shipments and destinations, and the **average** users stats on our website.

Interaction between front and back end

To make it possible to exchange data between front end and backend, we use json object as the mediat to send and recv through the socket. As we use different language for front and back end, and the data we need to transfer is not as abundant as the messages between Ups and Amazon, so we use a simpler and easy to use data structure instead of GPB.