**Developing a Reliable and Economical Web Portal for Meals on Wheels**

**A Technical Report submitted to the Department of Computer Science**

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Joshua Santana
Fall 2019, Spring 2020

Technical Project Team Members
Michael Benos
Alexander Hicks
Kyle Leisure
Kevin Naddoni
Maxwell Patek
Nathanael Strawser

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

**Developing a Reliable and Economical Web Portal for Meals on Wheels**

**Table of Contents**

**Abstract**

During the 2019-2020 school year, our capstone team developed a new web application to replace the existing portal for Meals On Wheels' (MOW) Charlottesville office. Over eight months, we worked collaboratively with each other, our customer, and our professors/evaluators to iteratively develop an application that allowed Meals On Wheels to plan, track, assign, and deliver meals to its clients in the greater Charlottesville area. In the early stages of our project, we spent several meeting sessions with MOW staff to gather requirements and observe the existing portal. In doing so, we discovered several organizational flaws and inefficiencies in the existing portal, prompting a complete rebuild.

We iteratively developed a new web portal in twelve sprints, each lasting two weeks. Using GitHub projects, we transformed our customer requirements into actionable issues of varying story point values, which were assigned to team members at each sprint planning meeting. Some of such issues were broken down into smaller items when needed and assigned to subsets of our team. Every other Friday, we met with MOW staff to show our progress and acquire feedback about our design choices. By doing so, we were able to develop an application that was significantly faster, better organized, and highly intuitive.

Our web application is highly maintainable and built with modern frameworks and dependencies. The system serves both staff and volunteers in the day-to-day details of job assignment, meal packing, and route delivery. Staff are enabled to assign volunteers to specific jobs, update client information, and generate statistical counts and billing reports. Volunteers can view their assigned jobs and delivery routes as well as request and fill substitutions. Our web

application allows for improved workflow for MOW coordinators by enabling them to better plan and distribute meals to a large number of clients. In creating this portal, we are contributing to mitigating nutritional scarcity and hunger in America's population, and we believe that our application will robustly serve the local Charlottesville community for the next several years.

## List of Figures



## 1. Introduction

### 1.1 Problem Statement

Despite America being one of the richest countries in the world, an estimated eight million of its aging citizens face the threat of hunger (World Bank, 2019; NCOA, 2015). Meals on Wheels is America's oldest and largest organization dedicated to mitigating this issue through community chapters (MOWA, 2019). The non-profit's local chapter delivers meals to disabled or

elderly people in the Charlottesville-Albemarle area who cannot cook or buy food themselves. With the help of volunteers, the organization packs, labels, and distributes meals to customers via various delivery routes. In addition, volunteers drive a few shuttle routes to deliver meals to locations outside of the Charlottesville-Albemarle area (A. Dudley, personal communication, September 27, 2019).

While the greater U.S. Meals on Wheels organization sells professional software to help staff manage the complexity of their tasks, the Charlottesville office cannot afford it (A. Dudley, personal communication, October 11, 2019). Thus, staff managed volunteers, customers, and routes by hand until approximately three years ago, when a University of Virginia computer science capstone team created a web portal for them. Adopting that web portal gave Meals on Wheels' staff more time to focus on essential management by automating physical reports and tedious manual tasks, which include: managing delivery routes, maintaining current and prospective customer information, and ensuring that all daily jobs are filled by at least one volunteer, can get rather complex due to a combination of daily, weekly, biweekly, monthly, and one-time volunteer shifts and customer needs (S. Bayker, personal communication, September 13, 2019).

Unfortunately, there were several issues with the existing portal. First, staff complained that the web application had become increasingly slow over time. After examining the existing codebase, we believed this slowness was likely due to its cluttered, unclear data storage and the use of a cheap, inefficient hosting solution. Second, staff identified several organizational oddities within the app layout, making some tasks take longer than required, sometimes necessitating twice the number of clicks and screens. Finally, staff requested the addition of new

features, including historical report generation and general search functionality. It was clear that the system needed an update; however, the technical debt accumulated by the separate capstone teams developing features over a two-year period necessitated a rewrite (Allman, 2012).

### 1.2 Contributions

In the end, we were able to successfully deliver an freshly-built, improved web application to increase the efficiency and effectiveness of meal planning, volunteer delivery, and job filling.

The new application satisfies Meals on Wheels' needs and has a more reasonable and maintainable backend for long-term deployment, including state-of-the-art modularity via Docker, normalized database models, and cost-effective cloud deployment via Amazon Web Services (Microsoft, 2017). By redesigning and modernizing from the ground up, our project enables Meals on Wheels to operate at lower costs and function more quickly; the organization should have more time and money to help customers in need. The new web application was released on March 27th, 2020 and includes numerous critical features, such as client management, volunteer assignment, and route delivery.

## 2. Related Work

Meals on Wheels of Charlottesville has been using a custom software solution to assist in daily operations for years. The national chapter of Meals on Wheels partners with software company *Accessible Solutions, Inc.* to offer licensing options of a software called *ServTracker*;

although this software would fulfill the Charlottesville chapter's needs, this solution is not affordable given the Charlottesville chapter's budget (Accessible Solutions, Inc., 2018; A. Dudley, personal communication, October 11, 2019). For this reason, a prior capstone team was recruited to provide a pro bono solution. Over time, this custom solution became ineffective for a variety of reasons, including overall slow speed of some features, unpredictable system crashes and some features becoming obsolete altogether. Our task involved rewriting the custom application in use, prioritizing usability and stability while introducing new functionality so that Meals on Wheels of Charlottesville could be more productive. Two notable features we added that did not exist in the previous portal include: the "Manage Jobs" page (Fig. 1) and the "Open in Google Maps" button. "Manage Jobs" allows staff to view who is working on a particular day without generating reports and "Open in Google Maps" allows users to export route directions directly into Google Maps from the portal.

