

**MIDDLE EAST TECHNICAL UNIVERSITY
NORTHERN CYPRUS CAMPUS**

**Departments of
Computer Engineering**

CNG 353 Software Design Patterns

Berkay Yericer 2385722

1 Table Of Contents

1	Table Of Figures	3
2	Introduction.....	3
3	Functional Requirements	5
3.1	User Management	5
3.2	Device Management.....	5
3.3	Analytics.....	5
3.4	Payment Processing.....	5
3.5	Notifications.....	5
3.6	Hybrid Cloud Integration	5
3.7	Utility Monitoring	5
3.8	Data Storage	5
4	System Design and Architecture.....	6
4.1	MemCa Diagram	6
4.2	Use Case Diagram	7
4.3	GUI Mockups	8
	9
4.4	Class Diagram.....	10
4.5	Database	11
5	Implementation	13
	13
6	Testing and Results	18
7	Conclusion	19
8	References	19

1 Table Of Figures

Figure 1 MemCa Diagram	6
Figure 2 Use Case Diagram	7
Figure 3 GUI Resident Portal.....	8
Figure 4 GUI Admin Panel	9
Figure 5 Class Diagram.....	10
Figure 6 SQL Database Entire Table.....	11
Figure 7 SQL Sensor Check Table	11
Figure 8 SQL IoT Devices Table.....	12
Figure 9 SQL Security Table.....	12
Figure 10 SQL Payment Table	12
Figure 11 Admin Dashboard Code.....	13
Figure 12 Resident Portal Code	14
Figure 13 Security Manager Code	15
Figure 14 Analytics Code.....	16
Figure 15 Cloud Service Code	17
Figure 16 Output	18

2 Introduction

in order to effectively manage the urban infrastructure of the cities established today, smart solutions are now needed due to the rapid increase in urbanization. The "Smart City System" project seeks to illustrate the use of design patterns, safe hybrid-cloud integration, and real-time data management in a city simulation. In addition to providing people with necessary services like utility monitoring, device management, and payment processing, the system gives municipal officials centralized control and monitoring capabilities. The system provides centralized control and monitoring capabilities to municipal authorities. Controlling such systems by computers provides advantages in many aspects

This project's major goal is to create and deploy a safe, scalable, and maintainable system that incorporates several features, such as:

- Controlling Internet of Things devices (such as thermostats and lamps).
- using predictive algorithms for energy and traffic data.
- enabling safe payment options (cryptocurrency and fiat).
- Notifying authorities and citizens in real-time.

Deliverable	Description
Class Diagram	A detailed UML class diagram illustrating relationships and design patterns used in the system.
Use Case Diagram	Use case diagram showing actor interactions with the system
MemCA Diagram	Holistic view of the system architecture in a tier-by-tier MemCA diagram.
Java Implementation	Fully functional Java code implementing the system using design patterns
GUI Mockups	GUI designs for Admin and Resident portals using Figma for interaction visualization.
SQL Database	SQL schema for managing users, utility data, and system attributes.
DigiBank Application	Integration with DigiBank app, including exported .txt file and real-time email notifications.
Screenshots and Outputs	Screenshots of the application running, exported files, and email notifications for bonus tasks.

Table 1 Deliverables

3 Functional Requirements

3.1 User Management

- The system shall allow admins to authenticate users through secure login control mechanisms.
- Residents should be able to register, update, and manage their profiles from the Resident Portal.

3.2 Device Management

- Residents should be able to control home devices (like thermostats, lights) from the Resident Portal.
- Admins should have access to manage public infrastructure devices (like streetlights, sensors) from the Admin Dashboard.

3.3 Analytics

- The system should generate real-time traffic and energy analytics using different analyzing techniques.
- Admins should be able to view detailed reports on energy consumption, traffic level , and system performance from the Admin Dashboard.

3.4 Payment Processing

- Residents should be able to make payments for utilities via fiat or cryptocurrency.
- The system should securely handle transactions using the DigiBank application.
- Payment details should be logged and stored in the database for future reference.

3.5 Notifications

- The system should notify residents about utility consumption, payment reminders, and system alerts in real-time.
- Public Safety Authorities shall receive notifications about security alerts triggered by the City Controller.

3.6 Hybrid Cloud Integration

- The system should utilize a Hybrid Cloud to store and retrieve analytics data.
- The Hybrid Cloud Proxy shall secure access to real cloud services, ensuring authenticated and authorized requests.

3.7 Utility Monitoring

- Residents shall be able to view real-time electricity and water consumption data via the Resident Portal.
- Admins should monitor overall utility usage and generate performance metrics.

3.8 Data Storage

- All user profiles, utility data, payments, and analytics results should be stored in the database.
- The system should maintain logs of user actions, notifications, and system operations for auditing purposes.

