

Code Management

Software implementation of planned user stories:

Sprint #3:

Sprint 3							
Request system	User Stories	Hassan	8	29/2/24	19/3/24	10	Complete
Requests endpoints	Sub user story	Hassan	3/8	29/2/24	1/3/24	2	Complete
Request submitter form	Sub user story	Vraj	3/8	1/3/24	3/3/24	2	Complete
Request list view (employee)	Sub user story	Vraj	2/8	1/2/24	3/3/24	2	Complete
Database web-socket connection for notification	Enhancement	Shivam		1/3/24	10/3/24	9	Complete
Add new role: Owner/Renter	Enhancement	Kaothar		1/3/24	3/3/24	2	Complete
Navigation/Routing	Enhancement	Dimitri		2/3/24	11/3/24	9	Complete
User roles management (testing)	User Stories	Shivam	7	2/3/24	10/3/24	8	Complete
Landing Page	Enhancement	Aly		19/3/24	21/3/24	3	Complete
Dashboard	Enhancement	Kaothar		19/3/24	21/3/24	3	Complete
Admin role tests	Sub user story	Omar	3/7	4/3/24	15/3/24	11	Complete
Public user tests	Sub user story	Jackson	2/7	2/3/24	15/3/24	13	Complete
Employee tests	Sub user story	Aly	2/7	2/2/24	15/3/24	13	Complete

Total effort = 71 story points

Story points completed in Iteration #1 = 4

Story points completed in Iteration #2 = 21

Story points completed in Iteration #3 = 15

Average per iteration = 13.33 story points / iteration

of iterations = $71 / 13.33 = 5.325$.

Since we were allotted 5 iterations to complete the project and most of the work done in the first sprint was focused on documentation, we believe we are on schedule to complete the project within the time provided.

Sprint #4:

Total effort = 71 story points

Story points completed in Iteration #1 = 4

Story points completed in Iteration #2 = 21

Story points completed in Iteration #3 = 15

Story points completed in Iteration #4 = 28

Average per iteration = 17 story points / iteration

of iterations = $71 / 17 = 4.18$.

After reevaluating, we can see that in sprint #4 we completed more story points and thus increased our average per iteration. We are perfectly on schedule and therefore can dedicate more time to refactoring and improvements in sprint #5 since most of the story points are complete.

Sprint 4			
Front end			
Reservation System	User Stories	Aly	4/8
Calendar	Sub user story	Kaothar	2/8
List of facilities per property	Sub user story	Vraj	1/8
Events Management	User Stories	Aly	4/6
Events Submission	Sub user story	Kaothar	2/6
Upcoming events (in dashboard)	Sub user story	Dimitri	1/6
Events view (admin)	Sub user story	Aly	1/6
Dashboard for admin	Enhancement	Jackson	
Backend			
<i>Reservation System US006</i>	User Stories	Shivam	4/8
Add facilities to database	Sub user story	Shivam	2/8
Reservation System endpoints	Sub user story	Hassan	1/8
Reservation system logic	Sub user story	Dimitri	1/8
<i>Events Management US010</i>	User Stories	Shivam	2/6
Events endpoints	Sub user story	Hassan	1/6
Events logic	Sub user story	Shivam	1/6

