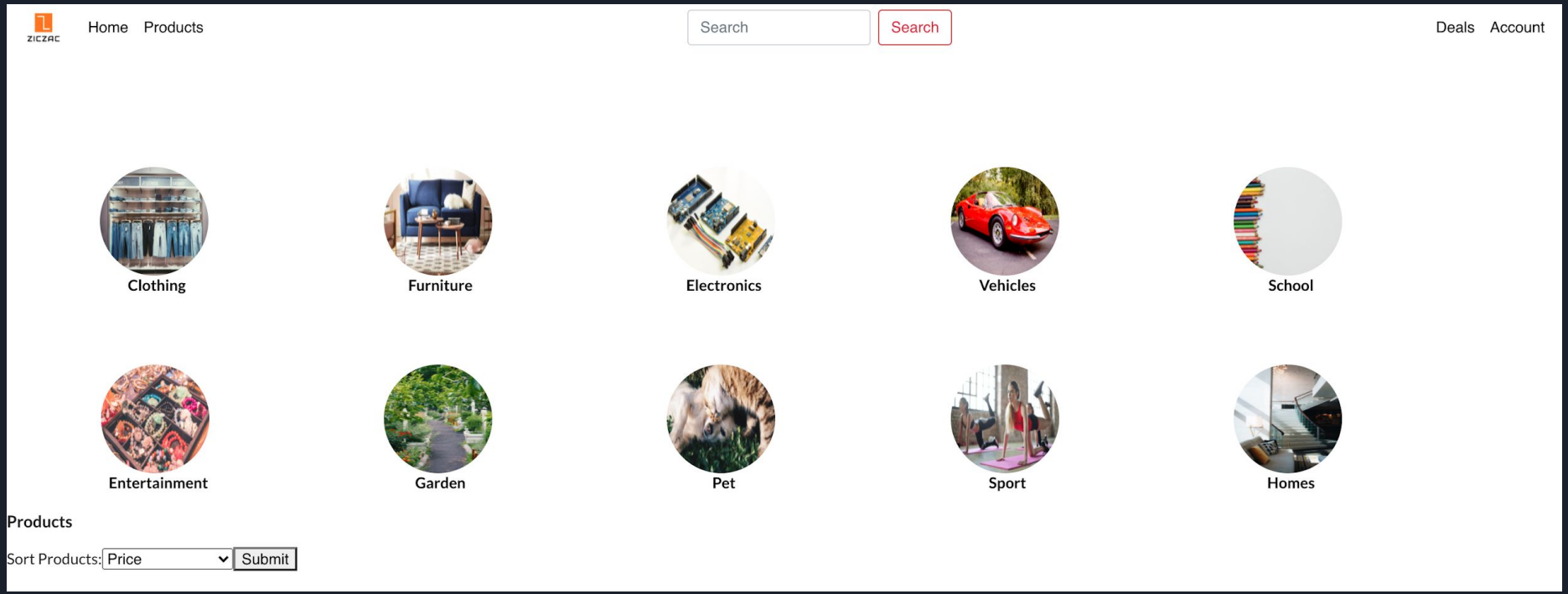
Iteration 2 Presentation

Team 3 - ZicZac

Demo


ZICZAC
Home
Products
Search
Search
Deals
Account
Clothing
Furniture
Electronics
Vehicles
School
Entertainment
Garden
Pet
Sport
Homes
Products
Sort Products:
Price
Submit

Chat System

Review (Iteration 1) :

- Started to implement the chat system to be used in web application.
- Following functional requirements were implemented :

1. Join the chat system
2. Type and send the messages to the others.
3. Receive the messages.
4. Check the others username.
5. Leave the chat system.

Chat System

Iteration 2 :

1. Integrating the web application and the chat system.
2. Researching and implementing the private chat system.

Chat System

Integrating the web application and the chat system

Integration Steps :

1. Combine the codes
2. Update the dependencies.
3. Execute both servers at the same time
    a. concurrently, express are used as dependencies.
    b. npm start = only web server
    c. npm run dev = both web server and chat server
4. UI : ‘Chat’ button

Iteration 2 Result : The chat system can be used in the web application by clicking the ‘Chat’ button.

Chat System

Researching and Implementing the private chat system.

What we need for the private chat system?

- socket.IO
- Database that contains user info (username, password, ect..)

How it works?

- Users send the direct messages to the selected users in database.

