Dispatcher - Clover Service
Design Documentation

Johan Burke, Justin Gavin, Stanimir Genov, Sean Fitzpatrick, Jake Schooley, Steve Snyder
Rowan University
Senior Project
Fall 2017

Table Of Contents

1. Overview

The Dispatcher project provides businesses with a way to send out jobs to be “claimed” to a pool of registered workers, one of whom can then claim the job to be performed. The merchant manages his or her pool of drivers and sends out jobs through a Clover app web interface. The Clover app makes api calls to the server [endpoints] using HTTP POST and GET requests and by sending and receiving JSON data. The server, upon receiving a request, uses Python scripts for each REST endpoint to get data from the MySQL database and send SMS messages to the merchant’s pool of drivers if necessary.

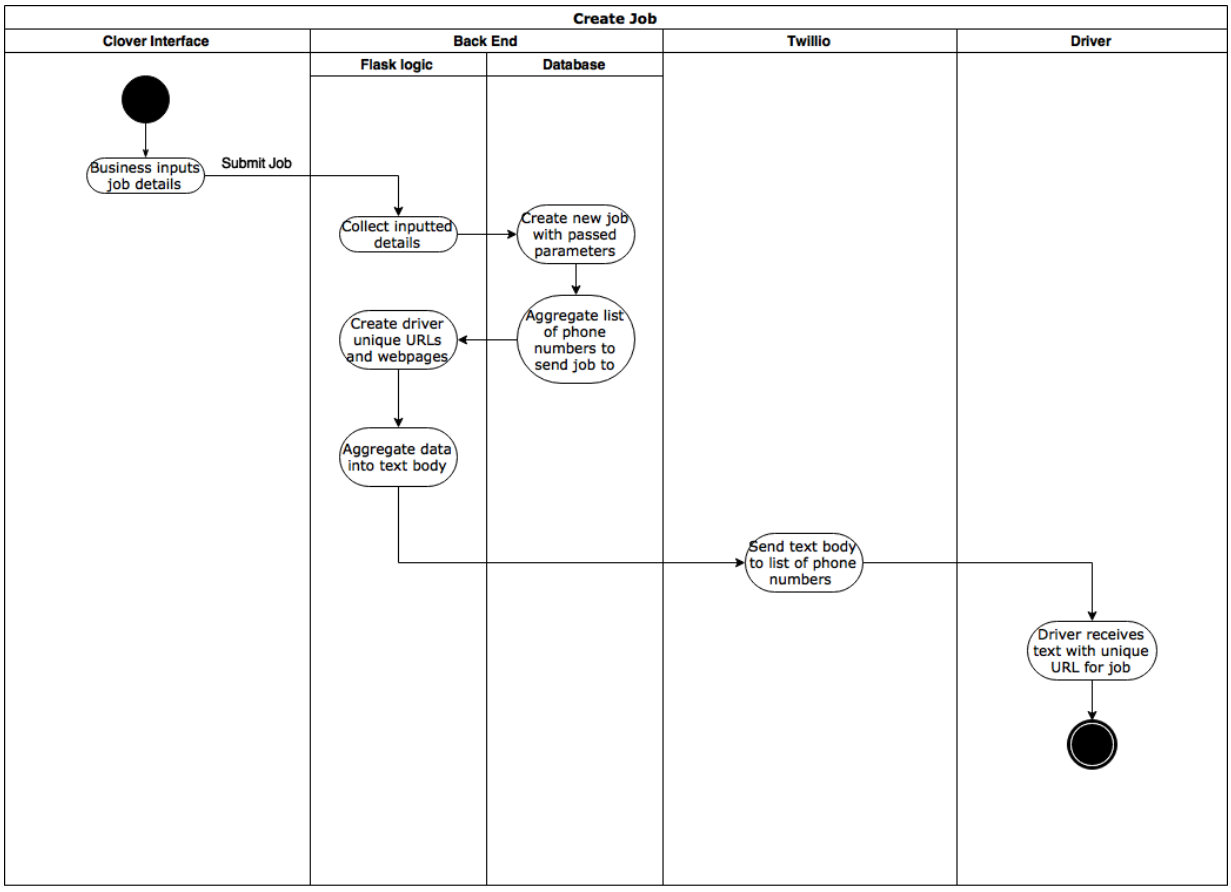
1.1. Introduction

1.2. RESTfulness, Client Server design, etc

REST, or Representational state transfer is a way of providing the ability to exchange information between different systems on the Internet. We chose a REST API as it allows us to easily communicate between the frontend UI, and our server through HTTP requests. The frontend web app leverages the Materialize CSS framework for the “look and feel”. All meaningful functionality is performed on the backend. The backend is built using Flask; a python-based micro web development framework.

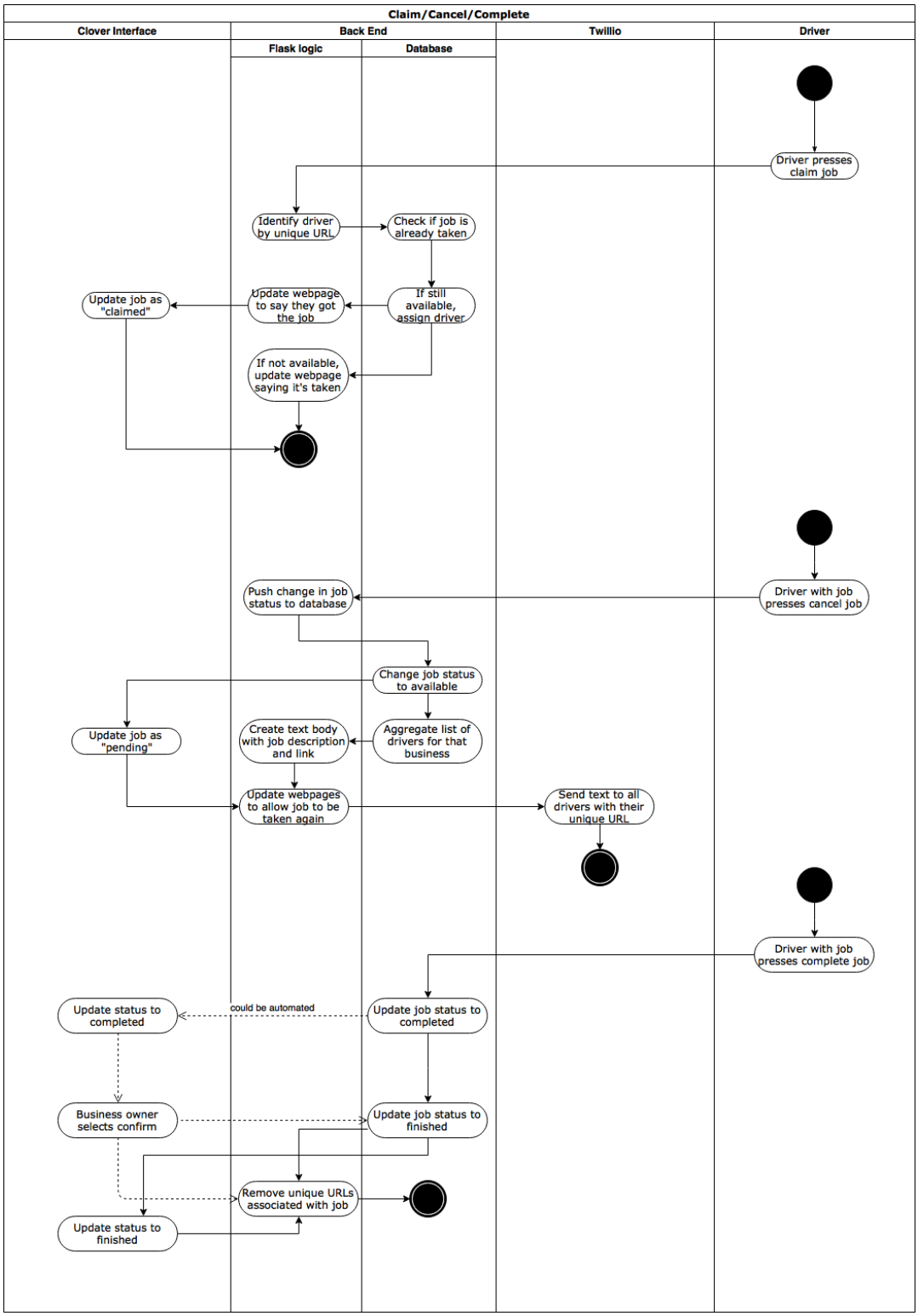
1.3. Major Features - Activity Diagram & Explanation

1.3.1. Create Job



The business opens up the screen to input the job details. Once done and they select submit job, the data is aggregated in a Json object and the corresponding endpoint is called in the server. Using the imputed data, a new job is created in the database with the passed parameters which returns a list of phone numbers of the drivers associated with that business. A unique URL will be created for each driver to a webpage where they can accept the job. Put all job details and link to job in a text body. Send off text body to each of the phone numbers. Driver receives URL for accepting job along with the job description.