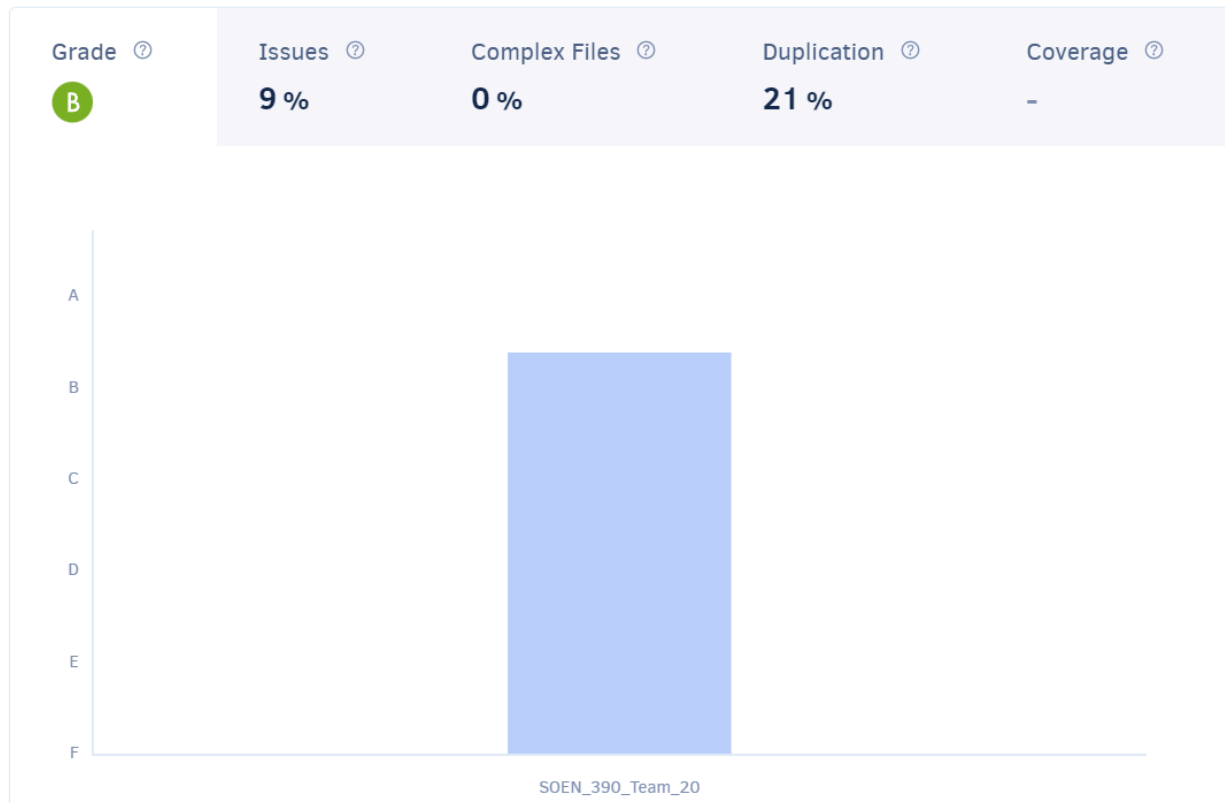
Sprint #5:

In sprint 5 we focused on implementing the financial system which was the last remaining core feature. We refined our code and deployed our application.

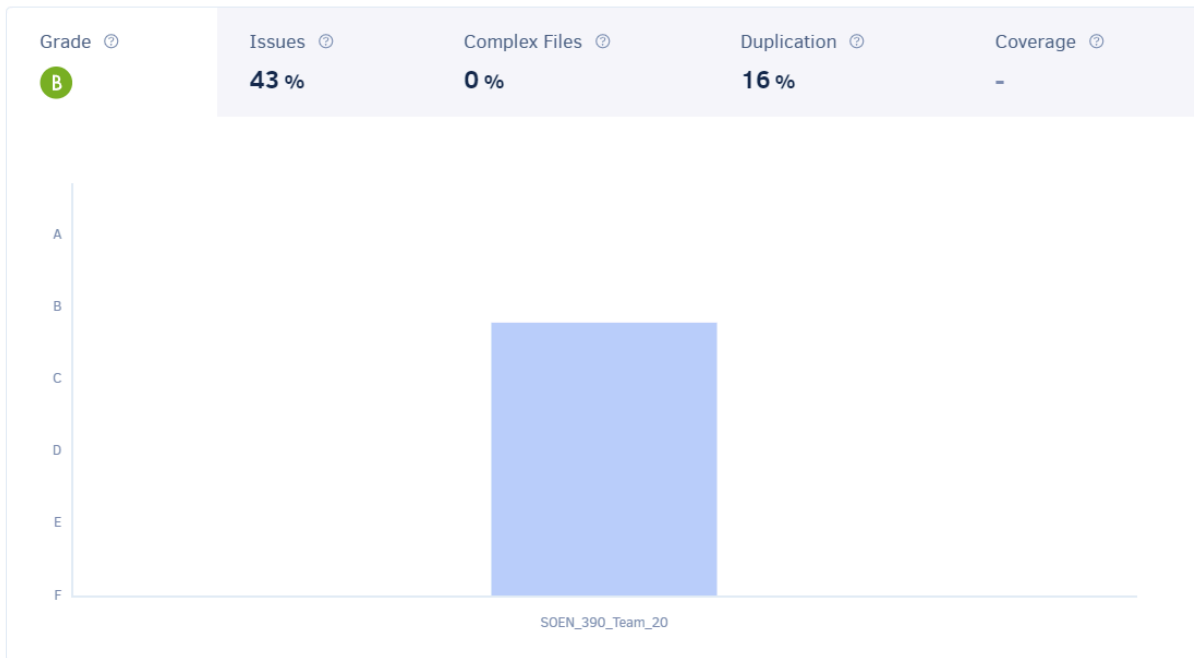
Code Review:

Using Codacy, we ran some tests and received a grade of B. For sprint #4 we can work on reducing our code duplication since it is slightly elevated at 21%

Sprint #3:

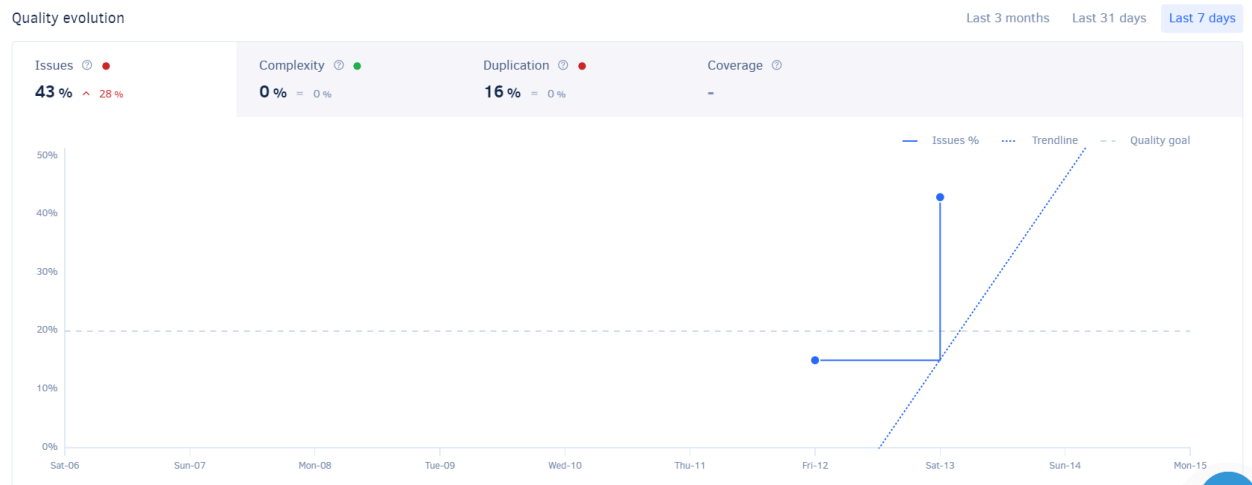


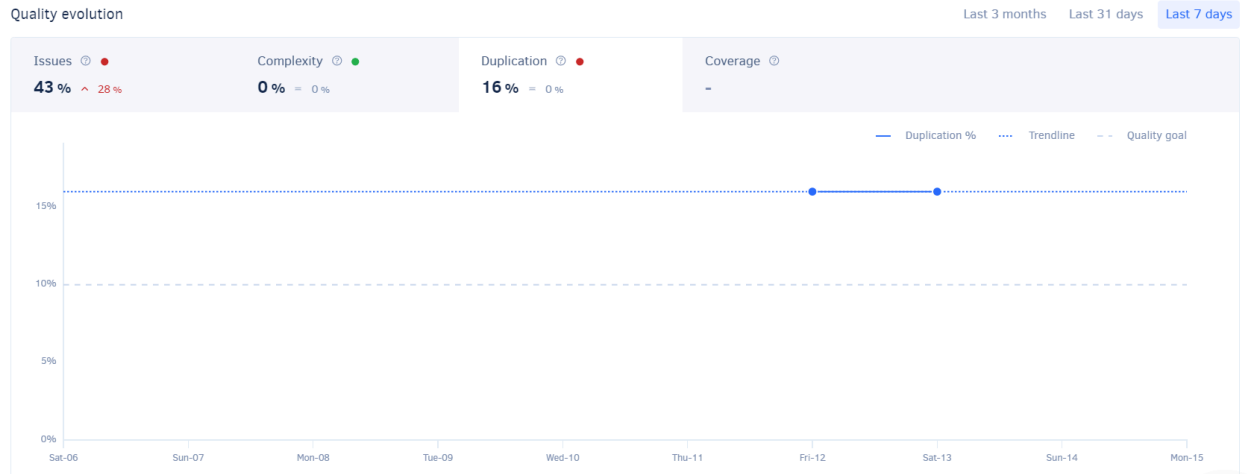
Sprint #4:



We reduced our code duplication from 21% to 16%.

Here are the graphs for the issues and code duplication metrics from Codacy:





Design Pattern:

For our database we are using a factory pattern implementing the IDBController interface.

```
// DBControllerFactory.js
import DBController from "../controllers/DBController";
import { IDBController } from "../types/DBTypes";

/**
 * The `DBControllerFactory` class in TypeScript provides a static method to create
 * instances of
 * `DBController`. */
Comment Code | Improve Code
class DBControllerFactory {
  /**
   * The function `createInstance` returns a new instance of `DBController` implementing
   * the
   * IDBController interface.
   * @returns An instance of the `DBController` class is being returned.
   */
  static createInstance(): IDBController {
    return new DBController();
  }
}

export default DBControllerFactory;
```

We then use Master classes so access the database with the dbController and return data to be routed:

TS accountsMaster.ts

TS postsMaster.ts

TS propertyMaster.ts

TS requestsMaster.ts

TS unitMaster.ts

```
class AccountsMaster {  
  readonly dbController: IDBController; // You might want to re  
  actual type of dbController  
  
  constructor() {  
    this.dbController = DBControllerFactory.createInstance();  