4 System Design and Architecture

4.1 MemCa Diagram

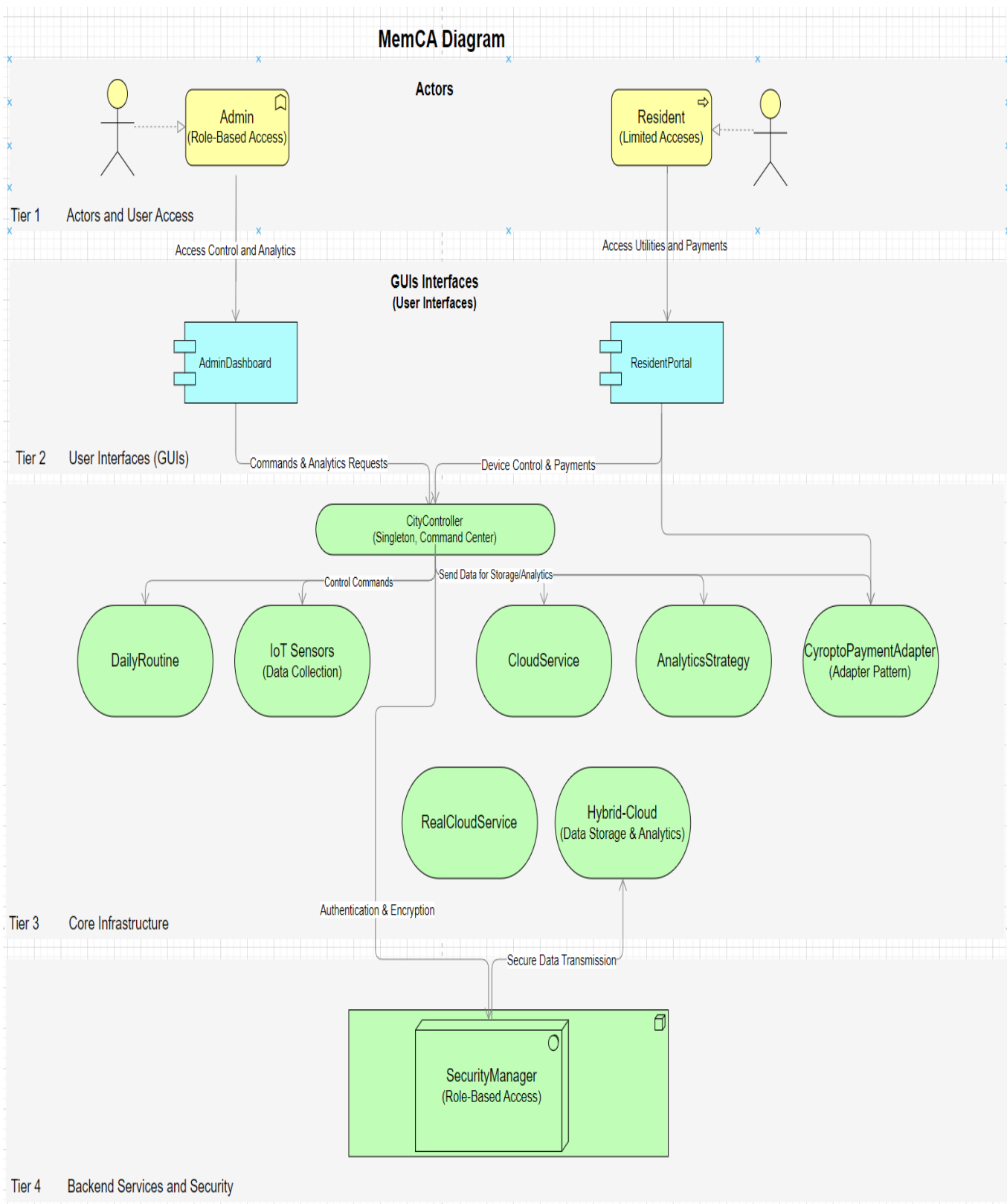


Figure 1 MemCa Diagram

4.2 Use Case Diagram

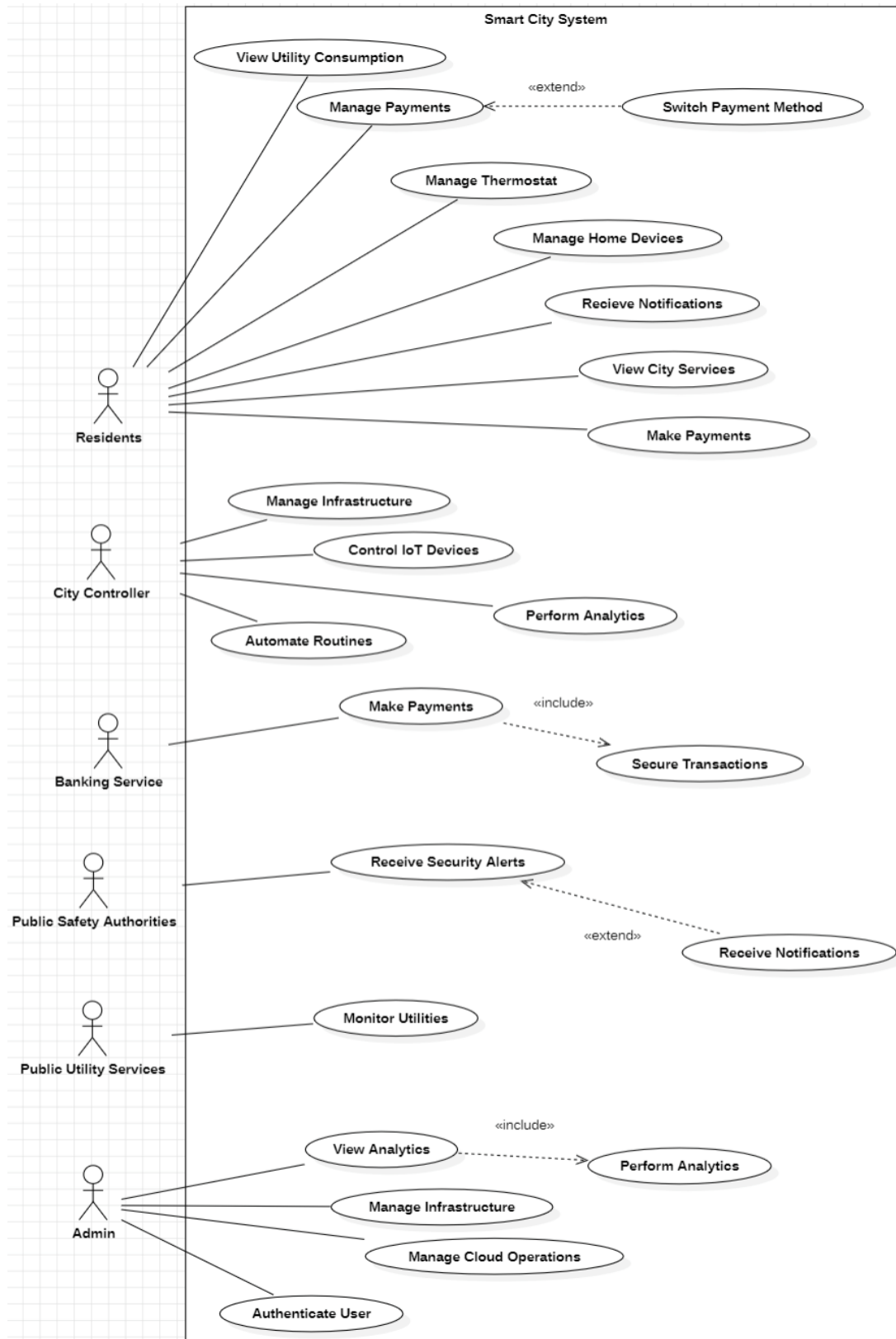


Figure 2 Use Case Diagram

Figure 2 illustrates the key interactions between actors and the Smart City System. Device management, utility viewing, payment processing, and notice receiving are all done by residents. The City Controller is in charge of IoT devices, analytics, and infrastructure. Admins are in charge of user authentication, cloud operations, and analytics. Secure transactions are handled by banking services, and alarms are sent to public safety authorities. Utility Services keep an eye on resources, guaranteeing smooth system integration and operation.