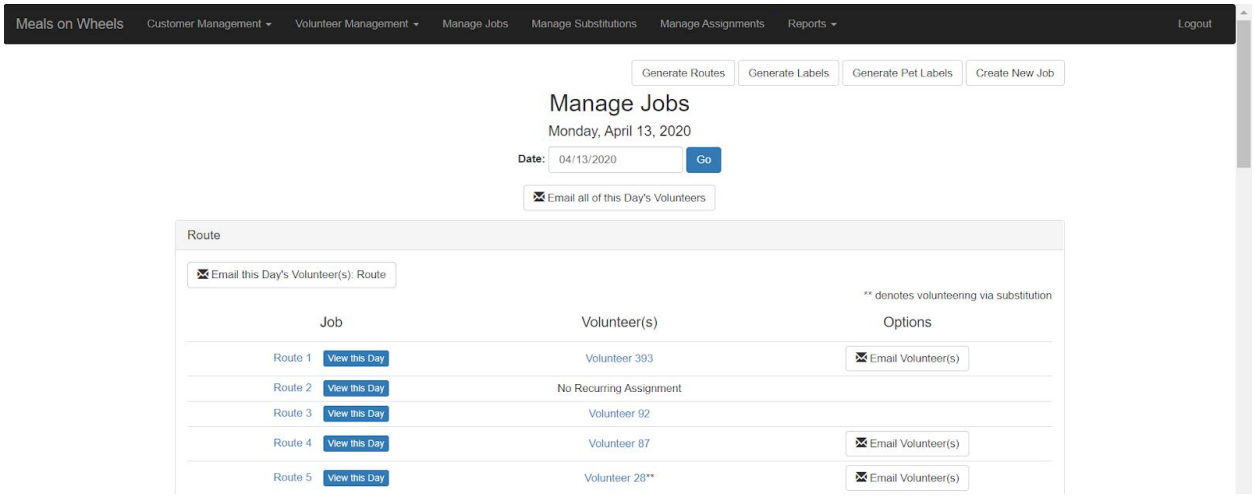


Fig 1. Screenshot of "Manage Jobs" page

## 3. System Design

At a high level, our application has two sets of users: the MOW staff and volunteers. The staff has the ability to manage jobs, customers, volunteers, assignments, routes, announcements, and substitutions. They can also view historical reports for billing and jobs. The volunteers have the ability to view the jobs they have signed up for, request a substitute for their assigned jobs, and take open substitutions.

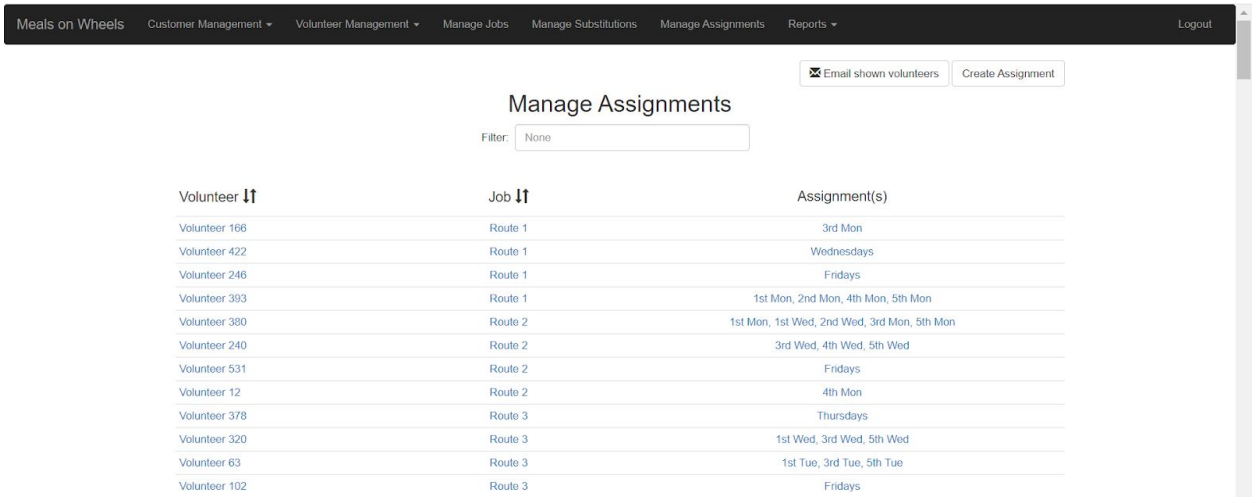


Fig 2. Screenshot of “Manage Assignments” page

This application is a rewrite of a previous capstone project. For this reason, we continued to use the GPL-3.0 license so that we could utilize the previous code. We decided to develop our application using Django, a python web framework, because it was used for the previous capstone project and because we all had prior experience with it.

### 3.1 System Requirements

By communicating with the MOW staff, we are able to gather system requirements and co-align our vision with their needs. Because we focused on consistent customer collaboration,

we were able to make healthy design decisions early on that improved our development speed later on. Furthermore, we spent less time rewriting features and more time robustly building out the rest of the application

Requirements for a Minimum Viable Product

- All Users
    - As a user, I should be able to create my own account (including custom username), so I can log in and see personalized information.
    - As a user, I should be able to request to change my password in case I forget it.
- Volunteers
    - As a volunteer, I should be able to release my route on a day, so someone else can substitute for that job.
    - As a volunteer, I should be able to pick up a released route on a particular day, so no routes go without a volunteer.
    - As a volunteer, I should be able to pick up a new route that has not been assigned to any volunteer, so I can plan my hours in advance.
- Staff
    - As staff, I should be able to create clients, so I can accommodate a growing client base.
    - As staff, I should be able to generate reports, so I can prepare daily operations.
    - As staff, I should be able to manually create delivery routes, so I can customize the volunteer's tasks.
    - As staff, I should be able to manually delete delivery routes, so I can avoid cluttering the portal with unused routes.
    - As staff, I should be able to assign volunteers to recurring routes, so I can plan delivery.
    - As staff, I should be able to substitute one-time volunteers for jobs, so I can ensure that all necessary jobs are filled.
    - As staff, I should be able to release volunteers from their recurring routes, so I can assign another volunteer to the recurring route.
    - As staff, I should be able to one-time release volunteers from their routes, so I can allow other volunteers to substitute.
    - As staff, I should be able to print reports that have been generated by any staff, so can have physical report copies.