1.3.2. Driver Claim/Cancel/Complete job



In order to claim a job, the driver selects claim job on the web page they received a URL to. The server then identifies driver and job by the unique URL. Then check the associated job if it has already been taken. If it is available, assign that driver to that job, and update their webpage to say that they have the job. Update the job status as “claimed” and update the clover interface to inform business. If the job has already been assigned another driver, the driver will have their webpage updated to say that the has been already taken.

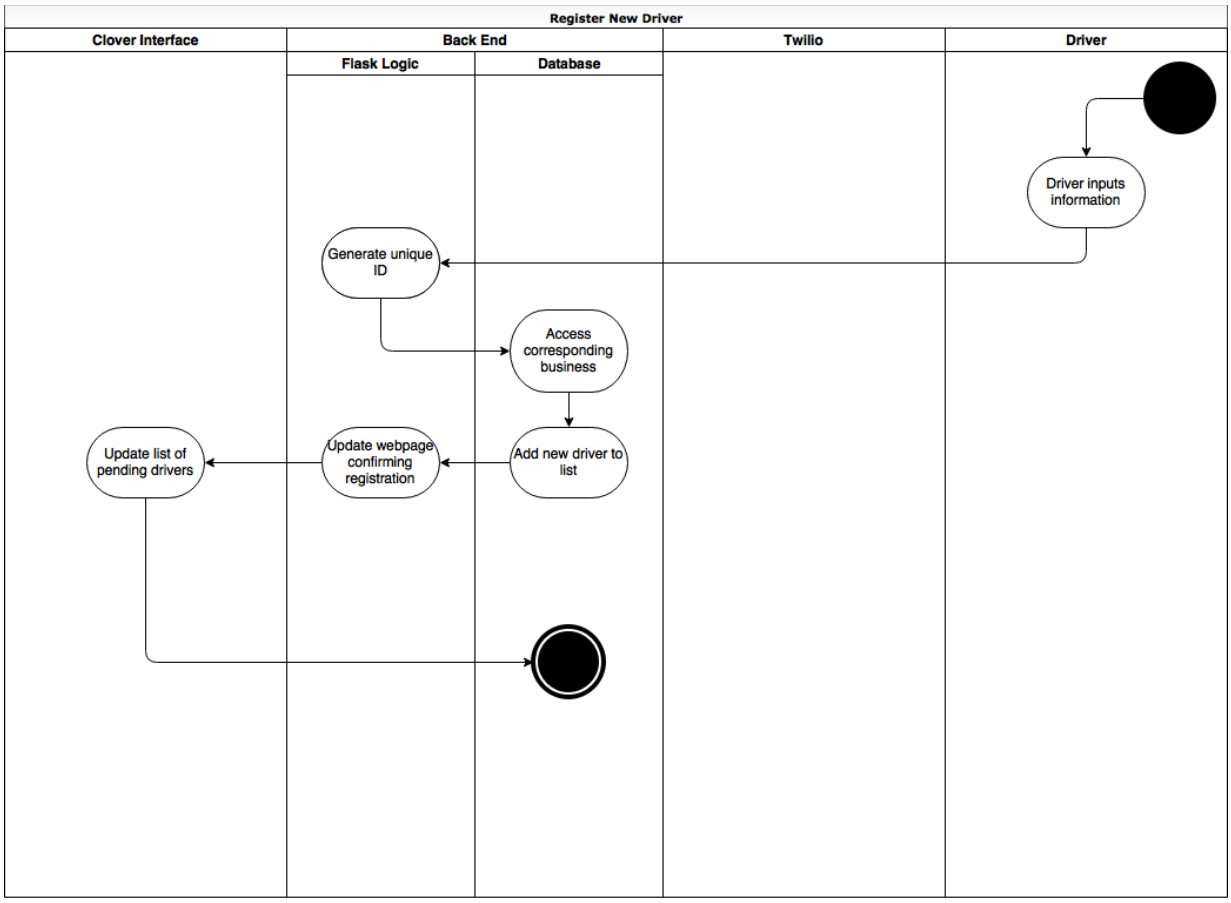
To cancel a job, the driver can press cancel job on their webpage. The change will then be pushed to the database, and the job will now be available for other drivers to claim. Update the job status on the clover interface as “not claimed”, and update webpages of drivers to allow the job to be taken again.

In order to complete a job, the driver presses complete on their webpage. The database is updated to say that the job has been completed. The clover user interface is updated to says that the job has been completed, which the owner can confirm. Upon confirming, the job status is changed to finished, and the URLs associated with the job are removed. The business owner could choose to not have to confirm job completion if they trust their drivers, in which case once the job status would be changed to finished, and the clover interface will be updated with a job status of “finished” for that job.

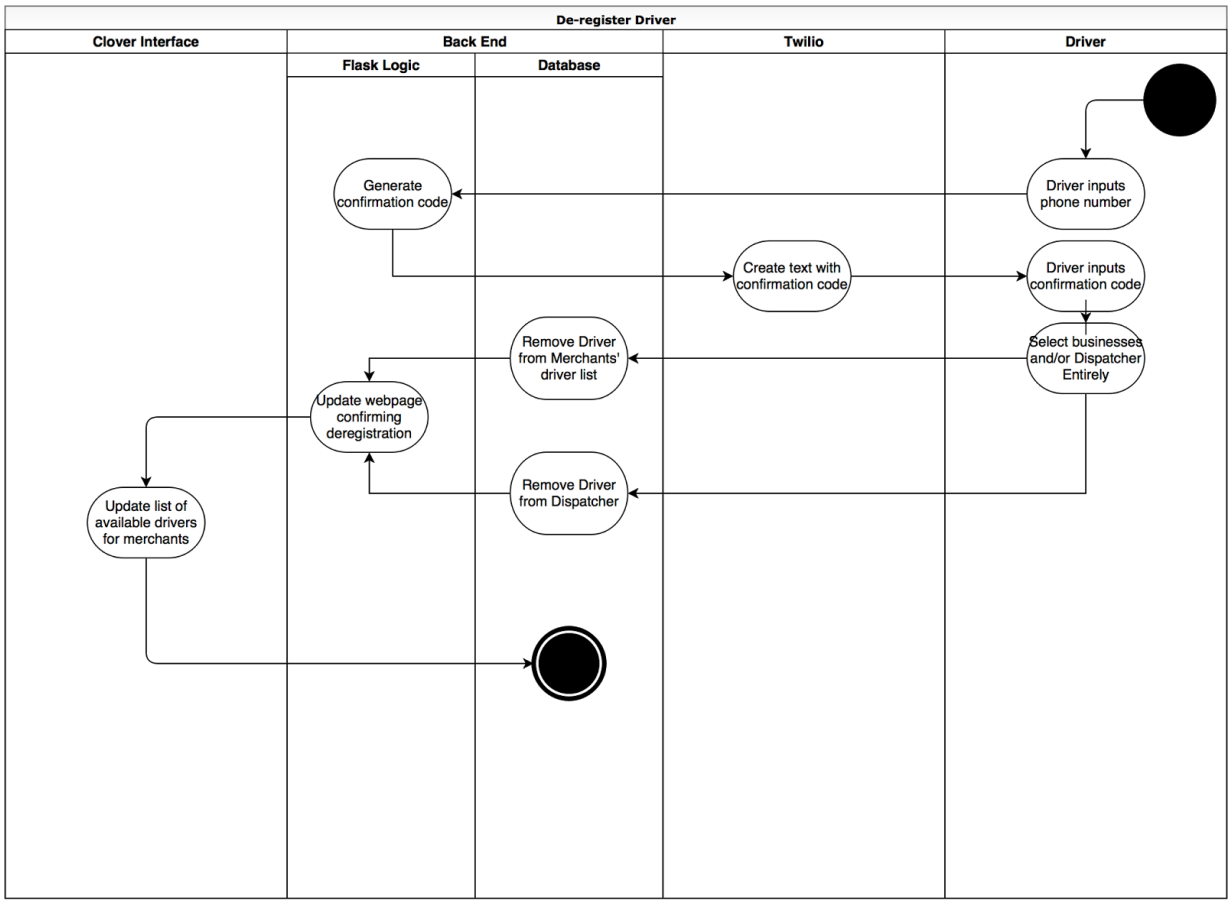
1.3.3. Register New Business

When a new business buys our app, in order to register them into our server. The app will first pull as much information as it needs from the clover API, such as business name, and phone number. The information it gathers is shown on screen, along with empty text boxes for any additional information we may need that can not be retrieved by the clover API. The business confirms all information and is then registered in our server.