4.3 GUI Mockups

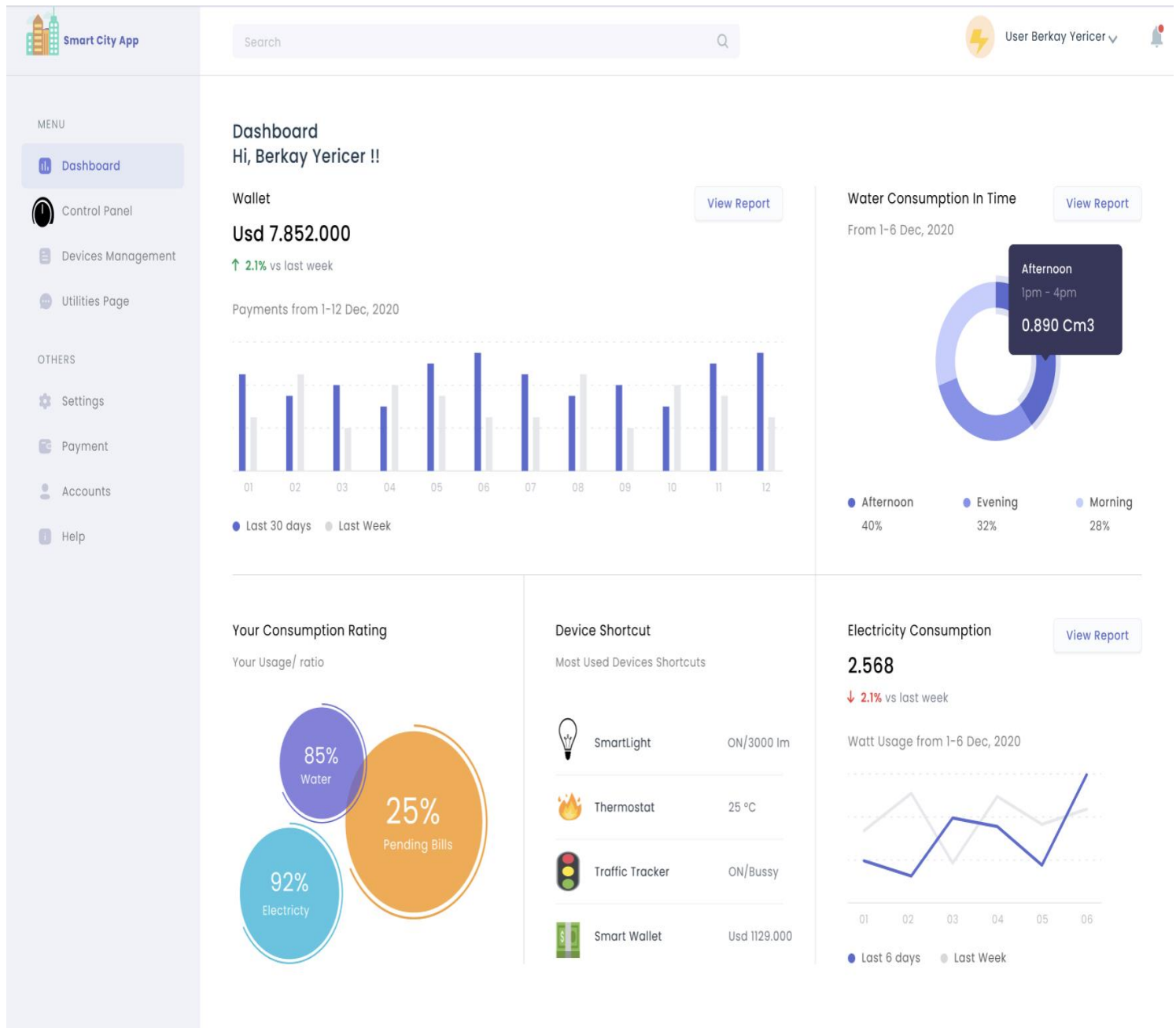


Figure 3 GUI Resident Portal

Figure 3 shows the user interface for the **Smart City App**. It provides an intuitive dashboard where users can view their wallet balance, track payments, monitor water and electricity consumption over time, and access their consumption rating. The interface includes device shortcuts for controlling SmartLights, Thermostats, and Traffic Trackers, ensuring efficient device management. This design integrates utility tracking and device control, enhancing user interaction with the system. interfaces are completely a product of imagination .there is no code implementation.

