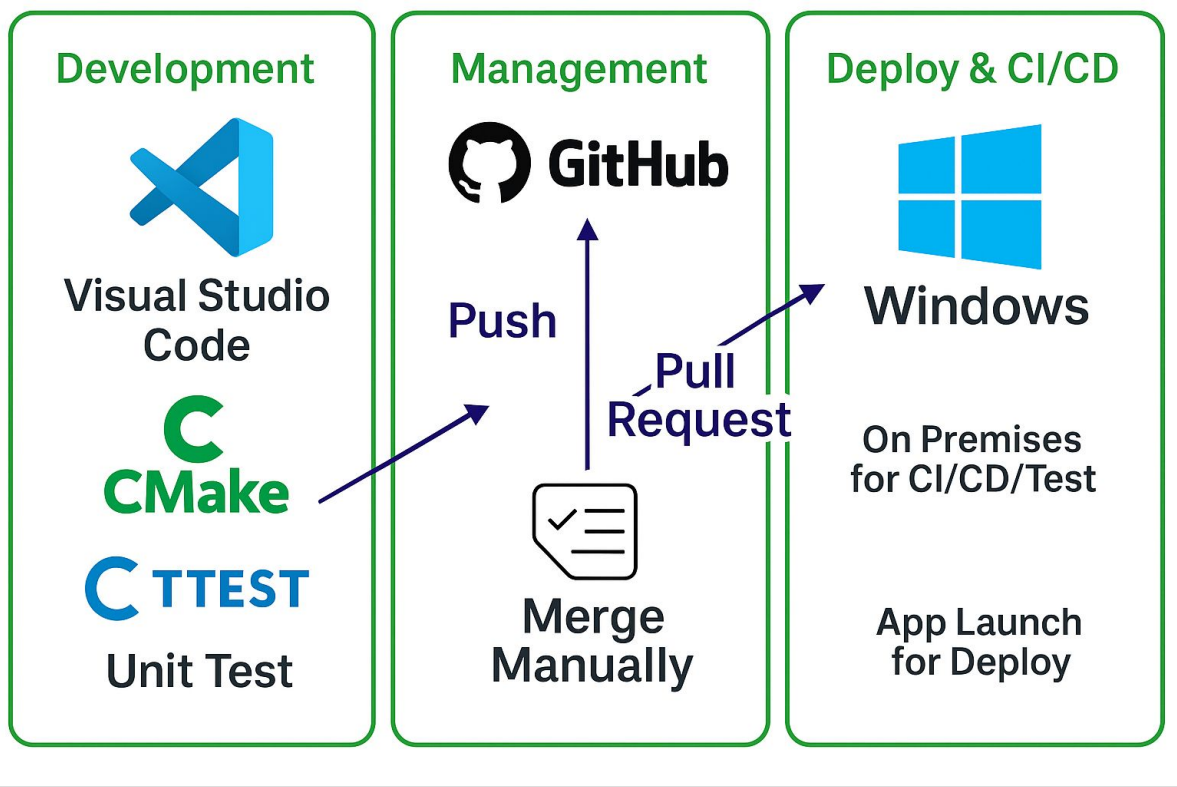
#4 - Constructure 2050

컴퓨터공학부 김진솔
컴퓨터공학부 백현준
컴퓨터공학부 이현수

1. 개발환경

로컬에서 **CMake** 기반 빌드 및 **GTest** 기반 단위 테스트를 수행. 기능 구현이 완료되면 **GitHub**에 브랜치를 푸시하고, **Pull Request**를 통해 코드 리뷰 및 테스트 결과 확인 후 **main** 브랜치에 병합



2. Unit Test Case 시나리오 및 결과

- persistence/inventoryRepository - 초기 재고리스트 저장 및 확인

```
46
47      //레퍼지토리 객체 생성 후 정했던 인벤토리(음료 + 수량) 내용 저장
48      inventoryRepository repo;
49      repo.setAllDrinks(inventoryList);
50
51      //저장된 리스트 반환
52      auto resultList = repo.getList();
53      EXPECT_EQ(resultList.size(), 20);
54
55      // 첫 번째 음료 검증
56      EXPECT_EQ(resultList[0].first, "콜라");
57      EXPECT_EQ(resultList[0].second, 10);
58
59      // 마지막 음료 검증
60      EXPECT_EQ(resultList[19].first, "칸타타");
61      EXPECT_EQ(resultList[19].second, 10);
62
63      // 중간 음료 검증
64      EXPECT_EQ(resultList[9].first, "비타500");
65      EXPECT_EQ(resultList[9].second, 10);
66
```

```
1: [----------] 1 test from InventoryRepositoryTest
1: [ RUN      ] InventoryRepositoryTest.SetAndGetAllDrinks
1: [       OK ] InventoryRepositoryTest.SetAndGetAllDrinks (0 ms)
1: [----------] 1 test from InventoryRepositoryTest (0 ms total)
```

