CS 410 - TymeWear
Backend Integration

CLIENTELE  Juan Morales
INSTRUCTOR  Marc Pomplun
DEV-TEAM  Endy I. Imam, Hanfei Xu, Cahao Vien
PRODUCT AND PROJECT OVERVIEW

ENDY IMAM
Virtual coaching powered by smart clothing technology.
Smart electronics
- Wireless communication
- 9 axis motion tracking
- Pace, Cadence, Movement Analysis

Soft sensing technology
- Seamless integration
- Breathing sensors
- Aerobic and Anaerobic Threshold
Threshold Test
App guided 10 min treadmill test

Personal Training Plan
Workouts personalized to your fitness level

Feedback & Coaching
Reach your goals faster
System Architecture

Sensors → Electronics → iOS App → Backend → Frontend
Data Sending

iOS ➔ Backend

TymeWear Smart Shirt USB Device ➔ iOS Phone + TymeWear App Display ➔ Backend Request Controller ➔ Output JSON File ➔ TymeWear Backend [AWS]

[Processor] AWS EC2 [from Elastic Beanstalk]

[Storage] AWS S3

[Database] AWS RDS
DEVELOPMENT ENVIRONMENT

ENDY IMAM
$ eb init
$ eb create huh?
$ eb deploy
$ eb help
$ eb idontgetit

Error: permission denied
* (Seriously, thank you for your help, Gabe)
Development Cycle Methodology

Source: Program and edit the codebase and collaborate with source control (ex. GitHub, AWS CodeCommit)

Build: Run and test the code ensuring that it works as intended (ex. Locally with Django’s Test)

Deploy: Push the code to production (or staging) release have it run in the public.

AWS Elastic Beanstalk can handle it for us (mostly)!
Elastic Beanstalk Full AWS Development Pipeline


TYMEwear
Elastic Beanstalk Alternate Development Pipeline

Source → Build → Deploy

- User
- GitHub Remote Repository
- Jenkins CI
- AWS CodeBuild
- AWS Elastic Beanstalk
Elastic Beanstalk **Our Development Pipeline**

Source → Build → Deploy

User → GitHub Remote Repository → Django Testing → AWS CodeBuild → AWS Elastic Beanstalk
How To Start

Please do it in a Linux environment or at least a VM.

Linux

0. Ensure you have Python3 at version 3.6.7 and Pip3 at version 9.0+

```bash
[-]$ python3 --version
Python 3.6.7
[-]$ pip3 --version
pip 9.0.1 /usr/lib/python3/dist-packages (python 3.6)
```

1. Find a directory in your local machine to clone the repo.

```bash
[-/thisDir]$ git clone https://github.com/EndyPremier/Backend.git
```

2. Install virtualenv and check, then setup and activate a virtual environment.

```bash
[-/thisDir]$ pip install virtualenv
[-/thisDir]$ virtualenv --version
16.4.3
[-/thisDir]$ virtualenv <env_name>
[-/thisDir]$ source ./<env_name>/bin/activate
(<env_name>)[-/thisDir]$
```

3. With the environment running, install all the required python packages in the requirements.txt file in the local repo.

```bash
(<env_name>)[-/thisDir]$ cd Backend
(<env_name>)[-/thisDir]Backend]$ pip install -r requirements.txt
```

4. Setup a PostgreSQL Server and link it to the project. (See Below)
5. Test the Django Application Locally. (See Below)
6. Deploy Django Application to Elastic Beanstalk. (See Below)
7. If you want to deactivate your running environment:

```bash
(<env_name>)[-]$ deactivate
```
MAIN TAKEAWAY COMMUNICATION IS KEY
POSTGRES DATABASE