    - As staff, I should be able to see who is volunteering on a particular day, so I can stay organized and communicate as necessary.

Desired Requirements

- All Users
    - As a user, I should be able to access the site from mobile platforms, so I can access the portal from my cell phone.
    - As a user, I should be able to navigate to each feature within 5 clicks, so that it is not too complicated to use.
    - As a user, I should be able to get familiar with the portal quickly, so I don't have to spend a lot of time learning how to navigate it.
    - As a user, I should be able to use the portal without it being slow so I can get things done efficiently.
- Staff
    - As staff, I should be able to sort volunteer schedules by the day of the week, so I can see who is scheduled for which routes each day.
    - As staff, I should be able to automatically email new volunteers through the portal, so I can welcome new volunteers to the portal.
    - As staff, I should be able to email volunteers scheduled for a given day, so I can have better day-to-day communication with the volunteers.
    - As staff, I should be able to store day-to-day statistics for meals for 3 months, so I can analyze substitutions and new volunteer counts.
    - As staff, I should be able to compile yearly reports, so I can submit them to the Board.
    - As staff, I should be able to access reports for at least a year, so I can refer back to past data if needed.
    - As admin-staff, I should be able to reset passwords for staff and volunteer accounts, so I can manage everyone's accounts if needed (lock-out, security, ...)
    - As staff, I should be able to remove clients, so I can eliminate confusion in meal planning
    - As staff, I should be able to update client data so I can accommodate any changes in their diet/address/...

## 3.2 Wireframes

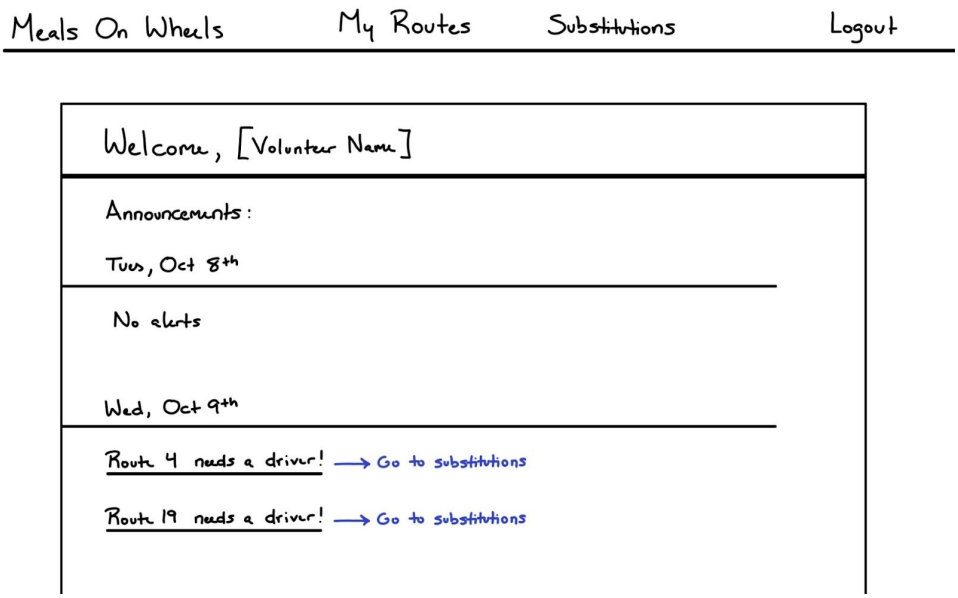


Fig. 4: High-fidelity wireframe

Wireframes are important to allow the customer to visualize the features proposed by the development team. Our team developed a series of wireframes to accomplish this, beginning with low-fidelity sketches. We used these sketches to develop higher fidelity wireframes, as shown in Figure 4. From this, we created an interactive prototype demo, as shown in Figure 5. Since our project involved maintaining the styling and much of the functionality of the legacy system, the demo focused on the changes that we were thinking to implement. The wireframes created before the interactive demo were useful for the development team to iterate on the proposed functionality, while the interactive demo itself was useful in conveying design decisions to the customer.

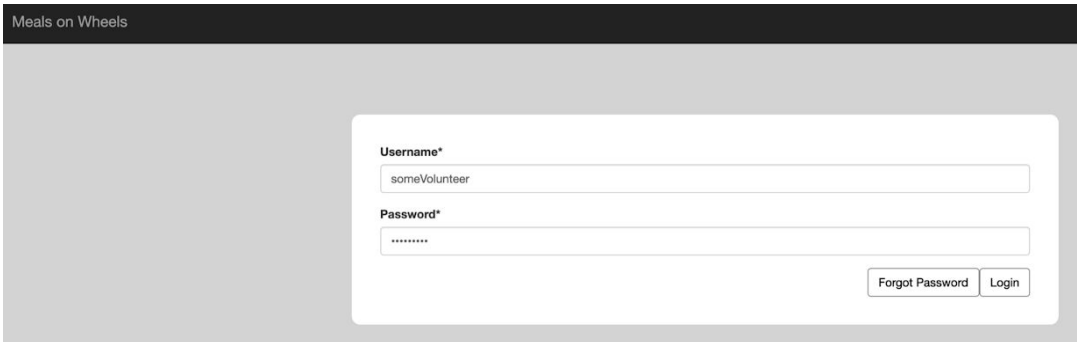


Fig. 5: Interactive demo presented to the customer

## 3.3 Sample Code

In this section are three sample views, three sample models, and three sample templates from our application.