2. Unit Test Case 시나리오 및 결과

- domain/inventory - 재고확인 관련

- 재고가 하나라도 있다면 **true,** 없다면 **false**를 반환하는지.
- 재고 수량에서 하나씩 **reduce** 했을 때 **isEmpty()** 값 체크

```
TEST(InventoryTest, IsEmpty) {
    Drink drink("환타", 1300, "D003");
    inventory inv1(drink, 0);                    //재고가 없을 때 반환 값 true
    ASSERT_TRUE(inv1.isEmpty());

    inventory inv2(drink, 2);                    //재고가 있을 때 반환 값 false
    ASSERT_FALSE(inv2.isEmpty());
}

TEST(InventoryTest, ReduceDrink) {
    Drink drink("몬스터", 2000, "D004");
    inventory inv(drink, 2);                     //재고가 2개에서 reduceDrink 후 재고확인 체크 -> false
    inv.reduceDrink();
    ASSERT_FALSE(inv.isEmpty());

    inv.reduceDrink();                       // 재고가 1개에서 reduceDrink 후 재고확인 체크 -> true
    ASSERT_TRUE(inv.isEmpty());
}
```

```
1: [ RUN      ] InventoryTest.Constructor
1: [       OK ] InventoryTest.Constructor (0 ms)
1: [ RUN      ] InventoryTest.GetDrink
1: [       OK ] InventoryTest.GetDrink (0 ms)
1: [ RUN      ] InventoryTest.IsEmpty
1: [       OK ] InventoryTest.IsEmpty (0 ms)
1: [ RUN      ] InventoryTest.ReduceDrink
1: [       OK ] InventoryTest.ReduceDrink (0 ms)
1: [----------] 4 tests from InventoryTest (0 ms total)
```

2. Unit Test Case 시나리오 및 결과

- domain/order - 주문 내용 생성 및 결제상태 반환

```cpp
// attachPrePay 메소드 테스트
TEST(OrderTest, AttachPrePay) {
    Drink drink("사이다", 1500, "D002");
    std::string code = "PRE001";

    Order order = Order().attachPrePay(drink, code);

    // attachPrePay로 생성된 주문이 올바른지 확인
    EXPECT_EQ(order.getDrink().getName(), "사이다");
    EXPECT_EQ(order.getDrink().getPrice(), 1500);
    EXPECT_EQ(order.getDrink().getCode(), "D002");
    EXPECT_EQ(order.getOrderID(), code);
    EXPECT_EQ(order.getPayStatus(), "Pending");
}
```

```cpp
// setStatus 메소드 테스트
TEST(OrderTest, SetStatus) {
    Order order;

    // 유효한 상태로 변경
    order.setStatus("Approved");
    EXPECT_EQ(order.getPayStatus(), "Approved");

    order.setStatus("Declined");
    EXPECT_EQ(order.getPayStatus(), "Declined");
```

```
1: [----------] 4 tests from OrderTest
1: [ RUN      ] OrderTest.DefaultConstructor
1: [       OK ] OrderTest.DefaultConstructor (0 ms)
1: [ RUN      ] OrderTest.ParameterizedConstructor
1: [       OK ] OrderTest.ParameterizedConstructor (0 ms)
1: [ RUN      ] OrderTest.AttachPrePay
1: [       OK ] OrderTest.AttachPrePay (0 ms)
1: [ RUN      ] OrderTest.SetStatus
1: [       OK ] OrderTest.SetStatus (0 ms)
1: [----------] 4 tests from OrderTest (0 ms total)
```

2. Unit Test Case 시나리오 및 결과

- domain/prepaymentCode - 인증코드 생성 및 사용가능한 인증코드인지 확인, 상태 **used**로 변경처리

```
// 생성자 테스트
TEST(PrepaymentCodeTest, Constructor) {
    PrepaymentCode code;
    // 생성자에서 code가 5자리이고 상태가 "Unused"인지 확인
    EXPECT_EQ(code.getCode().length(), 5);
    EXPECT_TRUE(PrepaymentCode::isUsable(code.getCode()));
}

// rand() 메소드 테스트
TEST(PrepaymentCodeTest, Rand) {
    PrepaymentCode code1;
    PrepaymentCode code2;
    // 두 개의 다른 코드가 생성되는지 확인
    EXPECT_NE(code1.getCode(), code2.getCode());
```

```
// rand() 메소드 테스트
TEST(PrepaymentCodeTest, Rand) {
    PrepaymentCode code1;
    PrepaymentCode code2;
    // 두 개의 다른 코드가 생성되는지 확인
    EXPECT_NE(code1.getCode(), code2.getCode());

    // 생성된 코드가 올바른 형식인지 확인
    std::string code = code1.getCode();
    EXPECT_EQ(code.length(), 5);
    for (char c : code) {
        EXPECT_TRUE(std::isalnum(c));
        if (std::isalpha(c)) {
            EXPECT_TRUE(std::isupper(c));
        }
    }
}

// isUsable() 메소드 테스트
TEST(PrepaymentCodeTest, IsUsable) {
    // 올바른 형식의 코드
    std::string validCode = "ABC12";
    EXPECT_TRUE(PrepaymentCode::isUsable(validCode));

    // 잘못된 길이
    std::string wrongLength = "ABC1";
    EXPECT_FALSE(PrepaymentCode::isUsable(wrongLength));

    // 소문자 포함
    std::string lowerCase = "abc12";
    EXPECT_FALSE(PrepaymentCode::isUsable(lowerCase));

    // 특수문자 포함
    std::string specialChar = "ABC!2";
    EXPECT_FALSE(PrepaymentCode::isUsable(specialChar));
}
```

```
// setStatus() 메소드 테스트
TEST(PrepaymentCodeTest, SetStatus) {
    PrepaymentCode code;
    std::string usedStatus = "Used";

    // 유효한 상태 변경
    code.setStatus(usedStatus);
    // 상태가 "Used"로 변경되었는지 확인
    EXPECT_EQ(usedStatus, code.getStatus());
```

```
1: [----------] 6 tests from PrepaymentCodeTest
1: [ RUN      ] PrepaymentCodeTest.Constructor
1: [       OK ] PrepaymentCodeTest.Constructor (0 ms)
1: [ RUN      ] PrepaymentCodeTest.Rand
1: [       OK ] PrepaymentCodeTest.Rand (0 ms)
1: [ RUN      ] PrepaymentCodeTest.IsUsable
1: [       OK ] PrepaymentCodeTest.IsUsable (0 ms)
1: [ RUN      ] PrepaymentCodeTest.Hold
1: [       OK ] PrepaymentCodeTest.Hold (0 ms)
1: [ RUN      ] PrepaymentCodeTest.SetStatus
1: [       OK ] PrepaymentCodeTest.SetStatus (0 ms)
1: [ RUN      ] PrepaymentCodeTest.GetCode
1: [       OK ] PrepaymentCodeTest.GetCode (0 ms)
1: [----------] 6 tests from PrepaymentCodeTest (0 ms total)
1:
```

