# Iteration 3 Presentation

ZICZAC

Team 3 - ZicZac

April 28, 2021

# Contents

# Project Overview

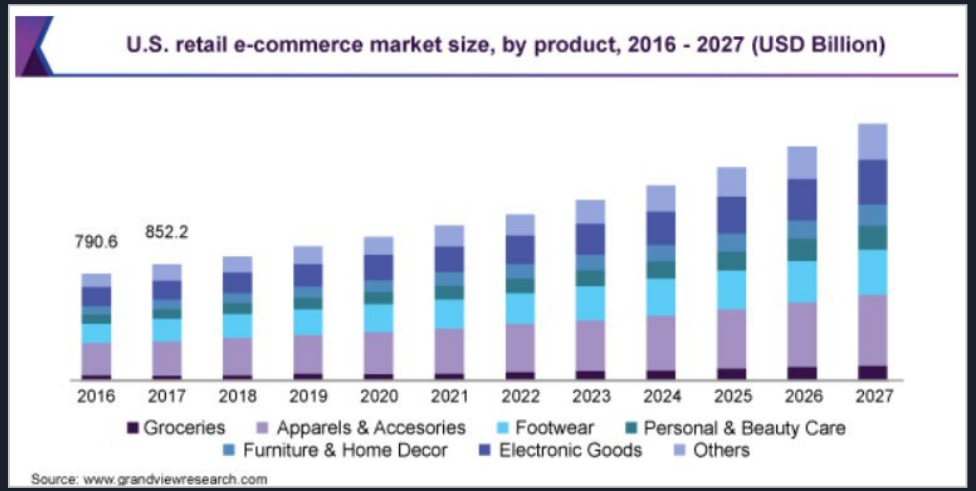
ZICZAC

## Motivation

1 . To encourage trade pre-owned items among people in local area for used recycling and reducing waste.

2. To become a community platform for nearby neighbors.

# Online Marketplace



# Differentiation

## The key features of ZICZAC

1. C2C (Customer to Customer)
2. Direct transaction
3. Communicate with private chat system

ZICZAC

Online Marketplace

Community Platform

# Project Requirements

## Essential Features

Sign up

Log in

Search

Item categories

Post items for sale

Sort (by price)

Private chat system

## Desirable / Optional Features

Account deletion

Account recovery

Review seller

Regulation by reviews

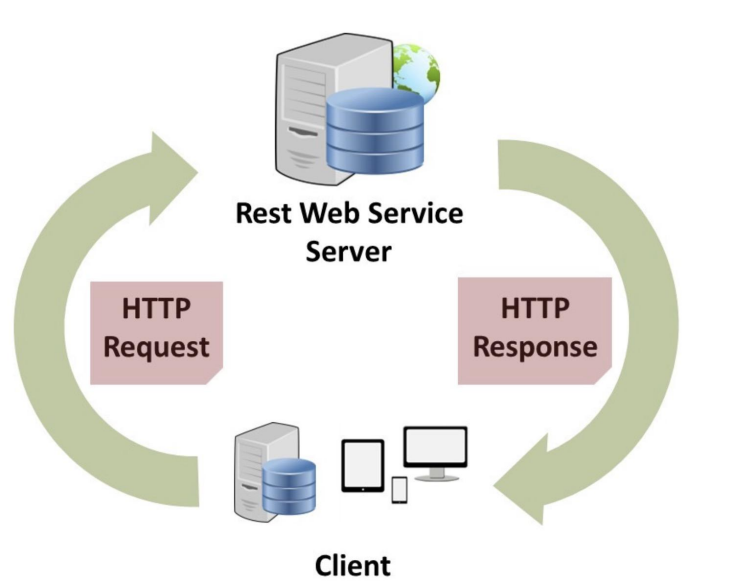
Customer service

# Software architecture

RESTful Web Service is an architectural style, where the data or the structural components of a system is described  in the form of URI ( Uniform Resource Identifier) and the behaviors are described in terms of methods. The resources can be manipulated using CRUD (Create, Read, Update and Delete) operations. The communication protocol for REST is HTTP since it suits the architecture requirement of being a stateless communication across the Client and Server.

