**Software Engineering Project**

**Section: B01**

**Date of Submission: 10th October 2021**

Group Members:

|  |  |
| --- | --- |
| **Name** | **QUID** |
| Talha Abdullah Punjabi | 201903446 |
| Ibrahim Demdoum | 201701515 |
| Abd Errahmane | 201803234 |
| Salem Rashid |  |

1. **Identify Constraints and group them into different categories; and recognize quality requirements (NFRs) of the system. You can propose more constraints and NFRs in addition to the requirements of the user**

|  |  |
| --- | --- |
| Constraint | Scenario |
| Business Constraints | * System must run as plug-in for the payment with the Google Pay system |
| Development Constraints | * The system must use structured development methods to ensure high portability |
| Schedule Constraints | * The deliverability is on 7th November |
| Technical Constraints | * New functionalities must be added to the system based on requirements * The system must be scalable * The existing system functionalities must be modifiable to enhance features * The system must not disclose the payment of information to an unauthorized entity * The system must be interoperable * The system must be loosely coupled with other components |
| Resource Constraints | * Budget for more than 8 new technical staff for project * 4 new servers are only available * Several backup modules must be managed to ensure availability of systems to members |
| Practical Constraints | * The privacy of the applicant must be respected * The proposed system should not have more than 1000 concurrent users |

|  |  |
| --- | --- |
| Quality Attribute | Architecture Requirement |
| Performance | * Application must provide fast response time for 97%+ of requests. |
| Portability | * it must be highly portable, meaning it can run on different operating systems such as Android,   Linux, iOS, Unix, etc. |
| Security | * The payment information of applicants is confidential; these should not be disclosed to unauthorized entities, they should not have direct exposure to any general user-level functions. * The data should be protected by separating them from high-level user interface. |
| Resource Management | * The system must run as plug-in for the payment with the Google Pay system |
| Usability | * If some functions were modified, the interfaces to the rest of the system may   remain same, so the user doesn’t have to  learn how to use it from the begging. |
| Availability | * Backup modules will be managed to support availability of the systems to the   users most of the time. |
| Reliability | * The system must give the same performance for a 2-year period of time under the load of 1000 concurrent visa application |
| Scalability | * The system can handle 1000 concurrent visa application and should handle more than 5000 applications within the next 2 years * The system could be distributed in future, different components may run on different machines in different locations (assumption) |
| Modifiability | * The system must support modifiability. The system should allow modification of   existing functionalities, adding new functionalities, or removing existing functionality. |
| High cohesion | * Various components of the software should be more focused |
| Low coupling | * Various components of the software should be less dependent on each other and more focused |

**2- (a) Propose a software architecture that addresses some or all of the technical constraints and quality properties identified in Task 1(b)**

**(b) Explain how your architecture achieves these properties. If any technical constraints and/or quality properties are not achieved in your architecture, explain why**

1. Based on the specified non-functional requirements and constraint, we chose layered architecture to help the system to counter constraints alongside giving more advantages, this will make our system take advantage of the layered architecture pros such us supporting modifiability, security, reusability, high cohesion, low coupling, availability, efficient, scalability, and portability, and those are all the NFRs our system needs, it also tackles some of the constraints the system faces.

The layered architecture divides into multiple horizontal layers. The most important layers of the system are business layer, presentation layer, service layer and the data layer. These layers focus on individual roles to accomplish certain tasks.

