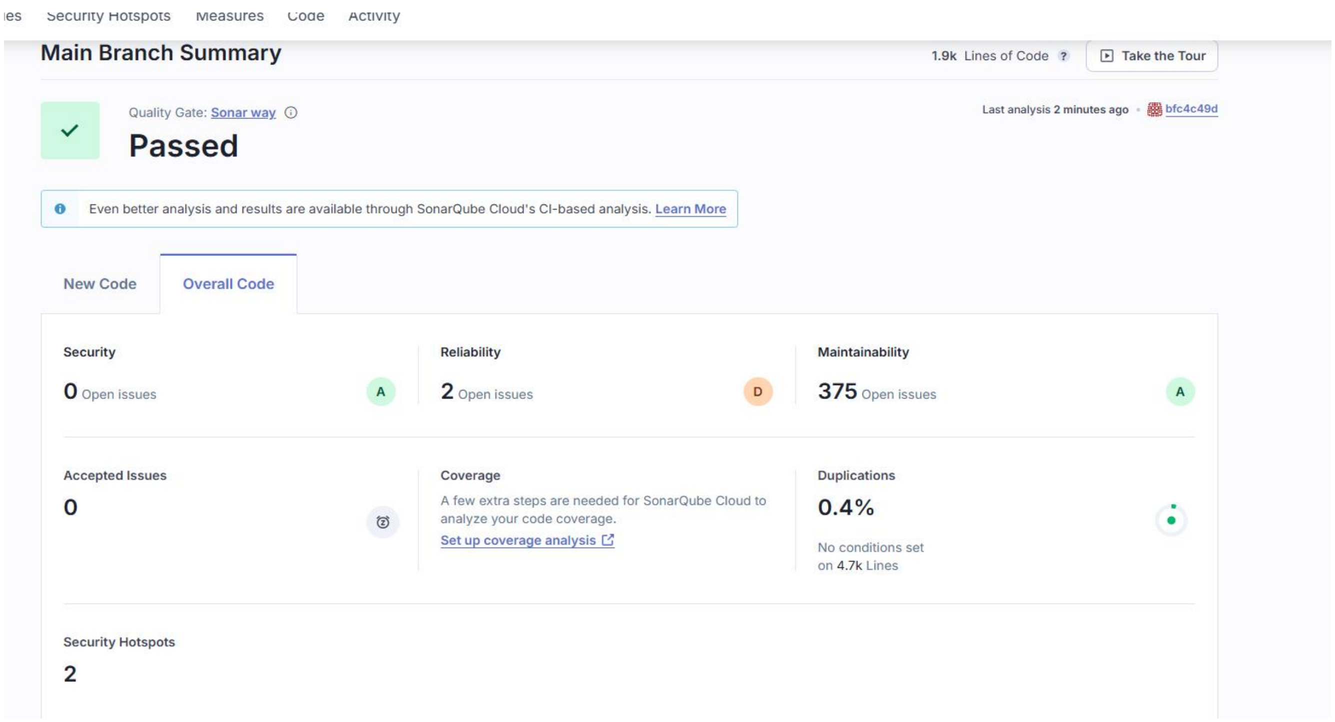
Software Project Report

- Quality Gate & Coverage Before Refactoring:

  1. **Quality Gate:**



ues Security Hotspots Measures Code Activity

Main Branch Summary

1.9k Lines of Code ? Take the Tour

Quality Gate: Sonar way

Passed

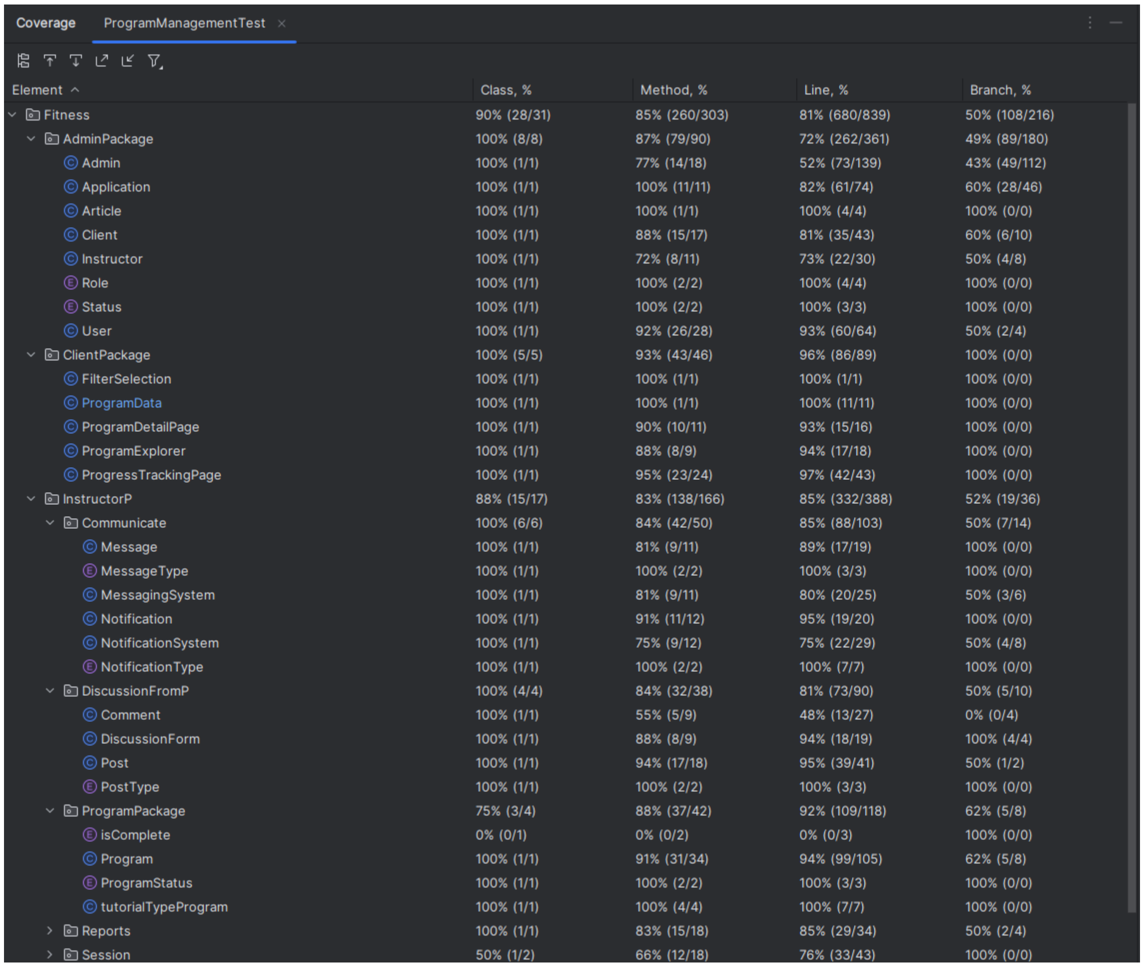
Last analysis 2 minutes ago • bfc4c49d

Even better analysis and results are available through SonarQube Cloud's CI-based analysis. Learn More

New Code | Overall Code

Security 0 Open issues — A	Reliability 2 Open issues — D	Maintainability 375 Open issues — A
Accepted Issues 0	Coverage A few extra steps are needed for SonarQube Cloud to analyze your code coverage. Set up coverage analysis	Duplications 0.4% No conditions set on 4.7k Lines
Security Hotspots 2		