This is the code for the view route on day functionality of the application. We have a more complex view for viewing and managing the general route, but this view is designed for the volunteer to be able to see all the necessary information for them to deliver to their route, including a google maps render of the route as well as instructions broken down by location. To do this, this view gets the route and the day. If the day is in a bad format, it will return volunteers to a 404 because there is nothing else for them to see, and it will return staff to the route management page.

```
@login_required
def view_route_day(request, route_number, date):
    """
    view route on a specific date
    """
    # get the route instance
    route = get_object_or_404(Route, number=route_number)
    is_staff = request.user.is_staff
    # convert the url to a datetime, if exception redirect based on auth
    try:
        date = datetime.datetime.strptime(date, "%m-%d-%Y")
    except ValueError:
```

```
    if is_staff:
        # go the the no specific day view if staff
        return HttpResponseRedirect(reverse("routes:view_route", args=[route.number]))
    else:
        # just 404 for the volunteers
        raise Http404

navbar = "navbar_staff.html" if is_staff else "navbar_volunteer.html"
customers = get_customers(route, date)

return render(
    request,
    "route-on-day.html",
    {
        "customers": get_customers(route, date),
        "route_name": route.name,
        "route_num": route_number,
        "date_picker_date": date.strftime("%Y-%m-%d"),
        "navbar": navbar,
        "MOW_LAT": MOW_LAT,
        "MOW_LON": MOW_LON,
    },
)
```

This view is the take substitution view that allows volunteers to fill a substitution that has been opened by another volunteer or staff member. This view posts the request from the volunteer, making sure that the substitution has been requested before assigning the new volunteer to the job.

```
@login_required
def take_substitution(request):
    """
    This function assigns a volunteer to a given job.
    """
    if request.method == "POST":
        try:
            sub = get_object_or_404(Substitution, pk=request.POST.get("pk"))
        except:
            log.error("Attempting to take substitution that does not exist.")
        if sub.assignment.volunteer == request.user.volunteer:
            # might as well say it is not via substitutions in this case
            sub.delete()
        else:
            sub.volunteer = request.user.volunteer
            sub.save()
        return HttpResponseRedirect(reverse("volunteers:open_jobs"))

    else:
        log.info(
            "Attempting to take substitution with {} method. Only POST allowed.".format(
                request.method
            )
        )
        return HttpResponseRedirect(reverse("volunteers:my_jobs"))
```

This view gathers all volunteers and renders a PDF of them ordered by the date they joined MOW. The to_pdf method takes in HTML and returns a FileResponse, making report generation simple.

```
@staff_member_required
def volunteer_join_date_report(request):
    """
    Generates a pdf list of volunteer join dates
    """

    # order clients by join date, then by name
```

```
    join_dates = Volunteer.objects.all().order_by("join_date", "user")
    template = get_template("pdfs/volunteer-join-date-report.html")

    return to_pdf(
        template.render({"join_dates": join_dates, "today": datetime.datetime.now(),})
    )
```

This model maintains all necessary data for the volunteers of MOW.

```
class Volunteer(models.Model):
    """
    Model for meals volunteers
    """

    user = models.OneToOneField(User, on_delete=models.CASCADE)
    organization = models.CharField(max_length=100, default="", blank=True)
    address = AddressField(null=True, blank=True, on_delete=models.PROTECT)
    home_phone = models.CharField(max_length=50, default="", blank=True)  # Home Phone
    cell_phone = models.CharField(max_length=50, default="", blank=True)
    work_phone = models.CharField(max_length=50, default="", blank=True)
    birth_date = models.DateField(null=True, blank=True)
    notes = models.TextField(default="", blank=True)
    join_date = models.DateField(default=date.today)
    number_of_people = models.IntegerField(default=1)
    dont_email = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.user.first_name} {self.user.last_name}"

    class Meta:
        ordering = ["user__last_name", "user__first_name"]
```

This model is for announcements that staff members post to the application. It keeps track of which user made it, as well as when it should stop being displayed in addition to the actual

content.

```
class ManagerAnnouncement(models.Model):
    created_by = models.ForeignKey(Volunteer, on_delete=models.PROTECT, null=True)
    display_until = models.DateField(null=True, blank=False)
    date_created = models.DateField(
        default=date.today, editable=False, blank=False, null=False
    )
    announcement = models.TextField(default="", blank=False)

    def __str__(self):
        return self.announcement
```

This model maintains historical records for volunteer data. A cron job that runs daily saves instances of this model to keep track of the volunteer, date, and job so that historical reports can be generated.

```
class VolunteerRecord(models.Model):
    """
    This serves to record-keep volunteers and what jobs they actually did
    """

    volunteer = models.ForeignKey(
        Volunteer, related_name="record", on_delete=models.SET_NULL, default=None,
null=True
    )
    job = models.ForeignKey(Job, on_delete=models.SET_NULL, default=None, null=True)
    date = models.DateField(default=date.today)
    original = models.ForeignKey(Volunteer, on_delete=models.SET_NULL, default=None,
null=True)
    is_substitution = models.BooleanField()

    class Meta:
        unique_together = ["volunteer", "job", "date"]
```