  }  
}
```

```
async getUserDetails(  
  email: string,  
  password: string  
): Promise<{ status: number; data: PublicUserData } | Error> {  
  let result = await this.dbController.getPublicUser(email, password);  
  if (result.message) return new Error(result.message);  
  
  return result as { status: number; data: PublicUserData };  
}
```

For access to the data we used a RESTful API design that creates routes where data can be accessed in a 3 layer system. Here is our routes:

routes

nested_routes

posts.ts

units.ts

accounts.ts

login.ts

properties.ts

requests.ts

signup.ts

users.ts

Example of a route “/requests/unit” that expects a unit_id in the body and returns the requests related to that unit:

```
router.post(
  "/unit",
  Comment Code | Improve Code
  async function (
    req: Request<{}, {}, { unit_id: string }>,
    res: Response<{ status: number; data?: RequestDetails[] } | { response: string }>,
    next: NextFunction
  ) {
    const { unit_id } = req.body;

    try {
      const result = await requestsMaster.getAllUnitRequests(unit_id);
      if (result instanceof Error) {
        throw result as Error;
      }
      res.status(result.status).send(result);
    } catch (error) {
      res.status(500).send({ response: (error as Error).message });
    }
  }
);
```

These can be used by the frontend as API calls and have the data returned.

Refactor Tags:

We included the use of refactor tags to indicate either features/files that needed refactoring. We also include the refactor tag for commits in which refactoring of a code was done.

