Architecture Diagram

With its efficient and modular components, the architecture seamlessly interacts for real-time transaction processing, ensuring high performance and reliability.

1. **Components**:
    - **User Module**: Ensures the highest level of security by handling user authentication, registration, and management.
    - **Transaction Module**: Processes real-time transactions.
    - **Fraud Detection Module**: Analyzes transactions for anomalies.
    - **Database**: Stores transaction and user data (PostgreSQL for structured data and MongoDB for logs).
    - **Monitoring Tools**: Prometheus and Grafana for system health and performance monitoring.
    - **APIs**: REST APIs for all modules, exposed via a Gateway (optional).
2. **Diagram Description**:
    - The Frontend communicates with the backend using RESTful APIs.
    - Microservices communicate internally and log data into databases.
    - Prometheus scrapes metrics from services while Grafana visualizes them.

1. Project Setup

- Create a **Spring Boot project** using a tool like Spring Initializr.
- Add dependencies:
    - Spring Web
    - Spring Data JPA
    - PostgreSQL Driver
    - MongoDB Driver
    - Spring Security (for JWT/OAuth2)
    - Actuator (for Prometheus metrics)

2. Code Modules

a) User Module

Handles registration, login, and JWT-based authentication.

Controller:

java
Copy code

```
@RestController
@RequestMapping("/api/users")
public class UserController {
```

```
    @Autowired
    private UserService userService;

    @PostMapping("/register")
    public ResponseEntity<String> register(@RequestBody User user) {
        userService.registerUser(user);
        return ResponseEntity.ok("User registered successfully!");
    }

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestBody LoginRequest
loginRequest) {
        String token = userService.authenticateUser(loginRequest);
        return ResponseEntity.ok(token);
    }
}
```

Service:

java
Copy code

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private JwtTokenProvider jwtTokenProvider;

    public void registerUser(User user) {
        user.setPassword(new
BCryptPasswordEncoder().encode(user.getPassword()));
        userRepository.save(user);
    }

    public String authenticateUser(LoginRequest loginRequest) {
        User user =
userRepository.findByUsername(loginRequest.getUsername())
```

```
                .orElseThrow(() -> new UsernameNotFoundException("User
not found"));
        if (!new
BCryptPasswordEncoder().matches(loginRequest.getPassword(),
user.getPassword())) {
            throw new BadCredentialsException("Invalid credentials");
        }
        return jwtTokenProvider.generateToken(user.getUsername());
    }
}
```

Model:

java
Copy code

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;
    private String password;
    private String role;

    // Getters and Setters
}
```

---

b) Transaction Module

Handles transaction processing.

Controller:

java
Copy code

```
@RestController
@RequestMapping("/api/transactions")
public class TransactionController {
```

```
    @Autowired
    private TransactionService transactionService;

    @PostMapping("/process")
    public ResponseEntity<String> processTransaction(@RequestBody
Transaction transaction) {
        transactionService.processTransaction(transaction);
        return ResponseEntity.ok("Transaction processed
successfully!");
    }
}
```

Service:

java
Copy code

```
@Service
public class TransactionService {

    @Autowired
    private TransactionRepository transactionRepository;

    public void processTransaction(Transaction transaction) {
        transactionRepository.save(transaction);
    }
}
```

Model:

java
Copy code

```
@Entity
public class Transaction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private Double amount;
```

```
    private String sourceAccount;
    private String destinationAccount;
    private LocalDateTime timestamp;

    // Getters and Setters
}
```

---

c) Fraud Detection Module

Checks for anomalies in transactions.

Service:

java
Copy code

```
@Service
public class FraudDetectionService {

    public boolean isFraudulent(Transaction transaction) {
        // Example rule: Flag transactions over $10,000
        return transaction.getAmount() > 10000;
    }
}
```

Integrate with the Transaction Service:

java
Copy code

```
@Autowired
private FraudDetectionService fraudDetectionService;

public void processTransaction(Transaction transaction) {
    if (fraudDetectionService.isFraudulent(transaction)) {
        throw new RuntimeException("Fraudulent transaction
detected!");
    }
    transactionRepository.save(transaction);
}
```

---

d) Prometheus and Grafana Monitoring

- Add Actuator dependency for Prometheus metrics.

Configure `application.properties`:
properties
Copy code
```
management.endpoints.web.exposure.include=*
management.endpoint.prometheus.enabled=true
```

-

Start Prometheus and Grafana containers:
yaml
Copy code
```
version: '3.8'
services:
  prometheus:
    image: prom/prometheus
    ports:
      - "9090:9090"
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml

  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
```

-

---

e) Dockerize Services

Write Dockerfile for each microservice:

dockerfile
Copy code
```
FROM openjdk:17
EXPOSE 8080
```

```
ADD target/user-service.jar user-service.jar
ENTRYPOINT ["java", "-jar", "user-service.jar"]
```

Build and run:

bash
Copy code

```
docker build -t user-service .
docker run -p 8080:8080 user-service
```