**The constraints the chosen architecture tackles and the quality attributes:**

1. **Business Constraints:** The business layer handles the server to run as a plug-in for the payment through Google pay system
2. **Development Constraints:** Layered structure allows the presentation layer to run the system on multiple operating systems
3. **Technical Constraints:**
4. It allows the system to be more scalable
5. Separation of concerns increases modifiability
6. It allows the system to support high performance
7. It allows low coupling to the system
8. **Practical Constraints:**  
   a. It allows security for the applicant of the system

b. It can expand the system storage to accommodate more concurrent users

1. **The chosen architecture achieves:**

* **High Cohesion:** Each layer will perform a specific task. Layered provide a high degree of cohesion.
* **Availability:** Servers in each layer can easy be replicated, so if one of them is going to fail, the server will respond with a different path or layer.
* **Portability:** The software can be run it in several environments such as Linux, windows etc.
* **Modifiability:** Since each layer is coupled to each other, so it’s easier to replace and makes it easier to reuse.
* **Low Coupling:** Each layer can use only the layers which is lower or higher than it.
* **Security:** The data layer isolates the infrastructure which in turn, protects the user data.

**Constraints not Countered by the Layered Approach:**

1. **Schedule Constraints**: It cannot deal with the users based on a limited schedule time. The running of the System in the layered approach will be consistent.
2. **Resource Constraints**: The layered approach is an architecture that deal with the system but cannot deal with outside resources which are working on the system.

**3**- **Implement one use case (available in Appendix B) using any programming language and provide screen shots demonstrating that your program works along with some sample output of the programs. Use nice user interface to give your program a professional look**

**Source Codes:**

**Tourist Class:**

**package** MainProject;

**import** java.io.Serializable;

**public** **class** Tourist **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 6894463765965627542L;

**private** **int** touristId,touristNumber,touristPassportNo,cardNumber;

**private** String touristName,touristAddress,touristEmail;

**public** Tourist(**int** touristId, **int** touristNumber, **int** touristPassportNo, **int** cardNumber, String touristName,

String touristAddress, String touristEmail) {

**super**();

**this**.touristId = touristId;

**this**.touristNumber = touristNumber;

**this**.touristPassportNo = touristPassportNo;

**this**.cardNumber = cardNumber;

**this**.touristName = touristName;

**this**.touristAddress = touristAddress;

**this**.touristEmail = touristEmail;

}

**public** **int** getTouristId() {

**return** touristId;

}

**public** **void** setTouristId(**int** touristId) {

**this**.touristId = touristId;

}

**public** **int** getTouristNumber() {

**return** touristNumber;

}

**public** **void** setTouristNumber(**int** touristNumber) {

**this**.touristNumber = touristNumber;

}

**public** **int** getTouristPassportNo() {

**return** touristPassportNo;

}

**public** **void** setTouristPassportNo(**int** touristPassportNo) {

**this**.touristPassportNo = touristPassportNo;

}

**public** **int** getCardNumber() {

**return** cardNumber;

}

**public** **void** setCardNumber(**int** cardNumber) {

**this**.cardNumber = cardNumber;

}

**public** String getTouristName() {

**return** touristName;

}

**public** **void** setTouristName(String touristName) {

**this**.touristName = touristName;

}

**public** String getTouristAddress() {

**return** touristAddress;

}

**public** **void** setTouristAddress(String touristAddress) {

**this**.touristAddress = touristAddress;

}

**public** String getTouristEmail() {

**return** touristEmail;

}

**public** **void** setTouristEmail(String touristEmail) {

**this**.touristEmail = touristEmail;

}

@Override

**public** String toString() {

**return** "Tourist [touristId=" + touristId + ", touristNumber=" + touristNumber + ", touristPassportNo="

+ touristPassportNo + ", cardNumber=" + cardNumber + ", touristName=" + touristName

+ ", touristAddress=" + touristAddress + ", touristEmail=" + touristEmail + "]";

}

}

**Issue Tourist Visa Class:**

package MainProject;

import java.io.EOFException;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.util.ArrayList;

public class IssueTouristVisa {

ArrayList<Tourist> tourists;

public static void saveTourist( ArrayList<Tourist> t ) {

ObjectOutputStream out;

try {

if(true) {

out = new ObjectOutputStream(new FileOutputStream ("touristDetails.txt"));

while(t.size()!=0)

{

Object o=t.remove(0);

out.writeObject(o);

}

out.close();

}

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.getMessage();

}catch(EOFException e) {

System.out.println("EOF Exception: This is the end of the file.");