This template is for the “My Jobs” page. It renders all the jobs for a given month based on a URL parameter and lists those jobs, along with a link to their detail page if the job is also a route. Buttons at the top allow the user to toggle to different months.

```
<div class="container" style="padding-bottom: 5%;">
  <div align="right">
     {% if request.GET.month %}
      {% if request.GET.month != "0" %}
       <a href="{% url
'volunteers:my_jobs'%}?month={{request.GET.monthladd:-1}}"><button type="button"
class="btn btn-default">Previous month</button></a>
      {% endif %}
       <a href="{% url
'volunteers:my_jobs'%}?month={{request.GET.monthladd:1}}"><button type="button"
class="btn btn-default">Next month</button></a>
      {% else %}
     <a href="{% url 'volunteers:my_jobs'%}?month=1"><button type="button" class="btn
btn-default">Next month</button></a>

     {% endif %}
   </div>
   <center><h1> My Jobs for {{month}}, {{year}}</h1></center>

   </br></br>
      <table class="table" style="width:100%; font-size:medium">
    <tr>
      <th>Job</th>
      <th>Type</th>
      <th>Date</th>
      <th> </th>
    </tr>
    {% for job, date, date_url, is_sub, is_route, sub_or_assignment_pk in my_jobs %}
       <tr>
         <td>
         {% if is_route %}
          <a href="{% url 'routes:view_route_day' route_number=job.job.route.number
```

```
date=date_url%}">{{job.job}}</a>
          {% else %}
            {{job.job}}
          {% endif %}

          {% if is_sub %}
            <td>Substitution</td>
          {% else %}
            <td>Recurring</td>

          {% endif %}
          <td>{{date}}</td>
          <td><button type="button" class="btn btn-default"
onclick="post_substitute_request({{sub_or_assignment_pk}}, '{{date}}',
'{{is_sub}}');">Request Substitute</button></td>
        </tr>
      {% empty %}
        <td>There are no jobs to display.</td><td></td><td></td><td></td>
      {% endfor %}
    </table>
</div>
{% endblock %}
```

This is the template rendered to create an announcement which simply displays a form.

```
<div class="container" style="padding-bottom: 5%;">
    <h1>Create an Announcement</h1>
    </br>

    <form method="post" class="post-form">
    {% csrf_token %}
    {{form.media}}
    <div class="form-row">
        <p>
          {{ form.display_untillas_crispy_field }}
        </p>
```

```
    <div class="form-row">
      <p>
        {{ form.announcementlas_crispy_field }}
      </p>
    </div>

    <div class ="form-row">
      <button type="submit" class="btn btn-primary">Create</button>
    </div>
  </form>
</div>
```

3.4 Sample Tests

Testing is necessary to ensure that all written code functions as intended and to ensure that no new code breaks previous functionality. In this project, we use Django's unit test framework to accomplish this task. Unit tests are intended to isolate a specific subroutine. By doing so, it becomes clear what functionality breaks if these tests begin to fail.

The test below is found in the volunteer section of our application. Volunteers are assigned jobs, and they can find a list of these upcoming jobs in the portal. This test creates a new job, assigns it to the test volunteer as a recurring assignment, and then ensures that the response returned from viewing the "My Jobs" page contains the name of the job just created.

```
def test_displays_correct_job_name(self):
    job_type = JobType.objects.create(name="test_type")
    job = Job.objects.create(
      name="TEST NAME OF JOB", num_vols_required=1, job_type=job_type
    )
```

```
        job.save()
        recurring = Assignment(
            volunteer=self.test_volunteer.volunteer,
            job_id=job.pk,
        day_of_week=date_to_day_of_month(datetime.date.today()).day_of_week,
            week_of_month=date_to_day_of_month(datetime.date.today()).week_of_month,
        )
        recurring.save()
        response = self.client.get("/volunteer/my_jobs/")
        self.assertContains(response, "TEST NAME OF JOB")
```

On the “My Jobs” page for volunteers to view, both routes and packer jobs are listed. The application should recognize which jobs are routes and display links to their corresponding pages. This test verifies that functionality.

```
def test_route_link_shown(self):
        """
        This test ensures the correct route link is shown on the my_jobs page.
        """
        job_type = JobType.objects.create(name="test_type")
        job = Route.objects.create(name="Test Route", number=1, job_type=job_type)
        job.save()
        recurring = Assignment(
            volunteer=self.test_volunteer.volunteer,
            job_id=job.pk,
            day_of_week=date_to_day_of_month(datetime.date.today()).day_of_week,
            week_of_month=date_to_day_of_month(datetime.date.today()).week_of_month,
        )
        recurring.save()
        response = self.client.get("/volunteer/my_jobs/")
        self.assertContains(response, f"/routes/{job.number}/")
```

The following test ensures that attempting to manage a job for an invalid date will redirect to the current day's manage jobs page.

```
@freeze_time("2020-03-13")
def test_manage_open_job_date_not_valid(self):
        """
        make sure it handles the case with a bad date
        """
        day = datetime.date.today()
        week_of_month = (day.day - 1) // 7 + 1
        # need this stuff to create a substitution
        job_type = JobType.objects.get_or_create(name="test_type")[0]
        job = Job.objects.get_or_create(
            name="new-job", num_vols_required=1, job_type=job_type )[0]
        assignment = Assignment.objects.get_or_create(
            volunteer=None, job=job, day_of_week=day.isoweekday(),
week_of_month=week_of_month,)[0]
        response = self.client.get(
            "/staff/manage-open-job/{}/{}/".format(assignment.id, "02-30-2019")) # date does not
exist
        self.assertRedirects(
            response, "/staff/manage-jobs/{}/".format(day.strftime("%m-%d-%Y")))
```

## 3.5 Code Coverage

We use coverage.py in our application, which can be installed via: "pip install coverage." Since our project is dockerized, adding "coverage" to our requirements.txt file was sufficient; all requirements are installed at start-up. We created a Makefile target that starts the application, runs the code coverage tool, and outputs this information. The result can be found below.