1.3.4. Register/Deregister Driver

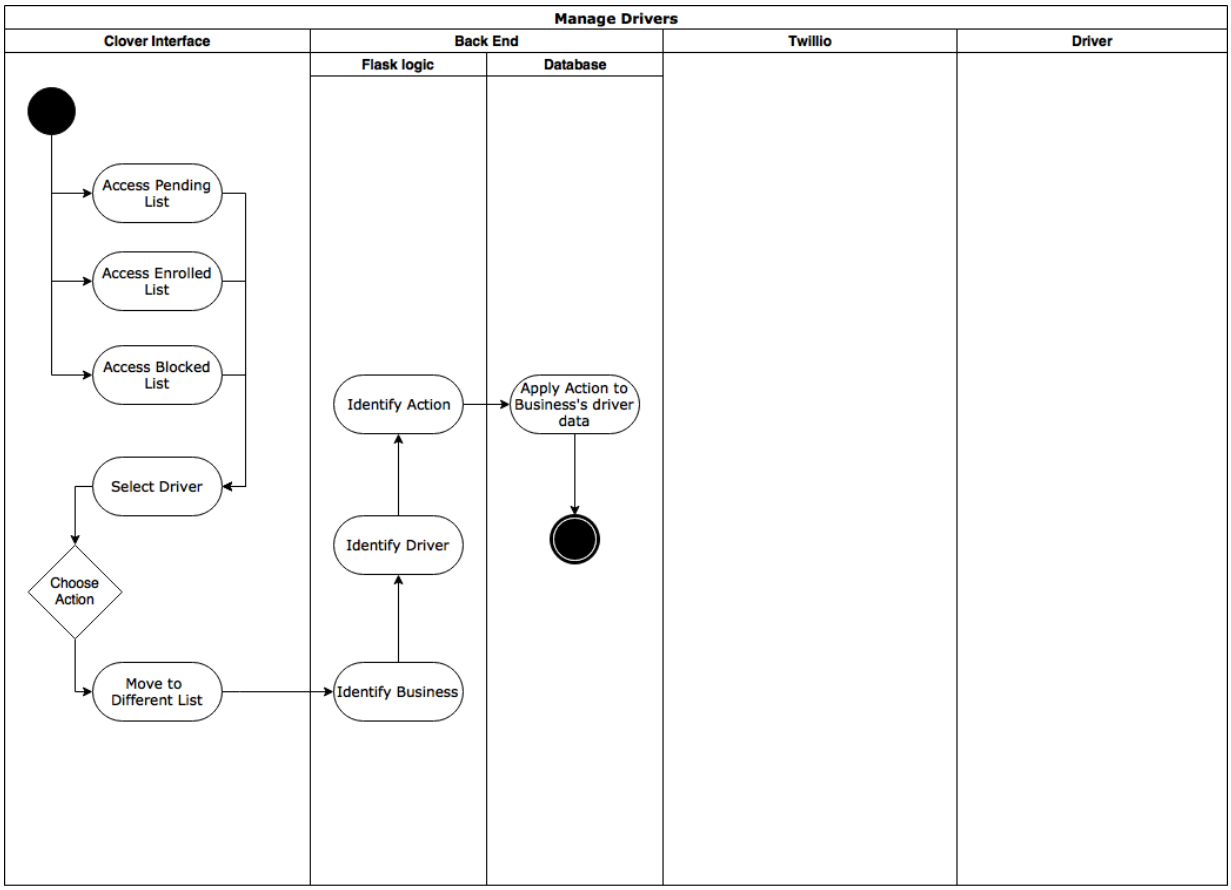


When a potential driver is interested in signing up to be included on the dispatcher list for a certain business, they will be given a link for that specific business. The Driver will fill out a form on the website providing their name and phone number, and then submit that form. That information will be passed to the Python Back End script which will give that information to the database. The database will create a new driver row and link it to the list of Drivers specific to that business. The Python script will then update the webpage informing the driver that their registration was successful as well as prompting an update of the screens on the Clover interface to include that driver on the business's list.



When a registered driver wishes to cease business with either a particular merchant or Dispatcher entirely, they can do so from the driver homepage. Choosing the deregister option will allow them to enter their phone number. This phone number is looked up in the database and a unique one time use confirmation code is generated. The driver can then enter this code into their web front end and choose from a list of merchants they are registered with to unregister with. Additionally if they choose to leave the dispatcher service all together, they may do that as well. Once a driver has chosen to unregister, all affected businesses with links to that driver will be updated in their Clover interface automatically.

1.3.5. Business Manage Associated Drivers



Businesses will have 3 lists to manage their driver pool. These lists pending, active, and blocked, will allow the business to manage who they want to work with. Each list will have the ability to remove a driver from the respective list or transfer a driver to another list. The businesses will use a Clover device to apply these decisions. The chosen action and data management will be applied by Flask and the Database.

Once a driver registers with a business they will be automatically added to the pending list for that business. How or why a business chooses to approve a driver to the active list is entirely up to the business. The active list is the list of all currently approved drivers who will receive updates of new jobs created. Drivers that the business wishes to have no further contact with can be placed into the blocked list. How or why a driver is blocked is entirely up to the business.

2. Supporting Technologies

2.1. Server Software

2.1.1. Operating System

Ubuntu 16.04.3 LTS (64-bit)
Ubuntu was chosen due to its widespread use, history of stability, availability of documentation, and large library of precompiled packages. The combination of these factors provide a reduction in risk and therefore the ability to focus on development of the Dispatcher project without much concern for the underlying software.

2.1.2. Web Server

Apache/2.4.18 (64-bit)
Apache HTTPd is the most widely used web server today[1]. It was chosen due to this widespread use, availability of documentation, ease of integration with the microframework, and out-of-the-box compatibility with the chosen OS.

- Apache HTTPd Module: mod_wsgi
  This module serves as the bridge between the Apache HTTPd web server and the python-based Dispatcher project. The existence of this module was an additional driving factor in choosing Apache HTTPd.

2.1.3. Database/RDBMS

MySQL 5.7.19-0ubuntu0.16.04.1
MySQL is the most widely-used open source [free to use] RDBMS[2]. It was chosen due to this widespread use, availability of documentation, ease of integration with the microframework, robust featureset, and out-of-the-box compatibility with the chosen OS.

2.2. Backend Language

Python 3.5
Python was chosen for its compatibility with the chosen OS for the backend server, extensive built-in support for JSON data through easy associative array creation and operations, and interoperability with the chosen server-side frameworks. The existence of the Python microframework, Flask, for easy setup of a REST api was especially a driving factor in the use

[1] https://w3techs.com/technologies/overview/web_server/all
[2] https://db-engines.com/en/ranking

of Python for backend server scripts. Python's ease-of-use and high readability are also helpful factors, as they allow all team members, even those unfamiliar with Python previously, to contribute to debugging server scripts and understanding backend logic.

2.3. Microframework - Flask

Flask allows us to do many things for building the server side of the app, as it allows for generating unique URLs, easy connection to the mysql server, and RESTful request dispatching. Easily allowing us to make HTTP requests to specific endpoints, it makes connections between separate pieces of the software effortless. Originally we considered having the drivers text back to accept a job, however thanks to unique URL generation and allowing a variable endpoint through flask, we can easily create dynamic web pages and have the driver accept, cancel, and complete jobs through a web page.

2.4. External SMS Messaging Service - Twilio

Twilio is a simple and cost effective solution for a SMS service provider. Many testimonials credited Twilio with outstanding service such as Uber, Twitter, Netflix, and many more. Additionally, after comparing Twilio to other SMS service providers Twilio appears to offer many tutorials and good documentation. These reasons directed us to choose the Twilio API for SMS messaging integration.

2.5. Clover

Clover gives us access to a unique marketplace that, considering our application, would benefit us in getting our app out to people who could most benefit from it. Clover is an Android Point of Sale platform and has physical hardware for transactions. It is used in commercial businesses, an area that our application would have a large focus on.

Additionally, Clover's API is used to handle all authentication, eventual billing, etc. The Clover API also allows for information about the merchant to be obtained programmatically (name, address, etc.).

2.6 Materialize CSS & jQuery

Materialize CSS is an open-source frontend web framework. It is utilized for the "look and feel" of the frontend webapp. It performs the majority of the heavy lifting; allowing for components to be pieced together in a very "plug and play" way.

jQuery is a dependency of the version of Materialize CSS used for the project. jQuery is also used to format and send various HTTP requests (often AJAX) to the project backend.

3. Backend Logic

3.1 Explanation of Python Logic

3.1.1 Register New Business

When a new business gets the app on their colver device, the Python Script will be passed information about the business such as the name, location, phone number, and pass that along to the database as well as generating a unique store ID to be passed into the database.

3.1.2 Create New Job

When a business creates a new job, information about the job entered through the UI, such as the job title, job description, location and phone number to contact, will be passed along to the Python script. The script will also pull the business's name and will then add a new row to the Database. These fields will then be bundled up into a message body that is prepared to be passed to Twilio. The Script will get a list of Driver's numbers from the Database, and then pass each number along with the message body to be sent along to Twilio.

3.1.3 Claim Job

When the driver clicks on the accept button on his or her unique link, the Python Script will use the unique id and use that to access the corresponding driver in the Database and initiate a link by prompting the Database to assign that driver as the driver for the job if the job is not already claimed. If already claimed, the driver will receive a message saying that the job had already been claimed. The business will see the job status updated from pending to claimed.

3.1.4 Cancel Job

If the driver were to then click the cancel button on the Unique link, the Script will call the job from the database again and remove the driver as the jobs driver. It will then retrieve the information about the job and business again and compile them into a message for Twilio. The script will retrieve the list of drivers and pass each one with the message body to Twilio.