The screenshot shows a GitHub issue titled "User data should be stored in app-wide state instead of browser local storage." with the issue number #41. The issue is marked as "Open" and was opened by Shiv2205 3 weeks ago. A comment from Shiv2205, posted 3 weeks ago, explains that user data is currently stored in browser local storage, which is accessible from the client-side, causing security issues. The comment mentions that a webhook for app-wide state has been created and that the Nav bar (issue #39) needs to be fixed first. The issue has several labels: "Front-End", "refactor", "Security", and "Sprint 4". The right sidebar shows the issue's metadata, including assignees (none), labels, projects (none yet), and milestones (none).

Refactoring Activity:

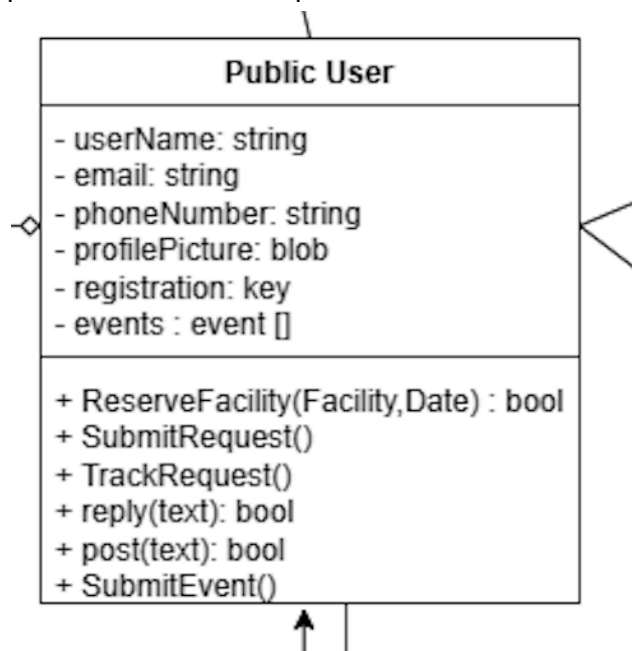
Github Commit involving refactoring	Associated issue closed
Commit #49	Issue #43
Commit #36	Issue #39
Commit #20	Issue #19
Commit #59	Issue #11

Feature Branches:

Since phase 1 of this project, the team has used the idea of feature branches. For every new feature, enhancement or bug we are working on, we create a separate branch from main and work on it. Once we add our new code, we check whether it conflicts with the original code in any way causing errors, bugs or breaks. If nothing is triggered, we are safe to merge back with the original main branch.

Measuring cohesion and coupling in our codebase:

We decided to measure the cohesion and coupling of our system using two distinct methods. The first method is using our class diagram to evaluate our design quality through the use of QMOOD; Specifically using the Cohesion among methods of a class (CAMC) and Direct Class Coupling (DCC) metrics. CAMC is calculated by summing all the distinct parameter types of each method in a class divided by the number of methods multiplied by the number of possible parameters. For example let us take the user class:



$CAMC = (1/24) * (3+1+1+2+2+1) = 0.41666$.

Let us now calculate the CAMC for a smaller class like manager:

Manager
+ PIN: string
+ assignRequest(request): bool

$CAMC = (1/2) * (2) = 1$.

We would then average the CAMC for the entire class diagram. The lower the score the better.

As for coupling, we would count all direct and indirect relationships between classes (inheritance, composition, etc). For the entire class diagram the DCC = 1.92. As we continue implementing our system and refining it we will look to increase the DCC and lower the CAMC.

Backend-Frontend Workflow:

Since we divided our team into back and front end, we needed an efficient way to communicate what we needed. We created a template filled with an example to help streamline

communication/requests to a simple format:

Name: Create Post
Endpoint: (follow convention, leave blank if unsure)

Method: POST

Description:

Creates a new post with the provided data.