}

catch (IOException e) {

// TODO Auto-generated catch block

e.getMessage();

}

}

public static ArrayList<Tourist> ReadTest (){

ArrayList<Tourist> tourists = new ArrayList<>();

ObjectInputStream in;

Object obj;

try {

in = new ObjectInputStream(new FileInputStream("touristDetails.txt"));

while((obj = in.readObject()) != null)

{

Tourist t= (Tourist) obj;

tourists.add(t);

// System.out.println(t);

}

in.close();

} catch (FileNotFoundException e) {

// // TODO Auto-generated catch block

System.out.println(e.getMessage());

} catch (IOException | ClassNotFoundException e) {

System.out.println( e.getMessage());

}

// System.out.println(tourists.size() + " tourists read from the file.");

return tourists;

}

public IssueTouristVisa(ArrayList<Tourist> tourists) {

this.tourists = tourists;

}

public IssueTouristVisa(){

tourists= ReadTest();

for(int i=0;i<tourists.size();i++) {

System.out.println( tourists.get(i));

}

}

public String addTourist(Tourist t) {

tourists.add(t);

return String.format("added Tourist successfully\n");

}

public Tourist findTourist(int passportno) {

for (var x : tourists)

if (x.getTouristPassportNo()==passportno)

return x;

return null;

}

public ArrayList<Tourist> getTourists() {

return tourists;

}

public void setTourists(ArrayList<Tourist> tourists) {

this.tourists = tourists;

}

}

**Main View:**

package mainview;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.stage.Stage;

import javafx.scene.Scene;

import javafx.scene.layout.AnchorPane;

public class Main extends Application {

@Override

public void start(Stage primaryStage) {

try {

AnchorPane root = FXMLLoader.load(getClass().getResource("MainView.fxml"));

Scene scene = new Scene(root);

primaryStage.setScene(scene);

primaryStage.setTitle("Tourist Application System!");

primaryStage.show();

} catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

launch(args);

}

}

**Main View Controller:**

package mainview;

import java.io.IOException;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.scene.layout.AnchorPane;

import javafx.stage.Stage;

public class MainViewController {

@FXML

void handleButtonAction(ActionEvent event) {

FXMLLoader loading = new FXMLLoader(getClass().getResource("Tourist\\touristView.fxml"));

try {

AnchorPane pane = (AnchorPane) loading.load();

Scene scene = new Scene(pane);

Stage stage = new Stage();

stage.setScene(scene);

stage.show();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

**Tourist Controller:**

package mainview.Tourist;

import javafx.application.Platform;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.TableColumn;

import javafx.scene.control.TableView;

import javafx.scene.control.TextField;

import javafx.scene.control.Alert.AlertType;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.layout.AnchorPane;

import javafx.stage.Stage;

import MainProject.Tourist;

import java.net.URL;

import java.util.ResourceBundle;

import javax.swing.JOptionPane;

import MainProject.IssueTouristVisa;

public class TouristController implements Initializable ,TouristInteractionForm {

@FXML

private Label bankDecisionLabel;

@FXML

private Label creditCardLabel;

@FXML

private Label resultNotice;

@FXML

private TextField taddress;

@FXML

private Button tbutton;

@FXML

private TextField tcardnumber;

@FXML

private TextField tdecision;

@FXML

private TextField temail;

@FXML

private TextField tid;

@FXML

private TextField tname;

@FXML

private TextField tnumber;

@FXML

private TextField tpassport;

@FXML

private Button tbutton1;

ObservableList<Integer> observableTObjectsID = FXCollections.observableArrayList(TouristRepository.getIds());

ObservableList<Integer> observableTObjectsPassport= FXCollections.observableArrayList(TouristRepository.getPassportNumber());

Tourist t;