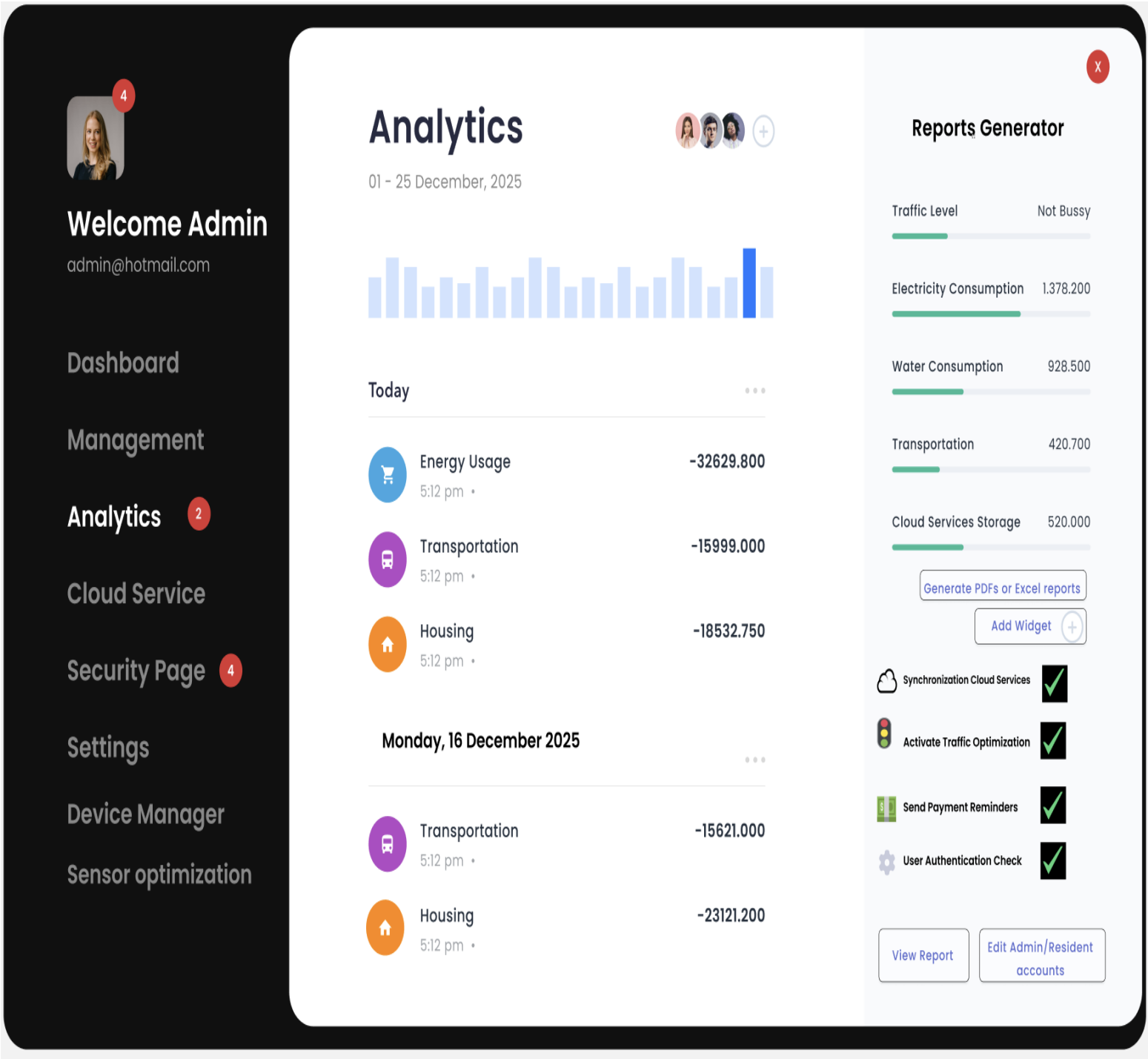


Figure 4 GUI Admin Panel

Figure 5 presents the **Class Diagram** for the Smart City System, showcasing the integration of various components and their relationships. The design employs several design patterns:

- **Singleton:** Ensures a single instance of CityController for centralized management.
- **Observer:** Used by CityMonitor to notify Residents and Authorities of real-time updates.
- **Command:** Allows flexible control of IoT devices (like streetlights) through RemoteControl.
- **Strategy:** Provides interchangeable analytics strategies (TrafficAnalytics and EnergyAnalytics) for predictive insights.
- **Proxy:** Secures cloud operations via HybridCloudProxy, mediating access to RealCloudService.
- **Template Method:** Automates routines in DailyRoutine and its subclass CityRoutine.
- **Adapter:** Integrates CryptoPaymentAdapter to enable cryptocurrency payments.

This diagram highlights the scalability, modularity, and maintainability of the Smart City System's architecture as we discuss in the class .

4.5 Database

id	name	surname	email	electricity_consumption	water_consumption	thermostat_degree	streetlight_status	payment_method	sensors
1	John	Doe	john.doe@gmail.com	350.75	1200.5	22	ON	Fiat	Traffic Sensor: ACTIVE
2	Jane	Smith	jane.smith@gmail.com	400.3	1300.8	24	OFF	Crypto	Energy Sensor: ACTIVE
3	Mike	Brown	mike.brown@gmail.com	300.1	1100.4	21	ON	Fiat	Water Sensor: INACTIVE
4	Lisa	Johnson	lisa.johnson@gmail.com	250	1000.25	20	OFF	Crypto	Temperature Sensor: ACTIVE
5	Tom	Clark	tom.clark@gmail.com	375.6	1250	23	ON	Fiat	Traffic Sensor: ACTIVE
6	Emma	Taylor	emma.taylor@gmail.com	325.5	1150.75	25	ON	Crypto	Energy Sensor: INACTIVE
7	James	Wilson	james.wilson@gmail.com	275.8	1050.3	18	OFF	Fiat	Water Sensor: ACTIVE

Figure 6 SQL Database Entire Table

name	surname	sensors
John	Doe	Traffic Sensor: ACTIVE
Jane	Smith	Energy Sensor: ACTIVE
Mike	Brown	Water Sensor: INACTIVE
Lisa	Johnson	Temperature Sensor: ACTIVE
Tom	Clark	Traffic Sensor: ACTIVE
Emma	Taylor	Energy Sensor: INACTIVE
James	Wilson	Water Sensor: ACTIVE

Figure 7 SQL Sensor Check Table

IoT_Devices				
device_id	device_name	device_type	status	last_updated
1	Streetlight Zone 1	Streetlight	ON	2024-12-15 22:31:07
2	Thermostat House A	Thermostat	OFF	2024-12-15 22:31:07
3	Traffic Sensor 1	Sensor	ACTIVE	2024-12-15 22:31:07