In our projects we have following sample APIs:

1. POST /api/sort - Sorts items by the given parameter such as recent or price
2. POST /api/category - Selects products pertinent to selected category such as furniture, books, etc.
3. POST /api/search - Searches items from the database based on the given string

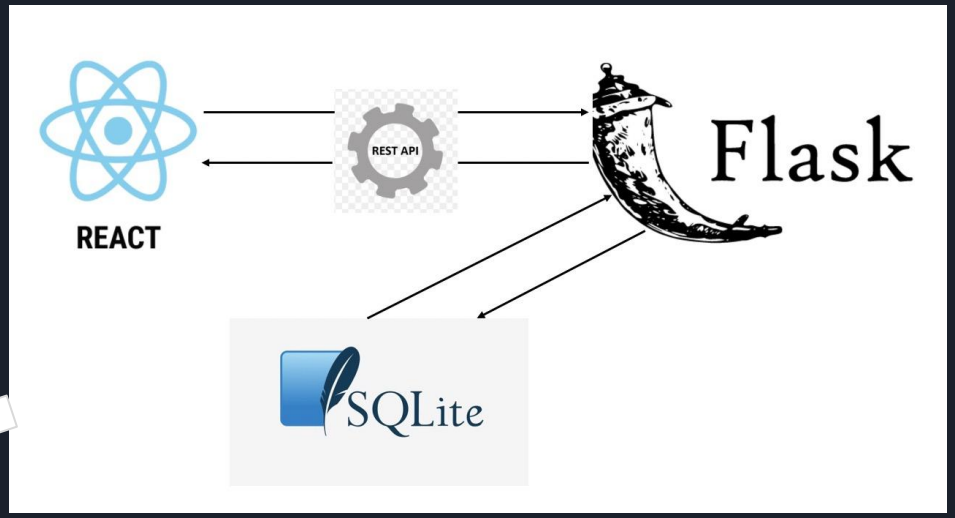


# Technical Stack

Front End - Reactjs

BackEnd -  Python Flask

Persistent data - SQLite




Flask static files

https://ziczac3.herokuapp.com

# Front End

Our front end code is in team3_webapp_react/src folder.

In our code we used recently introduced React Hooks such us useEffect, useState. Before we would have been using classes to identify state of the functional component but now we can just use React Hooks.

In src/components folder we identified all the UI components such as header, footer, etc and reused them in different pages.

For page routings we used BrowserRouter function of react-router-dom.

For communication with backend as we mentioned before we used API calls.

We used various Reactjs libraries such as react-bootstrap, semantic-ui-react, react-dropzone, etc

that gave our website desired functionality and styling without heavy-duty coding.

# Back End

Our backend code is located in the /dev directory.

Most of the functionality is in the app.py file.

Request from the front end comes on a specific endpoint.

Communication between front end and database happens via flask, reactJS doesn't talk to database directly.

For the list of all the classes used in app.py we used swagger library that allows us to get the list on /swagger and /swagger-ui

app.py  imports several other files such as passwordManager.py and passwordHash.py that give our

password systems desired functionally and security

# Persistent Data

SQLite was chosen to store all the persistent data because it is lightweight and marks all the checkboxes required for this project implementation.

We implemented following tables:

account: this table stores user data such as username, email, encrypted password.

Inventory is used to store information pertinent to a product such as name, price, images in the format of “blob”, etc.

We have 2 tables (inbox, outbox) connected to a specific user that hold all the private messaging information between the user and other members of the site that messaged this particular user.

