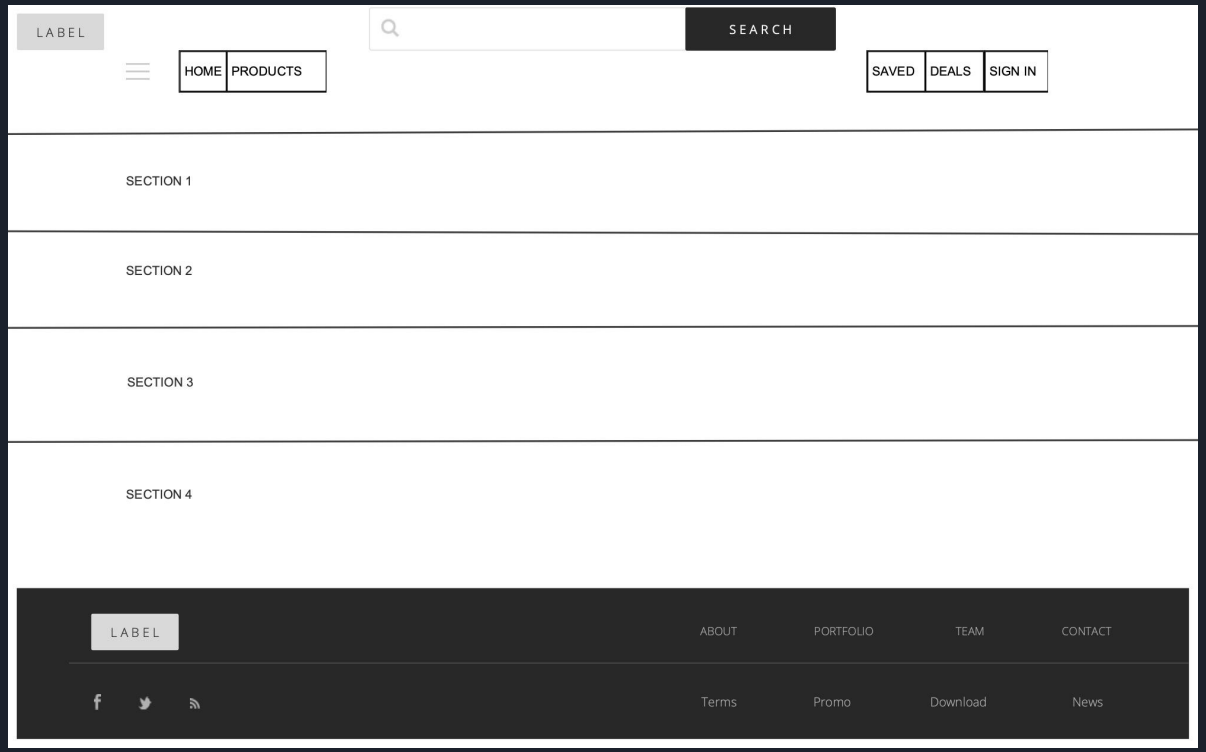
Iteration 1 Presentation

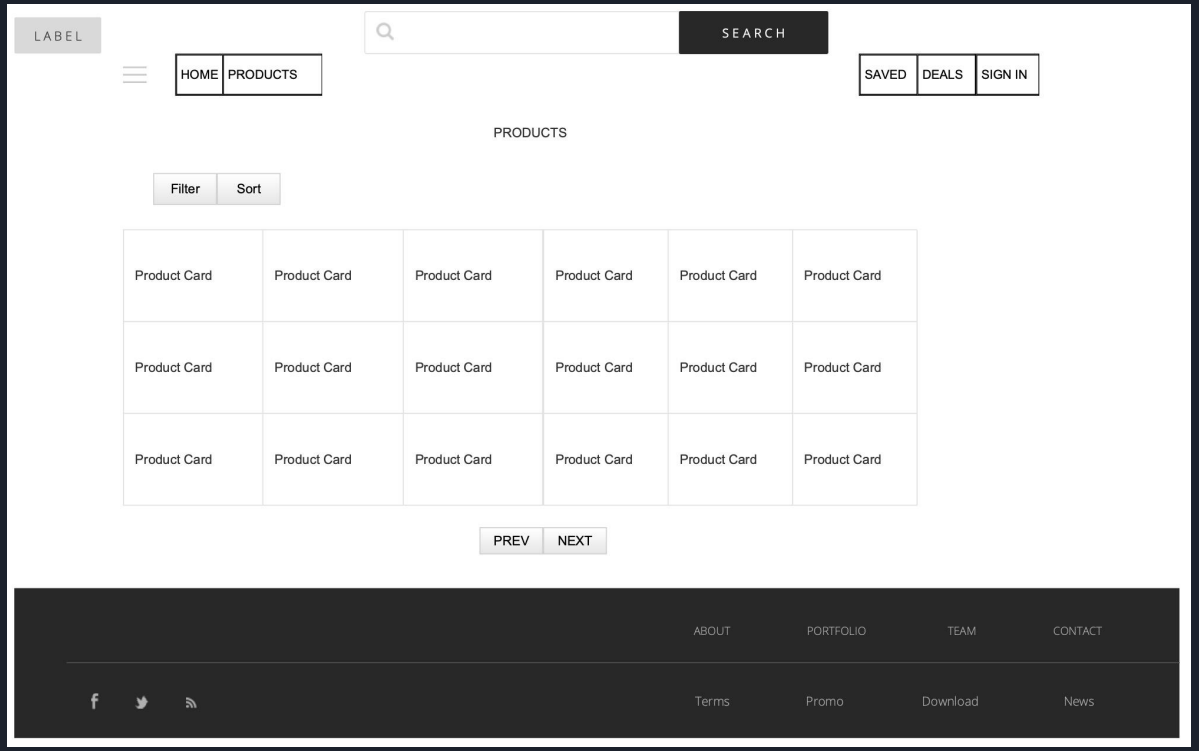
ZICZAC

Team 3