2. Unit Test Case 시나리오 및 결과

- domain/prepaymentCode - 인증코드 생성 및 사용가능한 인증코드인지 확인, 상태 **used**로 변경처리

```
// 생성자 테스트
TEST(PrepaymentCodeTest, Constructor) {
    PrepaymentCode code;
    // 생성자에서 code가 5자리이고 상태가 "Unused"인지 확인
    EXPECT_EQ(code.getCode().length(), 5);
    EXPECT_TRUE(PrepaymentCode::isUsable(code.getCode()));
}

// rand() 메소드 테스트
TEST(PrepaymentCodeTest, Rand) {
    PrepaymentCode code1;
    PrepaymentCode code2;
    // 두 개의 다른 코드가 생성되는지 확인
    EXPECT_NE(code1.getCode(), code2.getCode());
```

```
// rand() 메소드 테스트
TEST(PrepaymentCodeTest, Rand) {
    PrepaymentCode code1;
    PrepaymentCode code2;
    // 두 개의 다른 코드가 생성되는지 확인
    EXPECT_NE(code1.getCode(), code2.getCode());

    // 생성된 코드가 올바른 형식인지 확인
    std::string code = code1.getCode();
    EXPECT_EQ(code.length(), 5);
    for (char c : code) {
        EXPECT_TRUE(std::isalnum(c));
        if (std::isalpha(c)) {
            EXPECT_TRUE(std::isupper(c));
        }
    }
}

// isUsable() 메소드 테스트
TEST(PrepaymentCodeTest, IsUsable) {
    // 올바른 형식의 코드
    std::string validCode = "ABC12";
    EXPECT_TRUE(PrepaymentCode::isUsable(validCode));

    // 잘못된 길이
    std::string wrongLength = "ABC1";
    EXPECT_FALSE(PrepaymentCode::isUsable(wrongLength));

    // 소문자 포함
    std::string lowerCase = "abc12";
    EXPECT_FALSE(PrepaymentCode::isUsable(lowerCase));

    // 특수문자 포함
    std::string specialChar = "ABC!2";
    EXPECT_FALSE(PrepaymentCode::isUsable(specialChar));
}
```

```
// setStatus() 메소드 테스트
TEST(PrepaymentCodeTest, SetStatus) {
    PrepaymentCode code;
    std::string usedStatus = "Used";

    // 유효한 상태 변경
    code.setStatus(usedStatus);
    // 상태가 "Used"로 변경되었는지 확인
    EXPECT_EQ(usedStatus, code.getStatus());
```

```
1: [----------] 6 tests from PrepaymentCodeTest
1: [ RUN      ] PrepaymentCodeTest.Constructor
1: [       OK ] PrepaymentCodeTest.Constructor (0 ms)
1: [ RUN      ] PrepaymentCodeTest.Rand
1: [       OK ] PrepaymentCodeTest.Rand (0 ms)
1: [ RUN      ] PrepaymentCodeTest.IsUsable
1: [       OK ] PrepaymentCodeTest.IsUsable (0 ms)
1: [ RUN      ] PrepaymentCodeTest.Hold
1: [       OK ] PrepaymentCodeTest.Hold (0 ms)
1: [ RUN      ] PrepaymentCodeTest.SetStatus
1: [       OK ] PrepaymentCodeTest.SetStatus (0 ms)
1: [ RUN      ] PrepaymentCodeTest.GetCode
1: [       OK ] PrepaymentCodeTest.GetCode (0 ms)
1: [----------] 6 tests from PrepaymentCodeTest (0 ms total)
1:
```

2. Unit Test Case 시나리오 및 결과

MessageTest 테스트 결과

```
Running main() from /home/dev/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 11 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 4 tests from MessageSenderTest
[ RUN      ] MessageSenderTest.BroadcastSendMultipleEndpoints
[Sender] warning: failed to send to 127.0.0.1:9000: connect: Connection refused
[Sender] warning: failed to send to 127.0.0.1:9001: connect: Connection refused
[       OK ] MessageSenderTest.BroadcastSendMultipleEndpoints (3 ms)
[ RUN      ] MessageSenderTest.UnicastSendToTarget
[Sender] warning: failed to send to 127.0.0.1:9001: connect: Connection refused
[       OK ] MessageSenderTest.UnicastSendToTarget (0 ms)
[ RUN      ] MessageSenderTest.CanSerializeAndSendToCorrectEndpoint
[Sender] warning: failed to send to localhost:9000: connect: Cannot assign requested address
[       OK ] MessageSenderTest.CanSerializeAndSendToCorrectEndpoint (9 ms)
[ RUN      ] MessageSenderTest.BroadcastMessageToAllEndpoints
[Sender] warning: failed to send to 127.0.0.1:9000: connect: Connection refused
[Sender] warning: failed to send to 127.0.0.1:9001: connect: Connection refused
[       OK ] MessageSenderTest.BroadcastMessageToAllEndpoints (0 ms)
[----------] 4 tests from MessageSenderTest (14 ms total)
```