BEFORE

```
@app.route('/api/login', methods=['POST'])
def login():
    req = flask.request.get_json(force=True)
    username = req.get('username')
    password = req.get('password')

    with app.app_context():
        with sqlite3.connect(database) as con:
            cur = con.cursor()
            cur.execute("select * from accounts")
            accounts = cur.fetchall()
            for a in accounts:
                if a[0] == username:
                    message = 'Login accepted.' if a[1] == password else 'Incorrect password'
                    return {'message': message}, 200
            return {'message': 'Invalid username.'}, 200
```

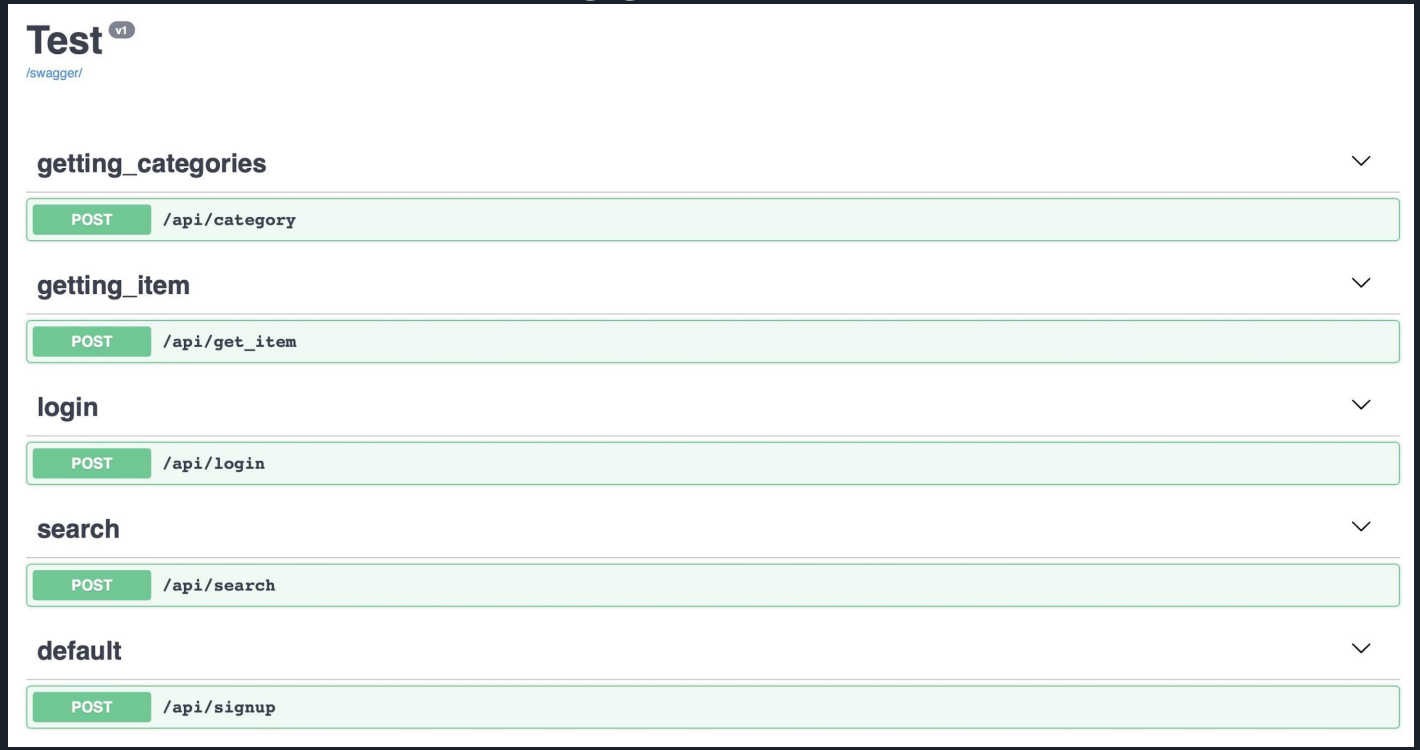
AFTER

```
class Login(MethodResource, Resource):
    @doc(description='login', tags=['login'])
    @marshal_with(ResponseSchema)  # marshalling
    def post(self):
        req = flask.request.get_json(force=True)
        username = req.get('username')
        password = req.get('password')
        ph = passwordHash()

        with app.app_context():
            with sqlite3.connect(database) as con:
                cur = con.cursor()
                cur.execute("select * from accounts")
                accounts = cur.fetchall()
                for a in accounts:
                    if a[0] == username:
                        message = 'Login accepted.' if ph.check_password(password, a[2], a[3]) else 'Incorrect password'
                        return {'message': message}, 200
                return {'message': 'Invalid username.'}, 200

api.add_resource(Login, '/api/login')
docs.register(Login)
```

SWAGGER API
localhost:5000/swagger-ui/



SWAGGER API
localhost:5000/swagger/


```
{
  "definitions": {
    "Response": {
      "properties": {
        "message": {
          "type": "string"
        }
      },
      "type": "object"
    }
  },
  "info": {
    "title": "Test",
    "version": "v1"
  },
  "paths": {
    "/api/category": {
      "post": {
        "description": "getting categories",
        "parameters": [],
        "responses": {},
        "tags": [
          "getting_categories"
        ]
      }
    },
    "/api/get_item": {
      "post": {
        "description": "get item",
        "parameters": [],
        "responses": {
          "default": {
            "description": "",
            "schema": {
              "$ref": "#/definitions/Response"
            }
          }
        },
        "tags": [
          "getting_item"
        ]
      }
    },
```

HEROKU DEPLOYMENT

Heroku git URL

`https://git.heroku.com/ziczac3.git`

https://ziczac3.herokuapp.com

currently:
*npm run build
*using dedicated branch so development doesn't break it

in the future:
*get rid of static files and render website dynamically
*continuous deployment (automation of deployment)
*including different metrics to evaluate website performance
*email integration

