Thursday, 3rd August 2023

Engineering WORKSHOP

Chalanika Sewwandi

Senior Software Engineer @ Rootcode Labs

Hasara Samson

Software Engineer @ Rootcode Labs

Hi there, I'm Chalanika Sewwandi.

Senior Software Engineer @ Rootcode Labs

rootcode LABS

&

Hi there, I'm Hasara Samson.

Software Engineer @ Rootcode Labs

rootcode LABS

Hackathon Challenge

15th August @ 00:01 hrs

6 Days / 144 Hours

Deliverables

1. Create a GitHub repo with your project *
   a. Repo name -: TeamName_ProductName
   b. Repo access -: public repo access
2. Include a README file with all the steps to run the application with configurations *
3. Provide a ER diagram and a DB dump *
4. Provide a demo video (Under 10 mins) with all the user flows *
5. Provide a document with limitations, assumptions and future improvements

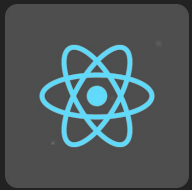
Submission

The team leader should complete the relevant form by providing the following details.

- Github repo url
- Upload the documents with instructions, future improvements and ER diagram (Rename as TeamName_ProductName)
- DB dump
- Demo Video (Rename as TeamName_ProductName_Dev_Demo)

Technologies

Front End






Back End

Database

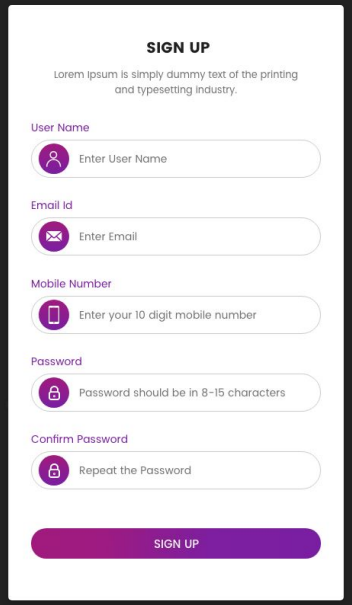
Evaluation Criterions

- Functionality
- Code Architecture
- Code quality
- DB design
- Best practices

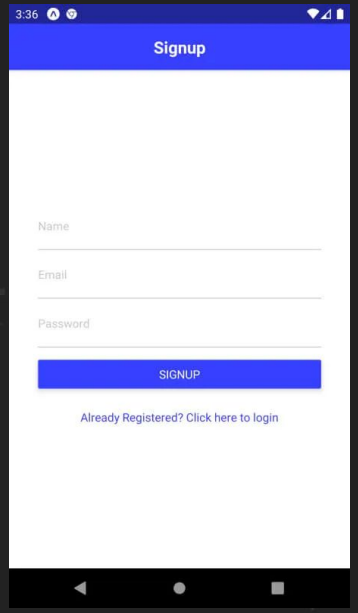
Functionality

- Align with the design
- Correctness & Completeness
- Mobile responsiveness
- Error handling & validations