-

2. Unit Test Case 시나리오 및 결과

- domain/prepaymentCodeRepository - 레퍼지토리에 인증코드 반복저장 및 검색

```
7
8   TEST(PrepaymentCodeRepositoryTest, MultipleSaves) {
9       PrepaymentCodeRepository repo;
10      std::vector<PrepaymentCode> codes;
11
12      // 여러 개의 코드 저장
13      for (int i = 0; i < 5; i++) {
14          PrepaymentCode code;
15          codes.push_back(code);
16          repo.save(code);
17      }
18
19      // 저장된 코드 중 하나를 선택하여 검증
20      EXPECT_TRUE(repo.isSameCode(codes[2].getCode()));
21  }
22
```

```
22
23  TEST(PrepaymentCodeRepositoryTest, InvalidCode) {
24      PrepaymentCodeRepository repo;
25
26      for (int i = 0; i < 5; i++) {
27          PrepaymentCode code;
28          repo.save(code);
29      }
30
31      // 존재하지 않는 코드 검색
32      EXPECT_FALSE(repo.isSameCode("INVALID"));
33      EXPECT_FALSE(repo.isSameCode(""));
34      EXPECT_FALSE(repo.isSameCode("ABC12"));  // 저장되지 않은 유효한 형식의 코드
35  }
```

```
1:
1: [----------] 2 tests from PrepaymentCodeRepositoryTest
1: [ RUN      ] PrepaymentCodeRepositoryTest.MultipleSaves
1: [       OK ] PrepaymentCodeRepositoryTest.MultipleSaves (0 ms)
1: [ RUN      ] PrepaymentCodeRepositoryTest.InvalidCode
1: [       OK ] PrepaymentCodeRepositoryTest.InvalidCode (0 ms)
1: [----------] 2 tests from PrepaymentCodeRepositoryTest (0 ms total)
1:
1: [----------] Global test environment tear-down
1: [==========] 19 tests from 6 test suites ran. (0 ms total)
1: [  PASSED  ] 19 tests.
1/1 Test #1: GoogleTests .....................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 1
```

2. Unit Test Case 시나리오 및 결과

- Service/OrderService - 결제 성공 여부(success)에 따라 주문 상태(status)가 **'Approved'** 또는 **'Declined'**로 정확히 설정되는지 검증하는 단위 테스트 수행

```
    Start 4: TestOrderService

4: Test command: /home/dev/build/test_order_service.out
4: Test timeout computed to be: 10000000
4: Running main() from /home/dev/lib/googletest/googletest/src/gtest_main.cc
4: [==========] Running 1 test from 1 test suite.
4: [----------] Global test environment set-up.
4: [----------] 1 test from OrderServiceTest
4: [ RUN      ] OrderServiceTest.ApproveHandlesTrueFalse
4: [승인 상태] Approved
4: [승인 상태] Declined
4: [       OK ] OrderServiceTest.ApproveHandlesTrueFalse (0 ms)
4: [----------] 1 test from OrderServiceTest (0 ms total)
4:
4: [----------] Global test environment tear-down
4: [==========] 1 test from 1 test suite ran. (0 ms total)
4: [  PASSED  ] 1 test.
4/5 Test #4: TestOrderService .................   Passed    0.10 sec
```

```cpp
// OrderService.cpp
// ------------------------------
#include "OrderService.hpp"
#include <iostream>

OrderService::OrderService() {}

void OrderService::approve(const std::string& paymentID, bool success) {
    if (success) {
        status = "Approved";
    } else {
        status = "Declined";
    }
    std::cerr << "[승인 상태] " << status << std::endl;
}

void OrderService::createOrder(const std::string& code) {
    drinkCode = code;
    std::cerr << "[주문 생성] DrinkCode: " << drinkCode << std::endl;
}

void OrderService::attachPrePay(const std::string& code) {
    prepayCode = code;
    std::cerr << "[선결제 코드 연결] Code: " << prepayCode << std::endl;
}
```

2. Unit Test Case 시나리오 및 결과

- Service/InventoryService - 자판기 내 음료의 재고 관리와 유효성 검사 기능을 담당

```
test 3
    Start 3: TestInventoryService

3: Test command: /home/dev/build/test_inventory_service.out
3: Test timeout computed to be: 10000000
3: Running main() from /home/dev/lib/googletest/googletest/src/gtest_main.cc
3: [==========] Running 2 tests from 1 test suite.
3: [----------] Global test environment set-up.
3: [----------] 2 tests from InventoryServiceTest
3: [ RUN      ] InventoryServiceTest.CallInventoryReturnsList
3: [       OK ] InventoryServiceTest.CallInventoryReturnsList (0 ms)
3: [ RUN      ] InventoryServiceTest.SaleValidReturnsTrue
3: [       OK ] InventoryServiceTest.SaleValidReturnsTrue (0 ms)
3: [----------] 2 tests from InventoryServiceTest (0 ms total)
3:
3: [----------] Global test environment tear-down
3: [==========] 2 tests from 1 test suite ran. (0 ms total)
3: [  PASSED  ] 2 tests.
3/5 Test #3: TestInventoryService .............   Passed    0.07 sec
```

test_inventory_service.cpp > TEST(InventoryServiceTest, CallInventoryReturnsList)

```cpp
#include <gtest/gtest.h>
#include "application/InventoryService.hpp"

TEST(InventoryServiceTest, CallInventoryReturnsList) {
    InventoryService service;
    auto result = service.CallInventorySer();
    EXPECT_TRUE(result.empty());
}

TEST(InventoryServiceTest, SaleValidReturnsTrue) {
    InventoryService service;
    EXPECT_TRUE(service.getSaleValid("콜라"));
}
```

