e6 - waitr

Sophie Sacks, Sam Lavelle, Dina Razek

Use Case and Use-Case Requirements

Scenario and Solution

- Hard to know how busy eateries are
- Combined hardware and software solution: two radio frequency identification (RFID) scanners + web application
- Scan in at beginning and end of line
- Web application for users to view with machine learning in backend

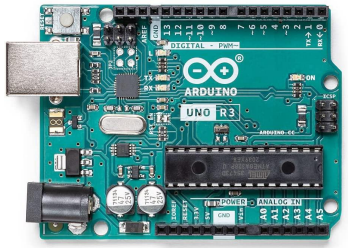
Quantitative Requirements

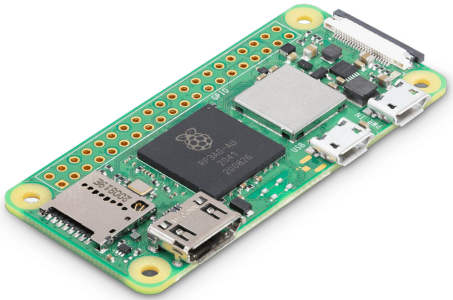
- AC-powered, OR battery powered scanner lasting **3 hours** or more
- Transmission from scanner to server within **200 milliseconds**
- Web application response time in under **2 seconds**
- Margin of error for wait time: **2 minutes**, or within **10%**
- Disregard ID numbers if 3 more ID numbers scan out before it
- Ability to keep track of up to **50 patrons**

Solution Approach

The Solution

- 2 RFID readers
- A board to send the data from the readers to the server
- A web application to publish the wait time
- Justified by accuracy, anonymity, and experienced





The Materials

- RFID Readers
  - 2 Arduinos
  - Mag wire for inductor coils, possible 3D printed mount
  - Power source - battery or cable
- Board
  - Raspberry Pi Zeros
- Web application
  - AWS server
  - SQL database through Django
  - Python backend (ML library)
  - HTML/CSS/Javascript frontend

System Specification - Hardware

Our inductor coil + circuit will detect the presence of the RFID chip inside of the user's ID, and send the bitcode to the Arduino

The Arduino will activate an LED light signaling the presence of and ID, and collect the ID's bitcode

The Raspberry Pi will wirelessly send the bitcode to our software

System Specification - Software Frontend

Specification

- **Tools**: HTML, CSS, and Javascript
- **On open and reload:** send request to backend to retrieve and display wait time
- **Security**: input validation through an allow list in *forms.py*
- **AWS**: EC2 instance to run entire web application

Low-Fidelity Design Plan

System Specification - Software Backend

Specification

- **Tools**: Django MVC architecture
- **Python packages:** numpy, pandas, joblib, sklearn
- **Stored in Django SQL Database:** data from RFID scanner, user input, ML model predictions
- **Security:** input validation through Django validators