@FXML

void handleButtonAction(ActionEvent event) {

if(tid.getText()!="" && tname.getText()!="" && tpassport.getText()!="" && tnumber.getText()!="" && temail.getText()!="" && taddress.getText()!="") {

t=new Tourist(0,0,0,0,"null","null","null");

t.setTouristId(Integer.parseInt(tid.getText()));

t.setTouristPassportNo(Integer.parseInt(tpassport.getText()));

t.setTouristNumber(Integer.parseInt(tnumber.getText()));

t.setTouristName(tname.getText());

t.setTouristEmail(temail.getText());

t.setTouristAddress(taddress.getText());

boolean booleanID=true,booleanPassport=true;

if(observableTObjectsID!=null) {

for(int i=0; i<observableTObjectsID.size();i++) {

if(observableTObjectsID.get(i)==Integer.parseInt(tid.getText()) || observableTObjectsPassport.get(i)==Integer.parseInt(tpassport.getText())) {

booleanID=false;

booleanPassport=false;

}

}

}

if(booleanID==true && booleanPassport==true) {

tbutton.setVisible(false);

tid.setEditable(false);

tpassport.setEditable(false);

tnumber.setEditable(false);

temail.setEditable(false);

taddress.setEditable(false);

tname.setEditable(false);

tid.setOpacity(0.7);

tpassport.setOpacity(0.7);

temail.setOpacity(0.7);

tnumber.setOpacity(0.7);

taddress.setOpacity(0.7);

tname.setOpacity(0.7);

creditCardLabel.setVisible(true);

tcardnumber.setVisible(true);

bankDecisionLabel.setVisible(true);

tdecision.setVisible(true);

tbutton1.setVisible(true);

}

else {

JOptionPane.showMessageDialog(null, "This user already exists");

}

}

else {

JOptionPane.showMessageDialog(null, "Missing Information! Try Again Later");

Platform.exit();

}

}

@FXML

void handleButton2Action(ActionEvent event) {

if(tcardnumber.getText()!=""){

t.setCardNumber(Integer.parseInt(tcardnumber.getText()));

if(tdecision.getText()!="") {

if(tdecision.getText().equalsIgnoreCase("f")) {

JOptionPane.showMessageDialog(null, "Bank Decision is Rejected ! Application Rejected !");

Platform.exit();

}

else {

addTourist(t);

JOptionPane.showMessageDialog(null, "Approved ! Visa Issued "+ "\n\nSummary : \n\n" + "Name of Person : "+tname.getText()+"\nID Number : "+tid.getText()+"\nPassport Number : "+tpassport.getText()+

"\nContact Number : "+tnumber.getText()+"\nContact Email ID :"+temail.getText()+"\nAddress of Person : "+taddress.getText());

}

}

else {

JOptionPane.showMessageDialog(null, "No Bank Decision Inputed ! Rejecting Application ... ");

}

}

else {

JOptionPane.showMessageDialog(null, "Credit Card Not Entered by the user!");

}

}

public interface touristInteraction {

public void addTourist(Tourist t);

}

@Override

public void initialize(URL arg0, ResourceBundle arg1) {

// TODO Auto-generated method stub

}

@Override

public void addTourist(Tourist t) {

TouristRepository.saveTourist(t);

}

}

**Tourist Interaction Form:**

**package** mainview.Tourist;

**import** MainProject.Tourist;

**public** **interface** TouristInteractionForm {

**public** **void** addTourist(Tourist t);

}

package mainview.Tourist;

import java.util.ArrayList;

import MainProject.IssueTouristVisa;

import MainProject.Tourist;

**Tourist Repository:**

public class TouristRepository {

public static ArrayList<Integer> getIds() {

ArrayList<Integer> touristIds = new ArrayList<>();

ArrayList<Tourist> t=IssueTouristVisa.ReadTest();

if(t.size()!=0) {

for (var tourist : t) {

touristIds.add(tourist.getTouristId());

}

}

return touristIds;

}

public static ArrayList<Integer> getPassportNumber() {

ArrayList<Integer> touristPassport = new ArrayList<>();