2. Unit Test Case 시나리오 및 결과

- Service/DistanceService - 가장 가까운 자판기 탐색 유닛 테스트

test_distance_service.cpp > ...

```cpp
#include <gtest/gtest.h>
#include "application/DistanceService.hpp"

TEST(DistanceServiceTest, ReturnsNearestWhenAllDifferent) {
    std::vector<VMInfo> machines = {
        {"VM003", {10.0, 10.0}},
        {"VM002", {5.0, 5.0}},
        {"VM001", {1.0, 1.0}}  // 가장 가까움
    };

    DistanceService service;
    std::string nearest = service.findNearest(0.0, 0.0, machines);

    EXPECT_EQ(nearest, "VM001");
}

TEST(DistanceServiceTest, ChoosesSmallerIdWhenDistancesAreEqual) {
    std::vector<VMInfo> machines = {
        {"VM010", {1.0, 1.0}},
        {"VM005", {-1.0, -1.0}}  // 거리 같고 ID 작음 → 선택됨
    };

    DistanceService service;
    std::string nearest = service.findNearest(0.0, 0.0, machines);

    EXPECT_EQ(nearest, "VM005");
}

TEST(DistanceServiceTest, ReturnsEmptyStringWhenNoMachines) {
    std::vector<VMInfo> machines;

    DistanceService service;
    std::string nearest = service.findNearest(0.0, 0.0, machines);

    EXPECT_EQ(nearest, "");
}
```

```
test 1
    Start 1: TestDistanceService

1: Test command: /home/dev/build/test_distance_service.out
1: Test timeout computed to be: 10000000
1: Running main() from /home/dev/lib/googletest/googletest/src/gtest_main.cc
1: [==========] Running 3 tests from 1 test suite.
1: [----------] Global test environment set-up.
1: [----------] 3 tests from DistanceServiceTest
1: [ RUN      ] DistanceServiceTest.ReturnsNearestWhenAllDifferent
1: [거리 계산] 가장 가까운 자판기 ID: VM001, 거리: 1.41421
1: [       OK ] DistanceServiceTest.ReturnsNearestWhenAllDifferent (0 ms)
1: [ RUN      ] DistanceServiceTest.ChoosesSmallerIdWhenDistancesAreEqual
1: [거리 계산] 가장 가까운 자판기 ID: VM005, 거리: 1.41421
1: [       OK ] DistanceServiceTest.ChoosesSmallerIdWhenDistancesAreEqual (0 ms)
1: [ RUN      ] DistanceServiceTest.ReturnsEmptyStringWhenNoMachines
1: [거리 계산] 가장 가까운 자판기 ID: , 거리: 1.79769e+308
1: [       OK ] DistanceServiceTest.ReturnsEmptyStringWhenNoMachines (0 ms)
1: [----------] 3 tests from DistanceServiceTest (0 ms total)
1:
1: [----------] Global test environment tear-down
1: [==========] 3 tests from 1 test suite ran. (0 ms total)
1: [  PASSED  ] 3 tests.
1/5 Test #1: TestDistanceService .............   Passed    0.08 sec
```

2. Unit Test Case 시나리오 및 결과

- MessageService::broadcastStock()
  호출 시 send가 정상 수행되는지 확인

```cpp
// MessageService::broadcastStock() 호출 시 send가 정상 수행되는지 확인
#include <gtest/gtest.h>
#include "network/MessageSender.hpp"
#include "network/MessageReceiver.hpp"
#include "service/MessageService.hpp"
#include "service/ErrorService.hpp"
#include "service/InventoryService.hpp"
#include "service/PrepaymentService.hpp"
#include "domain/drink.h"
#include "domain/inventory.h"
#include "persistence/OvmAddressRepository.hpp"

TEST(MessageServiceTest, BroadcastStockTest) {
    boost::asio::io_context io;

    // 가짜 repo, 에러서비스, 서비스
    persistence::OvmAddressRepository repo;
    service::ErrorService errSvc;
    service::InventoryService invSvc;
    service::PrepaymentService prepaySvc;
    domain::Drink d("콜라", 1500, "D005");
    domain::inventory drink(d, 10);

    std::vector<std::string> eps = {"127.0.0.1:9000"};
    std::unordered_map<std::string, std::string> id_map = {{"T1", "127.0.0.1:9000"}};
    network::MessageSender sender(io, eps, id_map);
    network::MessageReceiver receiver(io, 9000);

    application::MessageService msgSvc(sender, receiver, repo, errSvc, invSvc, prepaySvc, drink);

    EXPECT_NO_THROW(msgSvc.broadcastStock(d));
}
```

- MessageService::broadcastStock()
  호출 시 메시지 내용 검증

```cpp
// MessageService::broadcastStock() 호출 시 send가 정상 수행되는지 확인
#include <gtest/gtest.h>
#include "network/MessageSender.hpp"
#include "network/MessageReceiver.hpp"
#include "service/MessageService.hpp"
#include "service/ErrorService.hpp"
#include "service/InventoryService.hpp"
#include "service/PrepaymentService.hpp"
#include "domain/drink.h"
#include "domain/inventory.h"
#include "persistence/OvmAddressRepository.hpp"

TEST(MessageServiceTest, BroadcastStockTest) {
    boost::asio::io_context io;

    // 가짜 repo, 에러서비스, 서비스
    persistence::OvmAddressRepository repo;
    service::ErrorService errSvc;
    service::InventoryService invSvc;
    service::PrepaymentService prepaySvc;
    domain::Drink d("콜라", 1500, "D005");
    domain::inventory drink(d, 10);

    std::vector<std::string> eps = {"127.0.0.1:9000"};
    std::unordered_map<std::string, std::string> id_map = {{"T1", "127.0.0.1:9000"}};
    network::MessageSender sender(io, eps, id_map);
    network::MessageReceiver receiver(io, 9000);

    application::MessageService msgSvc(sender, receiver, repo, errSvc, invSvc, prepaySvc, drink);

    EXPECT_NO_THROW(msgSvc.broadcastStock(d));
}
```