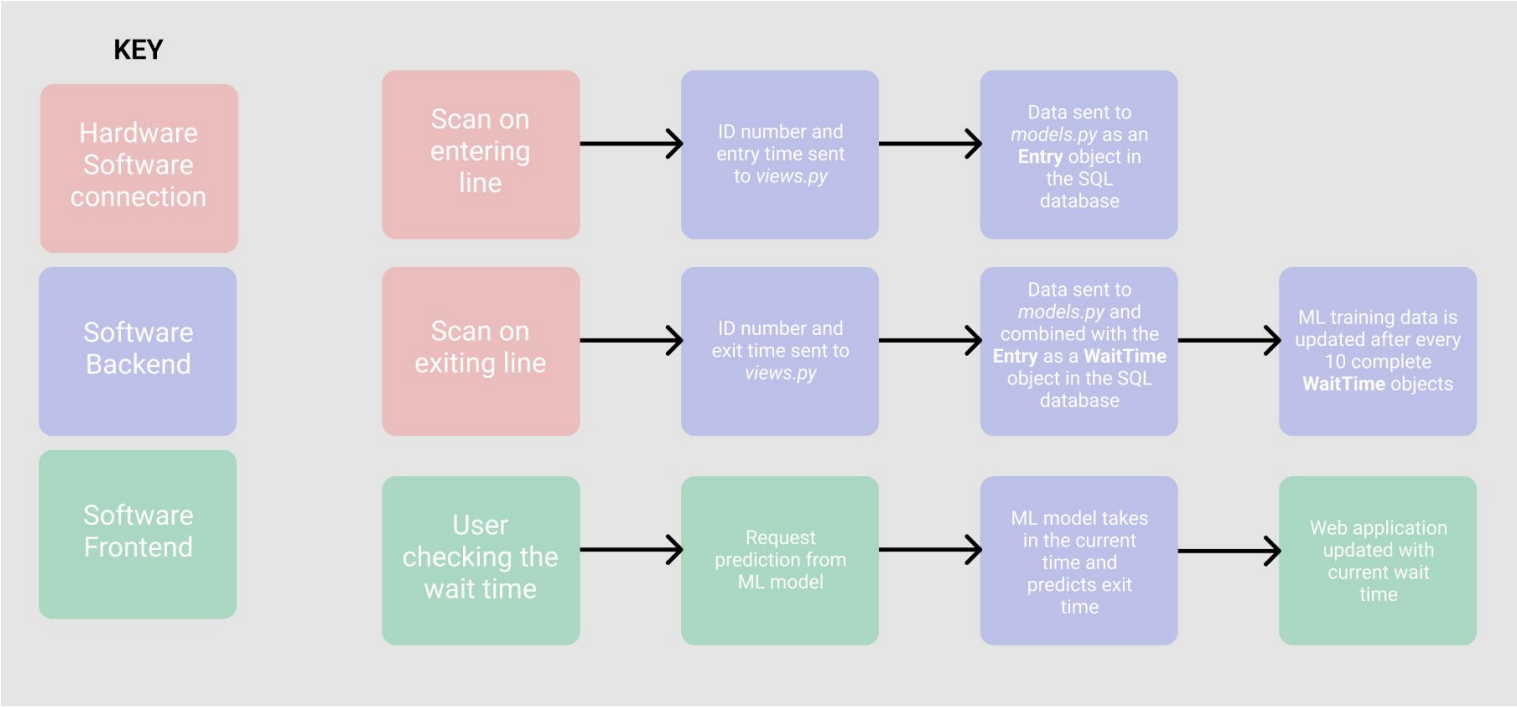
Backend Design Plan

models.py

waitr > waitr > models.py > ...

```python
from django.db import models

class Entry(models.Model):
    id_number = models.CharField(max_length=200)
    entry_time = models.DateTimeField(blank=True, null=True)

    def __str__(self):
        return str(self.id_number) + ',entry_time=' + self.entry_time

class WaitTime(models.Model):
    entry = models.DateTimeField(blank=True, null=True)
    exit = models.DateTimeField(blank=True, null=True)

    def __str__(self):
        return str('entry=' + self.entry) + ',exit=' + self.exit

    def save(self, entry_time, exit_time):
        self.entry = entry_time
        self.exit = exit_time
        super().save()
```