Coverage report: **93%**

| Module | statements | missing | excluded | coverage |
|---|---|---|---|---|
| Total | 5040 | 367 | 0 | 93% |
| accounts/__init__.py | 0 | 0 | 0 | 100% |
| accounts/admin.py | 0 | 0 | 0 | 100% |
| accounts/forms.py | 50 | 5 | 0 | 90% |
| accounts/migrations/__init__.py | 0 | 0 | 0 | 100% |
| accounts/tests.py | 73 | 0 | 0 | 100% |
| accounts/urls.py | 4 | 0 | 0 | 100% |
| accounts/views.py | 55 | 23 | 0 | 58% |
| config/config.py | 7 | 0 | 0 | 100% |
| interfaces/__init__.py | 0 | 0 | 0 | 100% |
| interfaces/address_lookup.py | 20 | 1 | 0 | 95% |
| interfaces/recurrence.py | 93 | 1 | 0 | 99% |
| interfaces/tests/__init__.py | 0 | 0 | 0 | 100% |
| interfaces/tests/test_recurrence.py | 112 | 2 | 0 | 98% |
| legacy/__init__.py | 0 | 0 | 0 | 100% |
| legacy/admin.py | 7 | 0 | 0 | 100% |
| legacy/management/commands/importlegacy.py | 62 | 15 | 0 | 76% |
| legacy/migrations/0001_initial.py | 7 | 0 | 0 | 100% |
| legacy/migrations/__init__.py | 0 | 0 | 0 | 100% |
| legacy/models.py | 408 | 40 | 0 | 90% |
| legacy/tests.py | 264 | 0 | 0 | 100% |
| manage.py | 9 | 2 | 0 | 78% |
| meals/__init__.py | 0 | 0 | 0 | 100% |
| meals/constants.py | 17 | 0 | 0 | 100% |
| meals/settings.py | 42 | 1 | 0 | 98% |
| meals/urls.py | 8 | 0 | 0 | 100% |
| models/__init__.py | 0 | 0 | 0 | 100% |
| models/admin.py | 7 | 0 | 0 | 100% |
| models/migrations/0001_initial.py | 10 | 0 | 0 | 100% |
| models/migrations/0002_auto_20200128_1619.py | 4 | 0 | 0 | 100% |
| models/migrations/0003_auto_20200128_1636.py | 4 | 0 | 0 | 100% |
| models/migrations/0004_remove_customer_route_order.py | 4 | 0 | 0 | 100% |
| models/migrations/0005_auto_20200206_1756.py | 5 | 0 | 0 | 100% |
| models/migrations/0005_customer_historical_route.py | 4 | 0 | 0 | 100% |
| models/migrations/0006_auto_20200207_2047.py | 4 | 0 | 0 | 100% |
| models/migrations/0006_volunteer_dont_email.py | 4 | 0 | 0 | 100% |
| models/migrations/0007_auto_20200209_1730.py | 4 | 0 | 0 | 100% |
| models/migrations/0008_merge_20200212_1354.py | 4 | 0 | 0 | 100% |
| models/migrations/0009_auto_20200225_2248.py | 4 | 0 | 0 | 100% |
| models/migrations/0009_auto_20200226_1738.py | 4 | 0 | 0 | 100% |
| models/migrations/0010_auto_20200225_2304.py | 4 | 0 | 0 | 100% |

```
models/migrations/0010_volunteerrecord.py 6      0      0      100%
models/migrations/0011_auto_20200226_1000.py   4      0      0      100%
models/migrations/0011_auto_20200301_0007.py   4      0      0      100%
models/migrations/0012_auto_20200301_0116.py   5      0      0      100%
models/migrations/0013_remove_volunteerrecord_is_substitution.py      4      0      0
      100%
models/migrations/0014_auto_20200304_0944.py   5      0      0      100%
models/migrations/0015_auto_20200304_0947.py   5      0      0      100%
models/migrations/0016_volunteerrecord_is_substitution.py      4      0      0
100%
models/migrations/0017_auto_20200305_1620.py   4      0      0      100%
models/migrations/0018_auto_20200316_2125.py   4      0      0      100%
models/migrations/0018_merge_20200315_1707.py4      0      0      100%
models/migrations/0019_auto_20200316_2302.py   4      0      0      100%
models/migrations/0020_merge_20200317_0900.py4      0      0      100%
models/migrations/__init__.py      0      0      0      100%
models/models.py     293    14     0      95%
models/tests.py      238    0      0      100%
pdfs/__init__.py     0      0      0      100%
pdfs/admin.py 1      0      0      100%
pdfs/cron.py  72     2      0      97%
pdfs/migrations/__init__.py  0      0      0      100%
pdfs/templatetags/route_extras.py     26     4      0      85%
pdfs/tests/__init__.py 0      0      0      100%
pdfs/tests/test_cron.py174    5      0      97%
pdfs/tests/test_views.py     233    0      0      100%
pdfs/urls.py   4      0      0      100%
pdfs/views.py  142    7      0      95%
routes/__init__.py     0      0      0      100%
routes/admin.py      1      0      0      100%
routes/forms.py      27     0      0      100%
routes/migrations/__init__.py 0      0      0      100%
routes/models.py     1      0      0      100%
routes/tests.py 286    0      0      100%
routes/urls.py  4      0      0      100%
routes/utility.py    15     0      0      100%
routes/views.py      93     3      0      97%
staff/__init__.py    0      0      0      100%
staff/admin.py 1     0      0      100%
staff/all_views.py   15     0      0      100%
staff/forms.py 206    18     0      91%
staff/migrations/__init__.py  0      0      0      100%
staff/tests/__init__.py 0     0      0      100%
staff/tests/test_announcements.py     0      0      0      100%
```

```
staff/tests/test_assignment_management.py  306    0    0  100%
staff/tests/test_autocompleter.py           36    0    0  100%
staff/tests/test_customer_management.py     56    0    0  100%
staff/tests/test_email.py                   17    0    0  100%
staff/tests/test_index.py                   49    0    0  100%
staff/tests/test_job_management.py         109    0    0  100%
staff/tests/test_othermodels_deletions.py    0    0    0  100%
staff/tests/test_reports.py                 24    0    0  100%
staff/tests/test_substitution_management.py 223    2    0   99%
staff/tests/test_volunteer_management.py     0    0    0  100%
staff/urls.py                               20    0    0  100%
staff/views/announcements.py                22   12    0   45%
staff/views/assignment_management.py       127    0    0  100%
staff/views/autocompleter.py                33   17    0   48%
staff/views/customer_management.py          67   25    0   63%
staff/views/email.py                        10    0    0  100%
staff/views/index.py                        24    0    0  100%
staff/views/job_management.py              202   42    0   79%
staff/views/othermodels_deletions.py        26   12    0   54%
staff/views/reports.py                      70   53    0   24%
staff/views/substitution_management.py      82    0    0  100%
staff/views/volunteer_management.py         65   45    0   31%
volunteers/__init__.py                       0    0    0  100%
volunteers/admin.py                          1    0    0  100%
volunteers/migrations/__init__.py            0    0    0  100%
volunteers/models.py                         1    0    0  100%
volunteers/tests.py                        182    0    0  100%
volunteers/urls.py                           5    0    0  100%
volunteers/views.py                         90   16    0   82%
No items found using the specified filter.

coverage.py v5.0.4, created at 2020-03-29 23:52
```