Align with the design

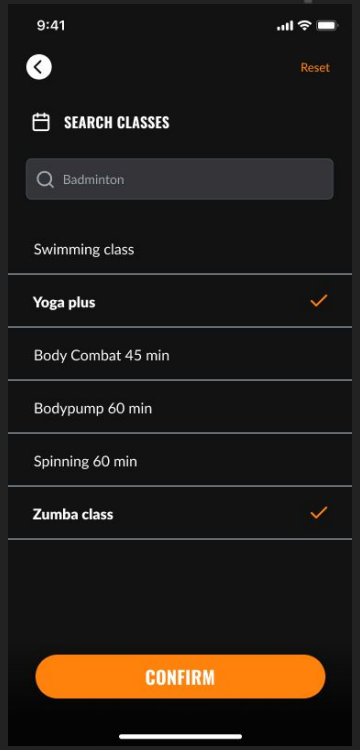


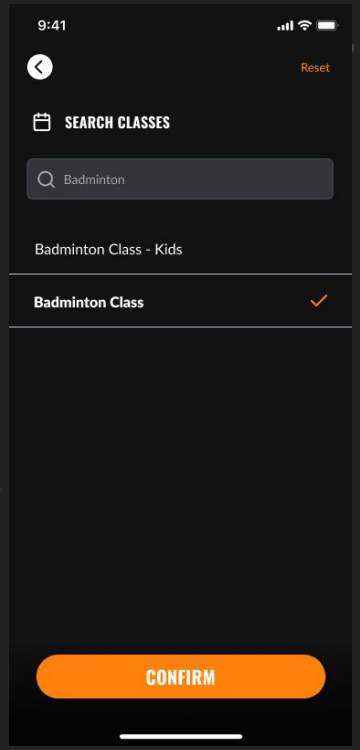
Design



Implementation

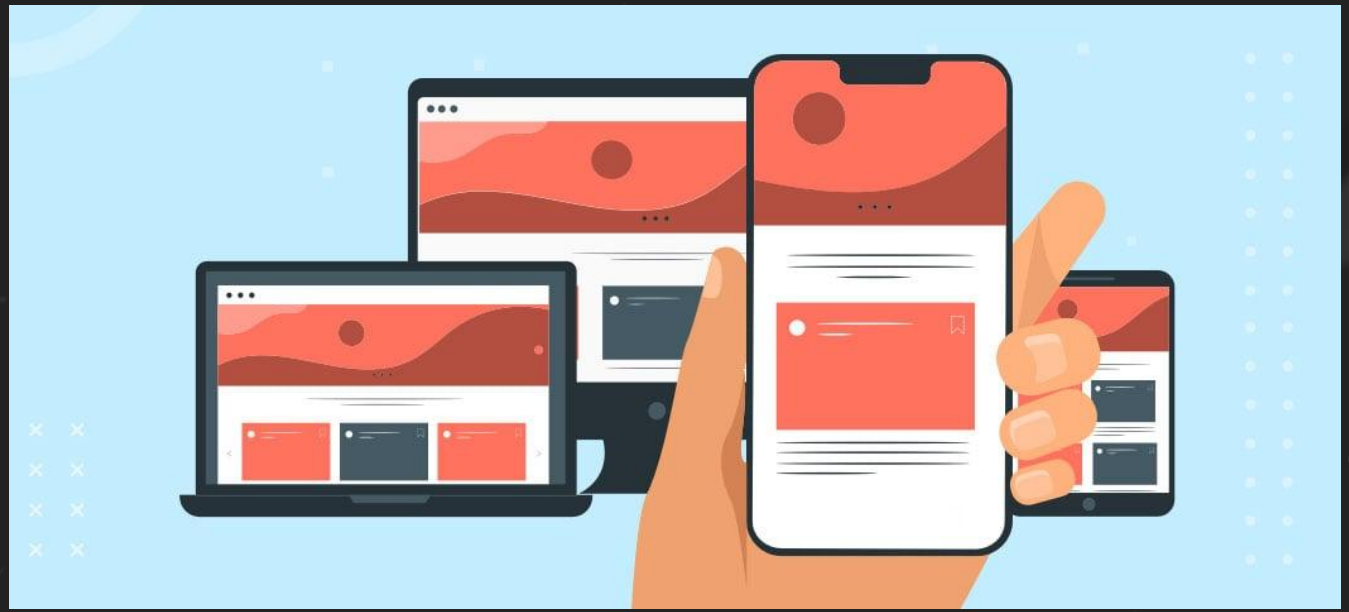
Correctness and Completeness





Mobile responsiveness

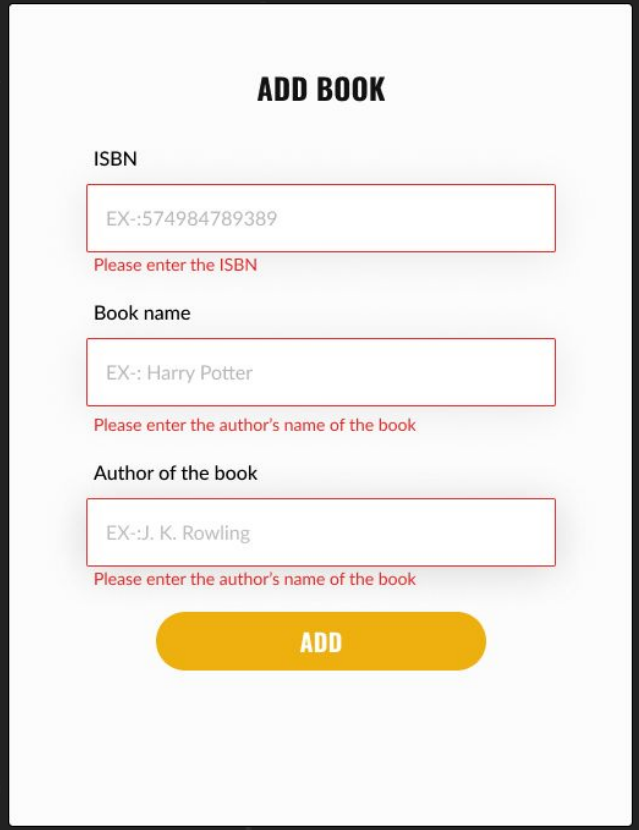
Mobile First Approach



Error handling and validations

1. Form validations
2. API error handling

```java
@PostMapping
public ResponseEntity<String> addBook(@RequestBody Book book) {
    try {
        if (isIsbnAlreadyExists(book.getIsbn())) {
            // ISBN number already exists, return meaningful error message
            return ResponseEntity.badRequest().body("ISBN number already exists in the database.");
        }

        bookList.add(book);
        return ResponseEntity.ok("Book added successfully.");
    } catch (Exception e) {

        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error occurred while
                processing the request.");
    }
}
```