3.1.5 Job Completed

Once the driver completes the assigned job, they will go to their unique link and confirm that the job has been completed. This will be passed to the Script and pass to the database that the job status is now complete.

3.1.6 Register New Driver

To register a new driver, the script will pull the driver's name and number from the web page, and add them our database of drivers. If that phone number is already registered with our service, the user will get an error message.

3.1.7 Apply to Business

The business will give their unique link to a driver that wants to register with them. From that link, the driver types in their phone number, that number is looked up in the database from that number as well as the id for the business from the unique link with which they are registering. The script will then pass the id of the business and the id of the driver to the Database to add the driver to the pending driver list for the corresponding business.

3.1.8 Managing Drivers

When a driver registers online, the driver will be added to the pending driver list for a business. If the business agrees to hire them, this will be passed to the Python script to request to the database to move the driver from pending to the approved driver list. If a business wants to block the driver, it will once again be passed to the script to request to the database to move the driver from the approved driver list to the blocked driver list.

3.1.9 Driver Deregistration

If a driver wants to deregister with either a business or our service entirely, they go to the deregistration page and enter their phone number. They will then be sent a confirmation code to that phone number with which to verify that they want to deregister. Once they enter the confirmation code into the form and it is confirmed that it is the correct code, they will prompted to select which services they want to deregister from. Once they check the boxes with which they want to deregister and hit submit, they will be deregistered from those entities.

3.2 Unique Links

The Flask api has the ability to generate unique links, so we plan on taking advantage of this feature. For each text to go out when a job is created, a unique link will also be created custom for each driver and also containing information about which business has requested the job. When the driver receives the link and requests to claim the job, the Python script will be able to get the information about which driver this specific page belongs to and what business requested the job. This prevents the Driver from having to send information along from their end to identify themselves and avoids any possible error that could come from that. It also could prevent any potential “crossed wires” if a driver happened to be registered for two businesses, the unique links for each job would prevent them from accidentally signing up for one job if they intended to for another.

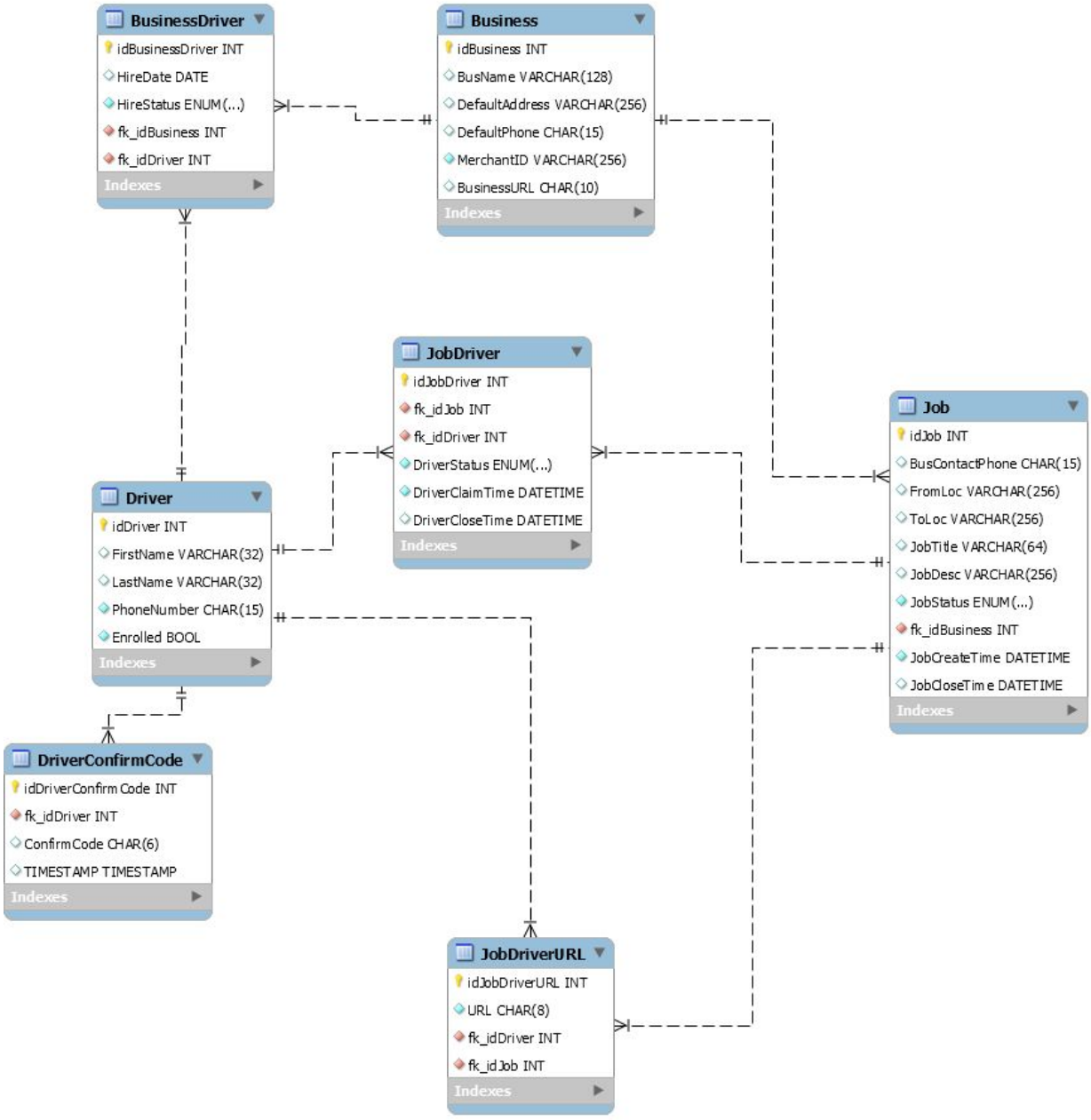
3.3 Web Registration

When a business wants to hire someone and add them to their list of drivers, all they have to do is have them go to the registration website for their business, which they will be able to find the address for on their Clover device. Each registration page will have a unique link for each

business. The driver will only need to fill out a form with their phone number, and the Registration page will pass that along to the Python Script to be added to the Database.

4. Database Design

4.1. Database Diagram



The above EER Diagram shows the entities, attributes, and relationships modeling the data storage requirements for the Dispatcher project.

4.2. High-Level Entity/Relationship Trace

A Business represents a merchant that subscribes to Dispatcher through clover. A **Driver** is a single person that has registered with Dispatcher. A **Driver** is unique to their phone number. A **Driver** may be associated with zero or more **Businesses**, through the **BusinessDriver** relationship/bridging table. This relationship has two attributes: *HireStatus* (must be either ['pending', 'hired', or 'blocked']; defaults to 'pending') *HireDate* (the date of the 'hire' status). This is modeling the "**Driver**'s application to a specific **Business**" behavior: when a **Driver** applies to drive for a **Business**, a new record will be created in the **BusinessDriver** table, status set to 'pending', the **Business** will be notified. An employee at the business then has the option to approve the application, status set to 'hired', or reject it: either a soft rejection (deleting the record from the bridging table) or a hard rejection, blocking, status set to 'blocked' (will prevent future applications).

A **Job** is created by a **Business**, it must be tied to a single specific **Business**. The value of the attributes of **Job** are provided by the **Business**, either directly at the time of **Job** creation or generated from information previously given. Upon creation, all **Drivers** hired by the **Business** will be notified. A **Driver** may be associated with zero or more **Jobs** through **JobDriver**. When a **Driver** claims a **Job**, a **JobDriver** record is created. At creation of the **JobDriver** record, the timestamp is recorded. The **DriverStatus** is initially set to 'claimed'. From this state, if the **Driver** successfully completes the **Job**, status is set to 'completed'; if they cancel, status 'canceled'. In either case, this closing action timestamp is recorded.