  2. **Coverage:**



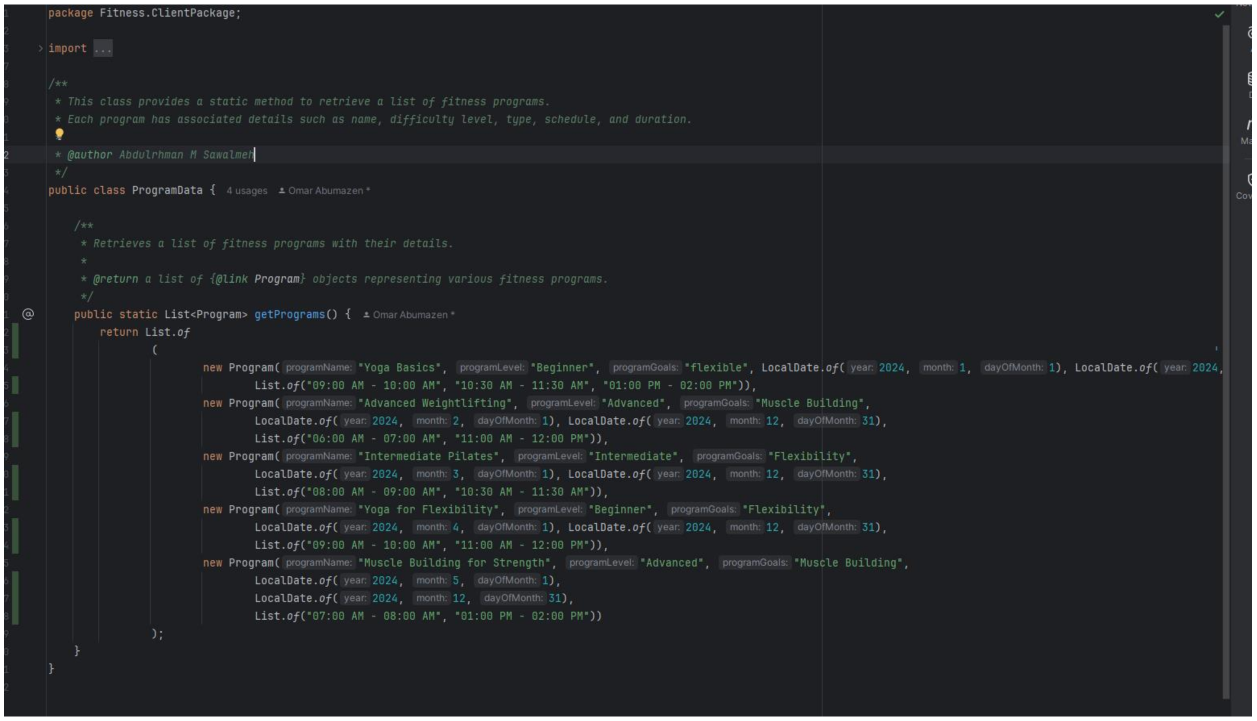
Coverage ProgramManagementTest

Element	Class, %	Method, %	Line, %	Branch, %
Fitness	90% (28/31)	85% (260/303)	81% (680/839)	50% (108/216)
AdminPackage	100% (8/8)	87% (79/90)	72% (262/361)	49% (89/180)
Admin	100% (1/1)	77% (14/18)	52% (73/139)	43% (49/112)
Application	100% (1/1)	100% (11/11)	82% (61/74)	60% (28/46)
Article	100% (1/1)	100% (1/1)	100% (4/4)	100% (0/0)
Client	100% (1/1)	88% (15/17)	81% (35/43)	60% (6/10)
Instructor	100% (1/1)	72% (8/11)	73% (22/30)	50% (4/8)
Role	100% (1/1)	100% (2/2)	100% (4/4)	100% (0/0)
Status	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
User	100% (1/1)	92% (26/28)	93% (60/64)	50% (2/4)
ClientPackage	100% (5/5)	93% (43/46)	96% (86/89)	100% (0/0)
FilterSelection	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
ProgramData	100% (1/1)	100% (1/1)	100% (11/11)	100% (0/0)
ProgramDetailPage	100% (1/1)	90% (10/11)	93% (15/16)	100% (0/0)
ProgramExplorer	100% (1/1)	88% (8/9)	94% (17/18)	100% (0/0)
ProgressTrackingPage	100% (1/1)	95% (23/24)	97% (42/43)	100% (0/0)
InstructorP	88% (15/17)	83% (138/166)	85% (332/388)	52% (19/36)
Communicate	100% (6/6)	84% (42/50)	85% (88/103)	50% (7/14)
Message	100% (1/1)	81% (9/11)	89% (17/19)	100% (0/0)
MessageType	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
MessagingSystem	100% (1/1)	81% (9/11)	80% (20/25)	50% (3/6)
Notification	100% (1/1)	91% (11/12)	95% (19/20)	100% (0/0)
NotificationSystem	100% (1/1)	75% (9/12)	75% (22/29)	50% (4/8)
NotificationType	100% (1/1)	100% (2/2)	100% (7/7)	100% (0/0)
DiscussionFromP	100% (4/4)	84% (32/38)	81% (73/90)	50% (5/10)
Comment	100% (1/1)	55% (5/9)	48% (13/27)	0% (0/4)
DiscussionForm	100% (1/1)	88% (8/9)	94% (18/19)	100% (4/4)
Post	100% (1/1)	94% (17/18)	95% (39/41)	50% (1/2)
PostType	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
ProgramPackage	75% (3/4)	88% (37/42)	92% (109/118)	62% (5/8)
isComplete	0% (0/1)	0% (0/2)	0% (0/3)	100% (0/0)
Program	100% (1/1)	91% (31/34)	94% (99/105)	62% (5/8)
ProgramStatus	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
tutorialTypeProgram	100% (1/1)	100% (4/4)	100% (7/7)	100% (0/0)
Reports	100% (1/1)	83% (15/18)	85% (29/34)	50% (2/4)
Session	50% (1/2)	66% (12/18)	76% (33/43)	100% (0/0)

- Refactoring 4 bad smell.

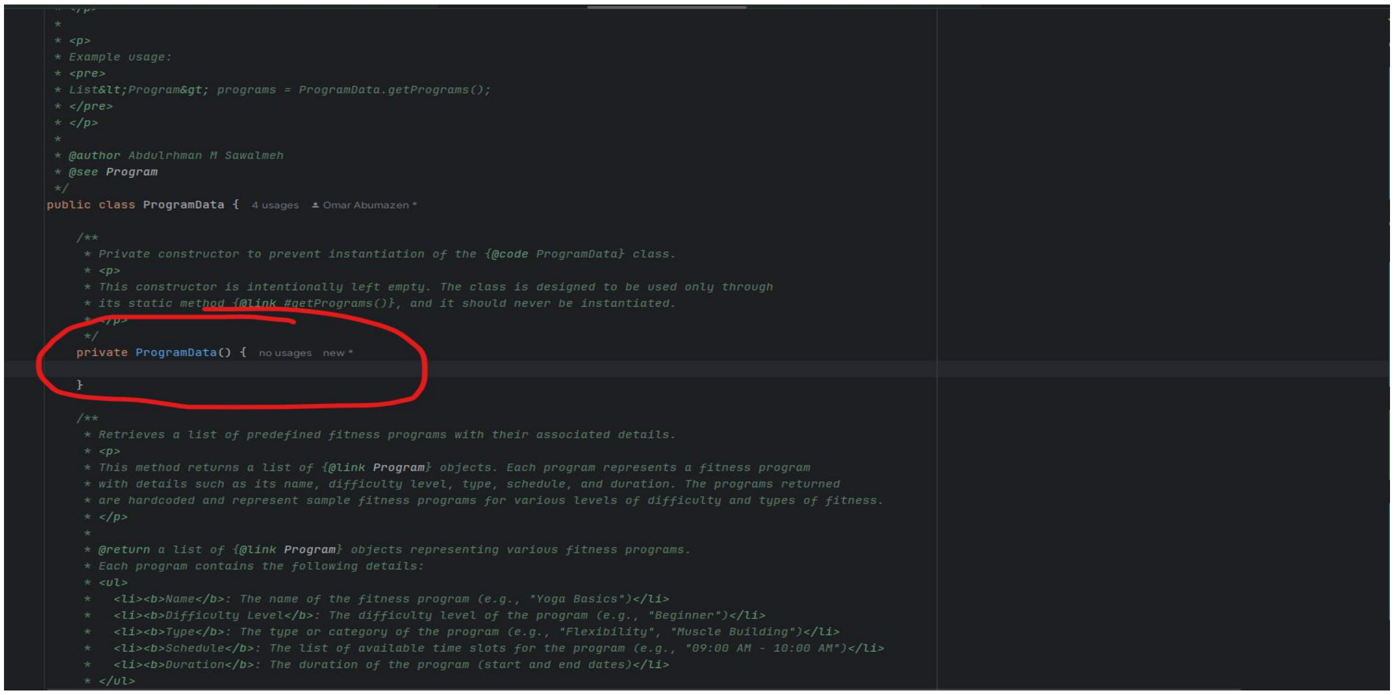
1. In *ClientPackage.ProgramData* Class, we add a private Constructor to the class.