2. Unit Test Case 시나리오 및 결과

- **MessageSender**가 단일 메시지를 올바르게 직렬화하고 전송하는지 확인

```cpp
#include <gtest/gtest.h>
#include <boost/asio/io_context.hpp>
#include "network/MessageSender.hpp"
#include "network/MessageSerializer.hpp"
#include <sstream>

using namespace network;

TEST(MessageSenderTest, CanSerializeAndSendToCorrectEndpoint) {
    boost::asio::io_context io;
    std::vector<std::string> endpoints = {"localhost:9000"};
    std::unordered_map<std::string, std::string> id_map = {{"T5", "localhost:9000"}};

    MessageSender sender(io, endpoints, id_map);

    Message msg;
    msg.msg_type = Message::Type::REQ_STOCK;
    msg.src_id = "T1";
    msg.dst_id = "T5";
    msg.msg_content = {{"item_code", "D001"}, {"item_num", "01"}};

    // 실제 sendOne은 네트워크 연결을 시도하므로 테스트에선 실패만 안 나면 통과
    EXPECT_NO_THROW(sender.send(msg));
}
```

- **MessageReceiver**가 핸들로 등록을 정상 처리하는지 확인

tests > network > test_MessageReceiver_subscribe.cpp > ...

```cpp
// 특정 메시지 타입에 대한 핸들러가 정상 등록되는지 테스트
#include <gtest/gtest.h>
#include <boost/asio.hpp>
#include "network/MessageReceiver.hpp"

TEST(MessageReceiverTest, SubscribeHandler) {
    boost::asio::io_context io;
    unsigned short port = 9002;
    network::MessageReceiver receiver(io, port);

    bool called = false;

    receiver.subscribe(network::Message::Type::REQ_STOCK, [&](const network::Message& m
        called = true;
    });

    // 핸들러 등록 확인용 - 내부 구조를 볼 수는 없지만 호출 구조 오류가 없는지만 본다
    EXPECT_NO_THROW(receiver.start());
}
```

2. Unit Test Case 시나리오 및 결과

- 브로드 캐스트메시지 요청 메시지 구조 확인

```cpp
#include <gtest/gtest.h>
#include "service/MessageService.hpp"
#include "domain/drink.h"

TEST(MessageServiceTest, BroadcastStockCreatesCorrectMessage) {
    domain::Drink drink("콜라", 1500, "D001");

    network::Message msg;
    msg.msg_type = network::Message::Type::REQ_STOCK;
    msg.dst_id = "0";
    msg.msg_content = {{"item_code", drink.getCode()}, {"item_num", "01"}};

    EXPECT_EQ(msg.msg_content.at("item_code"), "D001");
    EXPECT_EQ(msg.dst_id, "0");
}
```

- sendPrePayReq()메시지 생성로직 확인

```cpp
#include <gtest/gtest.h>
#include "domain/drink.h"
#include "domain/order.h"
#include "network/message.hpp"

TEST(MessageServiceTest, PrepayRequestMessageFields) {
    domain::Drink drink("사이다", 1500, "D002");
    domain::Order order("T2", drink, 2, "ABC123", "Pending");

    network::Message msg;
    msg.msg_type = network::Message::Type::REQ_PREPAY;
    msg.dst_id = order.vmId();
    msg.msg_content = {
        {"item_code", order.drink().getCode()},
        {"item_num", std::to_string(order.quantity())},
        {"cert_code", order.certCode()}
    };

    EXPECT_EQ(msg.msg_content.at("item_code"), "D002");
    EXPECT_EQ(msg.msg_content.at("cert_code"), "ABC123");
    EXPECT_EQ(msg.dst_id, "T2");
}
```

2. Unit Test Case 시나리오 및 결과

- respondPreqayReq()재고있음

```
#include <gtest/gtest.h>
#include "domain/drink.h"
#include "domain/order.h"
#include "network/message.hpp"

TEST(MessageServiceTest, PrepayResponseSuccessMessage) {
    domain::Drink drink("몬스터", 2500, "D005");
    domain::Order order("T2", drink, 1, "CERT123", "Pending");

    network::Message msg;
    msg.msg_type = network::Message::Type::RESP_PREPAY;
    msg.dst_id = order.vmId();
    msg.msg_content = {
        {"availability", "T"},
        {"item_num", std::to_string(order.quantity())},
        {"item_code", drink.getCode()}
    };

    EXPECT_EQ(msg.msg_content.at("availability"), "T");
    EXPECT_EQ(msg.msg_content.at("item_code"), "D005");
}
```

-

- respondPreqayReq()재고없음

```
#include <gtest/gtest.h>
#include "network/message.hpp"

TEST(MessageServiceTest, PrepayResponseFailMessage) {
    network::Message msg;
    msg.msg_type = network::Message::Type::RESP_PREPAY;
    msg.msg_content = {
        {"availability", "F"},
        {"item_num", "2"},
        {"item_code", "D009"}
    };

    EXPECT_EQ(msg.msg_content.at("availability"), "F");
    EXPECT_EQ(msg.msg_content.at("item_code"), "D009");
}
```