4.3. Selected Attribute Detail

Explaining the seemingly redundant fields:

Job

- BusContactPhone - phone number of the contact at the Business. May be different than the Business's default phone number.
- FromLoc - originating address, may be different from the Business's default address
- JobStatus; ENUM('pending', 'claimed', 'complete', 'canceled') - Status of Job from the perspective of the Job (business's perspective; business canceling the Job is different that the Driver canceling).
- JobCreateTime - when the Job was created by the Business
- JobCloseTime - when the overall Job was completed or canceled

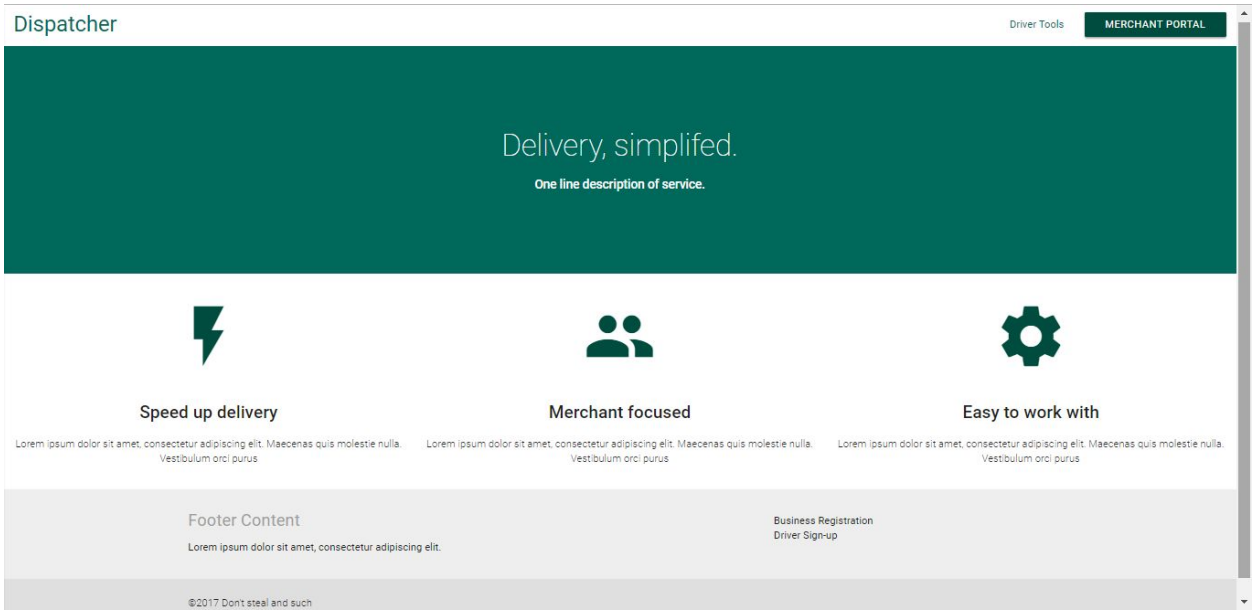
JobDriver

This table exists for auditing purposes; answering questions like "How many Jobs has this Driver canceled? How long on average has it taken for Driver to complete his 'complete' jobs? Multiple Drivers may be associated with a Job, in the case that a driver un-claims/cancels his involvement with a Job; another may claim and complete the Job.

- DriverStatus ENUM('claimed', 'complete', 'canceled') - Status of the Job from Driver's perspective. Has Driver claimed/canceled job?
- DriverClaimTime - the time the Driver claimed the Job. Always will be slightly later than JobCreateTime
- DriverCloseTime - the time the driver completed or canceled the job. May be different than the JobCloseTime (in the case that the Driver canceled the job and the job gets reassigned)

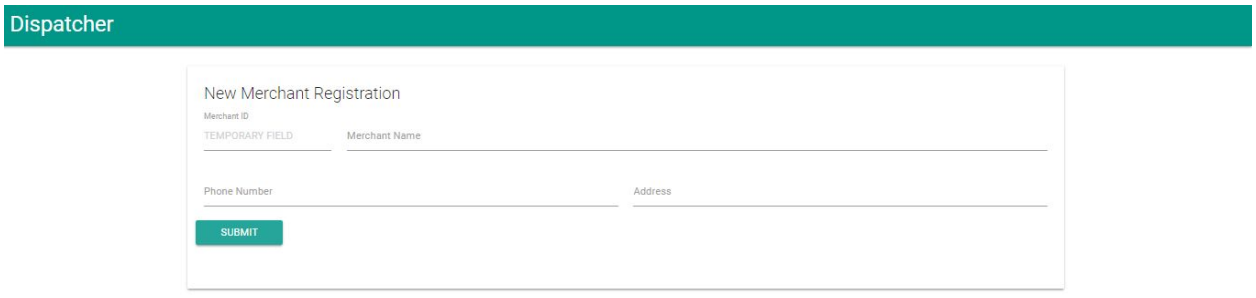
5. User Interface/Frontend: Clover App

5.1. Web Pages



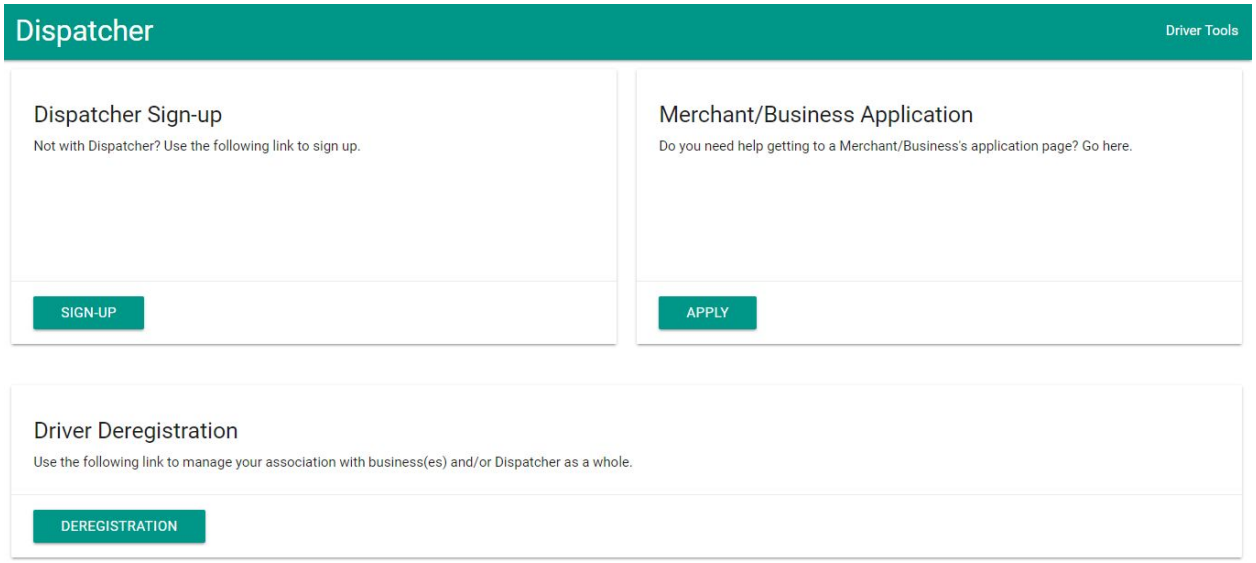
Main Menu

From the main menu, a user can access Merchant Registration, Driver Tools, and the Merchant Portal. This will not be seen by a merchant if they access the service directly from their 'clover apps' home screen.



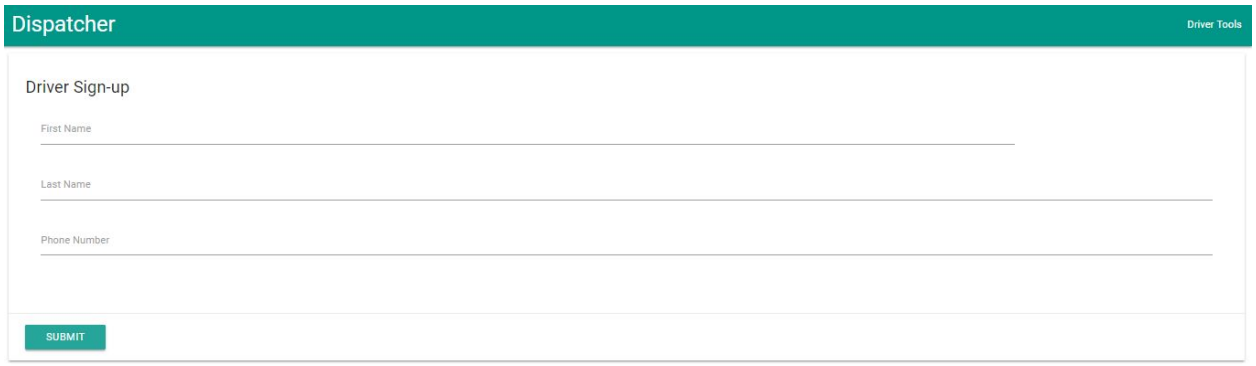
Merchant Registration

A new Merchant enters their information to register with Dispatcher. The required information that the merchant must give is the Merchant ID with Clover, the Merchant Name, their phone number, and their address.



Driver Tools

From Driver Tools, a driver can sign-up with Dispatcher, apply to a Merchant, and Deregister from Dispatcher.

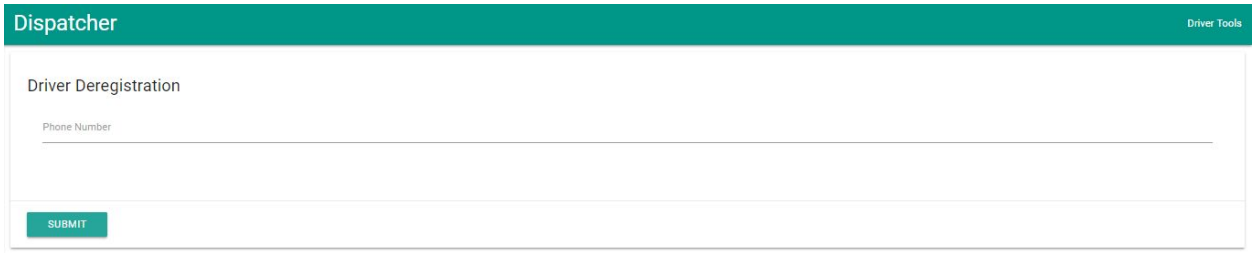


Driver Sign-Up

When a driver wishes to access the dispatcher service they must sign up with dispatcher first. The only information required is a first and last name and a valid telephone number.