3.6 Installation Instructions

The installation instructions for the Meals on Wheels management system are below and are broken down into three parts: provision an AWS instance, install Docker, Docker Compose, and Make, install the application.

*Provision an AWS instance*

First, register or login to an account on AWS and log in to the EC2 console in order to provision an instance. We recommend using "Ubuntu Server 18.04 LTS (HVM), SSD Volume Type" as the AMI. Next we recommend using at least a T2-small instance with protection for accidental termination (termination protection located on next page). We also recommend for storage a General Purpose SSD of 20 GB. There are no tags that need to be added, but in the configure security group page, add the default rules for HTTP and HTTPS from the Add Rule button in addition for the default rule for SSH. (If testing in a non-production environment, also allow port 8000). When prompted, create a new key pair and save the key somewhere safe and accessible.

*Install Docker, Docker Compose, and Make*

First, connect to your instance by right clicking on it in the dashboard and selecting Connect. Once connected, run the following commands

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER
sudo curl -L
https://github.com/docker/compose/releases/download/1.25.4/docker-compose-`uname
-s`-`uname -m` -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo apt update
sudo apt install make
sudo reboot
```

Now we let the app reboot before right clicking and connecting again.

*Install the Application*

To download the code and run the application run the following commands.

```
curl -fsSL http://cs.virginia.edu/~awh4kc/githubkey.gpg -o ~/.ssh/githubkey.gpg
cd ~/.ssh
gpg githubkey.gpg
```

Use password: M#gh7fRH06nD

```
eval "$(ssh-agent -s)"
chmod 600 ~/.ssh/githubkey
ssh-add ~/.ssh/githubkey
cd ~
git clone git@github.com:uva-cp-1920/Meals-on-Wheels.git
```

type yes when prompted