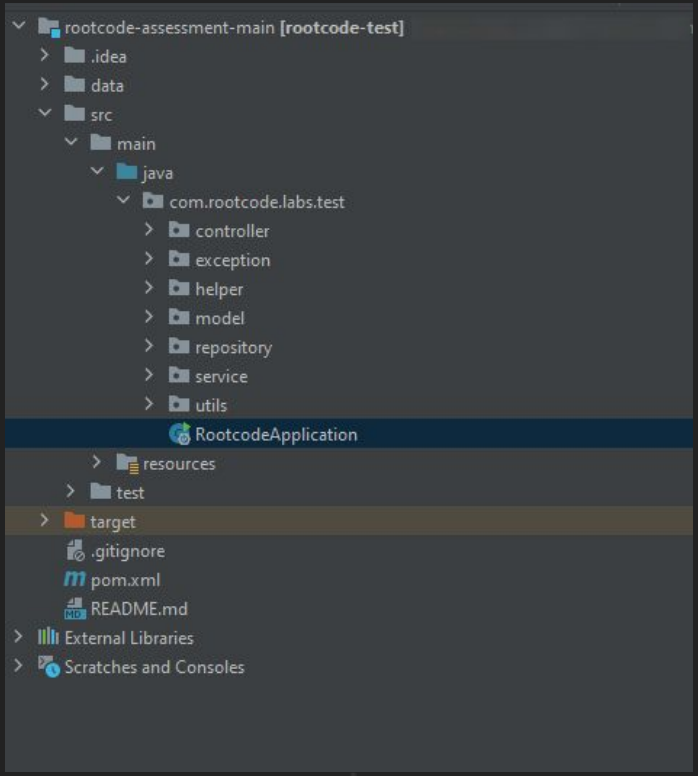
Code Architecture

- Folder structure
- Proper API methods definitions & distribution
- Component architecture

Folder structure

Example -:

```
project/
├── src/
|   ├── assets/
|   ├── components/
|   ├── helpers/
|   ├── pages/
|   ├── services/
|   └── utils/
```

