Destiny Vault Raider

Allyn Hunt
Blog: AllynH.com
Twitter: @Allyn_H_
Web application: DestinyVaultRaider.com
Introduction:

New father.
Engineer.
No prior experience with Python or Web Development.
Started Destiny Vault Raider as a hobby project and learning experience.
Blogging my experiences with Python and Destiny Vault Raider.
   Originally I wanted to keep project scope to pure Python, HTML and CSS.
Destiny the game:

Online first person shooter video game.
Developed by Bungie (Halo series).
Destiny 1 (September 2014).
Destiny 2 (September 2017).

Player vs Player:
  Casual and Competitive.
  Periodic events (Iron Banner).

Player vs Enemy:
  Strikes.
 Raids.
Destiny API:

Statistics:
  Number of kills, deaths, assists, precision kills.
  Game mode, time played, game score.
  Players met.

Activity / Milestone tracking:
  Track completion level of quests.
  Track steps on next quest.

Name: Savathun's Song
Des.: Delve deep into the Hive-infested Arcology in search of missing fireteams.
Mode: Strike
Kills: 64
Deaths: 5
Time: 22m 11s
Destiny API:

Interact with the in-game world:
  View items.
  Transfer items between characters.
  Equip items on character.

View vendor / collectable data:
  Weapons and armour for sale.
  View perks, modifications on weapons.
Inventory management solves a real problem:

About to raid and forget your favourite gun?

1. Change character
2. Fly to tower
3. Deposit weapon into vault
4. Change character
5. Fly to tower
6. Collect weapon from vault
7. Fly to activity
Destiny Manifest:

SQLite based lookup table. Contains definitions of all of the items in the Destiny world. Names, description, icon, activity name, enemy names, faction details, stats etc.

Download from (blog post with more details):
manifest_url = 'https://www.bungie.net/Platform/Destiny2/Manifest/
res = requests.get(manifest_url, headers=HEADERS)
App Structure:

Built in Flask with Python 2.7.13.

App structure is taken from Miguel Grinbergs blog and book.

Uses Flask-Login for user session management.

Flask-Script Manager to manage CLI in production and development environments.

SQLAlchemy used to manage user databases.

PostgreSQL database used in deployed environment (Heroku).

Flask Blueprint used to separate main app views from the API.

Jinja2 HTML template engine for Python.
App features:

- Log in / Log out
- Change account (PS4, Xbox, PC)
- Character select
- Character and account information
- Link to related blog posts
- Inventory management: Equip item, Send to vault
App features:

Select character
Click item to send to that character
User is redirected to that character
App Features:

Supports Destiny 2 (Released Sept 6th).

Custom OAuth 2.0 authentication flow.
  Handles multiple accounts.

Other functions:
  View vendor data (Bungie have not yet enabled for D2).
  View clan data.
  Compare active missions between clan members (Bungie have not yet enabled for D2).

API and debug:
  API to sync Manifest version with Bungie servers (Disabled due to Heroku charges).
  View list of usernames, sorted by last seen.
  Error reports emailed directly to my Gmail.
  Messages sent to private Slack account when specific errors are hit.
Presentation Scope:

What I’m planning on showing:
  - Viewing character vault.
  - How I created the character vault view.
  - Transferring an item.

For information on the following - Check out my blog:
  - Creating the Flask setup.
  - API quick start.
  - Downloading / formatting the Manifest.
  - Background jobs with Celery and Redis.
API Registration / storing auth token in Session:

Create an account and register as a developer:

https://www.bungie.net/en/User/API

This will give you your unique X-API-Key.

We need to send this API key in the HTTP header of the request.
Authorised requests will need you to authorise the user via the OAuth flow.