ArrayList<Tourist> t=IssueTouristVisa.ReadTest();

if(t.size()!=0) {

for (var tourist : t) {

touristPassport.add(tourist.getTouristPassportNo());

}

}

return touristPassport;

}

public static void saveTourist(Tourist T) {

ArrayList<Tourist> t = IssueTouristVisa.ReadTest();

t.add(T);

IssueTouristVisa.saveTourist(t);

}

}

**Outputs:**

1. **User Interface**

**Main Screen:**

Graphical user interface, application

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

1. **Adding a Tourist Visa**

Graphical user interface

Description automatically generated

Graphical user interface, website

Description automatically generated

Graphical user interface, website

Description automatically generated

**Notification of Visa Issued (Also added in the database)**

Graphical user interface, text, application, chat or text message

Description automatically generated

1. **If Visa application Already Exists**

Graphical user interface, website

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

1. **If Visa Application is rejected by the bank (Decision=F)**

Graphical user interface

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

1. **If Information is not complete, Missing Information**

Graphical user interface, website

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

**4- Test your program developed in Task 3. You must present test cases and test results. Explain how you have tested the program.**

1. The Program has been tested using Junit Testing method. A sample object is created, and different methods and functions are run separately to provide the expected results from them and compared to ensure different parts of the system are functioning well.

**Source Codes:**

**Class TouristTest:**

package test;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import model.Tourist;

class TouristTest{

private Tourist tourist = new Tourist (1, 1122, 29800021, 2338899,"Ahmed Saleh",

"Qatar-Al Sadd", "ahs332@gmail.com");

@Test

void testTourist() {

testGetTouristId();

testGetTouristNumber();

testGetTouristPassportNo() ;

testGetCardNumber();

testGetTouristName() ;

testGetTouristAddress() ;

testGetTouristEmail() ;

}

@Test

void testGetTouristId() {

assertEquals(1,tourist.getTouristId(),0);

}

@Test

void testGetTouristNumber() {

assertEquals(1122,tourist.getTouristNumber(),0);

}

@Test

void testGetTouristPassportNo() {

assertEquals(29800021,tourist.getTouristPassportNo(),0);

}

@Test

void testGetCardNumber() {

assertEquals( 2338899,tourist.getCardNumber(),0);

}

@Test

void testGetTouristName() {

assertEquals( "Ahmed Saleh",tourist.getTouristName());

}

@Test

void testGetTouristAddress() {

assertEquals( "Qatar-Al Sadd",tourist.getTouristAddress());