HANFEI XU
Raw Date from Front End

Single Json File for One Activity

```
{"pod_id": "F131",
"samples": [{"gz": 27, "my": 17, "mz": 49, "ax": 27, "c": 6, "gx": 23, "ay": 49, "ecg": 0, "mx": 20, "cr": 36, "gy": 9, "az": 22}, ...
],
"type": 0,
"time_stamp": "2019-03-18T13:37:10.679Z",
"info":{"end-time": "13:37:22",
"activity-type": "Normal Activity",
"ird": 60, "wd": 180, "ssi": 0.1, "feel": 0, "frd": 60,
"activity-name": "Simulated data ", "ws": 3,
"avg-speed": 0, "date": "03-18-2019",
"start-time": "13:37:10", "note-detail": "",
"miles": 0, "sd": 10, "duration": "00:12", "sis": 3,
"sfs": "VT", "max-speed": 0}
```

Samples Collected During One Activity

```
"samples":
[{
"gz": 27, "my": 17, "mz": 49, "ax": 27, "c": 6, "gx": 23, "ay": 49, "ecg": 0, "mx": 20, "cr": 36, "gy": 9, "az": 22},
{"gz": 22, "my": 9, "mz": 43, "ax": 0, "c": 22, "gx": 37, "ay": 42, "ecg": 0, "mx": 30, "cr": 10, "gy": 24, "az": 47},
{"gz": 43, "my": 41, "mz": 15, "ax": 49, "c": 20, "gx": 49, "ay": 24, "ecg": 0, "mx": 27, "cr": 8, "gy": 47, "az": 22},
{"gz": 41, "my": 47, "mz": 24, "ax": 19, "c": 37, "gx": 34, "ay": 2, "ecg": 0, "mx": 27, "cr": 35, "gy": 31, "az": 47},
{"gz": 6, "my": 6, "mz": 15, "ax": 25, "c": 19, "gx": 27, "ay": 25, "ecg": 0, "mx": 44, "cr": 48, "gy": 6, "az": 29},
{"gz": 39, "my": 13, "mz": 47, "ax": 35, "c": 39, "gx": 45, "ay": 17, "ecg": 0, "mx": 49, "cr": 4, "gy": 45, "az": 48},
{"gz": 25, "my": 16, "mz": 36, "ax": 44, "c": 3, "gx": 15, "ay": 10, "ecg": 0, "mx": 23, "cr": 7, "gy": 40, "az": 90},
{"gz": 31, "my": 35, "mz": 8, "ax": 49, "c": 3, "gx": 48, "ay": 39, "ecg": 0, "mx": 20, "cr": 20, "gy": 29, "az": 29}]
```
# Database Schema

## TYMEActivity
- **id**: uuid
- **name**: varchar(255)
- **user**: int
- **pod_id**: varchar(255)
- **type**: varchar(255)
- **duration**: varchar(20)
- **data_file**: file
- **time_stamp**: datetime
- **start_time**: time
- **end_time**: time
- **created**: datetime
- **average_speed**: float
- **feel**: int
- **frd**: int
- **ind**: int
- **max_speed**: float
- **miles**: float
- **sd**: int
- **sts**: varchar(10)
- **ssl**: int
- **ssi**: float
- **wd**: int
- **ws**: int
- **fs**: int

## TYMEActivityNote
- **note**: varchar(350)
- **tymeadvive**: int

## Additional/Data
- **id**: int
- **cbr_ren_samples**: int
- **data_file**: varchar(100)
- **tymeadvive_id**: uuid

## ThresholdPacePoint
- **date_activity**: date
- **threshold_speed**: float
- **threshold_pace**: time
- **recovery_pace**: time
- **sprint_pace**: time
- **tymeadvive**: table

## CBRPointsTV
- **init_time**: float
- **description**: varchar(500)
- **chest_breathing_rate**: int

## CBRPointsIOS
- **init_time**: float
- **description**: varchar(500)
- **chest_breathing_rate**: int

## CBRPointsNote
- **init_time**: float
- **title**: varchar(100)
- **description**: varchar(500)
- **chest_breathing_rate**: int

## MyUser
- **id**: int
- **email**: varchar(255)
- **password**: varchar(128)
- **user_type**: varchar(10)
- **first_name**: varchar(50)
- **last_name**: varchar(50)
- **is_active**: boolean
- **is_staff**: boolean
- **is_admin**: boolean
- **joined**: date
- **birthday**: date
- **address**: varchar(50)
- **gender**: varchar(50)
- **height**: float
- **weight**: float