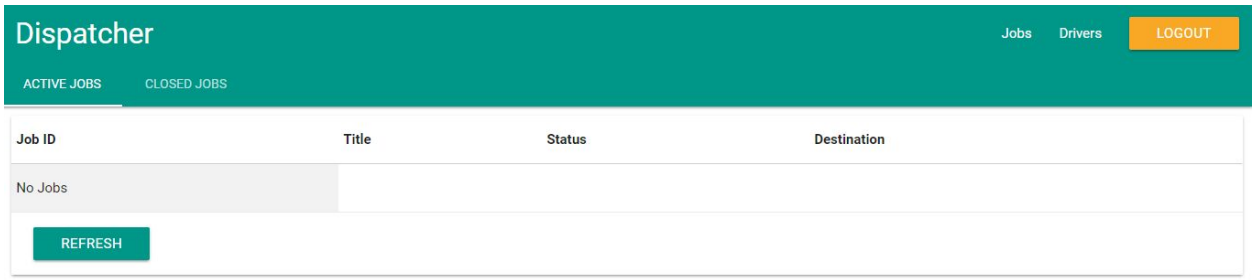
Driver Application to Business

When a driver wishes to work for a merchant, they are to enter a merchant's unique url/business code in the box. There interest will be pushed to the merchants pending drivers list where the merchant can then choose to approve them or not.



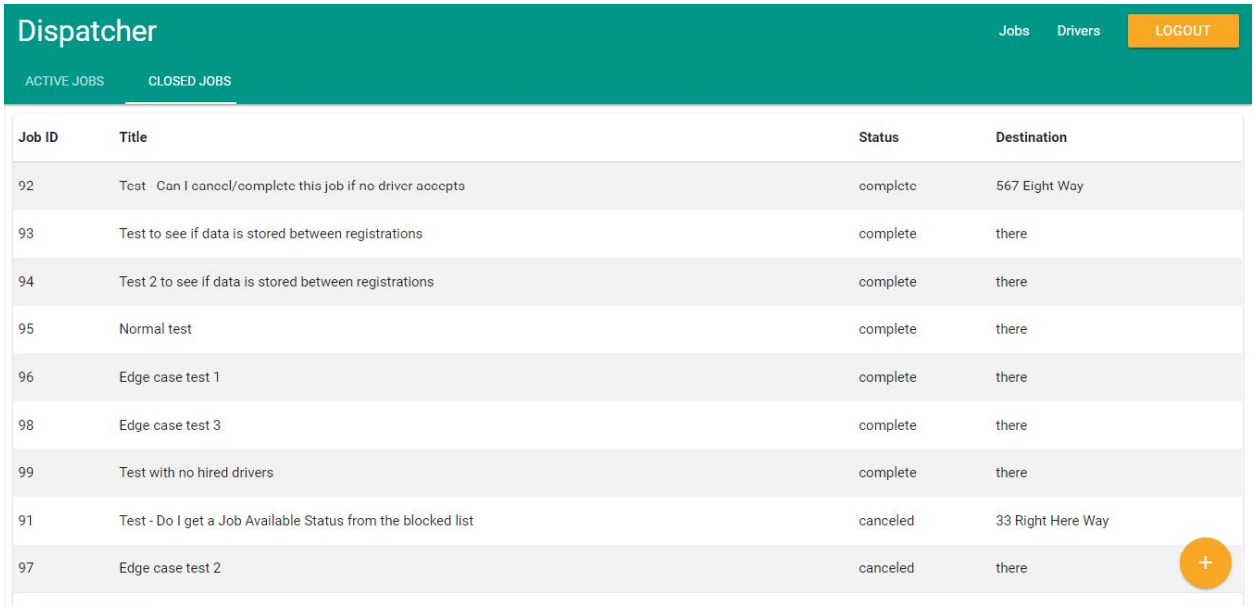
Driver Deregistration

When a driver wishes to cease working for either any merchants or dispatcher's service entirely the only information required to identify themselves is their phone number



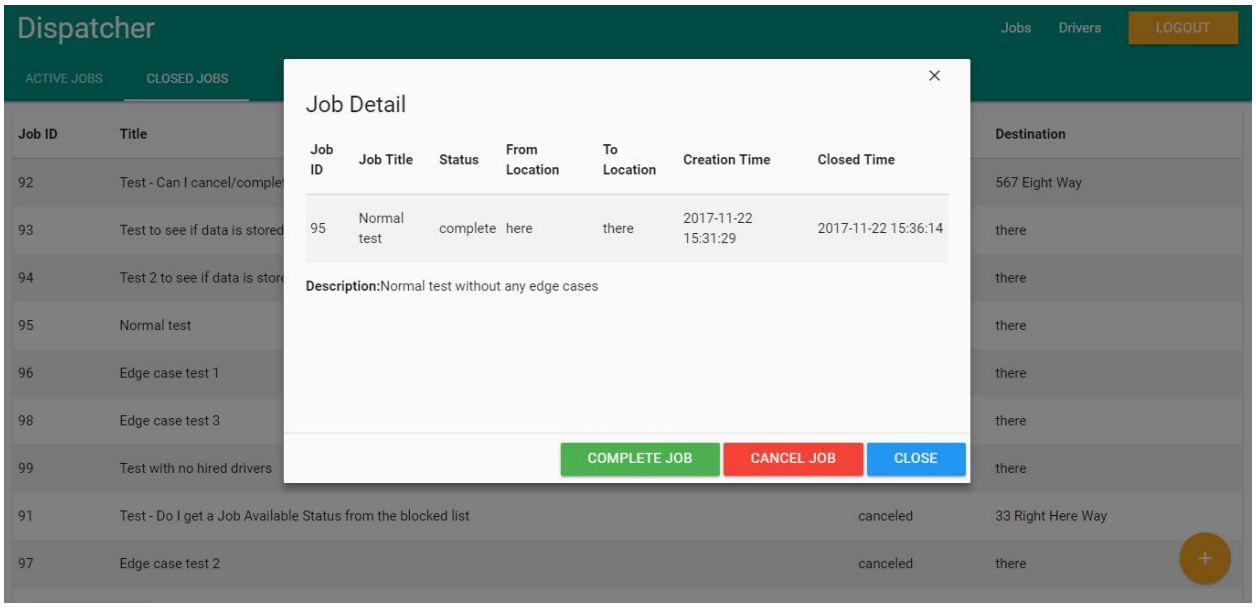
Active Jobs

A merchant can view all active jobs (in progress) through this portal. It relays the title of the job, the status, and the end delivery point for information at a quick glance.



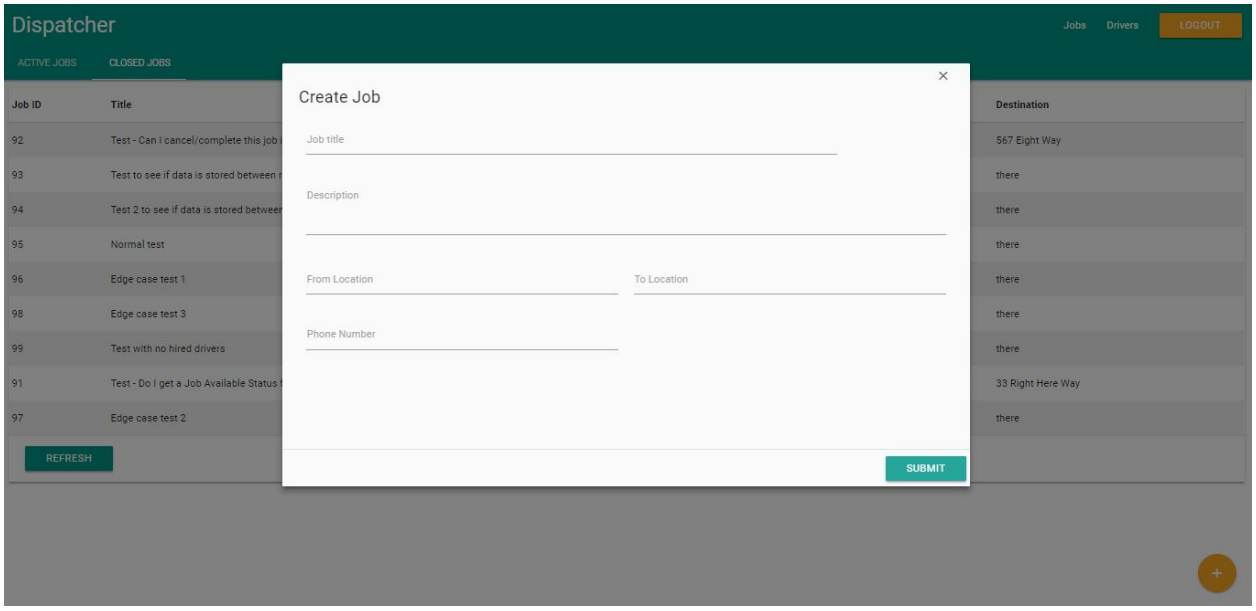
Closed Jobs

A merchant can view all closed jobs (no longer in progress) through this portal. It relays the title of the job, the final status, and the end delivery point for information at a quick glance.