# Configuration setup

Version control - Github

- Main branch - final branch with latest updates
- Development branch - intermediate branch that gets merged into main branch
- Personal branches for various work
- Integrated with slack so everyone is aware what gets posted to main branch
- Separate branch for heroku deployment
- .gitignore file that will prevent us from pushing files such as node modules, static files, etc.

Main means of communication - Slack

Weekly meetings and updates - Zoom

# Iteration 3 Updates

- Private message system: from product detail page and messages page
- Contact us
- Debugged default sort
- Improved ease of navigation with dropdown menus
- Refactoring
- Database restructuring

# App Refactoring: Before

```
214 @app.route('/api/post_product', methods=['POST'])
215 def post_product():
216     req = flask.request.get_json(force=True)
217     title = req.get('title')
218     price = req.get('price')
219     description = req.get('description')
220     category = req.get('category')
221     date_added = arrow.now().format('YYYY-MM-DD')
222     photo_filepath = f"/img/{title}.jpg"
223     seller = req.get('seller')
224     state = req.get('state')
225     image = req.get('image')
226     with app.app_context():
227         with sqlite3.connect(database) as con:
228             cur = con.cursor()
229             cur.execute(f"insert into inventory (title, price, description, category, c
230             con.commit()
231             return {'message' : 'success'}, 200
232
233 @app.route('/api/send_message', methods=['POST'])
234 def send_message():
235     req = flask.request.get_json(force=True)
236     seller = req.get('seller')
237     buyer = req.get('buyer')
238     message = req.get('message')
239     date = arrow.now().format('YYYY-MM-DD')
240     with app.app_context():
241         with sqlite3.connect(database) as con:
242             cur = con.cursor()
243             cur.execute(f"INSERT INTO {seller}_inbox (sender, body, date) VALUES (?, ?,
244             cur.execute(f"INSERT INTO {buyer}_outbox (recipient, body, date) VALUES (?,
245             con.commit()
246             return {'message' : 'success'}, 200
```

# App Refactoring: After

```
228 class PostProduct(MethodResource, Resource):
229     def post(self):
230         req = flask.request.get_json(force=True)
231         title = req.get('title')
232         params = {
233             'title' : title,
234             'price' : req.get('price'),
235             'description' : req.get('description'),
236             'category' : req.get('category'),
237             'date_added' : arrow.now().format('YYYY-MM-DD'),
238             'photo_filepath' : f"/img/{title}.jpg",
239             'seller' : req.get('seller'),
240             'state' : req.get('state'),
241             'image' : req.get('image')
242         }
243         queries = [f"insert into inventory (title, price, description, category, date_added, p
244         return query_db(queries = queries, method = 'post_product', params = params)
245
246 api.add_resource(PostProduct, '/api/post_product')
247 docs.register(PostProduct)
248
249
250 class SendMessage(MethodResource, Resource):
251     def post(self):
252         req = flask.request.get_json(force=True)
253         params = {
254             'seller' : req.get('seller'),
255             'buyer' : req.get('buyer'),
256             'message' : req.get('message'),
257             'date' : arrow.now().format('YYYY-MM-DD HH:MM:SS')
258         }
259         queries = [
260             f"INSERT INTO {params['seller']}_inbox (sender, body, date) VALUES ('{params['buye
261             f"INSERT INTO {params['buyer']}_outbox ([to], body, date) VALUES ('{params['selle
262         ]
263         return query_db(queries = queries, method = 'send_message', params = params)
264
265 api.add_resource(SendMessage, '/api/send_message')
```

# Testing

- Framework
  Selenium for automated UI testing

- Type
  Automated UI testing (black box testing) and Python unit testing (white box testing)

# Manual testing

**UI Test (Manual)**

Test name: Rating Seller date: 4/20/2021

- New or old: New
- Test items: (what do you test ) Testing if the user can rate the seller.
- Test priority (high/medium/low): Low
- Dependencies (to other test case/requirement if any): None
- Preconditions: (if any) None
- Input data: Code to run the react app in terminal.
- Test steps:
  - Login as a user
  - Select the number of stars to rate the seller
- Postconditions:None
- Expected output: Seller has a rating
- Actual output: Rating seller feature was not funtioning
- Pass or Fail: Fail