Figure 8 SQL IoT Devices Table

Security				
security_id	user_id	status	description	timestamp
1	1	Success	Admin login successful	2024-12-15 22:31:07
2	2	Failure	Invalid password attempt	2024-12-15 22:31:07

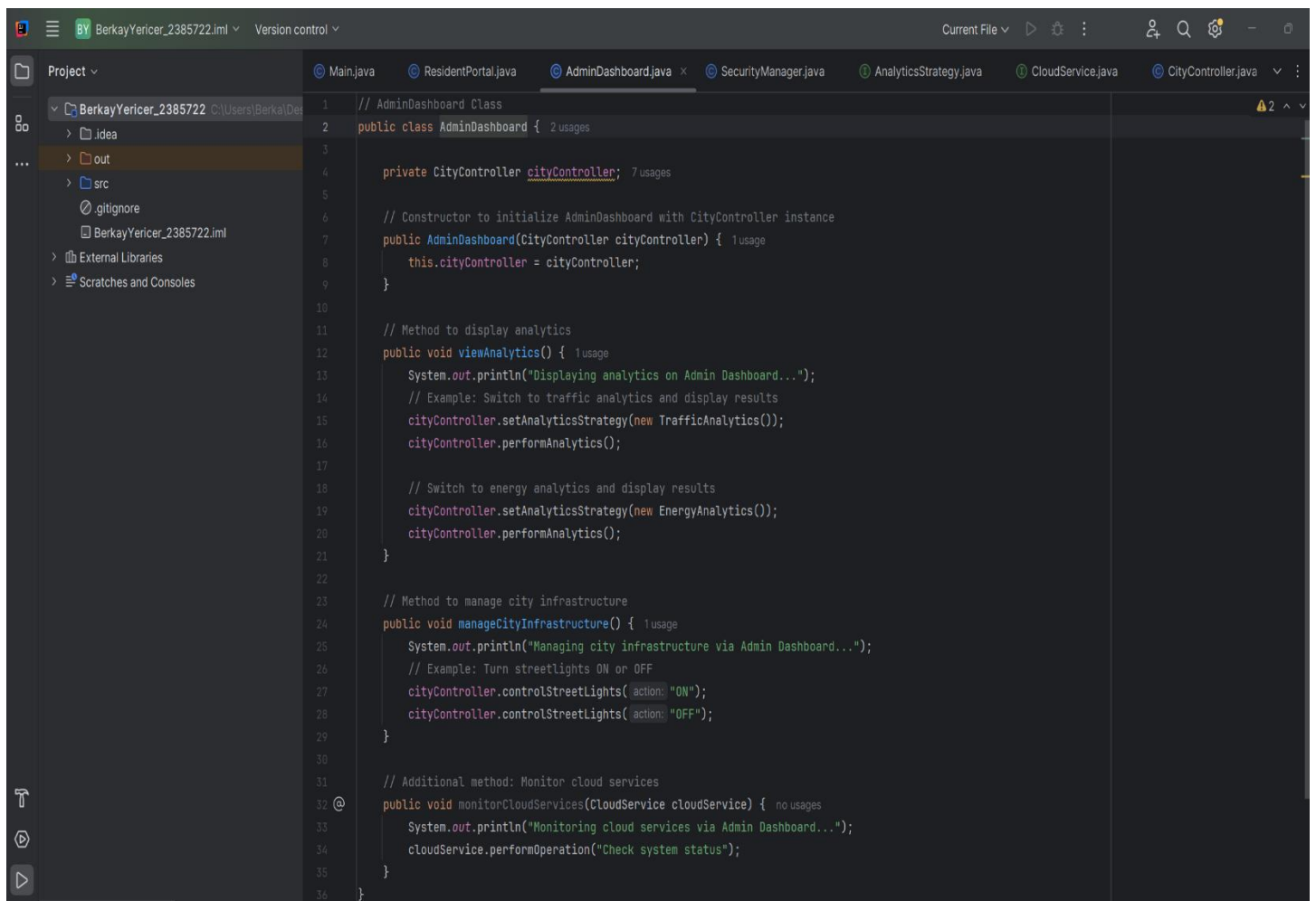
Figure 9 SQL Security Table

Payments				
payment_id	user_id	amount	payment_method	transaction_date
1	2	50	Fiat	2024-12-15 22:31:07
2	2	100	Crypto	2024-12-15 22:31:07