System Specification - Software Data Flow



Implementation Plan - Hardware

Design + Implementation

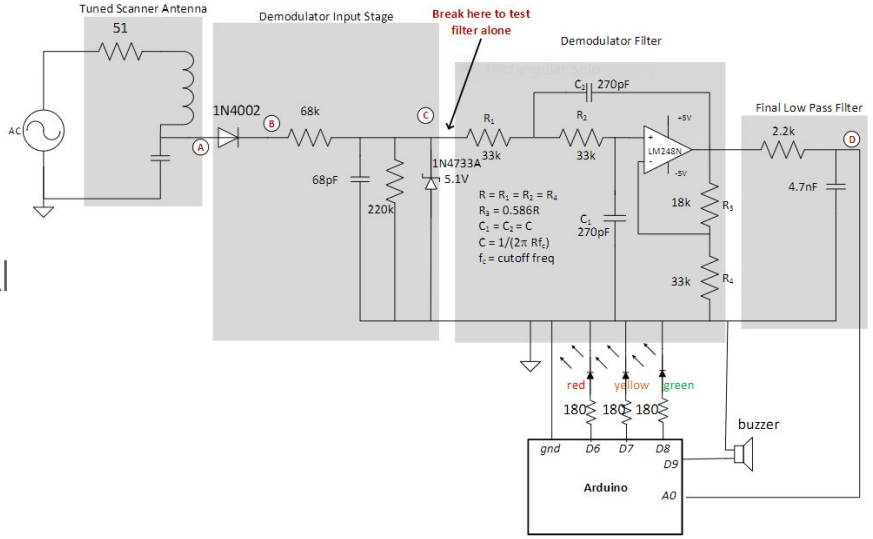
Copying
- RFID circuit design from 220

Modifying
- Arduino code from 220

Designing
- Custom inductor coil to read RFID signal

Buying
- Raspberry Pi Zero

Building
- Circuit (on breadboard + soldering)
- Inductor coil
- System housing (3D print)



Implementation Plan - Software

Design + Implementation

Modifying

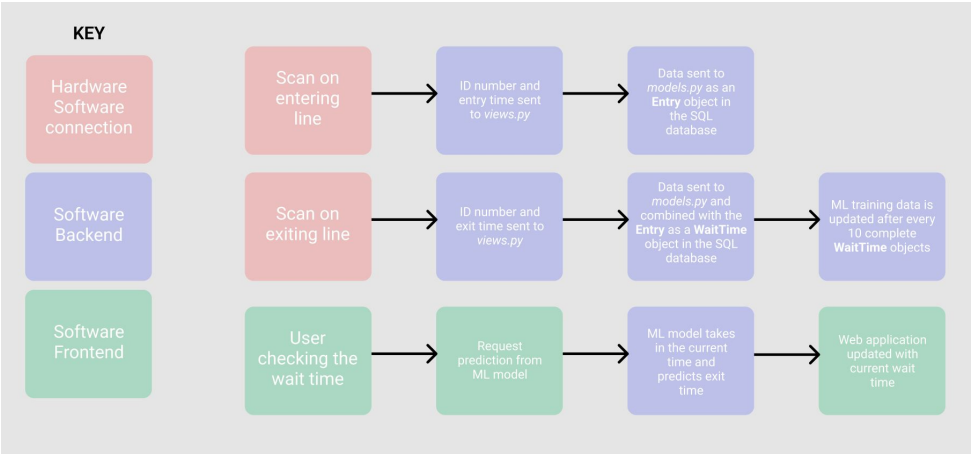
- Typical Django backend structure to fit our specifications

Downloading

- Python packages for ML model

Designing

- ML model to predict wait times
- Web application frontend
- Hardware software connection



Testing, Verification, and Validation

Test Inputs		Passing Test Outputs
ID card to be read on RFID scanner		RFID reader detects presence of ID card and its bit code
RFID scanner output (current date and time)		Received properly by the RPis within 200 ms
Date and time data from the RPis		Correct storage in the Django SQL database (entry vs. exit)
SQL database data		ML model predicting wait times
Web application frontend page reload		Accurate wait time prediction from the ML model (margin of error within 2 minutes or 10% of actual wait time)