# Refactoring

Before:

```
Run Test | Debug Test
def test_signup(self):

# checking to see if users can signup

    email = self.driver.find_element_by_xpath(
        '//*[@id="root"]/div/div/form/div[1]/input')
    email.send_keys("testingproject@gmail.com")
    username = self.driver.find_element_by_xpath(
        '//*[@id="root"]/div/div/form/div[2]/input')
    username.send_keys(''.join(random.choice('0123456789ABCDEF') for i in range(16)))
    password = self.driver.find_element_by_xpath(
        '//*[@id="root"]/div/div/form/div[3]/input')
    password.send_keys("Beauty1234@")
    self.driver.find_element_by_xpath('//*[@id="root"]/div/div/form/button').click()
    time.sleep(5)
    exps = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div[2]/div/div/h1').text
    print(exps)
    self.assertEqual(exps, "PRODUCTS")
    time.sleep(1)
```

# Refactoring

## Creating Page Object

```
class SignupPage:
    def __init__(self, driver):  # page object
        self.email = driver.find_element_by_xpath(
            '//*[@id="root"]/div/div/form/div[1]/input')
        self.username = driver.find_element_by_xpath(
            '//*[@id="root"]/div/div/form/div[2]/input')
        self.password = driver.find_element_by_xpath(
            '//*[@id="root"]/div/div/form/div[3]/input')
        self.submitbutton = driver.find_element_by_xpath(
            '//*[@id="root"]/div/div/form/button')
```

# Refactoring

After:

```
Run Test | Debug Test
def test_signup(self):

# checking to see if users can signup
    signuppage = SignupPage(self.driver)

    signuppage.email.send_keys("testingproject@gmail.com")

    signuppage.username.send_keys(''.join(random.choice('0123456789ABCDEF') for i in range(16)))

    signuppage.password.send_keys("Beauty1234@")
    time.sleep(1)
    signuppage.submitbutton.click()
    time.sleep(2)
    exps = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/form/div').text
    print(exps)
    self.assertEqual(exps, "ZicZac")
```

# Refactoring

Result:

RefactoringLab3 · BUMETCS673/BUMETCS673S21T3@496fdbb (github.com)

# Testing

## Lines deleted after refactoring

Showing 3 changed files with 80 additions and 72 deletions.

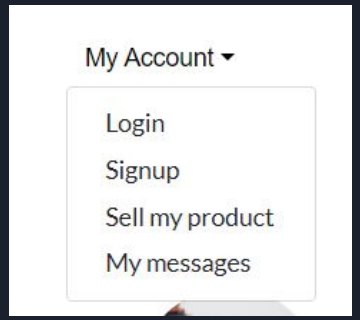
```
24  25      # checking to see if users can signup
    26 +        signuppage = SignupPage(self.driver)
    27 +
    28 +        signuppage.email.send_keys("testingproject@gmail.com")
    29 +
    30 +        signuppage.username.send_keys(
    31 +            ''.join(random.choice('0123456789ABCDEF') for i in range(16)))
25  32
26      -       email = self.driver.find_element_by_xpath(
27      -           '//*[@id="root"]/div/div/form/div[1]/input')
28      -       email.send_keys("testingproject@gmail.com")
29      -       username = self.driver.find_element_by_xpath(
30      -           '//*[@id="root"]/div/div/form/div[2]/input')
31      -       username.send_keys(''.join(random.choice('0123456789ABCDEF') for i in range(16)))
32      -       password = self.driver.find_element_by_xpath(
33      -           '//*[@id="root"]/div/div/form/div[3]/input')
34      -       password.send_keys("Beauty1234@")
35      -       self.driver.find_element_by_xpath('//*[@id="root"]/div/div/form/button').click()
36      -       time.sleep(5)
    33 +        signuppage.password.send_keys("Beauty1234@")
    34 +        time.sleep(1)
    35 +        signuppage.submitbutton.click()
    36 +        time.sleep(2)
37  37          exps = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div[2]/div/div/h1').tex
38  38          print(exps)
```

