HUMAN SECURITY SYSTEM

CODE FOR USE CASE OF

CALL FOR HELPER:

DOMAIN LAYER

In [2]:

```
from abc import ABC, abstractmethod
from typing import List, Dict

# User class representing a security system user
class User:
    def __init__(self, username: str, password: str, role: str):
        self.username = username
        self.password = password  # In a real system, use hashed passwords
        self.role = role

# Interface for authentication
class IAuthenticator(ABC):
    @abstractmethod
    def authenticate(self, username: str, password: str) -> User:
        pass

# Interface for access control
class IAccessControl(ABC):
    @abstractmethod
    def has_access(self, user: User, resource: str) -> bool:
        pass

# Interface for incident reporting
class IIncidentReporter(ABC):
    @abstractmethod
    def report_incident(self, user: User, description: str) -> None:
        pass
```

infrastructure layer

In [3]:

```python
# Concrete authentication class
class SimpleAuthenticator(IAuthenticator):
    def __init__(self, users: List[User]):
        self.users = {user.username: user for user in users}

    def authenticate(self, username: str, password: str) -> User:
        user = self.users.get(username)
        if user and user.password == password:
            return user
        raise ValueError("Authentication failed.")

# Concrete access control class
class RoleBasedAccessControl(IAccessControl):
    def has_access(self, user: User, resource: str) -> bool:
        if user.role == "admin":
            return True
        if user.role == "employee" and resource in ["dashboard", "profile"]:
            return True
        return False

# Concrete incident reporting class
class IncidentLogger(IIncidentReporter):
    def __init__(self):
        self.incidents: List[Dict] = []

    def report_incident(self, user: User, description: str) -> None:
        incident = {"user": user.username, "description": description}
        self.incidents.append(incident)
        print(f"Incident reported by {user.username}: {description}")
```

application layer

In [4]:

```python
class SecuritySystem:
    def __init__(self, authenticator: IAuthenticator, access_control: IAccessControl, incident_reporter: IIncidentRepo
rter):
        self.authenticator = authenticator
        self.access_control = access_control
        self.incident_reporter = incident_reporter

    def login(self, username: str, password: str) -> User:
        return self.authenticator.authenticate(username, password)

    def check_access(self, user: User, resource: str) -> bool:
        return self.access_control.has_access(user, resource)

    def report_incident(self, user: User, description: str) -> None:
        self.incident_reporter.report_incident(user, description)
```

presentation layers

In [5]:

```
def main():
    users = [
        User("admin", "adminpass", "admin"),
        User("employee", "employeepass", "employee"),
    ]

    authenticator = SimpleAuthenticator(users)
    access_control = RoleBasedAccessControl()
    incident_reporter = IncidentLogger()

    security_system = SecuritySystem(authenticator, access_control, incident_reporter)

    # User authentication and access control demonstration
    try:
        user = security_system.login("employee", "employeepass")
        print(f"{user.username} logged in successfully.")

        if security_system.check_access(user, "dashboard"):
            print(f"{user.username} has access to the dashboard.")
        else:
            print(f"{user.username} does not have access to the dashboard.")

        # Reporting an incident
        security_system.report_incident(user, "Unauthorized access attempt detected.")
```

```python
    except ValueError as e:
        print(e)

if __name__ == "__main__":
    main()
```

output:

```
employee logged in successfully.
employee has access to the dashboard.
Incident reported by employee: Unauthorized access attempt detected.
```

THE END