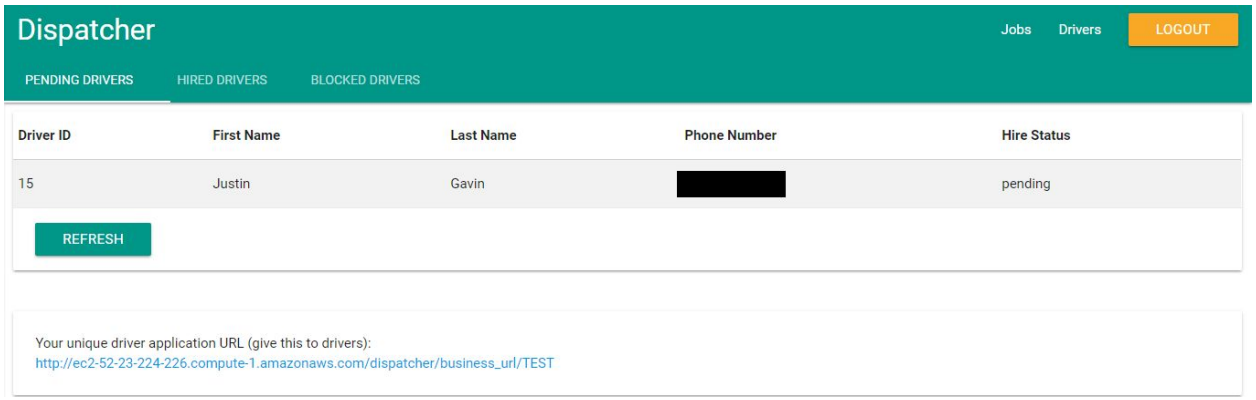
Job Detail

For a more indepth information about a specific job, a merchant just clicks on the job so have a pop-up appear. This pop-up will show all stored information about the job.



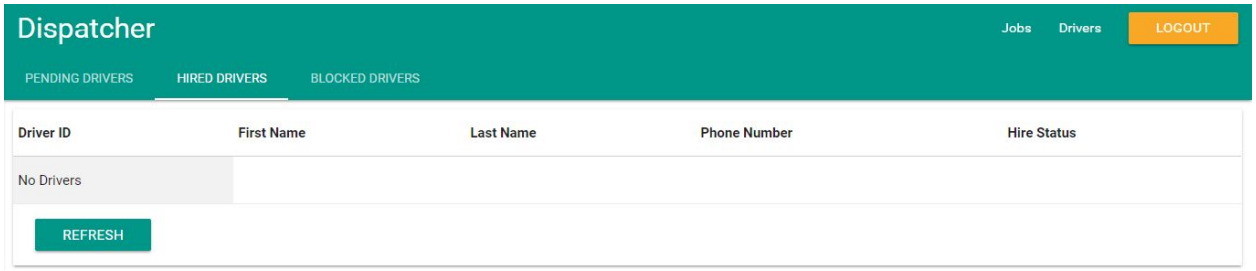
Create Job

When creating a new job, a form pop-up will appear. The merchant need only fill out the fields and click submit for the job to be created and pushed for drivers to accept.



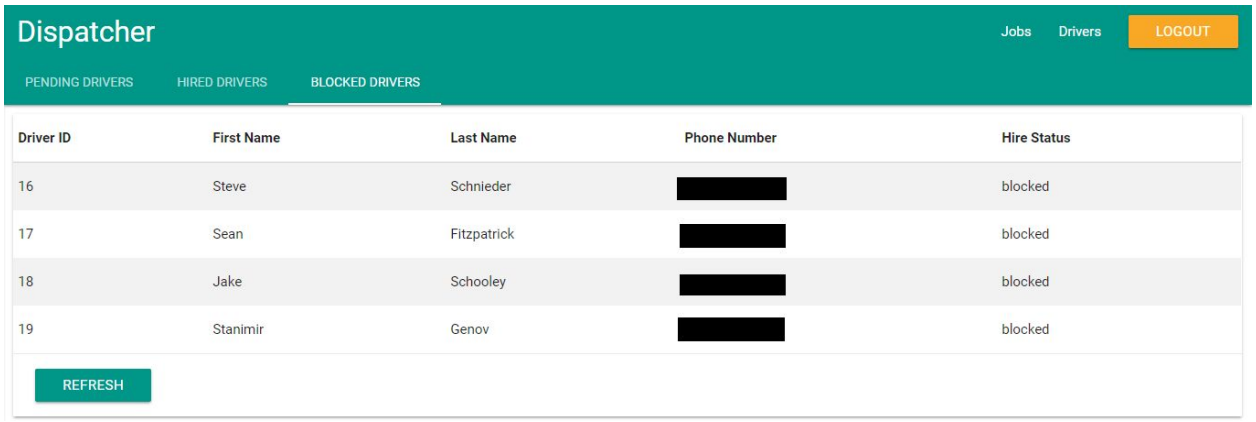
Pending Drivers

This list shows all drivers that are waiting for approval to receive job alerts from a merchant. From this list, drivers can either be transferred to the hired or blocked lists.



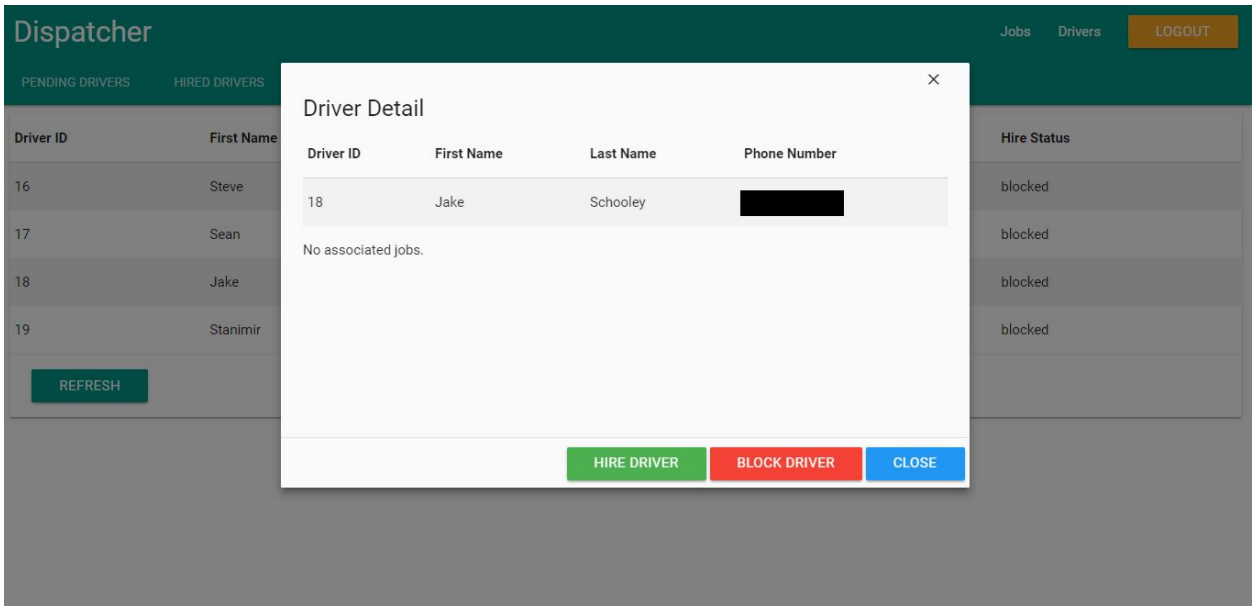
Hired Drivers

This list will display all drivers available to receive new jobs created by a merchant. From this list, drivers can be transferred to the blocked list.



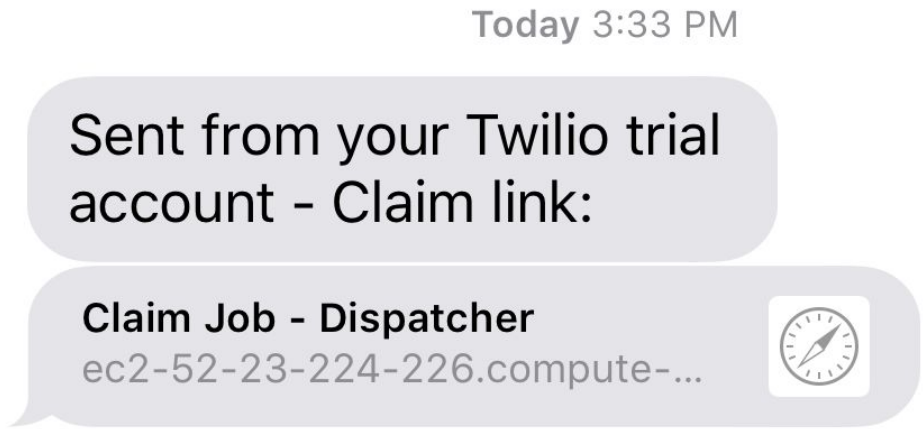
Blocked Drivers

This list will display all drivers that are still registered with a merchant, but the merchant does not wish to have jobs pushed to them. From this list, drivers can be transferred to the hired list.