## PermissionsCustom
- **id**: int
- **permission**: varchar(255)
- **description**: varchar(255)

## MyUserPermissionsCustom
- **id**: int
- **user_id**: int
- **permission_id**: int

## trainers
- **id**: int
- **from_myuser_id**: int
- **to_myuser_id**: int

## groups
- **id**: int
- **myuser_id**: int
- **group_id**: int
Manage Database

tymewear/apps/main/models.py

class TYMEActivity(models.Model):
    ...
    id = models.UUIDField(primary_key=True,
                           default=uuid.uuid4, editable=False)
    name = models.CharField(max_length=255,
                            default='Activity')
    user = models.ForeignKey(
        MyUser,
        on_delete=models.CASCADE
    )
    ...

$python3 manage.py makemigrations
$python3 manage.py migrate

manage.py
Read and Write Test Through Interactive API

http://twtestei-env.yw2meevthp.us-east-1.elasticbeanstalk.com/docs/#tyme-activities-list
Post & Get Data

tymewear/apps/main/urls.py

urlpatterns = [
    ...
    re_path(
        r'^api/tyme-activities/$',
        TYMEActivityListAPIView.as_view(),
        name='tyme_activities'   
    ),
    ...
]

tymewear/apps/main/views.py

class TYMEActivityListAPIView(generics.ListCreateAPIView):
    def create(self, request, *args, **kwargs):
        ...
        serializer_class = TYMEActivitySerializer
        if 'info' in data:
            info = data['info']
            keys = ['duration', 'feel', 'frd', 'ird', 'miles', 'sd', 'sfs', 'sis', 'ssi', 'wd', 'ws', 'pod_id', 'fs']
        ...
    def get_queryset(self, date_ini, date_end, type):
        ...

tymewear/apps/main/serializers.py

class TYMEActivitySerializer(serializers.ModelSerializer):
    ...
    class Meta:
        model = TYMEActivity
        fields = ('id', 'name', 'time_stamp', 'duration', 'user', 'type', 'created', 'start_time', 'end_time', 'fs', 'average_speed', 'feel', 'frd', 'ird', 'max_speed', 'miles', 'sd', 'sfs', 'sis', 'ssi', 'wd', 'ws')
To be continued

Not every data field is being used
- Serializer
- Models
- Views

Batch test
- Post json through Postman
- Post json through script
AWS INTEGRATION, RESEARCH AND TEAMBUILDING
CAHAO (HAO) VIEN
Refactored Django REST Service implementation to be layered in a Model View Controller Layout.

Poor knowledge of Django Framework made process longer.

Eventually got it running with the MVC Refactor but still awaiting development on fork.

Endy made a contribution to make the fork successfully implementable.
Hash#2: Multipart Multithreaded Encoded Transport

- Used an incremental approach to development: File Sendt → Partition → Encrypt/Decrypt → Combinations of prior

- Used validated 256 bit Hash over Advanced Encryption Standard/Cipher Block Chaining as the working procedure for Encryption

- File Transport using Sockets over HTTP was verified

- Ideas were sent to Gabriel for further research

- Prospectus: May vote for Django’s native support of Multipart instead and just modify to include encryption as needed
Hash#3: SOAP/REST Service
doGET and doPOST handlers

- doPost and doGet leverage same URL in Django dispatching
- Iteratively parsed PostGreSQL data for CBRSmootheredData Model
- Ran Tymewear Smoothing Algorithm over data
- Re-parceled data into JSON String for REST/SOAP response
- Further debugging and interfacing necessary
Hash#4: AWS Deployment and Debugging

- Learned that Elastic Beanstalk is an automated Command Line Wizard Utility that leveraged RDS, EC2, and S3

- Magic keywords: pip install awsebcli
git clone <your magic repo>
eb init [follow prompts to glory]
git add * [for CodeCommit]
git push
eb deploy
<magic>

------------------------------ *~~*~*

- Demonstrated this beauty

- Forked work to Endy because of cost concerns
QUESTIONS?
Thank You