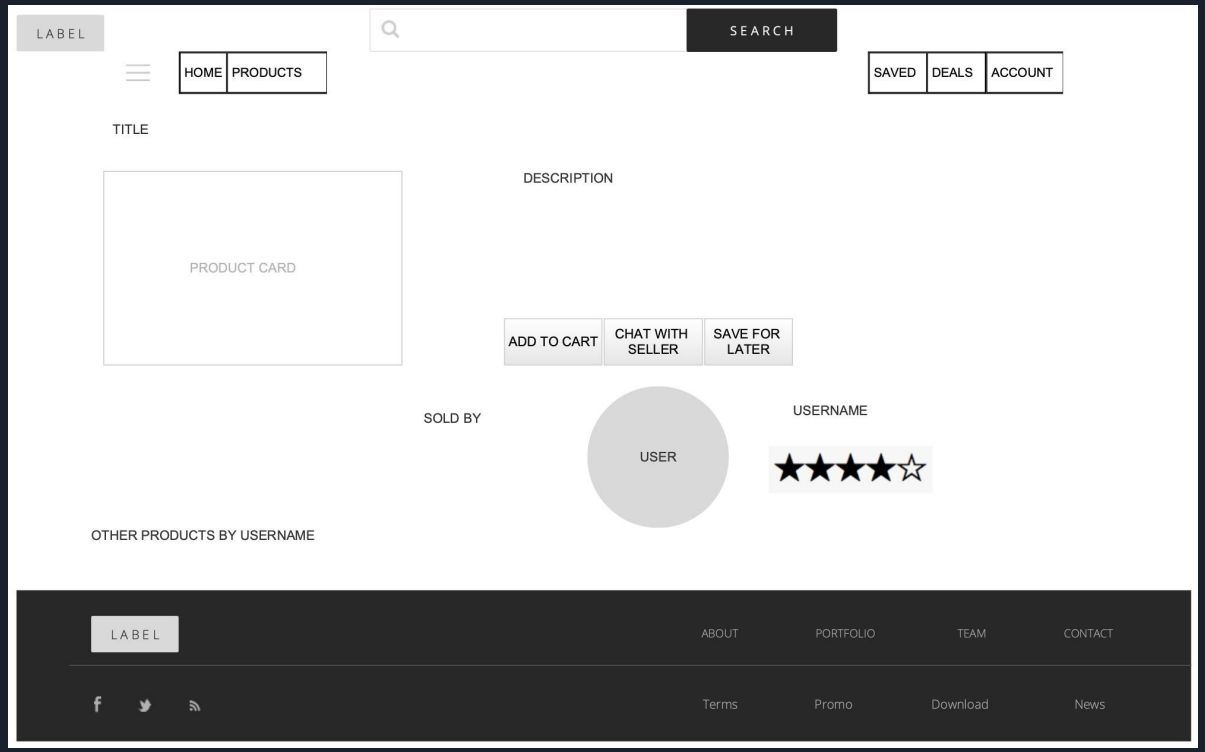
Wireframe diagrams
HOME



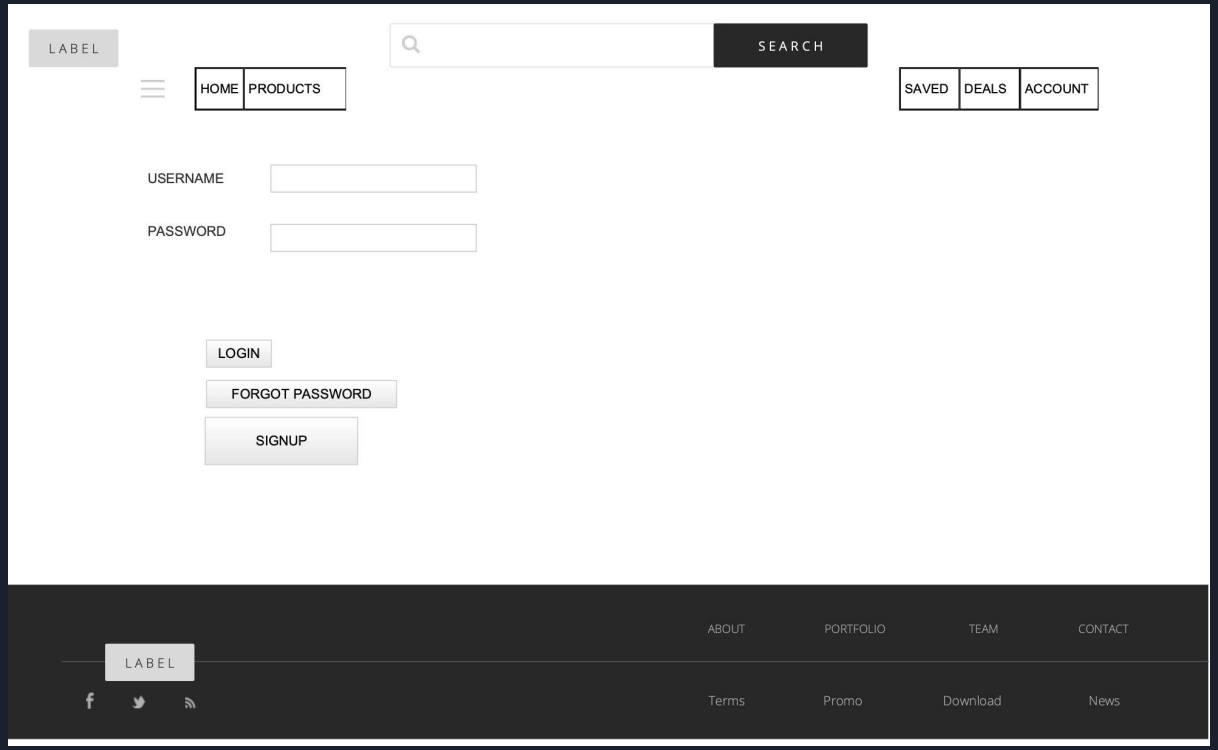
Wireframe diagrams
PRODUCTS