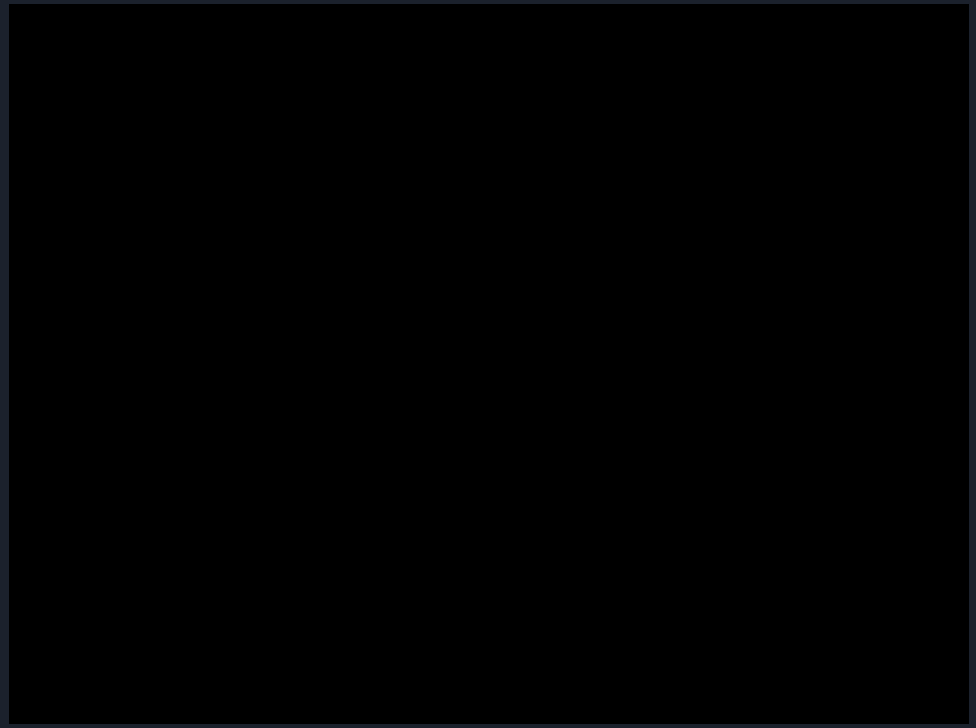
Testing

Black Box Testing(External Software Testing): High level testing that focuses on behaviour of the software. It involves testing from end-user perspective. Can be done without knowledge of internal structure of the application. Mostly done by software testers. Examples: System testing, acceptance testing and security testing.

White Box Testing(Structural Testing): Testing technique which checks the internal functioning of the system. Testing is focused on code structure, branches, paths and conditions. It is considered as low level testing. Knowledge of internal structure of the application is required. Mostly done by software developers. Examples: Unit testing, Integration testing

Automated UI Testing Demo

Selenium



Testing

Goals for next iteration

- Improving recently added sorting functionality in homepage
- Increasing item capacity of each category
- Functional contact us button at the footer
- Functional chat box button
- Adding user profile page
- Functional rating feature on seller’s profile
- Functional like and add to cart buttons on item page
- Fixing the bugs

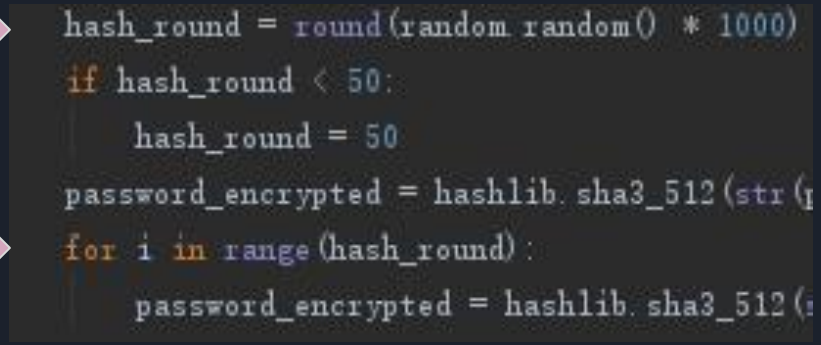
Security

- Previous characters for encryption: SHA-3 and salt
- First update: number of iterations

Security

Random number

For loop



Security

- PBKDF2 (Password-Based Key Derivation Function)
  - key derivation functions with a sliding computational cost, used to reduce vulnerabilities to brute-force attacks.
  - pseudorandom function + salt
  - Weakness - against GPU attacks

Security

```
Enter your password for sign up: 'Aa123456789'
Enter your password for login in: 'Aa123456789'
Hash round: 791
Salt: 44f1c2051ee342dda62f525e1dc19715
Encrypted password: $pbkdf2-sha512$791$NDRmMWMyMDUxZWUzNDJkZGE2MmY1MjVlMWRjMTk3MTU$URsU1DygsPgGfRUJKKuF0vAF7xUnvDZF
Checking password: True

Process finished with exit code 0
```