Request:

```
{  
  "body": {  
    "post_title": "Title of the post",  
    "post_content": "Content of the post",  
    "creator_id": "ID of the creator"  
  }  
}
```

Response:

```
{  
  "status": "success/error",  
  "message": "Success/Error message",  
  "data": {  
    "post_id": "ID of the created post"  
  }  
}
```

Code Coverage:

We tested coverage for our controller classes as well for some other components. For our controller class we maintain roughly 75 - 80 % coverage which is very good. This is a great indicator that our tests extensively cover our code base.

Sprint #3:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	84.14	62.58	86.2	89.35	
Factory	100	100	100	100	
DBControllerFactory.ts	100	100	100	100	
controllers	76.47	66.27	79.76	79.88	
DBController.ts	76.47	66.27	79.76	79.88	418-442,582-697
repo	76.62	6.25	95	95.16	
accountsMaster.ts	82.14	0	100	100	36-108
postsMaster.ts	80	25	100	100	34-67
propertyMaster.ts	80	0	100	100	22-52
unitMaster.ts	63.15	0	80	80	53-55
routes	97.45	70	100	99.09	
accounts.ts	100	85.71	100	100	21
login.ts	94.11	66.66	100	94.11	35
properties.ts	94.28	50	100	100	27-59
signup.ts	100	75	100	100	25
routes/nested_routes	89.74	84.61	87.5	90.62	
posts.ts	100	80	100	100	10
units.ts	81.81	87.5	77.77	83.33	46-52
tests/unit/utills	100	100	100	100	
recordExistsTest.js	100	100	100	100	
types	100	100	100	100	
DBTypes.ts	100	100	100	100	

Sprint #4:

All files

81.06% Statements 428/528 59.44% Branches 85/143 79.87% Functions 123/154 86.12% Lines 385/447

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements	Branches	Functions	Lines
Factory	100%	3/3	100%	0/0
controllers	69.95%	163/233	60.22%	53/88
repo	76.82%	63/82	16.66%	3/18
routes	97.45%	115/118	70%	14/20
routes/nested_routes	89.74%	70/78	84.61%	11/13
tests/unit/utills	100%	3/3	100%	0/0
types	100%	11/11	100%	4/4

Our code coverage remains high as over 85% of lines and over 80% of statements remain covered.

Sprint #5:

All files

76.05% Statements 486/639 55.36% Branches 98/177 76.43% Functions 146/191 80.84% Lines 439/543

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements	Branches	Functions	Lines
Factory	<div><div></div></div>	100%	3/3	100%	0/0
controllers	<div><div></div></div>	64.76%	204/315	55.93%	66/118
repo	<div><div></div></div>	72.07%	80/111	13.63%	3/22
routes	<div><div></div></div>	97.45%	115/118	70%	14/20
routes/nested_routes	<div><div></div></div>	89.74%	70/78	84.61%	11/13
tests/unit/utlis	<div><div></div></div>	100%	3/3	100%	0/0
types	<div><div></div></div>	100%	11/11	100%	4/4

Coding Guides:

We recently switched from javascript to typescript and within that transition adopted the google typescript coding style. Reference for this coding style is found here :

<https://google.github.io/styleguide/tsguide.html>

Bug Reports:

We are using github to document, label, categorize and assign our bugs.

	1 Open	2 Closed	Author	Label	Projects	Milestones
<input type="checkbox"/>	<input checked="" type="radio"/>			bug		
#33 opened 5 hours ago by KaotharReda						
<input type="checkbox"/>	<input checked="" type="radio"/>			Back-End	bug	Testing
#16 by Shiv2205 was closed 2 weeks ago						
<input type="checkbox"/>	<input checked="" type="radio"/>			bug		
#10 by Shiv2205 was closed 3 weeks ago						

Tests failing in accounts.test.js and login.test.js #16

Closed Shiv2205 opened this issue 2 weeks ago · 1 comment

Shiv2205 commented 2 weeks ago Owner ...