```
cd ~/Meals-on-Wheels/src
make env=prod deploy
```

Now you can verify the app is running by going to portal.cvillemeals.org in a browser.

## 4. Results

After spending nearly seven months iteratively developing the portal, we successfully deployed the application to production. The application meets all of the requirements mentioned in Section 3.1, thus solving our customer's issues. There are two main stakeholders that use the system, staff members and volunteers; the portal is designed to designate appropriate privileges and responsibilities to each account holder based upon stakeholder status.

Our customer, MOW administration, is in charge of the staff side responsibilities of the portal. To use the portal, staff log in and are redirected to the "Announcements" landing page,

where they can quickly see any open substitution requests. As staff, they have access to all of the management aspects of meal delivery; therefore, in the menu ribbon at the top of the page, they can manage jobs, customers, volunteers, assignments, and substitutions. The customer and volunteer management pages allow staff members to edit user information, including their meal recurrences and delivery availability. The "Manage Jobs" page provides staff an interface to view all of the daily jobs and assigned volunteers, including the ability to adjust the routes directly, generate the route delivery sheets for drivers, and open/fill substitution requests. The assignment and substitution management pages enable staff to view, edit, and create such data. Finally, staff can generate a variety of reports to count and summarize meal delivery, jobs, and substitutions over a specific time range via the reports dropdown.

MOW carries out its mission with a number of volunteers, the other type of primary stakeholder in the system. The portal interface for volunteers is simpler because volunteers only need minimal information about their specific role in meal delivery. On the volunteer-side, users can see a list of their monthly upcoming jobs, along with a map of how to deliver meals for each route. Furthermore, the menu ribbon allows volunteers to view open substitution requests and fill them if they want to volunteer for an extra job. Finally, the last ribbon option is for volunteers to view their own profile information. By having a simple interface for volunteers, we have designed an intuitive system allowing them to focus exclusively on their meal delivery job.

Comparing our new system to the old portal reveals a number of concrete improvements to speed and ease of use. While many of our optimizations consisted of backend organization or improvements to UI, they are most clearly reflected in the reports. The MOW staff rely on the reports to provide drivers with accurate information for their routes, for billing purposes, and for

a variety of other mission critical tasks. By consolidating/restructuring the existing Django models, we were able to cut the number of reports needed to encapsulate the same data from 16 on the old site to 7 on ours. For example, the job overview report now generates the data from what used to be three separate reports into one larger one. Reducing the number of required reports allows the MOW staff to more efficiently collect the information they need to operate on a day to day basis.

Perhaps the most significant improvement to the reports is the speed in which they are generated. One of MOW staff's biggest complaints about the existing portal was how long it took for it to create and display the printable pdf reports. If the aforementioned job overview report generated at all without a timeout error, it would take on the order of minutes. It now takes approximately 10 seconds to generate data for an entire month because of how we cleaned up the data storage and management on the backend. Other reports, like the billing report (Fig. 3), used to take about 10 seconds to load; they are now more or less instantaneous. Finally, there were some reports, such as the Daily Count sheet, that simply yielded incorrect data. In this particular case, the report would always show a count of 0 for meals for a day; this issue has been solved in our system.

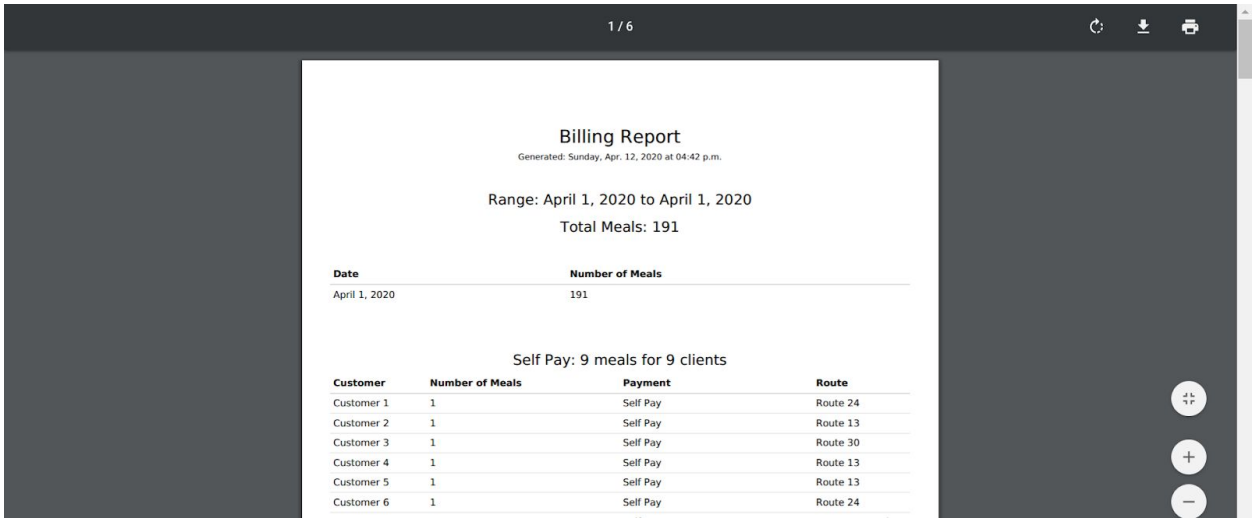


Fig 3. Screenshot of “Monthly Billing” report

## 5. Conclusions

As we have developed together over the course of the year, our team feels as though we have learned a lot in both the technical aspect and team portion of the capstone project. By working on a project that directly impacted real-world customers, we felt motivated and obligated to provide the best product to improve the lives of so many people. At the beginning of this project, we underestimated the complexity and depth that would be required in designing an application of this type from scratch. However, as we learned and communicated more with our customer, we realized the initial shortcomings in previous iterations of this app. Once we understood that these shortcomings were the real problems with the portal, we strove to design, code, and test extensively to ensure that every feature was intuitive, tested, and efficient.

In summary, we have proudly built an application that serves the local Charlottesville community. In doing so, we have become better developers, communicators, team members, and are eager to apply and improve our skills to serve society further in the future.

## 6. Future Work

The web application will likely require maintenance in the future as packages become outdated and/or Meal on Wheels' needs change. Changes in these needs are difficult to predict; though, they may entail improving the application's design to allow for better scaling or adjusting the time the application retains data. Aside from maintenance, further work could include acquiring volunteer input about their side of the web application. This side of the application has limited functionality and use, but substantial feedback about its user interface could be valuable as we will likely not have easy access to it due to Meals on Wheels' temporary COVID-19 closure around our deployment date.

## 7. References


Accessible Solutions, Inc. (2018, June, 13). Accessible Solutions, Inc. Announces Meals On Wheels America® Partnership. https://accessiblesolutions.com/news/news-2/accessible-solutions-inc-announces-meals-on-wheels-america-partnership

Allman, E. (2012). Managing technical debt. *Communications of the ACM*, *55*(5), 50. https://doi.org/10.1145/2160718.2160733

Meal Provider Software - ServTracker: Meals on Wheels: Senior Nutrition. (n.d.). Retrieved from https://accessiblesolutions.com/meal-delivery-software

Microsoft (2017). Description of the database normalization basics. https://support.microsoft.com/en-us/help/283878/description-of-the-database-normalization-basics.

MOWA. (2019). Meals on Wheels America. National Office. https://www.mealsonwheelsamerica.org/learn-more/national

NCOA. (2015, June 4). National Council On Aging. Facts About Senior Hunger. https://www.ncoa.org/news/resources-for-reporters/get-the-facts/senior-hunger-facts/

World Bank (2019). GDP per capita. https://data.worldbank.org/indicator/ny.gdp.pcap.cd