- Before Refactoring:



```
package Fitness.ClientPackage;

import ...

/**
 * This class provides a static method to retrieve a list of fitness programs.
 * Each program has associated details such as name, difficulty level, type, schedule, and duration.
 *
 * @author Abdulrhman M Sawalmeh
 */
public class ProgramData {  4 usages  Omar Abumazen *

    /**
     * Retrieves a list of fitness programs with their details.
     *
     * @return a list of {@link Program} objects representing various fitness programs.
     */
    public static List<Program> getPrograms() {  Omar Abumazen *
        return List.of
                (
                        new Program( programName: "Yoga Basics",  programLevel: "Beginner",  programGoals: "flexible", LocalDate.of( year: 2024,  month: 1,  dayOfMonth: 1), LocalDate.of( year: 2024,
                                List.of("09:00 AM - 10:00 AM", "10:30 AM - 11:30 AM", "01:00 PM - 02:00 PM")),
                        new Program( programName: "Advanced Weightlifting",  programLevel: "Advanced",  programGoals: "Muscle Building",
                                LocalDate.of( year: 2024,  month: 2,  dayOfMonth: 1), LocalDate.of( year: 2024,  month: 12,  dayOfMonth: 31),
                                List.of("06:00 AM - 07:00 AM", "11:00 AM - 12:00 PM")),
                        new Program( programName: "Intermediate Pilates",  programLevel: "Intermediate",  programGoals: "Flexibility",
                                LocalDate.of( year: 2024,  month: 3,  dayOfMonth: 1), LocalDate.of( year: 2024,  month: 12,  dayOfMonth: 31),
                                List.of("08:00 AM - 09:00 AM", "10:30 AM - 11:30 AM")),
                        new Program( programName: "Yoga for Flexibility",  programLevel: "Beginner",  programGoals: "Flexibility",
                                LocalDate.of( year: 2024,  month: 4,  dayOfMonth: 1), LocalDate.of( year: 2024,  month: 12,  dayOfMonth: 31),
                                List.of("09:00 AM - 10:00 AM", "11:00 AM - 12:00 PM")),
                        new Program( programName: "Muscle Building for Strength",  programLevel: "Advanced",  programGoals: "Muscle Building",
                                LocalDate.of( year: 2024,  month: 5,  dayOfMonth: 1),
                                LocalDate.of( year: 2024,  month: 12,  dayOfMonth: 31),
                                List.of("07:00 AM - 08:00 AM", "01:00 PM - 02:00 PM"))
                );
    }
}
```

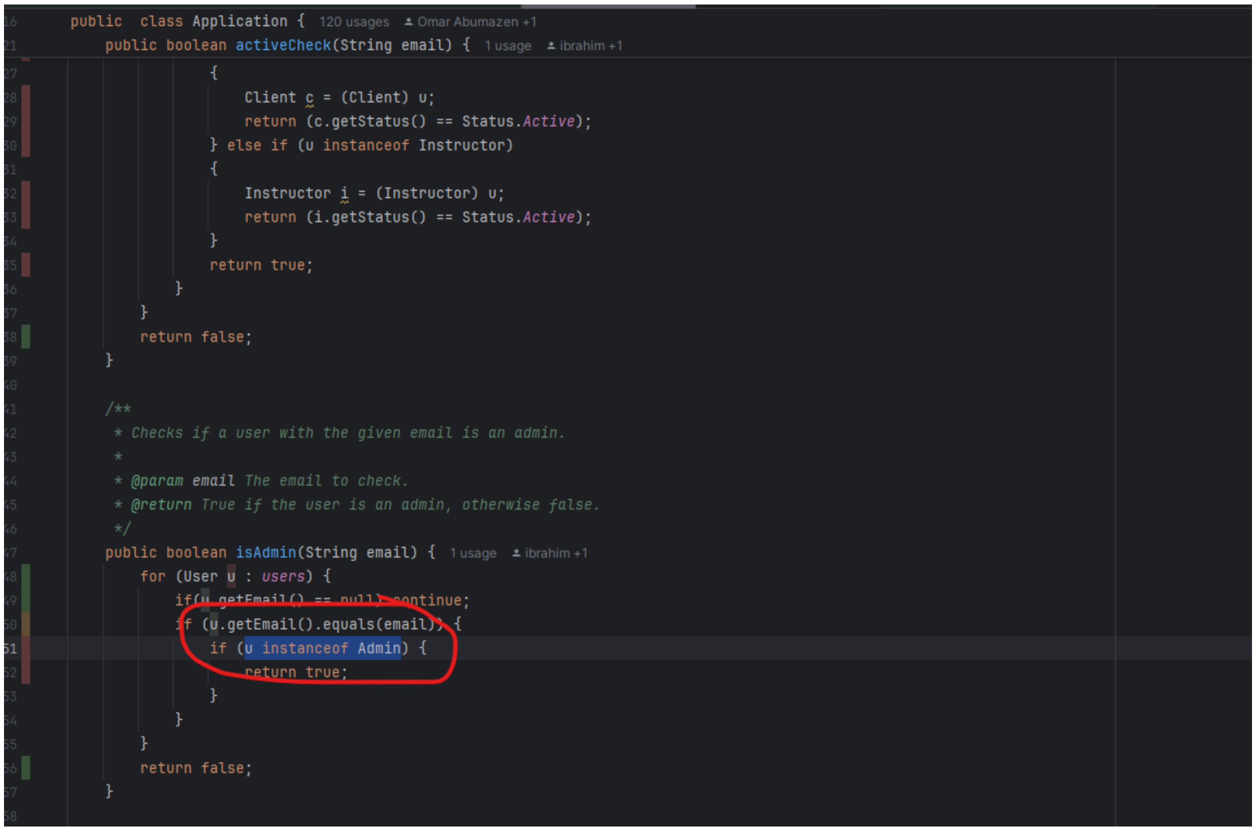
- After Refactoring:



```
 *
 * <p>
 * Example usage:
 * <pre>
 * List&lt;Program&gt; programs = ProgramData.getPrograms();
 * </pre>
 * </p>
 *
 * @author Abdulrhman M Sawalmeh
 * @see Program
 */
public class ProgramData {  4 usages  Omar Abumazen *

    /**
     * Private constructor to prevent instantiation of the {@code ProgramData} class.
     * <p>
     * This constructor is intentionally left empty. The class is designed to be used only through
     * its static method {@link #getPrograms()}, and it should never be instantiated.
     * </p>
     */
    private ProgramData() {  no usages  new *

    }

    /**
     * Retrieves a list of predefined fitness programs with their associated details.
     * <p>
     * This method returns a list of {@link Program} objects. Each program represents a fitness program
     * with details such as its name, difficulty level, type, schedule, and duration. The programs returned
     * are hardcoded and represent sample fitness programs for various levels of difficulty and types of fitness.
     * </p>
     *
     * @return a list of {@link Program} objects representing various fitness programs.
     * Each program contains the following details:
     * <ul>
     *   <li><b>Name</b>: The name of the fitness program (e.g., "Yoga Basics")</li>
     *   <li><b>Difficulty Level</b>: The difficulty level of the program (e.g., "Beginner")</li>
     *   <li><b>Type</b>: The type or category of the program (e.g., "Flexibility", "Muscle Building")</li>
     *   <li><b>Schedule</b>: The list of available time slots for the program (e.g., "09:00 AM - 10:00 AM")</li>
     *   <li><b>Duration</b>: The duration of the program (start and end dates)</li>
     * </ul>
```

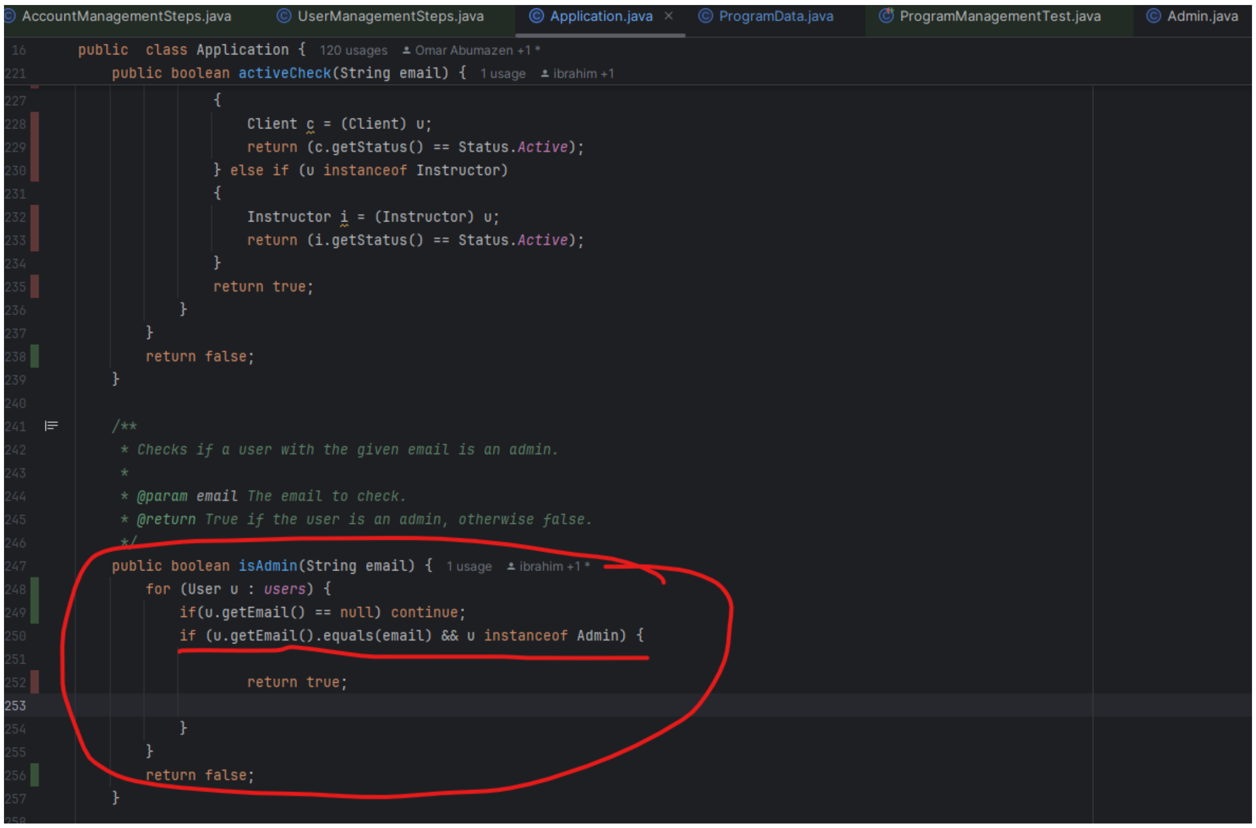
2. In *AdminPackage.Application* Class, we merge an if statement:

- Before Refactoring:



```
public  class Application {  120 usages  Omar Abumazen +1
    public boolean activeCheck(String email) {  1 usage  ibrahim +1
                {
                    Client c = (Client) u;
                    return (c.getStatus() == Status.Active);
                } else if (u instanceof Instructor)
                {
                    Instructor i = (Instructor) u;
                    return (i.getStatus() == Status.Active);
                }
                return true;
            }
        }
        return false;
    }

    /**
     * Checks if a user with the given email is an admin.
     *
     * @param email The email to check.
     * @return True if the user is an admin, otherwise false.
     */
    public boolean isAdmin(String email) {  1 usage  ibrahim +1
        for (User u : users) {
            if(u.getEmail() == null) continue;
            if (u.getEmail().equals(email)) {
                if (u instanceof Admin) {
                    return true;
                }
            }
        }
        return false;
    }
```

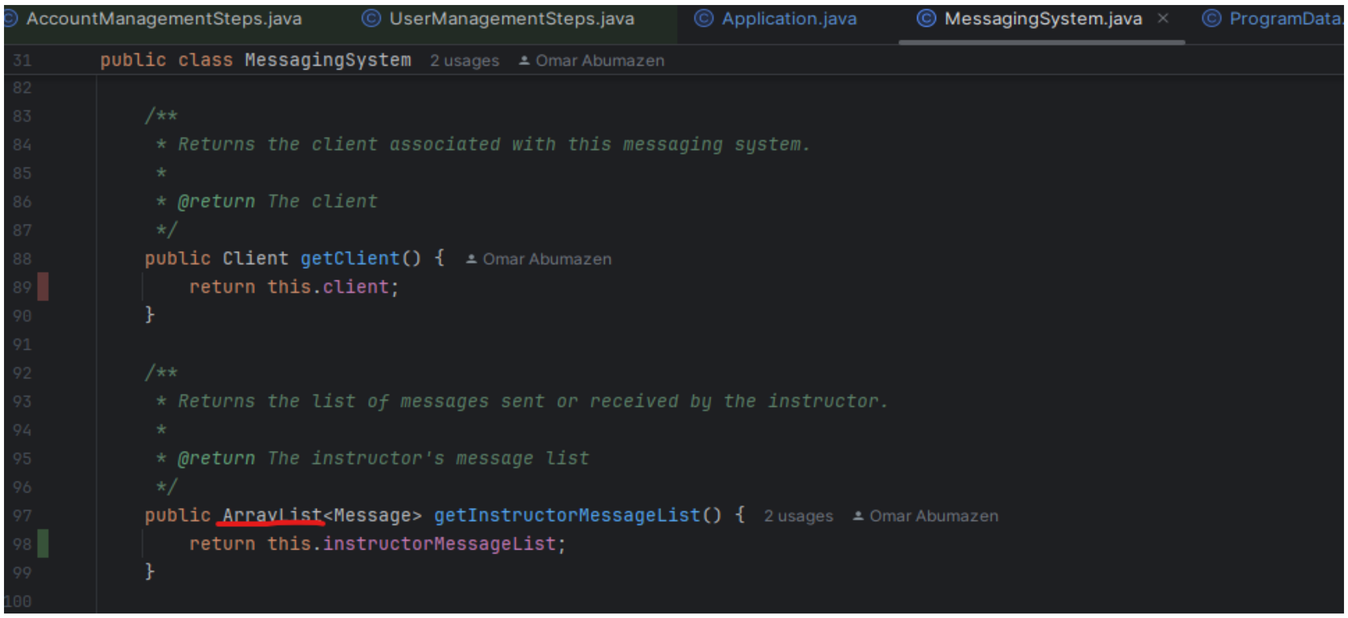
- After Refactoring:



```
AccountManagementSteps.java   UserManagementSteps.java   Application.java ×   ProgramData.java   ProgramManagementTest.java   Admin.java

public  class Application {  120 usages  Omar Abumazen +1 *
    public boolean activeCheck(String email) {  1 usage  ibrahim +1
                {
                    Client c = (Client) u;
                    return (c.getStatus() == Status.Active);
                } else if (u instanceof Instructor)
                {
                    Instructor i = (Instructor) u;
                    return (i.getStatus() == Status.Active);
                }
                return true;
            }
        }
        return false;
    }

    /**
     * Checks if a user with the given email is an admin.
     *
     * @param email The email to check.
     * @return True if the user is an admin, otherwise false.
     */
    public boolean isAdmin(String email) {  1 usage  ibrahim +1 *
        for (User u : users) {
            if(u.getEmail() == null) continue;
            if (u.getEmail().equals(email) && u instanceof Admin) {

                    return true;

            }
        }
        return false;
    }
```

3.In *InstructorP.Communicate.MessagingSystem* Class we change the type of the return type of a getter method.

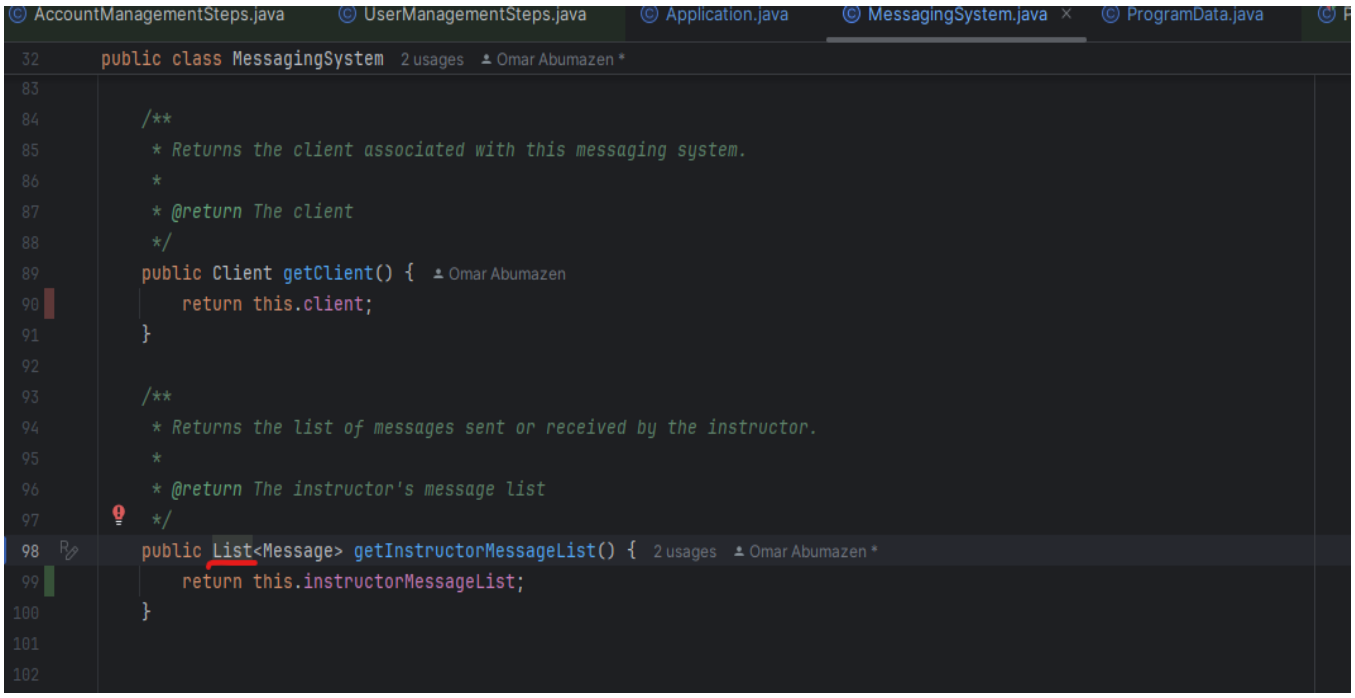
- Before Refactoring:



AccountManagementSteps.java | UserManagementSteps.java | Application.java | MessagingSystem.java | ProgramData.

```java
public class MessagingSystem  2 usages  Omar Abumazen

    /**
     * Returns the client associated with this messaging system.
     *
     * @return The client
     */
    public Client getClient() {  Omar Abumazen
        return this.client;
    }

    /**
     * Returns the list of messages sent or received by the instructor.
     *
     * @return The instructor's message list
     */
    public ArrayList<Message> getInstructorMessageList() {  2 usages  Omar Abumazen
        return this.instructorMessageList;
    }
```

- After Refactoring:

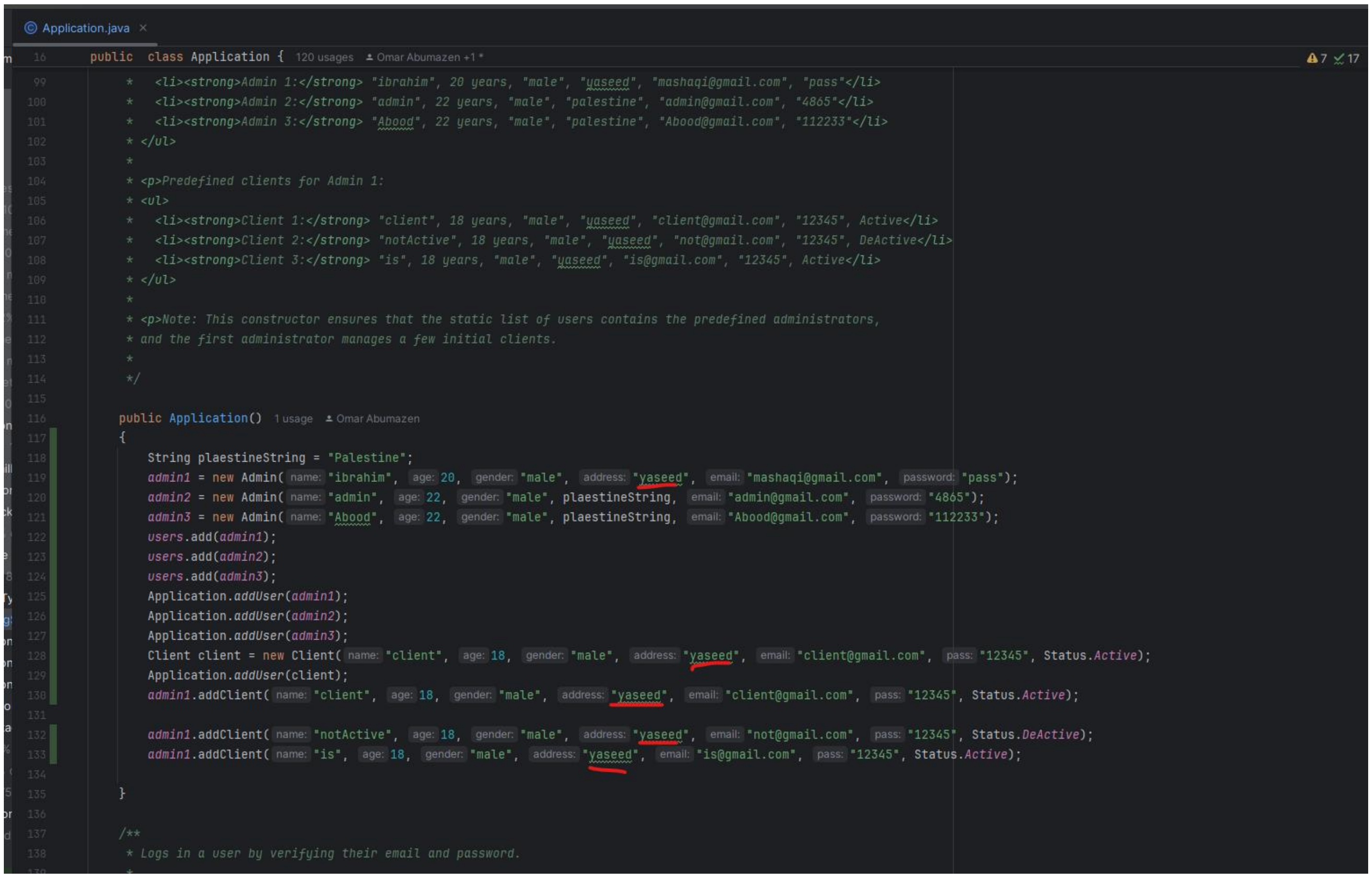


AccountManagementSteps.java | UserManagementSteps.java | Application.java | MessagingSystem.java | ProgramData.java

```java
public class MessagingSystem  2 usages  Omar Abumazen *

    /**
     * Returns the client associated with this messaging system.
     *
     * @return The client
     */
    public Client getClient() {  Omar Abumazen
        return this.client;
    }

    /**
     * Returns the list of messages sent or received by the instructor.
     *
     * @return The instructor's message list
     */
    public List<Message> getInstructorMessageList() {  2 usages  Omar Abumazen *
        return this.instructorMessageList;
    }
```

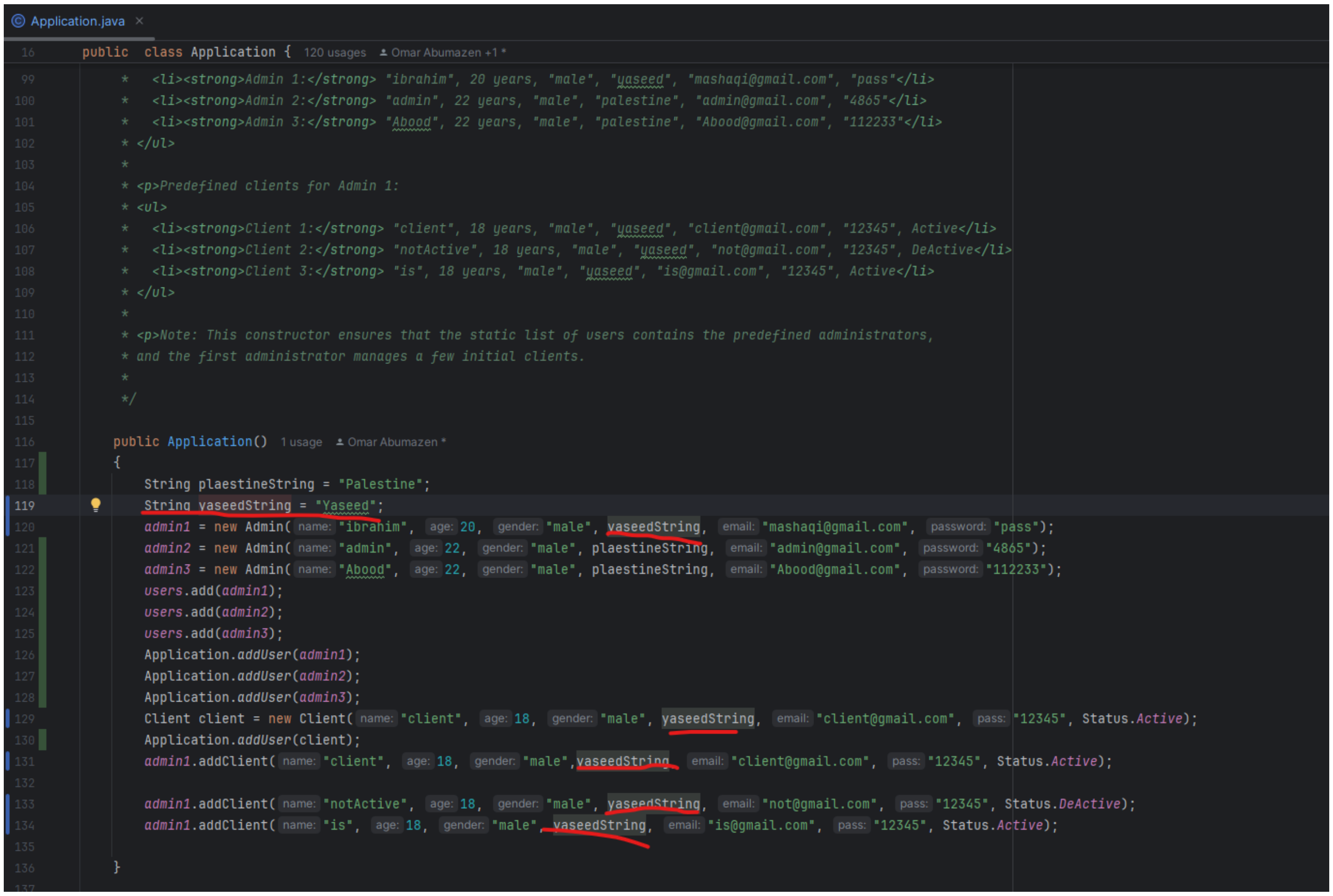
4. In *AdminPackage.Application* Class, we change the duplicate string values:

- Before Refactoring:


Application.java

```java
public  class Application {  120 usages  Omar Abumazen +1 *
     *   <li><strong>Admin 1:</strong> "ibrahim", 20 years, "male", "yaseed", "mashaqi@gmail.com", "pass"</li>
     *   <li><strong>Admin 2:</strong> "admin", 22 years, "male", "palestine", "admin@gmail.com", "4865"</li>
     *   <li><strong>Admin 3:</strong> "Abood", 22 years, "male", "palestine", "Abood@gmail.com", "112233"</li>
     * </ul>
     *
     * <p>Predefined clients for Admin 1:
     * <ul>
     *   <li><strong>Client 1:</strong> "client", 18 years, "male", "yaseed", "client@gmail.com", "12345", Active</li>
     *   <li><strong>Client 2:</strong> "notActive", 18 years, "male", "yaseed", "not@gmail.com", "12345", DeActive</li>
     *   <li><strong>Client 3:</strong> "is", 18 years, "male", "yaseed", "is@gmail.com", "12345", Active</li>
     * </ul>
     *
     * <p>Note: This constructor ensures that the static list of users contains the predefined administrators,
     * and the first administrator manages a few initial clients.
     *
     */

    public Application()  1 usage  Omar Abumazen
    {
        String plaestineString = "Palestine";
        admin1 = new Admin( name: "ibrahim",  age: 20,  gender: "male",  address: "yaseed",  email: "mashaqi@gmail.com",  password: "pass");
        admin2 = new Admin( name: "admin",  age: 22,  gender: "male", plaestineString,  email: "admin@gmail.com",  password: "4865");
        admin3 = new Admin( name: "Abood",  age: 22,  gender: "male", plaestineString,  email: "Abood@gmail.com",  password: "112233");
        users.add(admin1);
        users.add(admin2);
        users.add(admin3);
        Application.addUser(admin1);
        Application.addUser(admin2);
        Application.addUser(admin3);
        Client client = new Client( name: "client",  age: 18,  gender: "male",  address: "yaseed",  email: "client@gmail.com",  pass: "12345", Status.Active);
        Application.addUser(client);
        admin1.addClient( name: "client",  age: 18,  gender: "male",  address: "yaseed",  email: "client@gmail.com",  pass: "12345", Status.Active);

        admin1.addClient( name: "notActive",  age: 18,  gender: "male",  address: "yaseed",  email: "not@gmail.com",  pass: "12345", Status.DeActive);
        admin1.addClient( name: "is",  age: 18,  gender: "male",  address: "yaseed",  email: "is@gmail.com",  pass: "12345", Status.Active);

    }

    /**
     * Logs in a user by verifying their email and password.
```

- After Refactoring:

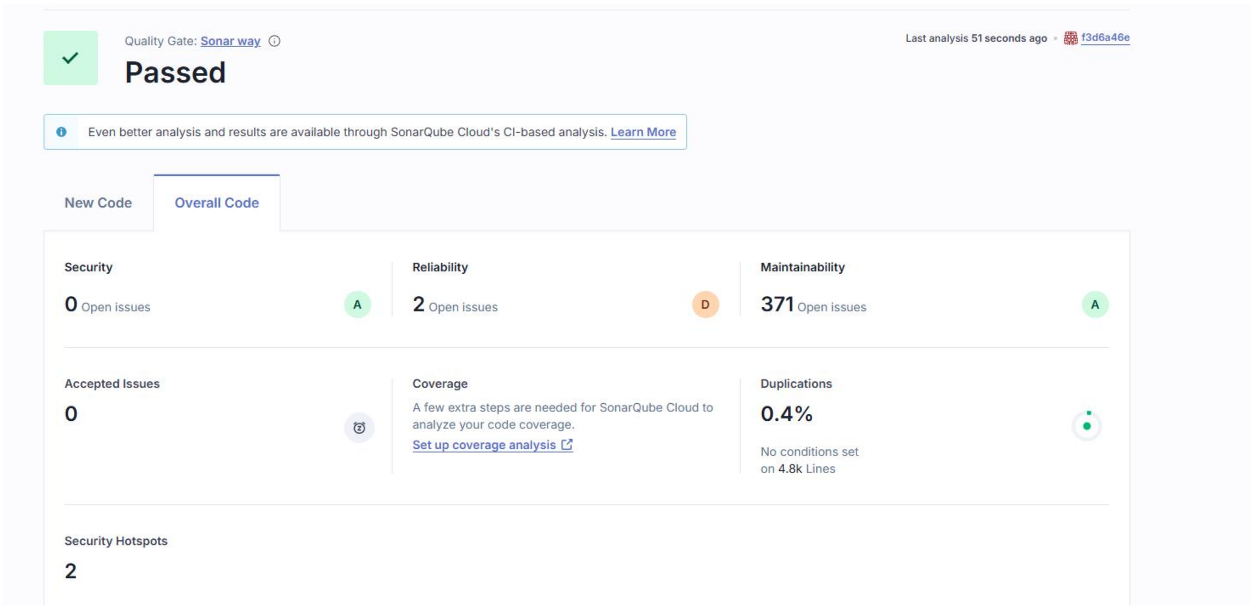


Application.java ×

```java
public  class Application {  120 usages  Omar Abumazen +1 *
        *   <li><strong>Admin 1:</strong> "ibrahim", 20 years, "male", "yaseed", "mashaqi@gmail.com", "pass"</li>
        *   <li><strong>Admin 2:</strong> "admin", 22 years, "male", "palestine", "admin@gmail.com", "4865"</li>
        *   <li><strong>Admin 3:</strong> "Abood", 22 years, "male", "palestine", "Abood@gmail.com", "112233"</li>
        * </ul>
        *
        * <p>Predefined clients for Admin 1:
        * <ul>
        *   <li><strong>Client 1:</strong> "client", 18 years, "male", "yaseed", "client@gmail.com", "12345", Active</li>
        *   <li><strong>Client 2:</strong> "notActive", 18 years, "male", "yaseed", "not@gmail.com", "12345", DeActive</li>
        *   <li><strong>Client 3:</strong> "is", 18 years, "male", "yaseed", "is@gmail.com", "12345", Active</li>
        * </ul>
        *
        * <p>Note: This constructor ensures that the static list of users contains the predefined administrators,
        * and the first administrator manages a few initial clients.
        *
        */

    public Application()  1 usage  Omar Abumazen *
    {
        String plaestineString = "Palestine";
        String yaseedString = "Yaseed";
        admin1 = new Admin( name: "ibrahim",  age: 20,  gender: "male", yaseedString,  email: "mashaqi@gmail.com",  password: "pass");
        admin2 = new Admin( name: "admin",  age: 22,  gender: "male", plaestineString,  email: "admin@gmail.com",  password: "4865");
        admin3 = new Admin( name: "Abood",  age: 22,  gender: "male", plaestineString,  email: "Abood@gmail.com",  password: "112233");
        users.add(admin1);
        users.add(admin2);
        users.add(admin3);
        Application.addUser(admin1);
        Application.addUser(admin2);
        Application.addUser(admin3);
        Client client = new Client( name: "client",  age: 18,  gender: "male", yaseedString,  email: "client@gmail.com",  pass: "12345", Status.Active);
        Application.addUser(client);
        admin1.addClient( name: "client",  age: 18,  gender: "male",yaseedString,  email: "client@gmail.com",  pass: "12345", Status.Active);

        admin1.addClient( name: "notActive",  age: 18,  gender: "male", yaseedString,  email: "not@gmail.com",  pass: "12345", Status.DeActive);
        admin1.addClient( name: "is",  age: 18,  gender: "male", yaseedString,  email: "is@gmail.com",  pass: "12345", Status.Active);

    }
```

- **Quality Gate & Coverage After Refactoring:**

1. Quality Gate:



Quality Gate: Sonar way

Passed

Last analysis 51 seconds ago · f3d6a46e

Even better analysis and results are available through SonarQube Cloud's CI-based analysis. Learn More