2. Unit Test Case 시나리오 및 결과

- handlePesaPrepay() T수신 후 pending 초기화

sts > network > msg_service_test_5.cpp > ...

```cpp
#include <gtest/gtest.h>
#include "network/message.hpp"

TEST(MessageServiceTest, HandleRespPrepay_ClearsPendingOnSuccess) {
    bool pendingExists = true; // 시뮬레이션

    network::Message msg;
    msg.msg_type = network::Message::Type::RESP_PREPAY;
    msg.msg_content = {
        {"availability", "T"},
        {"item_code", "D003"},
        {"item_num", "1"}
    };

    if (msg.msg_content.at("availability") == "T") {
        pendingExists = false; // 초기화 가정
    }

    EXPECT_FALSE(pendingExists);
}
```

2. Unit Test Case 시나리오 및 결과

MessageTest 테스트 결과

```
Running main() from /home/dev/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 11 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 4 tests from MessageSenderTest
[ RUN      ] MessageSenderTest.BroadcastSendMultipleEndpoints
[Sender] warning: failed to send to 127.0.0.1:9000: connect: Connection refused
[Sender] warning: failed to send to 127.0.0.1:9001: connect: Connection refused
[       OK ] MessageSenderTest.BroadcastSendMultipleEndpoints (3 ms)
[ RUN      ] MessageSenderTest.UnicastSendToTarget
[Sender] warning: failed to send to 127.0.0.1:9001: connect: Connection refused
[       OK ] MessageSenderTest.UnicastSendToTarget (0 ms)
[ RUN      ] MessageSenderTest.CanSerializeAndSendToCorrectEndpoint
[Sender] warning: failed to send to localhost:9000: connect: Cannot assign requested address
[       OK ] MessageSenderTest.CanSerializeAndSendToCorrectEndpoint (9 ms)
[ RUN      ] MessageSenderTest.BroadcastMessageToAllEndpoints
[Sender] warning: failed to send to 127.0.0.1:9000: connect: Connection refused
[Sender] warning: failed to send to 127.0.0.1:9001: connect: Connection refused
[       OK ] MessageSenderTest.BroadcastMessageToAllEndpoints (0 ms)
[----------] 4 tests from MessageSenderTest (14 ms total)
```

-

2. Unit Test Case 시나리오 및 결과

MessageTest 테스트 결과

```
[----------] 6 tests from MessageServiceTest
[ RUN      ] MessageServiceTest.BroadcastStockCreatesCorrectMessage
[       OK ] MessageServiceTest.BroadcastStockCreatesCorrectMessage (0 ms)
[ RUN      ] MessageServiceTest.PrepayRequestMessageFields
[       OK ] MessageServiceTest.PrepayRequestMessageFields (0 ms)
[ RUN      ] MessageServiceTest.PrepayResponseSuccessMessage
[       OK ] MessageServiceTest.PrepayResponseSuccessMessage (0 ms)
[ RUN      ] MessageServiceTest.PrepayResponseFailMessage
[       OK ] MessageServiceTest.PrepayResponseFailMessage (0 ms)
[ RUN      ] MessageServiceTest.HandleRespPrepay_ClearsPendingOnSuccess
[       OK ] MessageServiceTest.HandleRespPrepay_ClearsPendingOnSuccess (0 ms)
[ RUN      ] MessageServiceTest.BroadcastStockTest
[       OK ] MessageServiceTest.BroadcastStockTest (0 ms)
[----------] 6 tests from MessageServiceTest (1 ms total)

[----------] 1 test from MessageReceiverTest
[ RUN      ] MessageReceiverTest.SubscribeHandler
[       OK ] MessageReceiverTest.SubscribeHandler (0 ms)
[----------] 1 test from MessageReceiverTest (0 ms total)

[----------] Global test environment tear-down
[==========] 11 tests from 3 test suites ran. (17 ms total)
[  PASSED  ] 11 tests.
root@fb2ec081374c:/home/dev/build# ctest -V
```