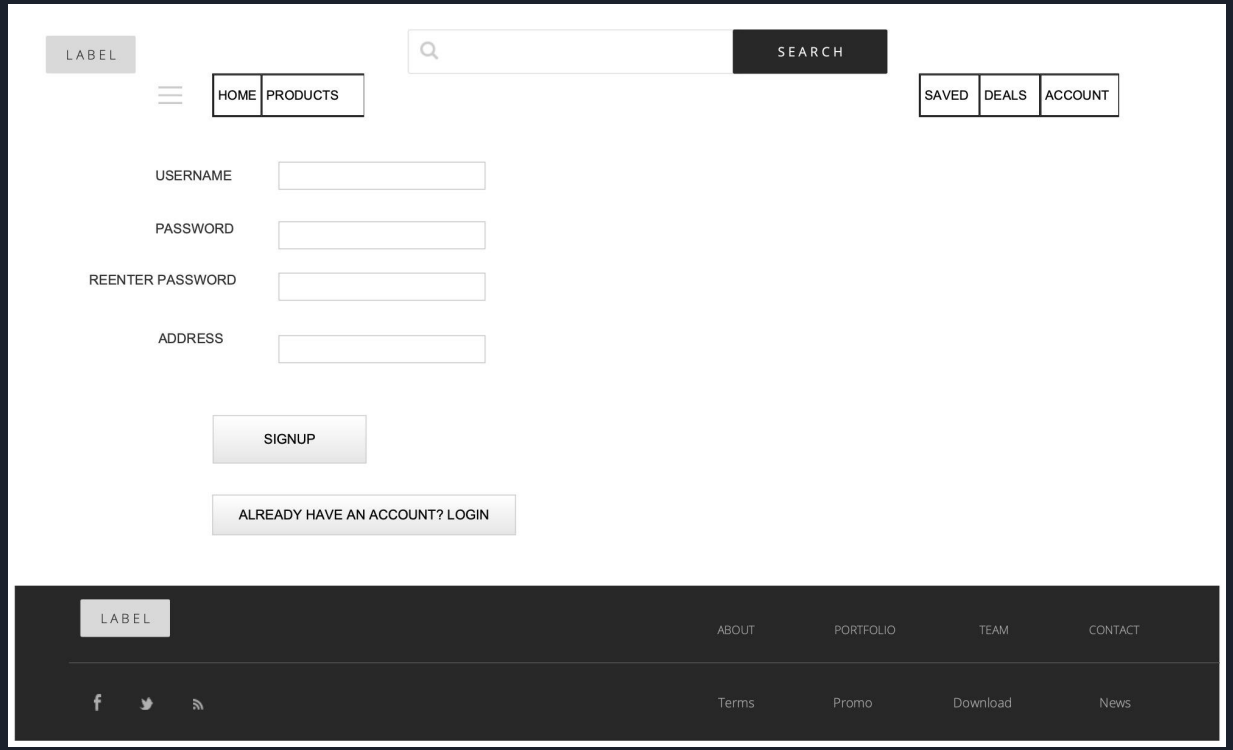
Wireframe diagrams
PRODUCT DETAILS



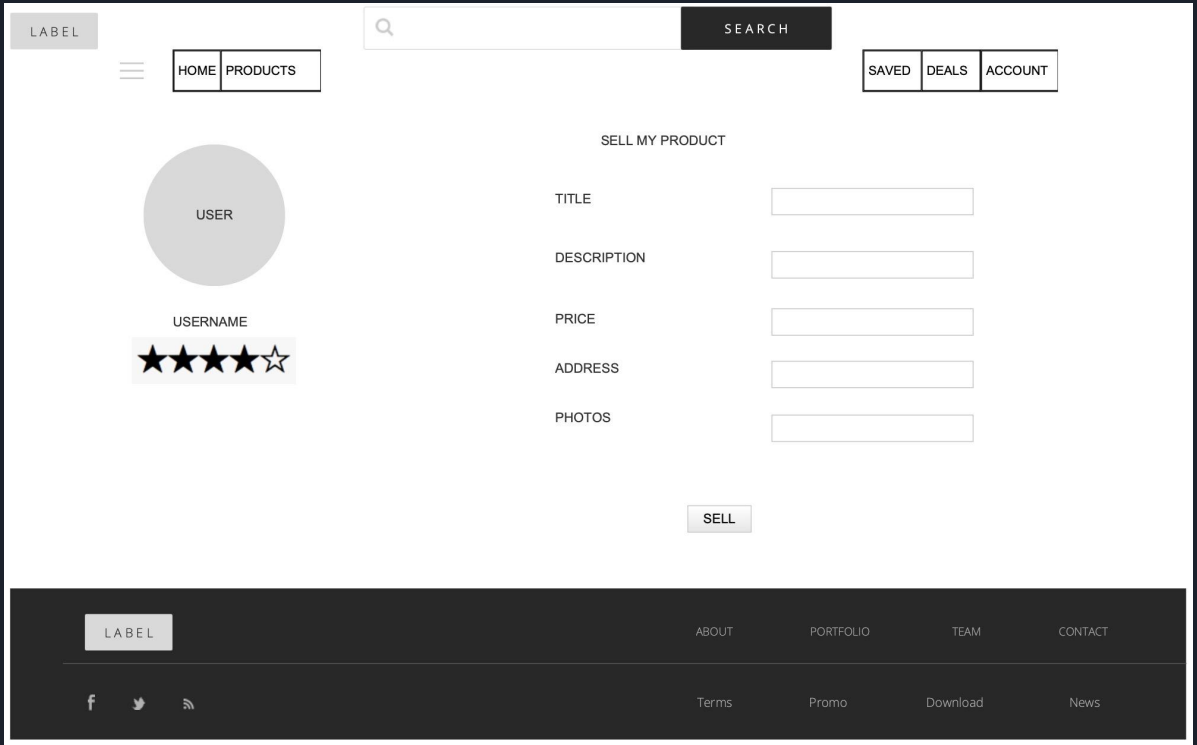
Wireframe diagrams
LOGIN