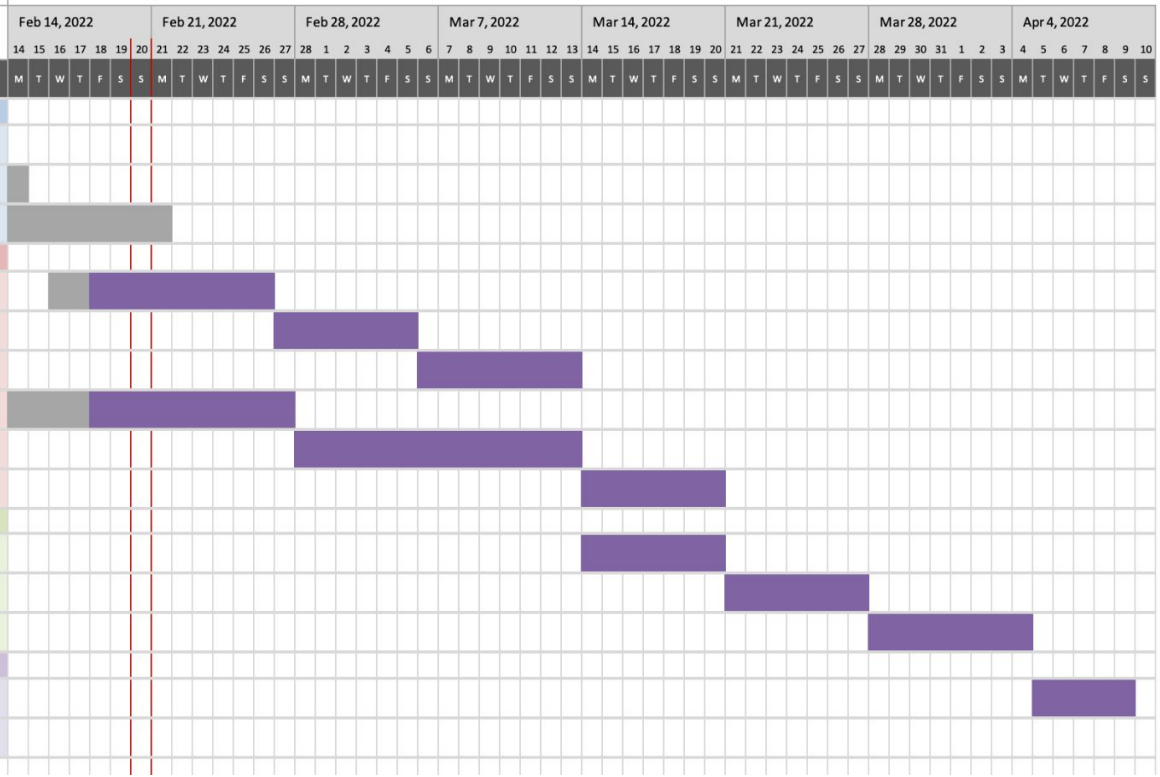
Project Management Gantt Chart

Sam Lavelle, Dina Razek, Sophie Sacks
18-500

https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

Project Start: Sun, 2/13/2022
Display Week: 1

TASK	ASSIGNED TO	PROGRESS	START	END
Setup				
Set Up Website	Sam, Sophie	100%	1/31/22	2/5/22
Design Solution Implementation	All	100%	1/31/22	2/14/22
Order Parts	Sam	100%	2/14/22	2/21/22
Build Solution				
HW: Circuit on breadboard	Sam	20%	2/16/22	2/26/22
HW: Arduino + Rpi	Sam	0%	2/27/22	3/5/22
HW: Solder	Sam	0%	3/6/22	3/13/22
SW: Frontend	Dina	30%	2/14/22	2/27/22
SW: Backend	Sophie	0%	2/28/22	3/13/22
SW: ML	Sophie, DIna	0%	3/14/22	3/20/22
Testing				
HW ability to read CMU ID	Sam	0%	3/14/22	3/20/22
Ability to transmit from HW to SW	All	0%	3/21/22	3/27/22
Ability to calculate wait times from data	Sophie, DIna	0%	3/28/22	4/4/22
Integration				
Integrate wait times into web app	Sophie, Dina	0%	4/5/22	4/19/22
Explore execution in eatery	All	0%	4/20/22	4/27/22



Next Steps

Hardware

- Build inductor coil
- Build circuit on breadboard
- Test with Arduino + RPi
- Solder final circuit

Software

- Finalize high-fidelity frontend design and begin implementation (HTML, CSS, JS)
- Set up backend views, forms, and SQL database
- Begin training ML model