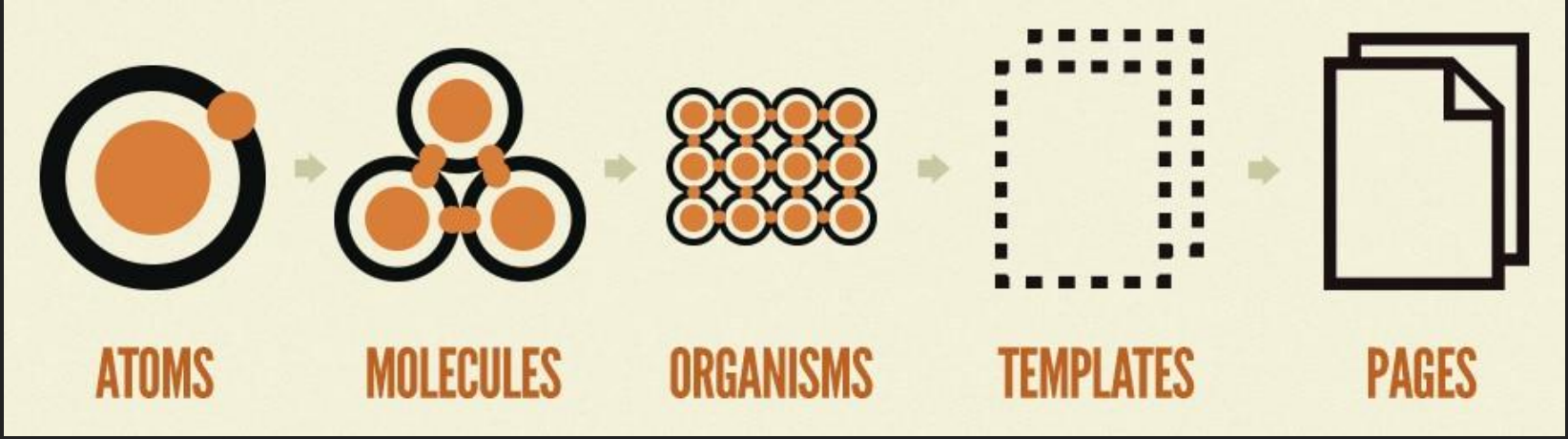


Proper API methods definitions

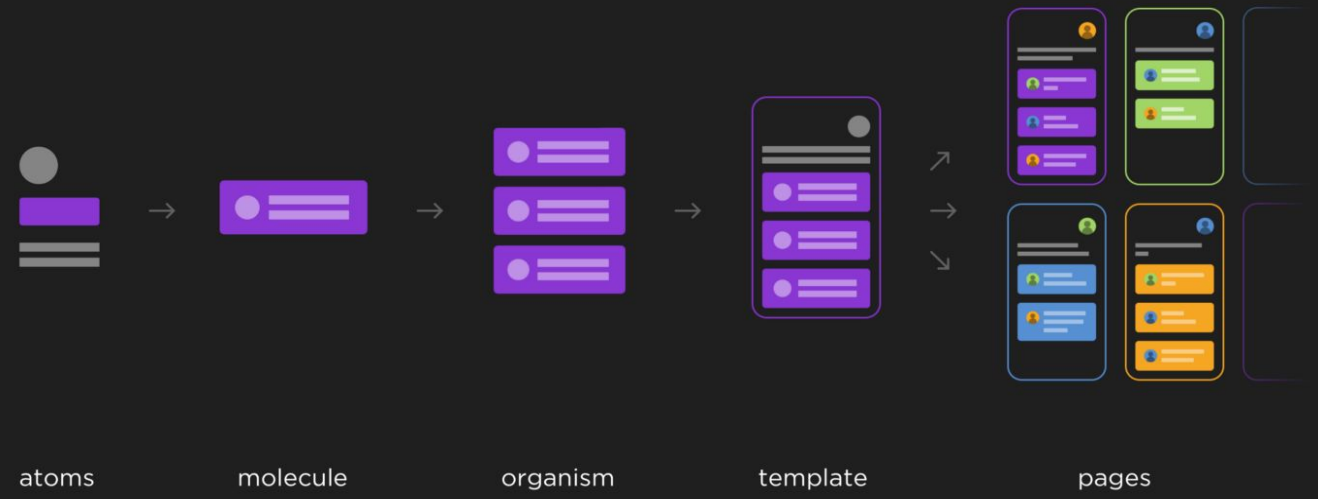
```java
@RestController
@RequestMapping("/products")
public class ProductController {

    // Get all products (GET)
    @GetMapping
    public ResponseEntity<List<Demo>> getAllProducts() {

    }

    // Get product by ID (GET)
    @GetMapping("/{id}")
    public ResponseEntity<Demo> getProductById(@PathVariable("id") long id) {

    }

    // Delete product by ID (DELETE)
    @DeleteMapping("/{id}")
    public ResponseEntity<String> deleteProductById(@PathVariable("id") long id) {

    }
```