FAIL tests/unit/accounts.test.js (5.296 s)
● Express accounts > /users > should get user details

```
expect(received).toEqual(expected) // deep equality

- Expected   - 3
+ Received   + 0

Object {
  - "data": Object {
    -   "name": "John Doe",
    - },
  - "status": 200,
}

85 |       const response = await request(app).post("/users").send(req.body);
86 |       expect(response.status).toEqual(200);
> 87 |       expect(response.body).toEqual({
    |                               ^
88 |         status: 200,
89 |         data: { name: "John Doe" },
90 |       });

at Object.toEqual (tests/unit/accounts.test.js:87:29)
```

Assignees
Shiv2205

Labels
Back-End bug Testing

Projects
None yet

Milestone
No milestone

Development
[Create a branch for this issue or link a pull request.](#)

Notifications Customize
[Unsubscribe](#)
You're receiving notifications because you're watching this repository.

1 participant

As you can see in the image below, we are referencing our issues when we implement/solve them in our commits:

Mega feature #36

Merged DobDub merged 19 commits into `main` from `megaFeature` 3 weeks ago

Conversation 3 Commits 19 Checks 3 Files changed 69

Aly-Hussein commented 3 weeks ago ...

PR for [issue #34](#)

Vraj2301 and others added 17 commits 3 weeks ago

- Service List 52857b8 ✓
- added router for front-end webpages 19beffd ✓
- Finalized landing page boiler plate 8abdfb1 ✓
- linked Nav to login 597e4d8
- Merge branch 'router-setup' into linkProfilePage e0e72e6
- linked login and sign up pages to landing page d4dd267 ✓

Bug Reporting/Fixing:

Bug ID	Issue #10	
Originator	Shivam	Signature : https://github.com/Shiv2205
Submit Date	March 3rd, 2024	
Summary	User data is not being received when attempting to login.	
Severity	Normal	
Product	website	
Component	Login frontend / backend	

Bug ID	Issue #16	
Originator	Shivam	Signature https://github.com/Shiv2205
Submit Date	March 4th, 2024	
Summary	Two tests are failing one in the account unit tests and one in the login unit tests.	

Severity	minor
Product	website
Component	Testing Suite

Bug ID	Issue #33	
Originator	Koathar	Signature https://github.com/KaotharReda
Submit Date	March 21st, 2024	
Summary	The signup on the landing page is not working.	
Severity	major	
Product	website	
Component	Frontend login component	

Bug ID	Issue #38
--------	---------------------------

Originator	Hassan	Signature :https://github.com/DobDub
Submit Date	March 22nd, 2024	
Summary	The sign in button only activates when clicking on the plaintext and not the actual button.	
Severity	Enhancement	
Product	website	
Component	Front end sign in page	

Bug ID	Issue #39	
Originator	Hassan	Signature :https://github.com/DobDub
Submit Date	March 22nd, 2024	
Summary	Upon signing in, the navigation bar is not being displayed.	
Severity	enhancement	
Product	website	

Component	Frontend component
------------------	--------------------

Bug ID	Issue #42	
Originator	Shivam	Signature https://github.com/Shiv2205
Submit Date	March 28th, 2024	
Summary	Websocket connection closes between client and server before it can be properly established.	
Severity	blocker	
Product	website	
Component	Websocket component	

Bug ID	Issue #43	
Originator	Shivam	Signature https://github.com/Shiv2205
Submit Date	March 31st, 2024	

Summary	We have failing unit tests for the DBController class more specifically the unit and posts components.
Severity	normal
Product	website
Component	Testing suite

Bug ID	Issue #50	
Originator	Hassan	Signature :https://github.com/DobDub
Submit Date	April 16th, 2024	
Summary	Redirecting the user to the login page even with incorrect credentials.	
Severity	major	
Product	website	
Component	Front end and Back end components	

Bug ID	Issue #71
---------------	---------------------------

Originator	Hassan	Signature :https://github.com/DobDub
Submit Date	April 30th, 2024	
Summary	A request returns status 200 but does not update or return anything.	
Severity	minor	
Product	website	
Component	Backend / Database	