}

@Test

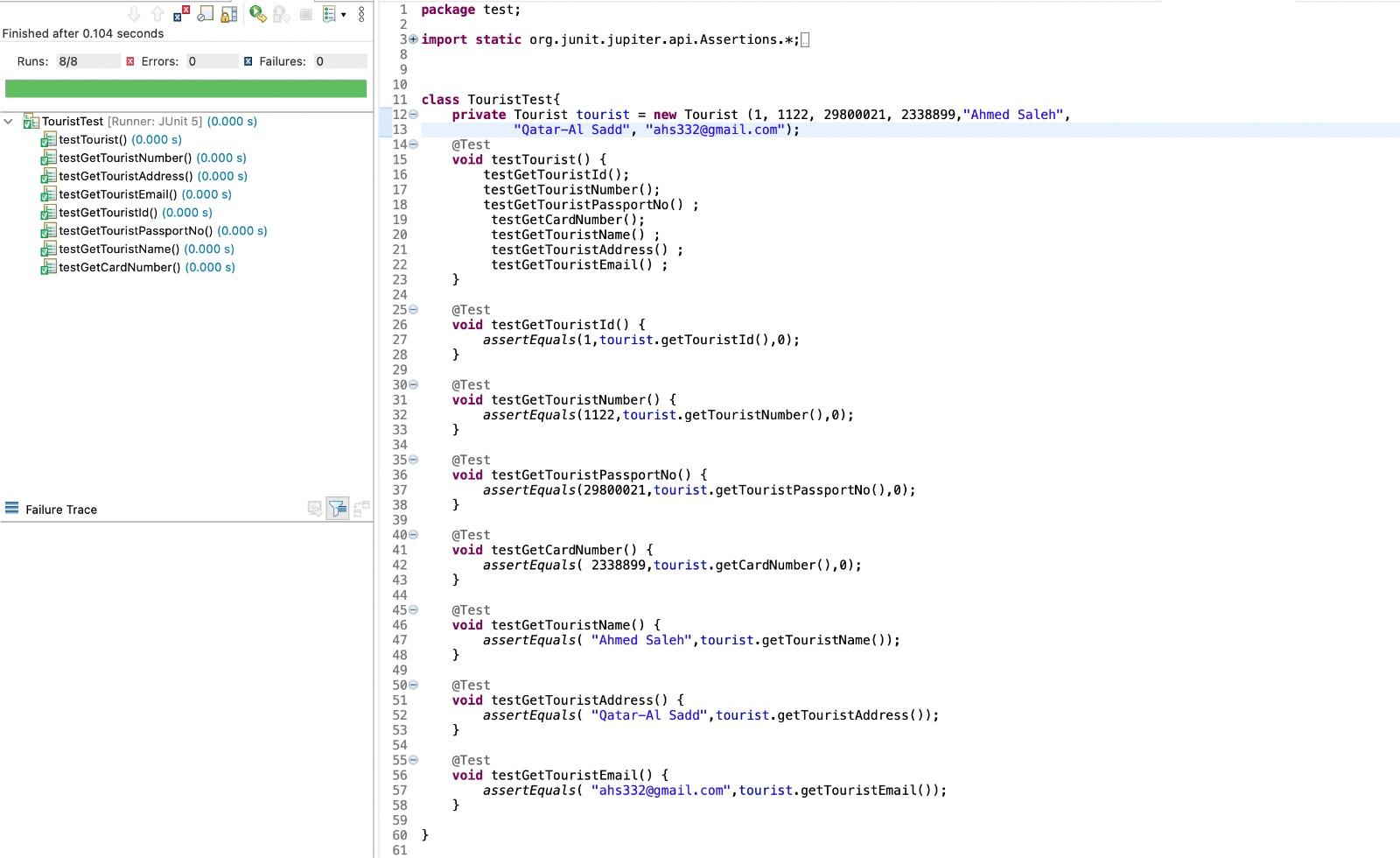
void testGetTouristEmail() {

assertEquals( "ahs332@gmail.com",tourist.getTouristEmail());

}

}

**Test Result:**



**Class IssueTouristVisaTest:**

package test;

import static org.junit.Assert.assertEquals;

import static org.junit.jupiter.api.Assertions.\*;

import java.util.ArrayList;

import org.junit.jupiter.api.Test;

import model.IssueTouristVisa;

import model.Tourist;

class IssueTouristVisaTest {

ArrayList<Tourist> tourists = new ArrayList <Tourist>();

IssueTouristVisa issueTouristVisa = new IssueTouristVisa(tourists);

@Test

void testReadTest() {

assertEquals(tourists,issueTouristVisa.ReadTest());

}

@Test

void testFindTourist() {

assertEquals(null,issueTouristVisa.findTourist(0));