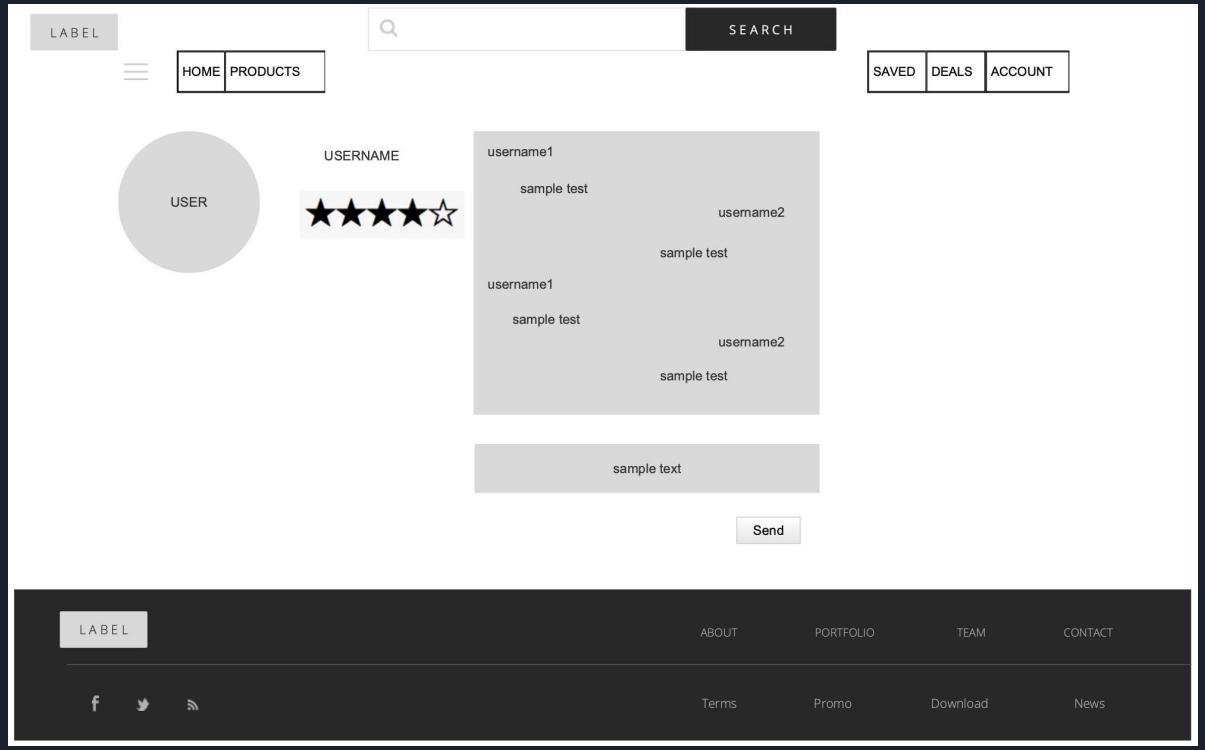
Wireframe diagrams
SIGNUP



Wireframe diagrams
SELL



Wireframe diagrams
CHAT SYSTEM



Web App Demo + Project Structure

Chat System

Purpose :

Ziczac Chat System gives the buyers an opportunity to communicate with the sellers for direct transactions.