# Refactoring Before:

```
    def test_loginbutton(self):
            #checking to see if login button works
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/button').click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/div/a[1]').click()
        expr2=self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/h3')
.text
        print(expr2)
        self.assertEqual(expr2, 'Sign In')
        time.sleep(1)

    def test_signup(self):
        #checking signinbutton
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/button').click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/div/a[2]').click()
        expr4 = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/fo
rm/h3').text
        print(expr4)
        self.assertEqual(expr4, 'Sign Up')
        time.sleep(1)


    def test_sellyourproduct(self):
        #checking sell my products button
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/button').click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/div/a[3]').click()
        exp = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/div/
div[2]/h2').text
```

```
        self.assertEqual(exp, 'Sell Your Product')
        time.sleep(1)

    def test_mymessages(self):
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/button').click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div
/div/a[4]').click
        exp3 = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/div
/div[1]/div/h4').text
        print(exp3)
        self.assertEqual(exp3, 'View your conversations')
        time.sleep(1)
```

# Refactoring
## Creating Account Page object




```python
class AccountPage:
    def __init__(self, driver):  # page object
        self.driver = driver
        self.dropdown = driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div/button')
    def clicksignup(self):

        self.dropdown.click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div/div/a[2]').click()
    def clicklogin(self):

        self.dropdown.click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div/div/a[1]').click()
    def clicksellmyproduct(self):

        self.dropdown.click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div/div/a[3]').click()

    def clickmymessages(self):

        self.dropdown.click()
        self.driver.find_element_by_xpath('//*[@id="root"]/div/div/nav/div[2]/div/div/a[4]').click()
```

# Refactoring
## After:

```
Run Test | Debug Test
def test_loginbutton(self):
        #checking to see if login button works
    accountpage=AccountPage(self.driver)
    accountpage.clicklogin()
    time.sleep(5)
    expr2=self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/div/div[1]/label').text
    print(expr2)
    self.assertEqual(expr2, 'Username')


Run Test | Debug Test
def test_signup(self):
    #checking signinbutton
    accountpage=AccountPage(self.driver)
    accountpage.clicksignup()
    time.sleep(5)
    expr4 = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/form/div[1]/label').text
    print(expr4)
    self.assertEqual(expr4, 'Email address')


Run Test | Debug Test
def test_sellyourproduct(self):
    #checking sell my products button
    accountpage=AccountPage(self.driver)
    accountpage.clicksellmyproduct()
    time.sleep(5)
    exp = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/div/div[2]/h2').text
    print(exp)
    self.assertEqual(exp, 'Sell Your Product')


Run Test | Debug Test
def test_mymessages(self):
    accountpage=AccountPage(self.driver)
    accountpage.clickmymessages()
    time.sleep(5)
    exp3 = self.driver.find_element_by_xpath('//*[@id="root"]/div/div/div/div/div[1]/div/h4').text
    print(exp3)
    self.assertEqual(exp3, 'View your conversations')
```

# Testing

## Updates since iteration 2

- In iteration 2, sort products by under featured products in home page was not sorting properly.


bug no longer exists and test result is a pass.