Figure 10 SQL Payment Table

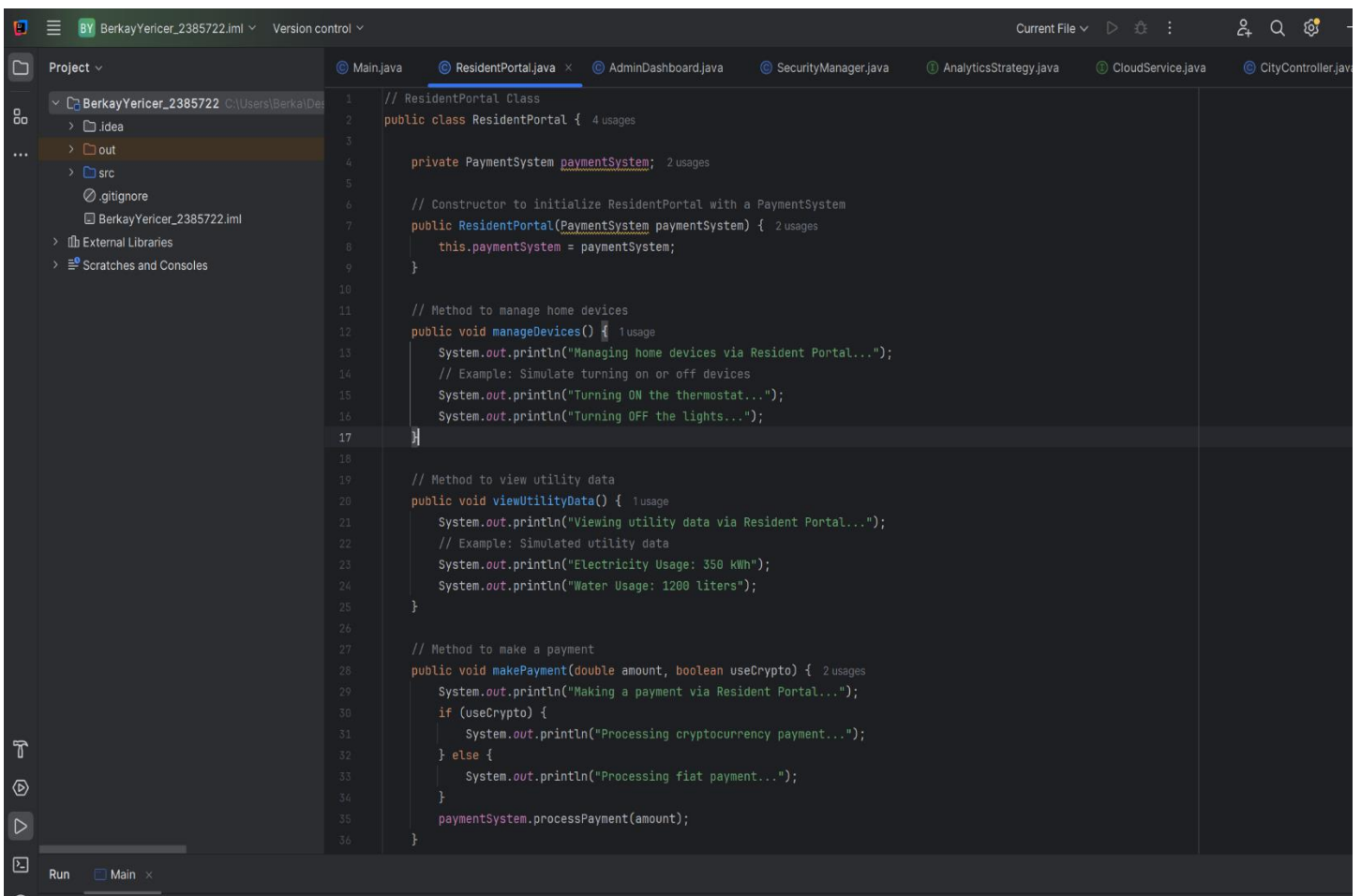
Figures between 6-10 shows the example Database visualization of Smart City . I thought there should be a database for such a comprehensive assignment. I was able to do it quickly because I had taken a database course before, and I thought it would be suitable for this assignment. names and data are given completely randomly.

5 Implementation

The image is a screenshot of an IDE window showing the implementation of the AdminDashboard class. The project name is 'BerkayYericer_2385722'. The file 'AdminDashboard.java' is open, showing the following code:

```
1 // AdminDashboard Class
2 public class AdminDashboard { 2 usages
3
4     private CityController cityController; 7 usages
5
6     // Constructor to initialize AdminDashboard with CityController instance
7     public AdminDashboard(CityController cityController) { 1 usage
8         this.cityController = cityController;
9     }
10
11     // Method to display analytics
12     public void viewAnalytics() { 1 usage
13         System.out.println("Displaying analytics on Admin Dashboard...");
14         // Example: Switch to traffic analytics and display results
15         cityController.setAnalyticsStrategy(new TrafficAnalytics());
16         cityController.performAnalytics();
17
18         // Switch to energy analytics and display results
19         cityController.setAnalyticsStrategy(new EnergyAnalytics());
20         cityController.performAnalytics();
21     }
22
23     // Method to manage city infrastructure
24     public void manageCityInfrastructure() { 1 usage
25         System.out.println("Managing city infrastructure via Admin Dashboard...");
26         // Example: Turn streetlights ON or OFF
27         cityController.controlStreetLights( action: "ON");
28         cityController.controlStreetLights( action: "OFF");
29     }
30
31     // Additional method: Monitor cloud services
32     public void monitorCloudServices(CloudService cloudService) { no usages
33         System.out.println("Monitoring cloud services via Admin Dashboard...");
34         cloudService.performOperation("Check system status");
35     }
36 }
```

Figure 11 Admin Dashboard Code



The screenshot shows an IDE window with the project 'BerkayYericer_2385722' open. The file 'ResidentPortal.java' is selected in the editor. The code defines a 'ResidentPortal' class with three methods: 'manageDevices()', 'viewUtilityData()', and 'makePayment()'. The 'manageDevices()' method prints messages about managing home devices, including turning on/off the thermostat and lights. The 'viewUtilityData()' method prints simulated utility data for electricity and water usage. The 'makePayment()' method prints messages about making a payment, including processing cryptocurrency or fiat payments, and then calls 'paymentSystem.processPayment()'. The IDE interface includes a project explorer on the left, a tab bar at the top, and a run button at the bottom left.

```
1 // ResidentPortal Class
2 public class ResidentPortal { 4 usages
3
4     private PaymentSystem paymentSystem; 2 usages
5
6     // Constructor to initialize ResidentPortal with a PaymentSystem
7     public ResidentPortal(PaymentSystem paymentSystem) { 2 usages
8         this.paymentSystem = paymentSystem;
9     }
10
11    // Method to manage home devices
12    public void manageDevices() { 1 usage
13        System.out.println("Managing home devices via Resident Portal...");
14        // Example: Simulate turning on or off devices
15        System.out.println("Turning ON the thermostat...");
16        System.out.println("Turning OFF the lights...");
17    }
18
19    // Method to view utility data
20    public void viewUtilityData() { 1 usage
21        System.out.println("Viewing utility data via Resident Portal...");
22        // Example: Simulated utility data
23        System.out.println("Electricity Usage: 350 kWh");
24        System.out.println("Water Usage: 1200 liters");
25    }
26
27    // Method to make a payment
28    public void makePayment(double amount, boolean useCrypto) { 2 usages
29        System.out.println("Making a payment via Resident Portal...");
30        if (useCrypto) {
31            System.out.println("Processing cryptocurrency payment...");
32        } else {
33            System.out.println("Processing fiat payment...");
34        }
35        paymentSystem.processPayment(amount);
36    }
}
```