Python Requests library used to store Session data:

```python
oauth_session = requests.Session()
oauth_session.headers['X-API-Key'] = 'abcd12345'
oauth_session.headers['Authorization'] = 'Bearer ' + str(oauth_token)
```
Sending a HTTP GET request to view vault:

Sending the GET request:

```python
D2_BASE_URL = "https://www.bungie.net/Platform/Destiny2/"
req_string = D2_BASE_URL + str(membershipType) + "/Profile/" + str(membershipId) + "?components=100,102"
res = session.get(req_string)
```

Which allows us to do some cool stuff:

```python
print res.url
print res.status_code
print res.headers
print res.text
print res.json()
```

We can store session data in the headers and cookies too:

```python
print session.headers
print session.cookies
```
Send a request to Bungie.net to get the user's Vault details:
Parse the Vault response and return only the required data:
Populating the vault route with our data:

```python
# main.route('/vault', methods=['GET', 'POST'])
@login_required
def vault():
    user = g.user

    # Get profile information:
    GetProfile_res = GetProfile(south_session, destinyMembershipId, membershipType)
    weaponlist = parseDVault(south_session, GetProfile_res, all_data)

    return render_template('vault.html',
        categories = categories,
        weaponlist = weaponlist,
        character = user.username,
        charid = charid,
        lightlevel = GetProfile_res.json()['Response']['characters'][0]['light'],
        emblemImage = GetProfile_res.json()['Response']['characters'][0]['charId']['emblemImage'],
        backgroundImage = GetProfile_res.json()['Response']['characters'][0]['data'][charid]['emblemBackgroundPath'],
        character_details = character_details,
        form = form)
```
Categorising the data:

**Item categories:**

category = {
    1 : 'Kinetic Weapons',
    2 : 'Energy Weapons',
    3 : 'Power Weapons',
    4 : 'Ghost',
    5 : 'Helmet',
    6 : 'Gauntlets',
    7 : 'Chest Armor',
    8 : 'Leg Armor',
    9 : 'Class Armor',
    :
    :
}

**Dictionary response:**

weaponList = {
    'itemName': u'MIDA Multi-Tool',
    'itemTypeName': u'Scout Rifle',
    'bucket': u'Kinetic Weapons',
    'tierTypeName': u'Exotic',
    'itemReferenceHash': u'6917529035440581369',
    'itemLightLevel': '',
    'stackSize': 1,
    'equipped': False,
    'icon': u'https://www.bungie.net/common/destiny2_content/icons/077e9577fb39cb521b49048db236e39d.jpg',
    'itemHash': 1331482397,
    'quantity': 1
}
Populating categories:

Vault.html:

```html
<div class="inventory-container">

<!-- Loop through all item categories: -->
{% for item in category %}

<!-- Loop through list of dictionaries: -->
{% for dict_item in weaponList %}

<!-- if dict is in this category, create the HTML to display item: -->
{% if category[item] in dict_item['bucket'] %}

{% include 'itemBlock.html' %}

{% endif %}

{% endfor %}

{% endfor %}

</div>
```
Populating the HTML:

itemBlock.html:

```html
<div class="thumbnail">
  <img src="{{ dict_item['icon'] }}" title="{{ dict_item['itemName'] }}">
  <p>{{ dict_item['itemName'] }}: {{ dict_item['itemLightLevel'] }}</p>
  <p>{{ dict_item['tierTypeName'] }} {{ dict_item['itemTypeName'] }}</p>
</div>
```
Completed Vault view:

**Kinetic Weapons:**

- **Vigilance Wing:** Exotic Pulse Rifle
- **Haunted Earth:** Legendary Scout Rifle
- **SUROS Throwback:** Uncommon Auto Rifle
- **Nameless Midnight:** Legendary Scout Rifle
- **Sweet Business:** Exotic Auto Rifle
- **The Old Fashioned:** Legendary Hand Cannon
- **The Steady Hand:** Legendary Hand Cannon
- **The Guiding Sight:** Legendary Scout Rifle
- **Cydonia-AR1:** Uncommon Auto Rifle
- **Song of Justice VI:** Legendary Scout Rifle
- **Lincoln Green:** Legendary Pulse Rifle
- **Bad News:** Legendary Hand Cannon
- **Scathelocks:** Legendary Auto Rifle
- **Minimum Distance:** Legendary Sidearm
- **Traveler’s Chosen (Damaged):** Common Sidearm
- **The Forward Path:** Legendary Auto Rifle
- **Frontier Justice:** Legendary Scout Rifle
- **Does Not Compute:** Legendary Scout Rifle
The transferItem endpoint:

Create the POST data:

```python
transfer_url = "https://www.bungie.net/Platform/Destiny2/Actions/Items/TransferItem/

payload = {
    'itemReferenceHash': Unique reference number,
    'stackSize': '1',
    'transferToVault': True / False,
    'itemId': Generic item hash,
    'characterId': charId,
    'membershipType': 1 = Xbox, 2 = Playstation, 4 = PC
}

Send the POST request:

res = session.post(transfer_url, json=payload)
```
Transferring an item:

HTML:
```
<a href="transferItem/{{ charId }}/\{ dict_item['itemReferenceHash'] \}/\{ dict_item['itemHash'] \}/\{ False \}/.charInventory">
  <img src="\{ dict_item['icon'] \}" title="\{ dict_item['itemName'] \}" />
</a>
```

https://www.destinyvaultraider.com/transferItem/2305843009260647152/6917529038721127558/2014642399/False/.charInventory
The transferItem route:

```python
@main.route('/transferItem/<charId>/<itemInstanceId>/<itemReferenceHash>/<to_vault>/<view>/')
@login_required
def transferItem(charId, itemInstanceId, itemReferenceHash, to_vault, view):
    user = g.user
    payload = {
        'itemReferenceHash': itemReferenceHash,
        'stackSize': '1',
        'transferToVault': to_vault,
        'itemId': itemInstanceId,
        'characterId': charId,
        'membershipType': g.user.membershipType
    }
    transferItem_res = D2transferItem(payload, oauth_session)
    return redirect(url_for(view))
```
Completed character view:

Current Character: ChimpAhoy
Select Character: Warlock

Subclass:
- Downblade
- Skirmisher
- Voidwalker

Kinetic Weapons:
Equipped Item: Reckless
Carried Items:
- MEGA Multi Tool
- Relics of the Golden Age
- Nightblade
- Better Devils
- The Steady Hand
- Home Again
- Bayou Sting
- The Time-Warp Splat

Send to vault:
Video:

- See here for video: [https://vimeo.com/240543497](https://vimeo.com/240543497)
  - Currently in vault view.
  - Transferring Helmet to character,
  - Equip it, helmet changes.
  - Transferring Mida-Multi tool,
  - Equip it, gun on back changes.
  - Transfer a sword for PvP.
  - Equip it, No animation as Mida is currently equipped.
Deployment issues / challenges:

Multiple accounts / accounts activated only on Beta version of the game.
  Very difficult to simulate correctly in development.
  Email error reports and Slack messaging service really helped here.
  Bungie now return account information in numerical order.

Automating deployment of new Manifest files still tricky.
  Hobby plan + Celery worker + Redis database >€60/month.
  No easy way to push a JSON file to a Heroku repo.

By the time you spot an issue, users are very unlikely to return.
  Get it right first time!
Conclusions:

Core app works well - Flask is great:
   Needs some unique features in order to progress from tutorial into full product.

Lots of redundant API calls and refreshing of data:
   Can I cache data between item transfers?
   Can updating view be done on the front-end with AJAX?

Lots of front-end work needed to build features:
   Hover over an item to display screenshot, stats, additional transfer options.
   Create a loadout builder on the front-end.

Possibly move hosting service:
   Heroku is very easy to get started with but very expensive when adding features.
Questions?:

Get in touch:

Allyn Hunt
Blog: AllynH.com
Twitter: @Allyn_H_
Web application: DestinyVaultRaider.com
GitHub: https://github.com/AllynH