-

3. OOD에서 수정된 부분 정리

Activity 2033. 기존 Define Domain Model

Drink	
name	String
price	Float
stock	Integer

AuthCode	
code	String
expiry	Date

VendingMachine	
id	String
location	String

CardReader	
readerID	String

BroadcastMessage	
messageId	String
targetDrink	String

Order	
id	String
SelectedDrink	Drink
time	Date

Payment	
paymentId	String
status	String
method	String

CardSystem	
SystemName	String

otherVendingMachine	
id	String
location	String

InventoryStatus	
status	String
availableQty	Integer

3. OOD에서 수정된 부분 정리

Activity 2033. Define Domain Model

Drink	
drinkCode	String
name	String
price	Int

Order	
vmid	String
drink	Drink
qty	Int
certcode	String
paystatus	String

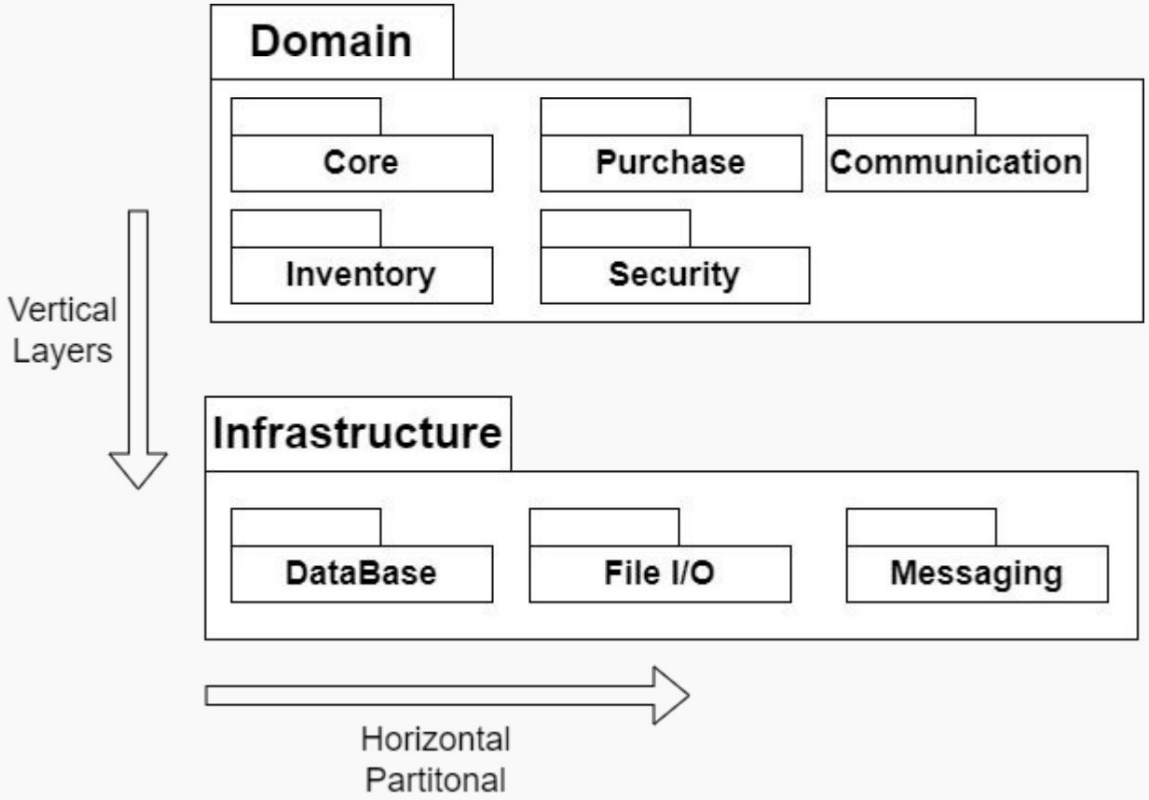
PrePaymentCode	
code	String
status	CodeStatus
heldorder	Order

Inventory	
drink	Drink
qty	Int

VendingMachine	
id	String
location	<int,int>
port	String

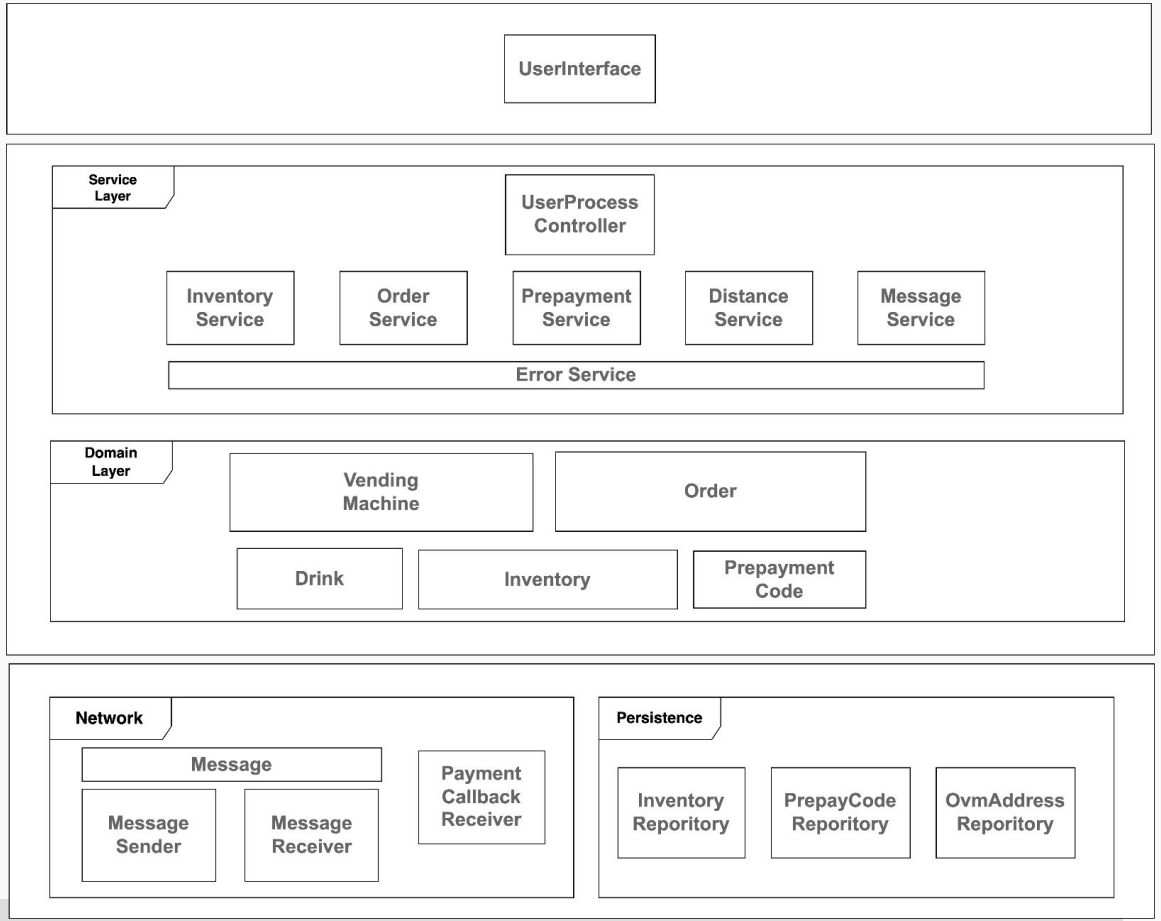
3. OOD에서 수정된 부분 정리

Activity 2043.
기존 **Refine System Archi tecture**



3. OOD에서 수정된 부분 정리

Activity 2043. Refine System Architecture



3. OOD에서 수정된 부분 정리

Activity 2045. Define Design Class Diagrams