Software:

React JS

Dependencies :

1. **Socket.io**
2. Moment
3. Express
4. nodemon

Chat System

Functional Requirements :

1. As a buyer, I want to join the chat system for sending a message to a seller.
2. As a users (buyer/ seller) in the chat system, I want to type the messages into the message box.
3. As a user (buyer/ seller) in the chat system, I want to click a send button (or enter) to send the messages.
4. As a user (buyer/seller) in the chat system, I want to receive the messages that has been sent from the other users.
5. As a user (buyer/ seller) in the chat system, I want to check the time of the messages.
6. As a user (buyer/seller) in the chat system, I want to check the other user's username for confirmation.
7. As a user (buyer/ seller) in the chat system, I want to click a leave button to leave the chat system.

Chat System

Further Steps :

1. Uploading the images or videos
2. Private messaging (one to one)
3. Design
4. Integrating the chat system to the main website

Testing

Unit Testing

- **Python's unit test module will be used to do unit testing.**

UI (User Interface) Testing

Launching React App

- Test case name: Starting the webapp
- New or old: New
- Test items: Testing if the react app will launch correctly, and the homepage will be displayed once the app is activated.
- Test priority (high/medium/low) High
- Dependencies : Node.js
- Preconditions: None
- input data: Code to run the react app in terminal.
- Test steps:
  - Install necessary tools to run react code
  - Run npm build to run the frontend and then python app.py command for backend in terminal
- Postconditions:None
- Expected output: ZicZac webapp homepage displayed
- Actual output: Ziczac webapp homepage displayed
- Pass or Fail: Pass

Testing

App Login:

- Test case name: Login
- New or old: New
- Test items: An existing user with a registered email and password can successfully login
- Test priority: High
- Dependencies : None
- Preconditions : User has a ZicZac account
- input data: Username, Password
- Test steps: Activate react app and navigate to home page
    - Click “Account” and then “Login”
    - Enter credentials
    - Click “Submit”
- Postconditions: None
- Expected output: Login successful
- Actual output: Login successful
- Pass or Fail: PASS

Testing

New User Signup:

- Test case name: Signup
- New or old: New
- Test items: New user registration
- Test priority: High
- Dependencies: None
- Preconditions: None
- input data: Email address, username, password
- Test steps:
  - Activate webapp and navigate to home page
  - Click “Account” then “Sign up”
  - Enter credentials
  - Click “Submit”
- Postconditions: None
- Expected output: Flask message indicating successful or failed registration
- Actual output: Flask message indicating successful or failed registration
- Pass or Fail: PASS

Testing

Sorting products:

- Test case name: Sorting products by price
- New or old: New
- Test items: Sorting items by price
- Test priority : Low
- Dependencies : None
- Preconditions: None
- input data: Price
- Test steps:
  - Activate webapp, navigate to home page
  - Click on the drop down next to “Sort Products”
  - Select price and click “submit”
- Postconditions: None
- Expected output: Products sorted from lowest price to highest
- Actual output: Products sorted from lowest price to highest
- Pass or Fail: PASS

Testing

Item Categories

- Test case name: Filtering products by category
- New or old: New
- Test items: Displaying items by category
- Test priority: High
- Dependencies: None
- Preconditions: None
- input data: None
- Test steps:
  - Activate webapp, navigate to home page
  - Click on any category
- Postconditions: None
- Expected output: Only the products related to the selected category are displayed
- Actual output: Only the products related to the selected category are displayed
- Pass or Fail: PASS

Testing

Products

- Test case name: Product information
- New or old: New
- Test items: Displaying product information on click
- Test priority: High
- Preconditions: None
- input data: None
- Test steps:
  - Activate webapp, navigate to home page
  - Click on any category
  - Click on a product
- Postconditions: None
- Expected output: Product description, seller name, seller rating , add to cart and save for later features displayed
- Actual output: Product description, seller name, seller rating , add to cart and save for later features displayed
- Pass or Fail: PASS