New Code | Overall Code

Security
0 Open issues — A

Reliability
2 Open issues — D

Maintainability
371 Open issues — A

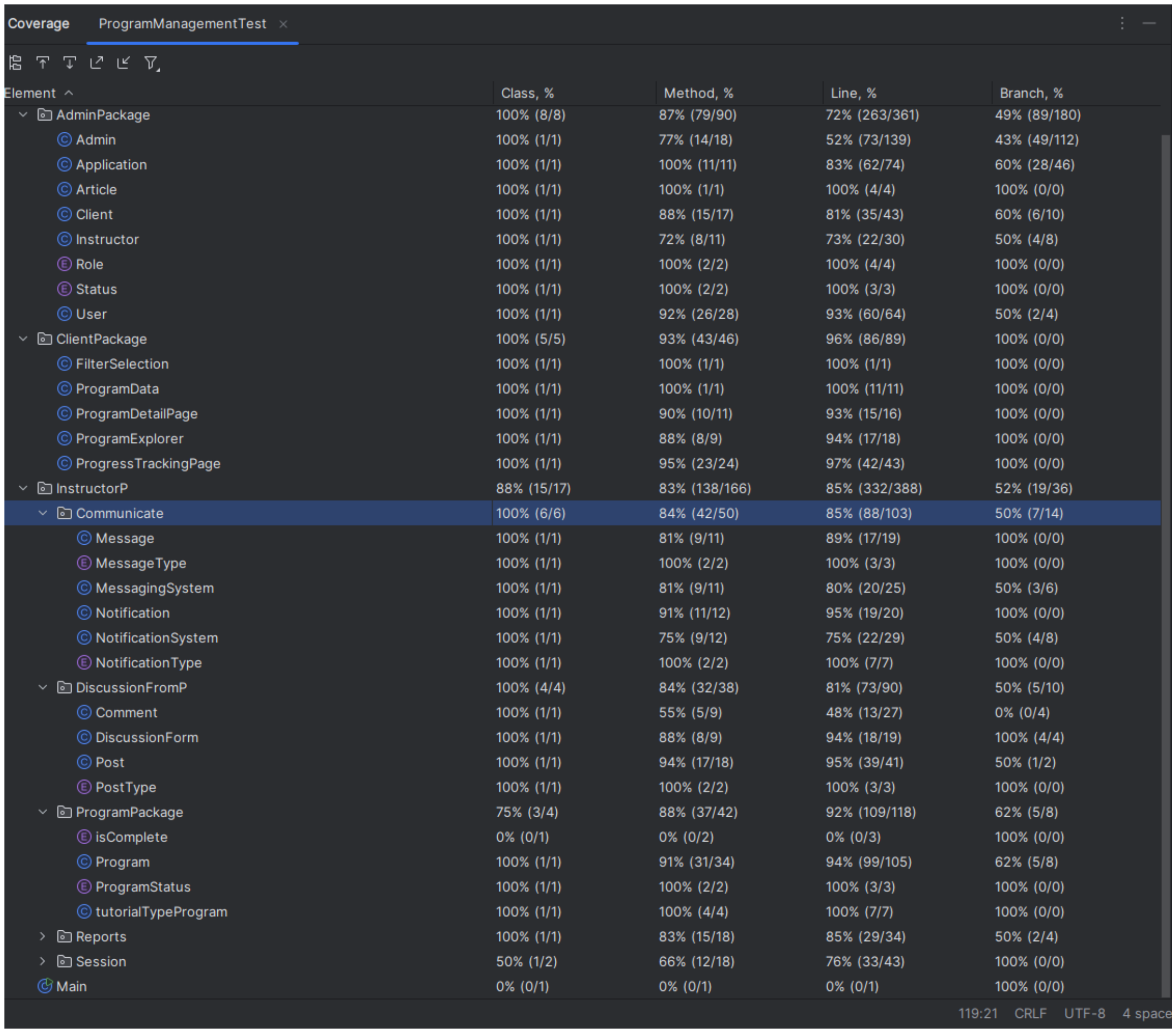
Accepted Issues
0

Coverage
A few extra steps are needed for SonarQube Cloud to analyze your code coverage.
Set up coverage analysis

