

WPL Report

Gift Registry WebApp

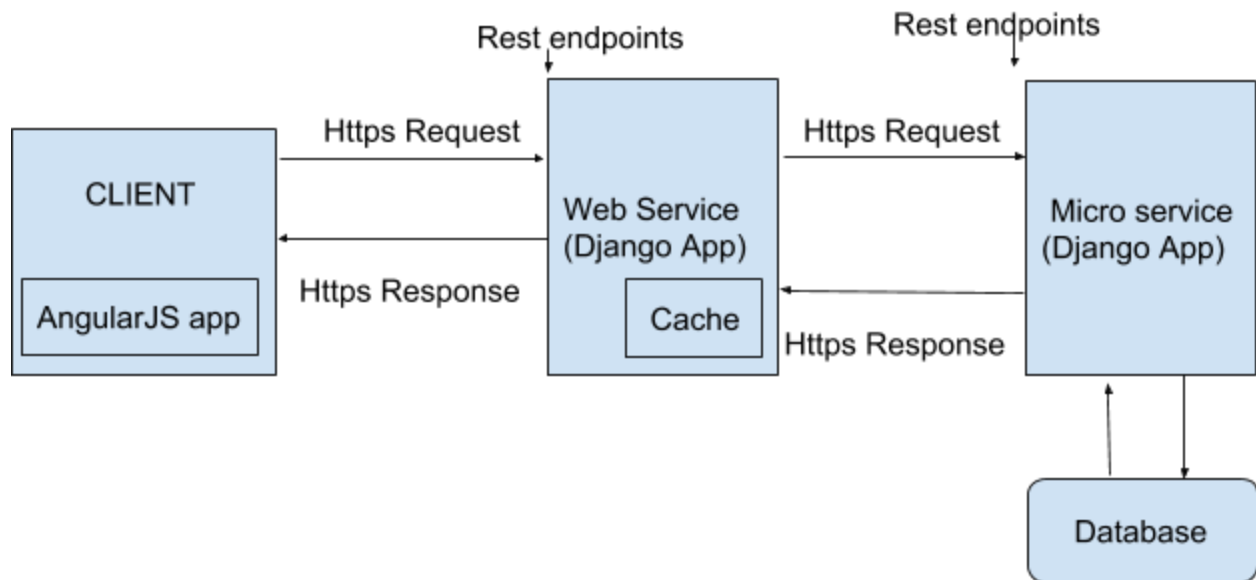
Madhuri Palagummi vxp171130

Sathya Pooja sxr176830

Shashidhar sxd173730

5th December, 2017

Architecture



Tech Stack Used

Frontend:

Considered developing the app using plain html, JS and CSS but decided that using a framework will save a lot more time. So we decided to use AngularJS. Using a CSS framework would give better styling to our website, so we used Materialized CSS framework as it has better components than Bootstrap.

1. Angular JS
2. Materialized CSS
3. HTML, Javascript, CSS

Backend:

Again Using a framework would help us save time as we need not write code for routing, Sqlqueries, authentication etc. So we considered using Php-Symfony and python-Django framework and finally decided to use Django because it is a simpler framework and perfect for light weight applns like ours. And we used Sqlite, not SQL because it is lighter and good for apps like ours where there is not a lot of data.

We used Django's inbuilt cache for caching data

1. Django Framework(Python)
2. Sqlite Database
3. Django's inbuilt cache

Servers:

We decide to use Nginx for serving. And since Nginx can't directly serve python apps, we used uWsgi (Web server gate interface) for serving Django Apps.

1. Nginx
2. uWSGI for Django App

Communication flow:

Client ---> nginx ---> uWsgi ----> Django Application



Functionalities Available

Admin


1. Login
2. Logout
3. View Items in Inventory
4. Add/Remove items from Inventory

User:

1. New user registration
2. Login
3. Logout
4. View own registries
5. Add new Registry
6. Make the registry public/share it with specific set of people
7. Add/Remove items from own registries
8. View shared registries
9. Assign/Unassign item in shared registries
10. View account information
11. Change password
12. Search/Sort for items in inventory
13. Reset password using forgot password functionality

Web Services:

1. Login Api - /login
Request Parameters: username, password
Response: Auth token if successful else error message.
Microservices: /createtoken api to create token
2. Logout Api - /logout
Request Parameters: UserId
Response: Success message
Microservices: /logout api
3. ChangePassword - /changepassword
Request Parameters: new_password
Response: Success message
Microservices: /userfromtoken api to get user details from token
/changepassword to change the password
4. New user Api - /newuser
Request Parameters: userDetails
Response: Success message
Microservices: /registeruser api to get user details from token
5. Create Registry - /createregistry
Request Parameters: new_password
Response: Success message
Microservices: /userfromtoken api to get user details from token
/createregistry to create new registry