Component Architecture

Atomic Design Methodology



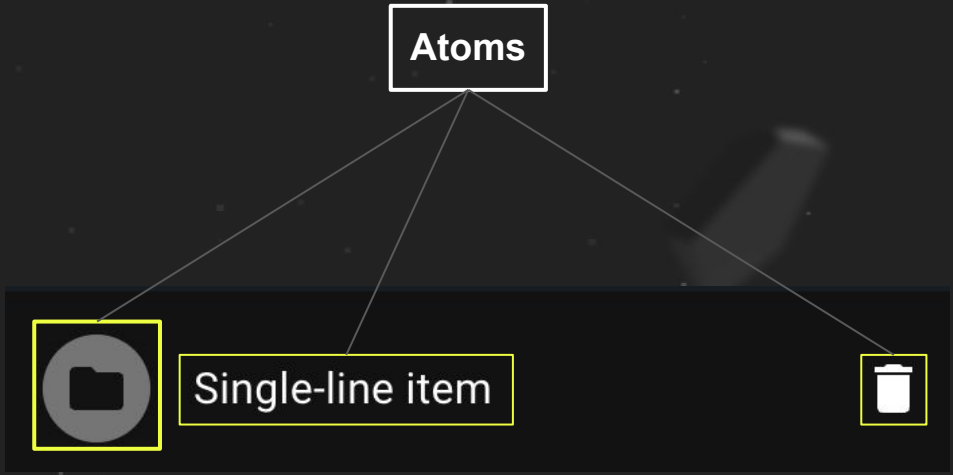
Atomic Design Methodology



Atoms

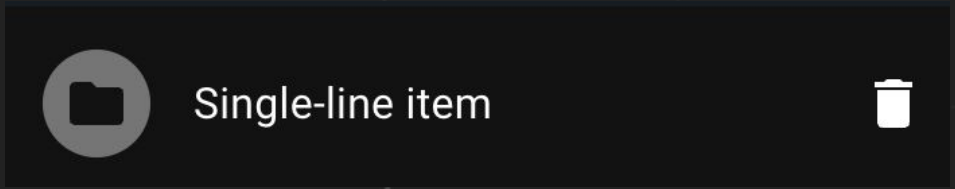
Atoms are the smallest and most basic building blocks of a design system.

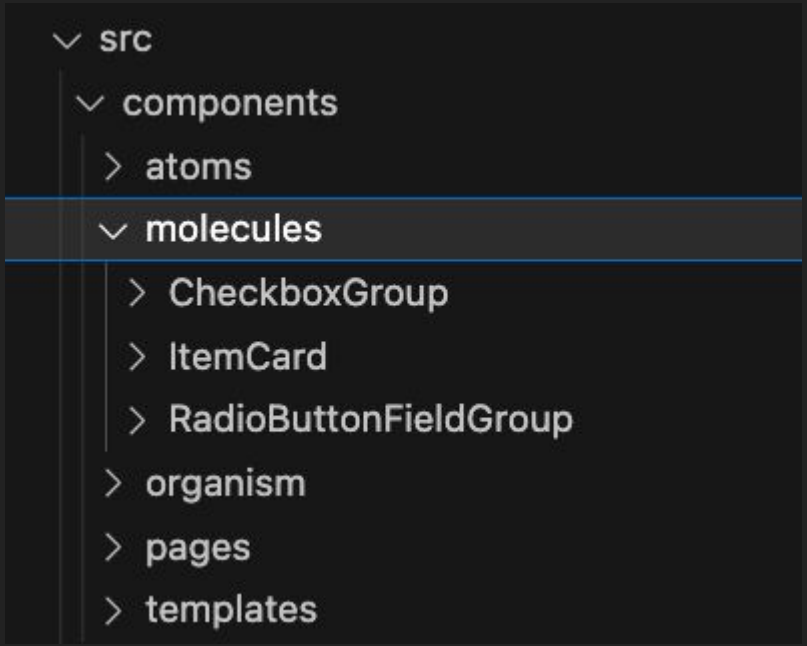
- src
  - components
    - atoms
      - Avatar
      - Button
      - CheckboxField
      - Icon
      - RadioButtonField
      - Text
      - TextField
    - molecules
    - organism
    - pages
    - templates



Molecules

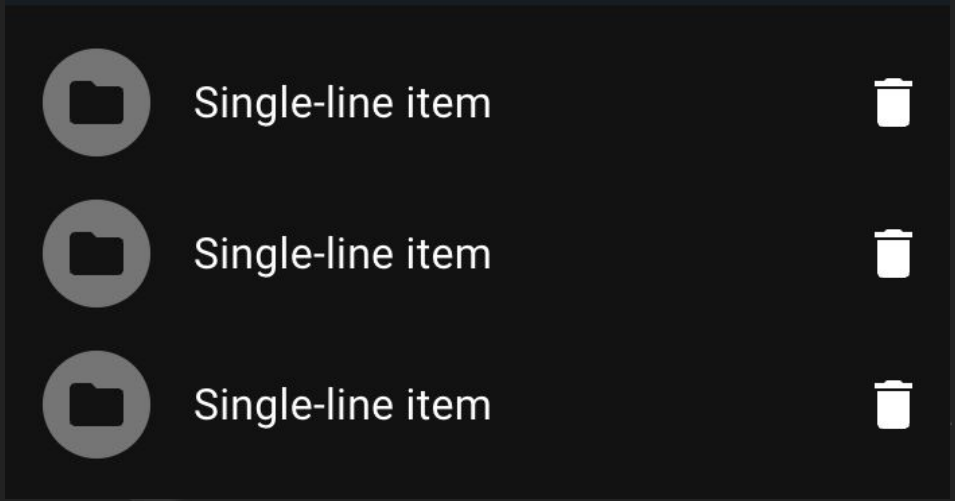
A molecule is a collection of atoms that combine to create a group of repeatable elements.