Figure 12 Resident Portal Code

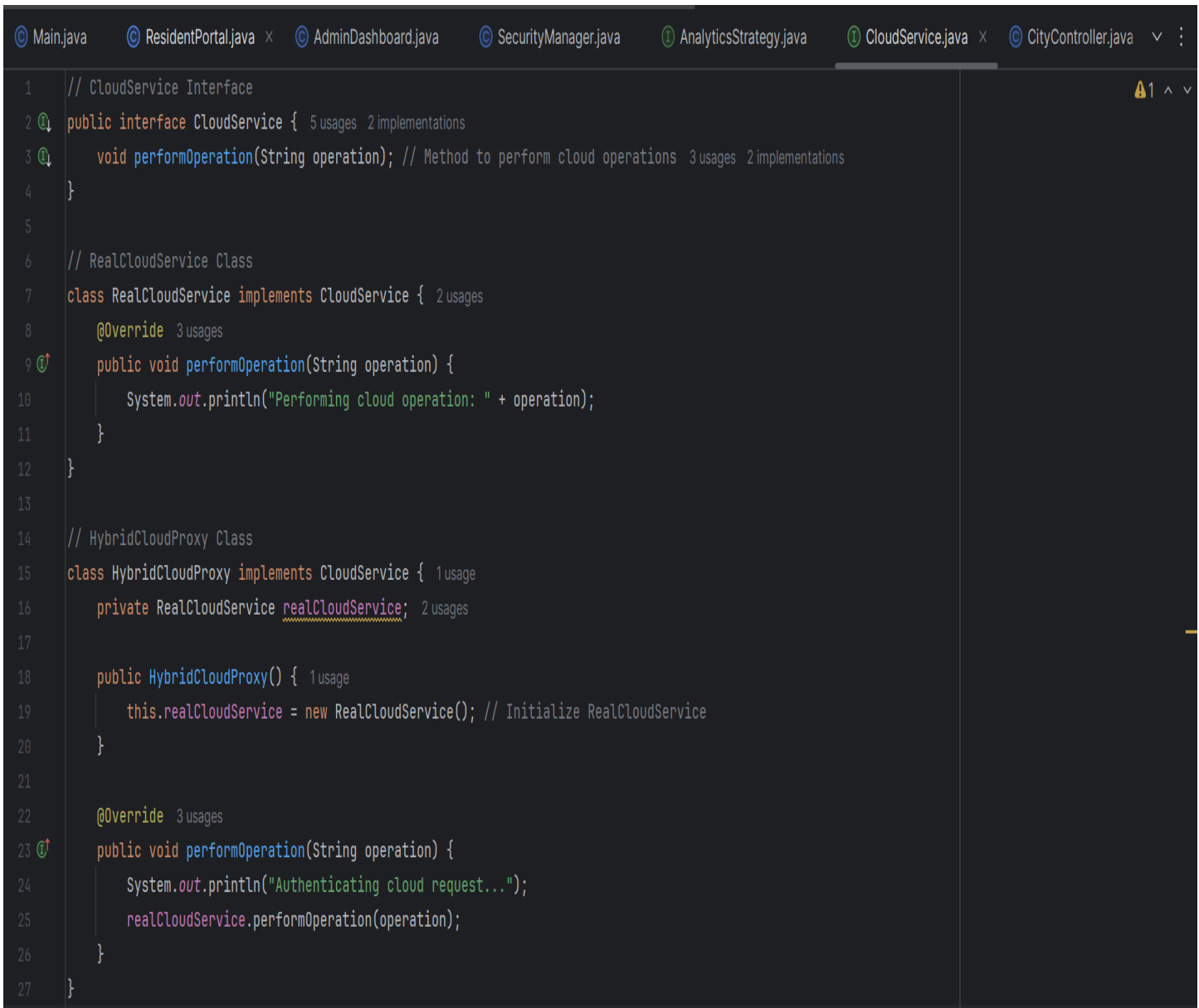
```
© Main.java © ResidentPortal.java © AdminDashboard.java © SecurityManager.java × AnalyticsStrategy.java CloudService.java CityController.java
1 // SecurityManager Class
2 public class SecurityManager { 2 usages
3
4     // Method to authenticate a user with a specific method
5     public void authenticate(String user, String method) { 1 usage
6         System.out.println("Authenticating " + user + " using " + method + "...");
7         // Simulate authentication logic
8         if ("Multi-Factor Authentication".equalsIgnoreCase(method)) {
9             System.out.println(user + " has been successfully authenticated using Multi-Factor Authentication.");
10        } else {
11            System.out.println(user + " authentication failed. Unsupported method: " + method);
12        }
13    }
14
15    // Method to encrypt data
16    public String encrypt(String data) { 1 usage
17        // Simulate encryption logic
18        String encryptedData = "Encrypted(" + data + ")";
19        System.out.println("Encrypting data: " + data);
20        System.out.println("Encrypted data: " + encryptedData);
21        return encryptedData;
22    }
23
24    // Method to decrypt data
25    public String decrypt(String encryptedData) { no usages
26        // Simulate decryption logic
27        if (encryptedData.startsWith("Encrypted(") && encryptedData.endsWith(")") {
28            String decryptedData = encryptedData.substring(10, encryptedData.length() - 1);
29            System.out.println("Decrypting data: " + encryptedData);
30            System.out.println("Decrypted data: " + decryptedData);
31            return decryptedData;
32        } else {
33            System.out.println("Invalid encrypted data format.");
34            return null;
35        }
36    }
}
```