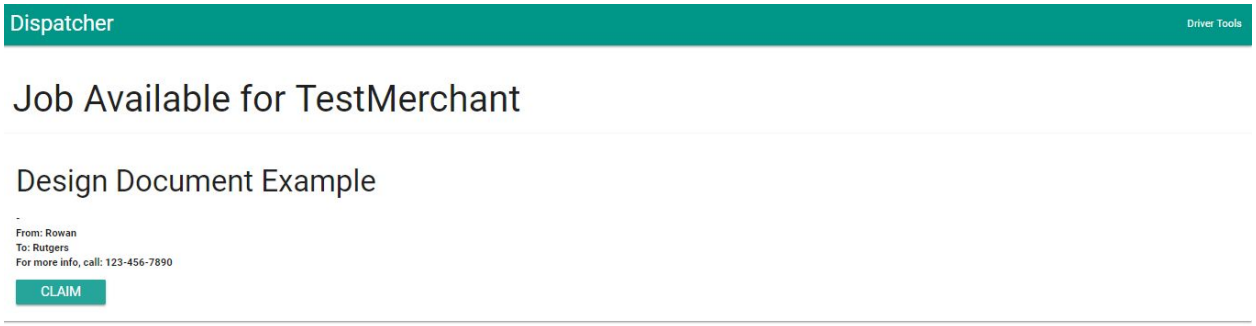
Driver Detail

After a merchant clicks a driver, their information is shown. This includes name, phone number, as well as all completed jobs done by the driver for the merchant and the status of those jobs upon completion.



Job Received

When a driver receives a SMS notification about a new available job, they are sent a unique link to open in which then can claim said job.

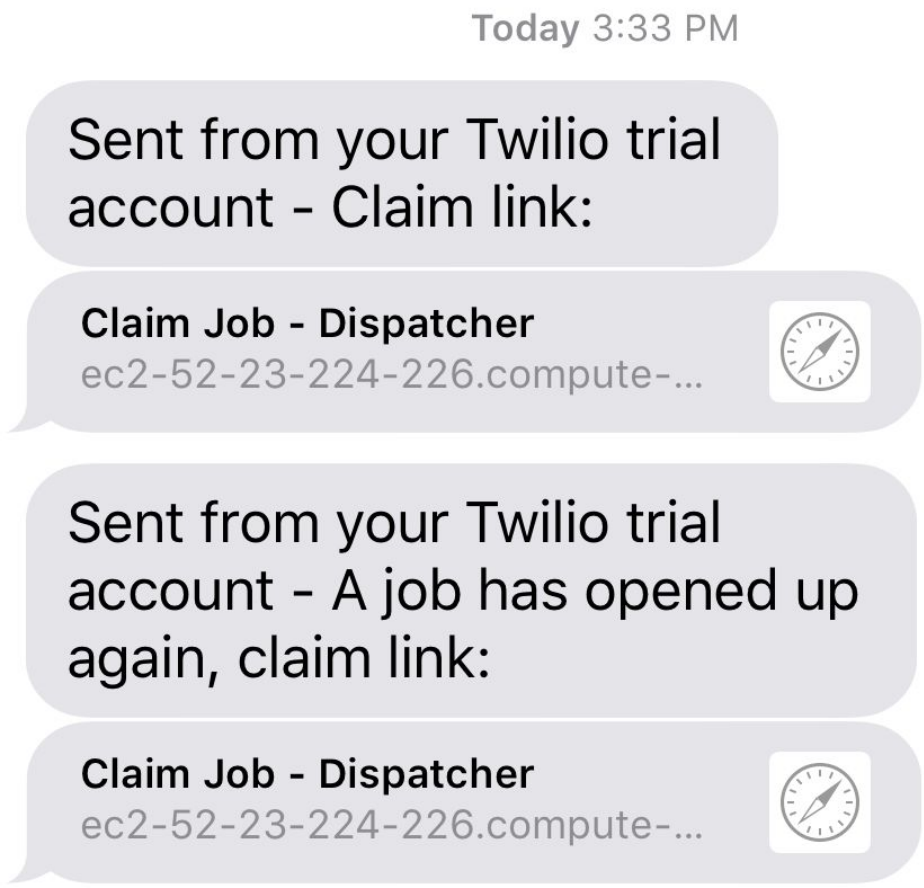


Claim Page

A job link will direct the driver to a page with all of the job information available to see. The driver can then choose to claim the link or ignore it.

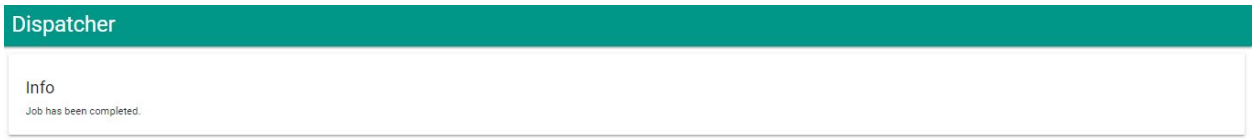
Complete or Cancel

After a driver claims a job, by using same job link, they are then able to either cancel or complete the job. Hopefully they are able to complete it.



Job Cancelled

If a job is cancelled, drivers for that merchant will be alerted that the job has opened up and is available to claim once again.



Job Completed

Once a job has been completed, the job link will confirm that information to the driver.

5.1.1. Register Business

When a merchant purchases the service, initial registration is performed through the user inputing desired information into a form. This is then packaged into a JSON object and sent to the business registration REST endpoint on the server, and upon successful registration (or an error), the server sends a response indicating either a successful status or an error message.5.1.2. Create Job

When the Submit button is tapped, the clover app constructs a JSON object containing the job title and job description entered by the user. The app then sends a request to the /business/{unique_id}/create_job REST endpoint on the server, and upon success a label will indicate that an SMS message has been sent to that business' registered drivers.

5.1.3. Current and Past Jobs

Merchant can view the status of each current and past job. While on this screen, the app will send GET requests to the server every 5 seconds to check if any jobs have changed status. A job is pending if it has not yet been accepted by a driver, claimed if it has been claimed by a driver but not yet completed, completed if it has been completed by a driver, and cancelled if the merchant has decided to roll back the job. In the latter two cases, the job will be moved from the current job list to past job list.

5.1.4. Current Drivers

Merchants can view the status of each of the drivers they have accepted, as well as accept or reject applicant drivers who have yet to be accepted into the pool of drivers. A driver's status is claimed if the driver has claimed a job and has not yet completed that job, cancelled if the driver has cancelled their job, completed if the driver has just completed a job, or idle if none of the above. If any drivers are pending approval, the pending tab will display the number of drivers that are pending.

5.1.5. Home Screen

The home screen allows merchants to navigate to the screens detailed in sections 5.1.2-4.

6. SMS Module

6.1. Logic Trace

A new message can be sent through Twilio with 5 parameters.

- Account ID
- Auth Token
- Registered Twilio Number (Outgoing)
- Destination Number
- Message Body

These parameters are then sent to Twilio through their API where they are processed and a message is sent out relative to those parameters.

6.2. Twilio Integration/Reliance

While Twilio appears to be the best choice for a SMS messaging service provider, our choice to use them is fairly flexible if a need to change does arrise. Because we are using SMS just for outgoing message alerts the system would only require a refactoring of single python script currently. If the project does grow to include SMS features over time we can maintain this flexibility by making sure to keep the script abstracted as much as possible. Ideally the only changes that would need to be made would be to the specific outgoing url as well as account credentials.

7. Test Plan

The project can be considered complete once all major components have been fully executed. Additionally no errors will be displayed to any client side features. Furthermore, all standards and features as outlined below have implemented with corresponding unit tests to ensure functionality with potential future features and upgrades.

7.1 Validation Standards

- Business is able to create a new account with Dispatcher
- Business is able to receive an ID to display for Driver registration
- Drivers are able to enroll with business through web link
- Business is able to see pending drivers
- Business is able to move a driver from pending to active
- Business is able to move a driver from pending to blocked

- Business is able to move a driver from active to blocked
- Business is able to move a driver from blocked to active
- Business is able to remove a driver from pending list
- Business is able to remove a driver from active list
- Business is able to remove a driver from blocked list
- Business is able to create a new job
- A dynamic web link is created for each active driver
- Each driver's dynamic web link displays that the job is open or claimed
- All active drivers receive a SMS message upon creation of a new job
- A driver is able to “claim” a job through a weblink
- Once a job is “claimed” another driver cannot claim that job
- Business is able to see status of active jobs