- Private chat system has been implemented and the result of testing is a pass (manual test).
- Contact us button at the footer became functional. The button works perfectly fine but requirements to send a message have not been implemented. Therefore, the test fails when the user enters invalid data in contact us form.
- Each category still displays only four items. The test fails, when the user posts more than four items in each category.
- Items posted do not show up in recently added items. New bug after iteration 2
- More testing files have been refactored since iteration 2. The testing files look more readable and concise

# Automated UI Testing

TEST EXPLORER

- accountpageloads
  - test_loginbutton
  - test_mymessages
  - test_sellyourproduct
  - test_signup
- contactusfunctionality
  - test_blankemail
  - test_blankmessage
  - test_contactusloads
  - test_invalidemail
- homepagetest
  - test_clothingbutton
  - test_contactus
  - test_electronicsbutton
  - test_entertaintmentbutton
  - test_furniturebutton

TEST EXPLORER

- test_furniturebutton
- test_gardenbutton
- test_homebutton
- test_Homepageloads
- test_homesbutton
- test_petbutton
- test_schoolbutton
- test_searching
- test_signupbutton
- test_sorting
- test_sportbutton
- test_vechilesbutton
- test_viewrecentlyadded
- iteminrecentlyadded
  - test_fillingformtosell
- loginpagefunctionality

- loginpagefunctionality
  - test_loginpage
  - test_loginpagefunction
  - test_loginpagemessage
- sellingproduct test_itempostedincate...
  - test_sellingproduct
- sellingproduct test_sellyourproduct
  - test_sellingproduct
- signuppagefunctionality
  - test_allfieldsempty
  - test_blankemail
  - test_emptyusername
  - test_existinguser
  - test_passwordfieldempty
  - test_passwordrequirements
  - test_signup

- testingchatsystem
  - test_loginasbuyer

# Testing

- Number of Test cases
  44 test cases. 14 more test cases since iteration 2
- Functionality coverage
  100%. Every single feature has been tasted. 10% higher than iteration 2
  Functionality of contact us button, Functionality of chat button, forgot password in login page, seller rating, Like button for seller have been included.

- Defect rate
  Including all manual and automated tests, 20% defect rate.

# Security Review

- Authentication
  - The positive identification of the person or system seeking access to secured information or system. Password, Kerberos, token, and biometric are forms of authentication. (Greene).

    Source: Sari Stern Greene. (2014). *Security Program and Policies: Principles and Practices*.

- Authentication Mechanisms
  - Knowledge: Something a person knows (e.g. password)
  - Possession: Something a person has (e.g. cryptographic token or smart card)
  - Inherence: Something a person can produce (e.g. fingerprint)
- Single-factor Authentication vs. Multi-factor authentication
- Real world Example: shopping (bank card and PIN)

# Security Review

- Password
  - Standard
  - Cryptography
- Plan: Session Tokens and Automatically Logout
- Password - the most commonly used single-factor network authentication method
- Connection to the future

# Conclusion

- ZicZac: Online trading marketplace
- Build:
  - ReactJS
  - Python + Flask
  - SQLite
- Major features:
  - Intuitive product posting
  - Platform for communication between users
  - Search engine
  - Filtering: categorical, chronological, and price
  - Account security

# Takeaways - Eli

- This project is my first experience with:
  - Web application
  - Code collaboration
  - React or Javascript
  - Management

- Given lack of experience, important to get hands dirty with implementation early on. Otherwise hard to know what will be possible for requirements
- Next web application: familiarize self with stack beforehand. This way all the learning won't take away from important early stage planning

# Takeaways - Dinara

- Learned reactjs, flask and sqlite
- Learned heroku deployment
- Learned API
- Learned team communication, better Github practices
- Could not implement some features but that allowed me to learn some aspects of development and deployment

# Takeaways-Pelin

-Learned about automated testing and Selenium framework

-Developed interest in QA

-Time management is the key

-Learned about Java react and sqlite.

-General idea about how backend and frontend works

Challenges:

- Do not have so much coding experience.
- Never built an application before

Overall it was a great learning experience

# Takeaways - Chenghao

- What I learn from the project:
  - From the group
  - Security
  - Other fields

# Takeaways - Jay

- Learned new frameworks ReactJS, Flask
- Learned basics of chat system
- First time to participate on building a web application
- Experience team collaboration development


Thank you