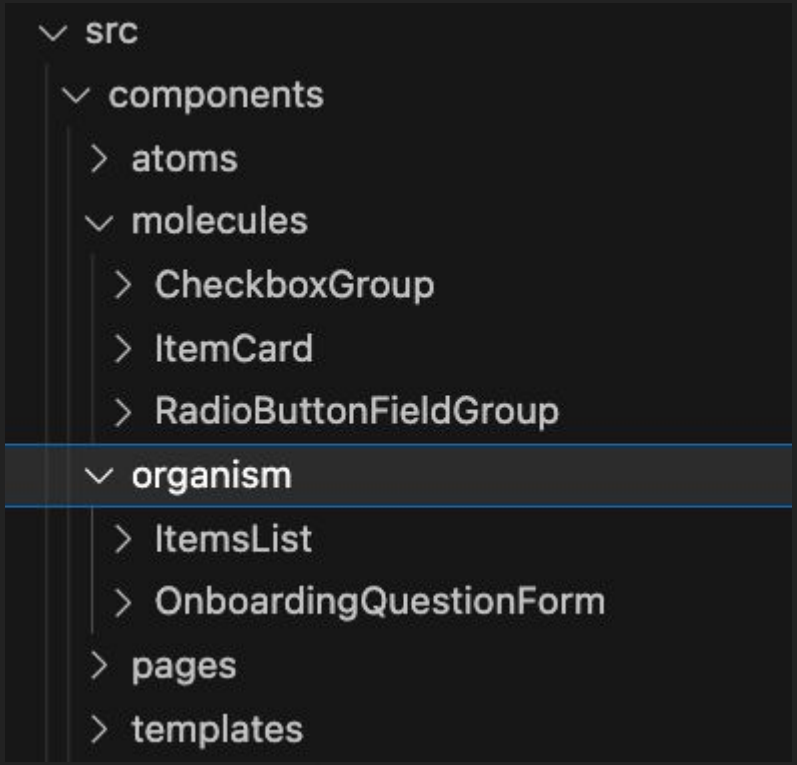




Organism

Organisms are made up of one or more molecular or atomic components and are used to create even more complex UI elements.





Templates and pages

In Atomic Design, templates and pages are the final building blocks after organisms.

Templates

- Ability to reuse common functionality or props across multiple pages.
- Provide the structure for a page but don't contain real content.

Pages

Pages are specific instances of templates where real content is populated

Code Quality

- Type usage
- Code analysis and formatting
- Code refactoring
- Comments
- Style usage
- Meaningful names
- Class and methods distribution

Why Types ??

- Early error detection
- Code readability
- Code maintainability

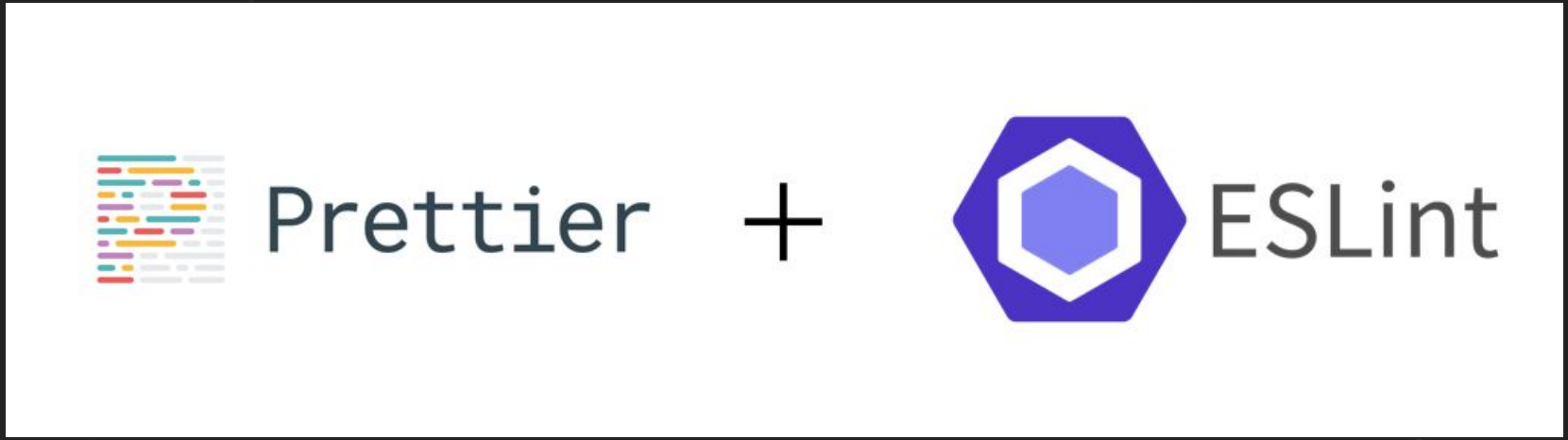
JavaScript

```
let bookName = 'Harry Potter';

bookName = 10; // allowed in JavaScript
```