Figure 13 Security Manager Code

```
© Main.java x © ResidentPortal.java © AdminDashboard.java © SecurityManager.java 1 AnalyticsStrategy.java x 1 CloudService.java © CityController.java v ⋮

1 // AnalyticsStrategy Interface
2 public interface AnalyticsStrategy { 4 usages 2 implementations
3     void analyze(); 1 usage 2 implementations
4 }
5
6 // TrafficAnalytics Class
7 class TrafficAnalytics implements AnalyticsStrategy { 2 usages
8     @Override 1 usage
9     public void analyze() {
10         // Example Logic: Calculate congestion level
11         double congestionLevel = Math.random() * 100; // Simulating traffic congestion percentage
12         if (congestionLevel < 30) {
13             System.out.println("Traffic is smooth with low congestion: " + congestionLevel + "%");
14         } else if (congestionLevel < 70) {
15             System.out.println("Traffic is moderate with medium congestion: " + congestionLevel + "%");
16         } else {
17             System.out.println("Traffic is heavy with high congestion: " + congestionLevel + "%");
18         }
19     }
20 }
21
22 // EnergyAnalytics Class
23 class EnergyAnalytics implements AnalyticsStrategy { 2 usages
24     @Override 1 usage
25     public void analyze() {
26         // Example Logic: Predict energy consumption
27         double energyDemand = Math.random() * 1000; // Simulating energy demand in kWh
28         if (energyDemand < 300) {
29             System.out.println("Energy demand is low: " + energyDemand + " kWh");
30         } else if (energyDemand < 700) {
31             System.out.println("Energy demand is moderate: " + energyDemand + " kWh");
32         } else {
33             System.out.println("Energy demand is high: " + energyDemand + " kWh");
34         }
35     }
36 }
```

Figure 14 Analytics Code



```

1 // CloudService Interface
2 public interface CloudService { 5 usages 2 implementations
3     void performOperation(String operation); // Method to perform cloud operations 3 usages 2 implementations
4 }
5
6 // RealCloudService Class
7 class RealCloudService implements CloudService { 2 usages
8     @Override 3 usages
9     public void performOperation(String operation) {
10         System.out.println("Performing cloud operation: " + operation);
11     }
12 }
13
14 // HybridCloudProxy Class
15 class HybridCloudProxy implements CloudService { 1 usage
16     private RealCloudService realCloudService; 2 usages
17
18     public HybridCloudProxy() { 1 usage
19         this.realCloudService = new RealCloudService(); // Initialize RealCloudService
20     }
21
22     @Override 3 usages
23     public void performOperation(String operation) {
24         System.out.println("Authenticating cloud request...");
25         realCloudService.performOperation(operation);
26     }
27 }

```

Figure 15 Cloud Service Code

These Figures show that, the provided Java implementation for the Smart City System highlights key design patterns to ensure modularity and functionality. The AdminDashboard leverages the Strategy Pattern for dynamic analytics (e.g., traffic and energy) and the Proxy Pattern for secure cloud operations. The ResidentPortal enables users to manage devices, monitor utility data, and make payments, integrating the Adapter Pattern to support both fiat and cryptocurrency transactions. The AnalyticsStrategy and its implementations (e.g., TrafficAnalytics and EnergyAnalytics) demonstrate the Strategy Pattern for predictive insights. Similarly, the CloudService and HybridCloudProxy employ the Proxy Pattern to authenticate and secure cloud operations. Together, these implementations showcase a robust and scalable architecture for the Smart City System.

6 Testing and Results

```
Processing cryptocurrency payment of 100.0 units.|
Processing fiat payment of $50.0
Turning on the street lights...
Checking public safety...
Managing utilities like water and electricity...
Authenticating Admin using Multi-Factor Authentication...
Admin has been successfully authenticated using Multi-Factor Authentication.
Encrypting data: Sensitive Data
Encrypted data: Encrypted(Sensitive Data)
Encrypted(Sensitive Data)
Authenticating cloud request...
Performing cloud operation: Store traffic data
Performing analytics...
Traffic is moderate with medium congestion: 47.53856166034859%
Performing analytics...
Energy demand is low: 255.95963932507826 kWh
Displaying analytics on Admin Dashboard...
Performing analytics...
Traffic is smooth with low congestion: 20.30214389875581%
Performing analytics...
Energy demand is high: 910.9371377166694 kWh
Managing city infrastructure via Admin Dashboard...
Street Lights are now ON
Street Lights are now OFF
Managing home devices via Resident Portal...
Turning ON the thermostat...
Turning OFF the lights...
Viewing utility data via Resident Portal...
Electricity Usage: 350 kWh
Water Usage: 1200 liters
Making a payment via Resident Portal...
Processing fiat payment...
Processing fiat payment of $50.0
Switching ResidentPortal to CryptoPayment...
Making a payment via Resident Portal...
Processing cryptocurrency payment...
Processing cryptocurrency payment of 100.0 units.
```

Figure 16 Output

7 Conclusion

The development of the **Smart City System** demonstrates the practical application of advanced software design principles and patterns to solve real-world challenges in urban management. Through the use of modular and scalable architecture, the system successfully integrates key functionalities such as IoT device management, predictive analytics for traffic and energy, secure payment processing, and real-time notifications.

By leveraging design patterns like Singleton, Observer, Template Method, Strategy, and Proxy, the system ensures a high degree of maintainability and flexibility. The integration of a **Hybrid-Cloud** provides scalability for data storage and analytics, highlighting the importance of cloud-based solutions in modern systems. Additionally, the secure transaction handling through the Digi Bank application underscores the emphasis on data security and reliability.

The testing phase validated the system's core functionalities, including device control, payment processing, and analytics generation. The successful monitoring and real-time email notifications using the Observer Pattern further enhance the system's usability and responsiveness.

Overall, the **Smart City System** showcases the power of design patterns, secure integration, and cloud computing in building intelligent and efficient urban management systems. This project not only meets the requirements of HW2 but also serves as a foundation for further exploration and development in smart city technologies.

8 References

1. Refactoring.Guru. (n.d.). *Design patterns*. <https://refactoring.guru/design-patterns>
2. Uzayr, S. B. (2022). *Software Design Patterns*. <https://doi.org/10.1201/9781003308461>
3. *A Survey on Trusted Distributed Artificial Intelligence*. (2022). IEEE Journals & Magazine | IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/9777972>