Duplications
0.4%
No conditions set on 4.8k Lines

Security Hotspots
2

2. Coverage:

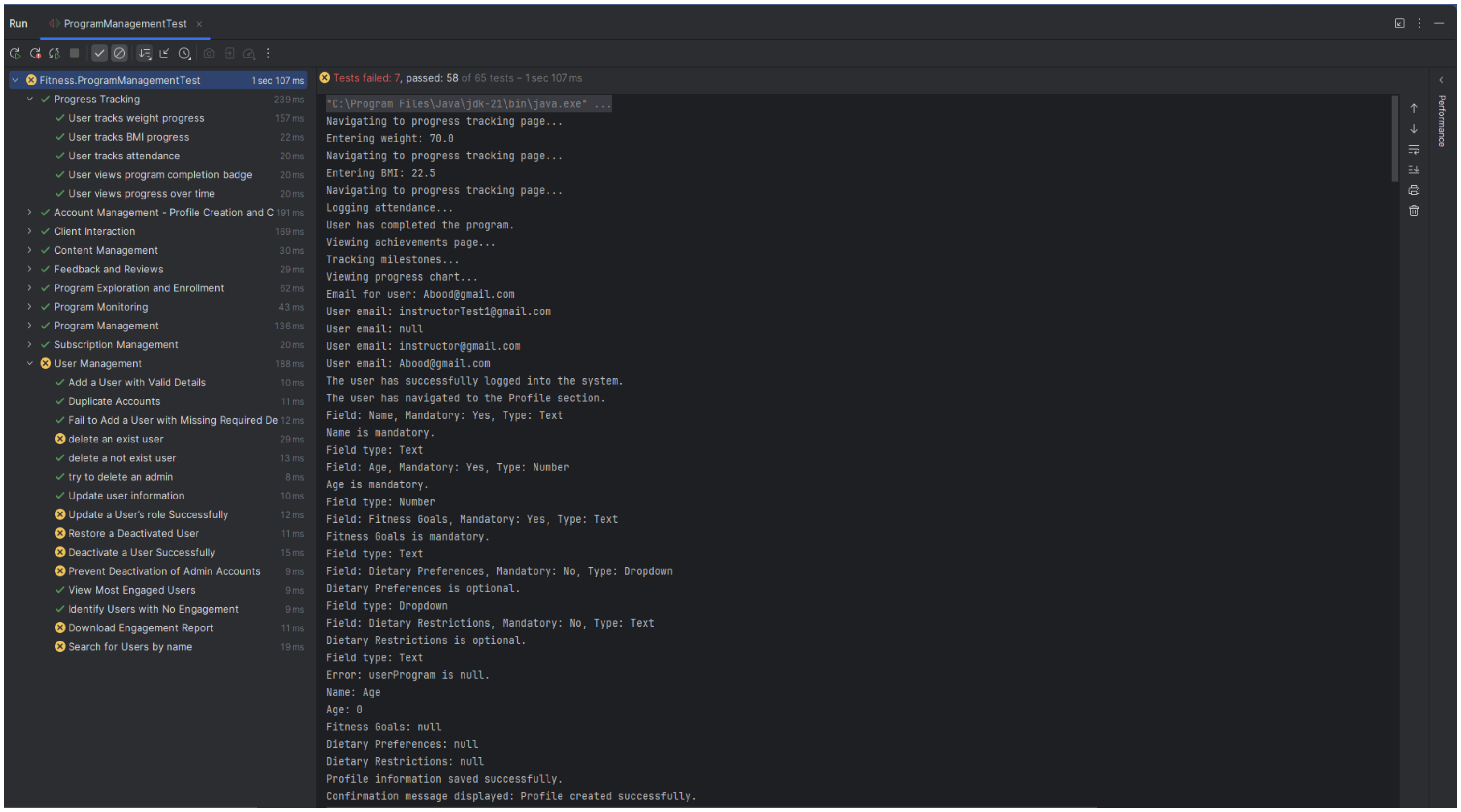


Coverage | ProgramManagementTest

Element	Class, %	Method, %	Line, %	Branch, %
AdminPackage	100% (8/8)	87% (79/90)	72% (263/361)	49% (89/180)
Admin	100% (1/1)	77% (14/18)	52% (73/139)	43% (49/112)
Application	100% (1/1)	100% (11/11)	83% (62/74)	60% (28/46)
Article	100% (1/1)	100% (1/1)	100% (4/4)	100% (0/0)
Client	100% (1/1)	88% (15/17)	81% (35/43)	60% (6/10)
Instructor	100% (1/1)	72% (8/11)	73% (22/30)	50% (4/8)
Role	100% (1/1)	100% (2/2)	100% (4/4)	100% (0/0)
Status	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
User	100% (1/1)	92% (26/28)	93% (60/64)	50% (2/4)
ClientPackage	100% (5/5)	93% (43/46)	96% (86/89)	100% (0/0)
FilterSelection	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
ProgramData	100% (1/1)	100% (1/1)	100% (11/11)	100% (0/0)
ProgramDetailPage	100% (1/1)	90% (10/11)	93% (15/16)	100% (0/0)
ProgramExplorer	100% (1/1)	88% (8/9)	94% (17/18)	100% (0/0)
ProgressTrackingPage	100% (1/1)	95% (23/24)	97% (42/43)	100% (0/0)
InstructorP	88% (15/17)	83% (138/166)	85% (332/388)	52% (19/36)
Communicate	100% (6/6)	84% (42/50)	85% (88/103)	50% (7/14)
Message	100% (1/1)	81% (9/11)	89% (17/19)	100% (0/0)
MessageType	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
MessagingSystem	100% (1/1)	81% (9/11)	80% (20/25)	50% (3/6)
Notification	100% (1/1)	91% (11/12)	95% (19/20)	100% (0/0)
NotificationSystem	100% (1/1)	75% (9/12)	75% (22/29)	50% (4/8)
NotificationType	100% (1/1)	100% (2/2)	100% (7/7)	100% (0/0)
DiscussionFromP	100% (4/4)	84% (32/38)	81% (73/90)	50% (5/10)
Comment	100% (1/1)	55% (5/9)	48% (13/27)	0% (0/4)
DiscussionForm	100% (1/1)	88% (8/9)	94% (18/19)	100% (4/4)
Post	100% (1/1)	94% (17/18)	95% (39/41)	50% (1/2)
PostType	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
ProgramPackage	75% (3/4)	88% (37/42)	92% (109/118)	62% (5/8)
isComplete	0% (0/1)	0% (0/2)	0% (0/3)	100% (0/0)
Program	100% (1/1)	91% (31/34)	94% (99/105)	62% (5/8)
ProgramStatus	100% (1/1)	100% (2/2)	100% (3/3)	100% (0/0)
tutorialTypeProgram	100% (1/1)	100% (4/4)	100% (7/7)	100% (0/0)
Reports	100% (1/1)	83% (15/18)	85% (29/34)	50% (2/4)
Session	50% (1/2)	66% (12/18)	76% (33/43)	100% (0/0)
Main	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)

119:21 CRLF UTF-8 4 space

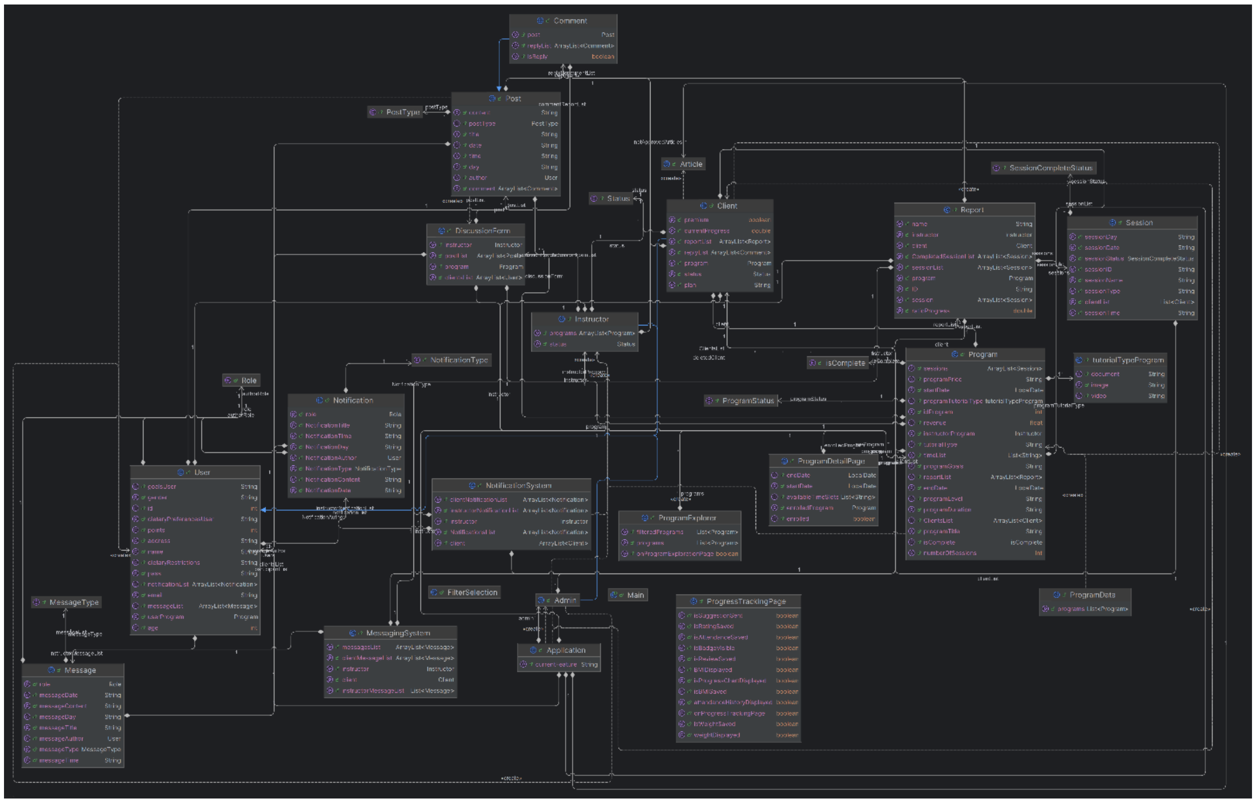
- Last Result of Building Steps:



```
Fitness.ProgramManagementTest                1 sec 107 ms
  Progress Tracking                          239 ms
    User tracks weight progress              157 ms
    User tracks BMI progress                  22 ms
    User tracks attendance                    20 ms
    User views program completion badge       20 ms
    User views progress over time             20 ms
  Account Management - Profile Creation and C 191 ms
  Client Interaction                         169 ms
  Content Management                          30 ms
  Feedback and Reviews                        29 ms
  Program Exploration and Enrollment          62 ms
  Program Monitoring                          43 ms
  Program Management                         136 ms
  Subscription Management                     20 ms
  User Management                            188 ms
    Add a User with Valid Details             10 ms
    Duplicate Accounts                        11 ms
    Fail to Add a User with Missing Required De 12 ms
    delete an exist user                      29 ms
    delete a not exist user                   13 ms
    try to delete an admin                     8 ms
    Update user information                   10 ms
    Update a User's role Successfully         12 ms
    Restore a Deactivated User                11 ms
    Deactivate a User Successfully            15 ms
    Prevent Deactivation of Admin Accounts     9 ms
    View Most Engaged Users                    9 ms
    Identify Users with No Engagement          9 ms
    Download Engagement Report                11 ms
    Search for Users by name                  19 ms
```

```
Tests failed: 7, passed: 58 of 65 tests – 1 sec 107 ms
"C:\Program Files\Java\jdk-21\bin\java.exe" ...
Navigating to progress tracking page...
Entering weight: 70.0
Navigating to progress tracking page...
Entering BMI: 22.5
Navigating to progress tracking page...
Logging attendance...
User has completed the program.
Viewing achievements page...
Tracking milestones...
Viewing progress chart...
Email for user: Abood@gmail.com
User email: instructorTest1@gmail.com
User email: null
User email: instructor@gmail.com
User email: Abood@gmail.com
The user has successfully logged into the system.
The user has navigated to the Profile section.
Field: Name, Mandatory: Yes, Type: Text
Name is mandatory.
Field type: Text
Field: Age, Mandatory: Yes, Type: Number
Age is mandatory.
Field type: Number
Field: Fitness Goals, Mandatory: Yes, Type: Text
Fitness Goals is mandatory.
Field type: Text
Field: Dietary Preferences, Mandatory: No, Type: Dropdown
Dietary Preferences is optional.
Field type: Dropdown
Field: Dietary Restrictions, Mandatory: No, Type: Text
Dietary Restrictions is optional.
Field type: Text
Error: userProgram is null.
Name: Age
Age: 0
Fitness Goals: null
Dietary Preferences: null
Dietary Restrictions: null
Profile information saved successfully.
Confirmation message displayed: Profile created successfully.
```

- UML class Diagram (there's an image with project folder more clear):



Comment
Post
PostType
Article
SessionCompleteStatus
Status
Client
Report
Session
DiscussionForm
Instructor
NotificationType
isComplete
Program
TutorialTypeProgram
Role
Notification
ProgramStatus
User
NotificationSystem
ProgramDetailPage
ProgramExplorer
MessageType
FilterSelection
Admin
Main
ProgressTrackingPage
ProgramData
MessagingSystem
Application
Message