- 
6. Get registries - /registries -- Returns all the registries of user
Request Parameters: new_password
Response: All registry details that user owns/shares
Checks for user_id in cache, if it does not find it will hit the /userfromtoken microservice api
Microservices: /userfromtoken api to get user details from token
/registries to create new registry
 7. Get user details - /userdetails
Request Parameters: None
Response: User details
Microservices: /userfromtoken api to get user details from token
 8. Assign item - /assignitem
Request Parameters: registry_item_id
Response: Success message
Microservices: /userfromtoken api to get user details from token
/assignitem to assign item to user
 9. Unassign item - /unassignitem
Request Parameters: registry_item_id
Response: Success message
Microservices: /userfromtoken api to get user details from token
/unassignitem to assign item to user
 10. Get registry details - /getregistry
Request Parameters: registry_id
Response: Registry details
Microservices: /userfromtoken api to get user details from token
/getregistry to get registry details

11. Add item to registry - /additemtoregistry

Request Parameters: item_id, registry_id

Response: Success message

Microservices: /userfromtoken api to get user details from token
/additemtoregistry to add item

12. Remove item from registry - /removeitemfromregistry

Request Parameters: item_id, registry_id

Response: Success message

Microservices: /userfromtoken api to get user details from token
/removeitemfromregistry to remove item

13. Fetch all items - /items

Request Parameters: None

Response: Details of all items in inventory

Microservices: /items api to get item details

14. Get a list of users - /getusers

Request Parameters: None

Response: Details of all users

Microservices: /getusers api to get item details

15. Forgot password - /forgotpassword

Request Parameters: email_id

Response: Success message

Microservices: /forgotpassword api to reset password

16. Add item to inventory - /additemtoinventory

Request Parameters: item_details

Response: Success message

Microservices: /userfromtoken api to get user details from token
/additemtoinventory to add item

-
17. Remove item to inventory - /removeitemtoinventory
Request Parameters: item_id
Response: Success message
Microservices: /userfromtoken api to get user details from token
/removeitemtoinventory to remove item
 18. Remove item from registry - /removeitemfromregistry
Request Parameters: item_id, registry_id
Response: Success message
Microservices: /userfromtoken api to get user details from token
/removeitemfromregistry to remove item

Authentication in the system:

Authentication for the user is token based and authentication between microservice and web service is based on shared secret key.

User authentication:

- When the user logs in for the first time with username and password, the microservice creates a token, stores it in DB against that user id and sends that token to the client. The client sends this token in the headers for every request so that the user is identified uniquely.

Web service - Microservice Authentication:

- Authentication between web service and microservice is based on a pre-shared secret token which web service sets in the header in every request to microservice.

Flow:

Initial Login:

- User signs in with username and password
- Web service receives the request and it forwards to microservice.
- Microservice validates the details, creates a random token, stores it DB against that user id.
- Microservice returns the token to web service which sends
To client
- The client sets this token in headers in every request

Other requests:

- Web service receives request from client
- Web service looks for user's auth-token in the headers.
- Web service requests microservice for user auth sending user's auth token in parameters. It sets the pre shared secret token in the headers.
- Microservice receives request
 - Checks for the shared key in the headers
 - Looks for user's auth-token in parameters , checks it in the user table and returns user details
- Web service receives user details from web service. It sends the original request (from client) adding user_id in params.
- Microservice receives the request, check for user_id in params and checks if the user is authorized to perform the operation

Cache:

We used Django's inbuilt cache to store user id, token pairs.
This is in Web service app.



Tech challenges encountered

1. Setting up Nginx for angularjs
Had to figure out how nginx.conf works and route the urls properly
2. Setting up Nginx for django application
Since Nginx directly cant communicate with django app, we had to use uWSGI to communicate with django app.In the nginx config we had to set upstream as the django app's server and route the proper urls to uwsgi
3. Flow of authentication
We had to research a lot and figure out a flow for authentication.This includes user authentication and web services authentication.
4. Understanding Auth in Django
Since django already provides a framework for auth, we had little trouble figuring out the token authentication.
5. CORS issue
We had to fix the CORS issue by allowing all hosts in django app.Took a while to figure this out.
6. Setting up the whole angular app
Took a while to figure out how to write controllers, services and templates.And we had to import all the files properly in index file.And the routing had to be done properly.So all this took considerable effort
7. Setting up django app
We had to figure out how routing works, apis work and had to install django-rest lib for the rest apis.
8. Db Design
We had to design db properly so that it is in normalized form.

—

—