Security

Security of Password

- Strength of Password (Standard)
    - Minimum of eight upper- and lowercase alphanumeric characters
    - Include at least one special character (such as *, &, $, #, !, or @)

- Cryptography (Encryption and Checking)
    - SHA-3 (Secure Hashing Algorithm 3)
    - Salt

Security

SHA-3 (Secure Hashing Algorithm 3)

——the latest member of the Secure Hash Algorithm family of standards, released by NIST on August 5, 2015. Although part of the same series of standards, SHA-3 is internally different from the MD5-like structure of SHA-1 and SHA-2.

(NIST. (2017, January 4). Retrieved from: https://csrc.nist.gov/projects/hash-functions)

Comparison of SHA functions [edit]

In the table below, *internal state* means the number of bits that are carried over to the next block.

Comparison of SHA functions view • talk • edit

Algorithm and variant		Output size (bits)	Internal state size (bits)	Block size (bits)	Rounds	Operations	Security (in bits) against collision attacks	Capacity against length extension attacks	Performance on Skylake (median cpb)[58] Long messages	Performance on Skylake (median cpb)[58] 8 bytes	First published
MD5 (as reference)		128	128 (4 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or	≤18 (collisions found)[59]	0	4.99	55.00	1992
SHA-0		160	160 (5 × 32)	512	80	And, Xor, Rot, Add (mod 2^{32}), Or	<34 (collisions found)	0	≈ SHA-1	≈ SHA-1	1993
SHA-1							<63 (collisions found)[60]		3.47	52.00	1995
SHA-2	*SHA-224*	224	256 (8 × 32)	512	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112	32	7.62	84.50	2004
	SHA-256	256					128	0	7.63	85.25	2001
	SHA-384	384	512 (8 × 64)	1024	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192	128 (≤ 384)	5.12	135.75	2001
	SHA-512	512					256	0[61]	5.06	135.50	
	SHA-512/224	224					112	288	≈ SHA-384	≈ SHA-384	2012
	SHA-512/256	256					128	256			
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	24[62]	And, Xor, Rot, Not	112	448	8.12	154.25	2015
	SHA3-256	256		1088			128	512	8.59	155.50	
	SHA3-384	384		832			192	768	11.06	164.00	
	SHA3-512	512		576			256	1024	15.88	164.00	
	SHAKE128	*d* (arbitrary)		1344			min(*d*/2, 128)	256	7.08	155.25	
	SHAKE256	*d* (arbitrary)		1088			min(*d*/2, 256)	512	8.59	155.50	

Screenshot Reference: https://en.wikipedia.org/wiki/SHA-3

Security

Salt

——A randomly-generated string. Restraint: dictionary attack and rainbow table attack on password.

Password	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz
Salt	-	-	et52ed	ye5sf8
Hash	f4c31aa	f4c31aa	lvn49sa	z32i6t0

Screenshot Reference: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/

Security

Output of Strength Part (sample):

```
Enter the password: 1Aa!
Testing...
['1Aa!', False, 'too short.']

Enter the password: AaAaAaAaAaAaAa
Testing...
['AaAaAaAaAaAaAa', False, 'no number; no symbol.']

Enter the password: !Aa12345678901234
Testing...
['!Aa12345678901234', True, 'good.']
```

Security

Output of Cryptography Part (sample):

```
Enter your password for sign up: !Aa12345
Enter your password for login in: !Aa12345
Salt: 5f82d89da2ca475da39b9d0432b93387
Encrypted password: 182e180492aeae8f924f70b5e6e7ad2e5d2e118d8e5b96af0ec202989a866391e617e94f7682e31116e9237d70e0908f1def82d2f4f085926cb29936c4e30e32
Checking password: True
```

```
Enter your password for sign up: !Aa12345
Enter your password for login in: 12345!Aa
Salt: b27a914b6a9d4f6f928b20aa0db14abe
Encrypted password: a64b7091a2259a45ecdff8054168175851bc0bab0d193f3bc1e9a9bb2e02147c0079f59f73c86b5155d6fd156c0bb712a00ae4b52bcec74576b9d3978fd970f9
Checking password: False
```