}

@Test

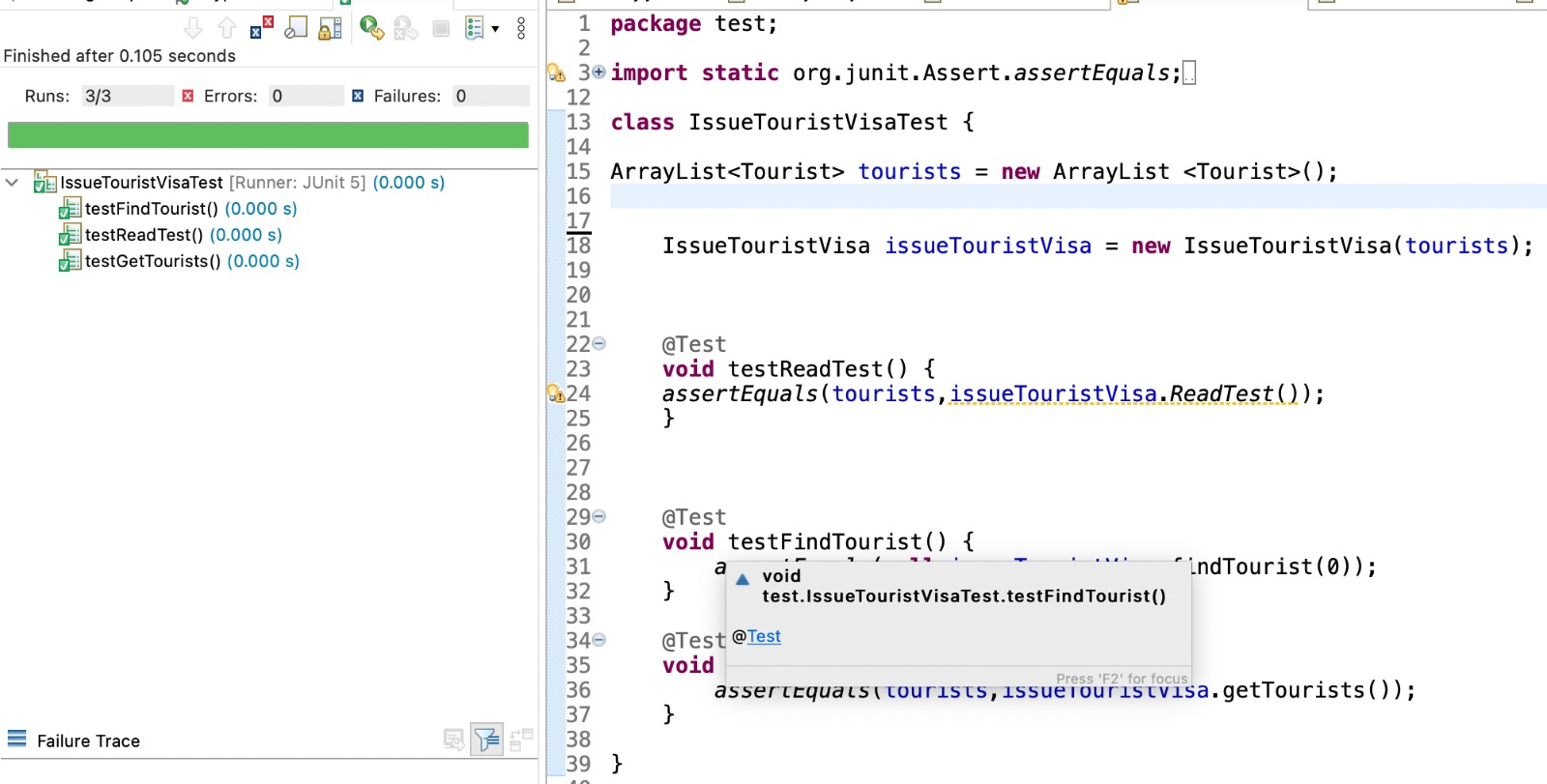
void testGetTourists() {

assertEquals(tourists,issueTouristVisa.getTourists());

}

}

**Test Result:**



**5. Test at least two quality requirements (NFRs) identified in Task (1).**

**NFR Test 1:**

**Performance: Load Testing:**

Graphical user interface, text, application, email

Description automatically generated

Inputting 1000 users inside the system file:

File (1000 users):

Text

Description automatically generated

Testing if the System can work with concurrent users

Adding another applicant after the system has 1000 concurrent users

Graphical user interface, application

Description automatically generated

The System functions well with 1000 concurrent users and the required performance is satisfied

Test Passed! Requirement is met

**NFR Test 2:**