TypeScript

```
let bookName:string = 'Harry Potter';

bookName = 10; // not allowed in TypeScript
```

Code Analysis And Formatting



Code Refactoring

```typescript
function calculateTotalPriceWithTax(items: Item[], taxRate: number): number {
  let total = 0;
  for (const item of items) {
    total += item.price;
  }
  const taxAmount = total * taxRate;
  total += taxAmount;
  return total;
}
```

Before refactoring

Code Refactoring

```typescript
function calculateTotalPriceWithTax(items: Item[], taxRate: number): number {
  let total = 0;
  for (const item of items) {
    total += item.price;
  }
  const taxAmount = total * taxRate;
  total += taxAmount;
  return total;
}
```

Before refactoring

```typescript
function calculateTotalPrice(items: Item[]): number {
  let subtotal = 0;
  for (const item of items) {
    subtotal += item.price;
  }
  return subtotal;
}

function calculateTax(amount: number, taxRate: number): number {
  return amount * taxRate);
}

function calculateTotalPriceWithTax(items: Item[], taxRate: number): number {
  const totalPrice = calculateTotalPrice(items);
  const totalWithTax = totalPrice + calculateTax(totalPrice, taxRate);
  return totalWithTax;
}
```

After refactoring

Code Comments

Where do we need to add code comments ?

- Complex algorithm
- Design decision
- TODOs and Future improvements
- Edge Cases and Assumptions

What makes a good comment ?

- Clear and concise
- Explain purpose
- Updates with code changes

Style Usage

```
import React from 'react';
import {StyleSheet, Text, View} from 'react-native';

const LotsOfStyles = () => {
  return (
    <View style={{ marginTop: 50}}>
      <Text>blue</Text>
    </View>
  );
};

export default LotsOfStyles;
```

style.ts

```
import { StyleSheet, ViewStyle } from 'react-native';

interface Styles {
  container: ViewStyle;
}

export default (): Styles =>
  StyleSheet.create({
    container: {
      marginTop: 20,
      flex: 1,
      paddingBottom: 30,
    },
});
```

```
import React from 'react';
import {StyleSheet, Text, View} from 'react-native';
import styles from './styles';

const LotsOfStyles = () => {
    const { container} = styles();
  return (
    <View style={container}>
      <Text>blue</Text>
    </View>
  );
};

export default LotsOfStyles;
```

Meaningful Names

- Use intention revealing names
- Use suitable naming conventions

```
const x = 10;
const arr = [1, 2, 3];
const res = calculate(x, arr);
```

```
const numberOfStudents = 10;
const grades = [85, 92, 78];
const averageGrade =
calculateAverageGrade(numberOfStudents, grades);
```

Naming conventions

```java
public class NamingConventionExample {

    // Naming convention for variables: camelCase
    private double x; //this is wrong
    private double totalPrice;

    // Naming convention for methods: camelCase
    public void calculateTotalPrice(int quantity, double pricePerItem) {
        totalPrice = quantity * pricePerItem;
        System.out.println("Total Price: " + totalPrice);
    }

}
```

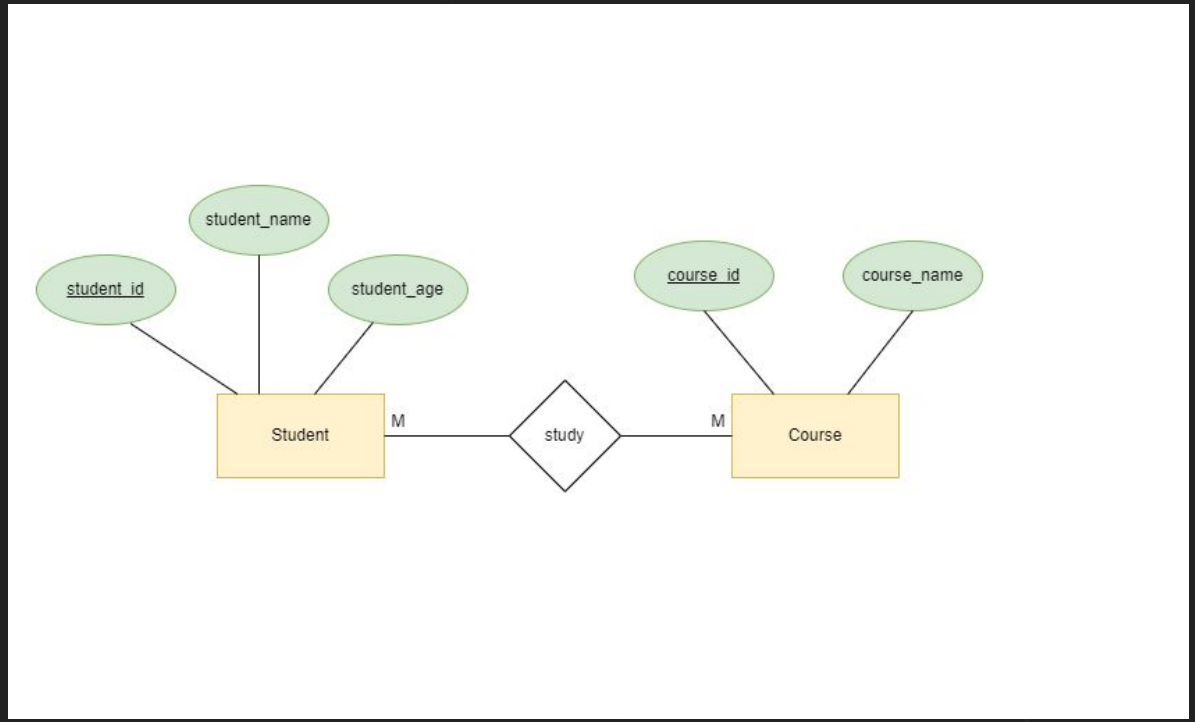
Class and method distribution

```java
public class Order {

    public void processOrder(Order order) {

            if (isValidOrder(order)) {

                double totalPrice = calculateTotalPrice(order);

                processPayment(order, totalPrice);

                updateInventory(order);

                sendOrderConfirmationEmail(order);
            } else {
                System.out.println("Invalid order. Please check order details.");
            }
    }

    private boolean isValidOrder(Order order) {
        // Validate order details and return true if valid, otherwise false
        return true;
    }

    private double calculateTotalPrice(Order order) {
        // Calculate total price of the order and return it
        return 0.0;
    }

    private void processPayment(Order order, double totalPrice) {
        // Process payment for the order
    }

    private void updateInventory(Order order) {
        // Update inventory based on the order items
    }

    private void sendOrderConfirmationEmail(Order order) {
        // Send order confirmation email to the customer
    }
}
```

DB design

- Normalization
- Data integrity
- Scalability & extensibility
- Best practices for each database technology
- ER Diagram and data model

ER Diagram



Best Practices

- Security
- Documentation
- Hosting
- Test cases
- Optimizations
- Logs
- Dependency Management

Test cases

```java
public class UserRepositoryTest {
    private UserRepository userRepository;

    @BeforeEach
    public void setUp() {
        userRepository = new UserRepository();
    }

    @Test
    public void testCreateUser() {

        User user = new User(1, "John Doe", 30);

        userRepository.createUser(user);
        User retrievedUser = userRepository.readUser(1);

        Assertions.assertNotNull(retrievedUser);
        Assertions.assertEquals(user.getName(), retrievedUser.getName());
        Assertions.assertEquals(user.getAge(), retrievedUser.getAge());
    }
}
```

Optimizations

```java
//first approach
for (Product product : productList) {
    DbHelper.executeQuery("update product set price = 1000 where", product.getId());
}

//second approach
List<Integer> productIds = new ArrayList<>();
for (Product product : productList) {
    productIds.add(product.getId());
}

DbHelper.executeQuery("update product set price = 1000 where id IN (" + productIds + ")");
```

Logs

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LogExample {

    private static final Logger log = LoggerFactory.getLogger(LogExample.class);

    public static void main(String[] args) {
        int a = 10;
        int b = 0;

        try {
            int result = divide(a, b);
            log.info("Division result: {}", result);
        } catch (ArithmeticException ex) {
            log.error("Error occurred during division: {}", ex.getMessage(), ex);
        }
    }

    public static int divide(int dividend, int divisor) {
        if (divisor == 0) {
            throw new ArithmeticException("Division by zero is not allowed!");
        }
        return dividend / divisor;
    }
}
```

Dependency management

```xml
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

<dependency>
    <groupId>com.opencsv</groupId>
    <artifactId>opencsv</artifactId>
    <version>5.0</version>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

package.json

```json
{
  "name": "tech-triathlon",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event":
"^13.5.0",
    "@types/jest": "^27.5.2",
    "@types/node": "^16.18.39",
    "@types/react": "^18.2.17",
    "@types/react-dom": "^18.2.7",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "typescript": "^4.9.5",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

yarn

npm

Q & A