프로젝트 발표

STM32F746을 이용한 Cafe Order System 구현

KOSTA 290기 노현
24.11.04 - 12.09

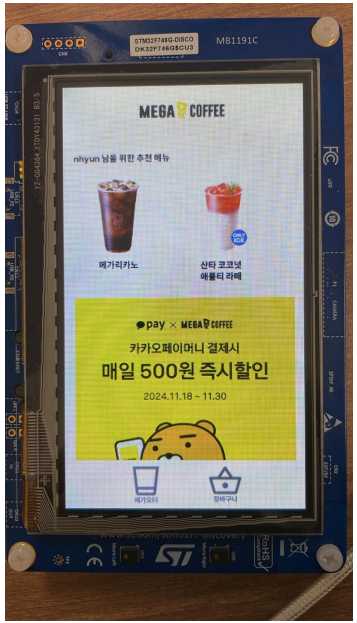
목 차

프로젝트 개요

1. Cafe Order System
2. 하드웨어
3. 소프트웨어

프로젝트 개요 - Cafe Order System

실제 어플 사진



구현한 사진

MEGA COFFEE

※ 카페 오더 어플리케이션

일상 생활에서 빠르게 커피를 주문할 수 있는 시스템
메가 커피 브랜드의 오더 앱을 카피하여 제작

프로젝트 개요 : 하드웨어

STM32F746-Discovery

※ 특징

ARM Cortex-M7 기반 MCU
480 X 272 해상도의 디스플레이
내부 메모리 : Flash, SRAM
외부 메모리 : SDRAM, QSPI(N25Q128A)
다양한 인터페이스(GPIO, UART, I2C, SPI 등)를 지원하여 확장성 제공

프로젝트 개요 : 하드웨어

RaspberryPi

Raspberry Pi 5

※ 특징

ARM Cortex-A76 기반의 싱글 보드 컴퓨터
라즈베리 파이 OS(라즈비안, 리눅스-데비안 기반)
소형, 저전력으로 임베디드 어플리케이션과 및 서버에 적합

프로젝트 개요 : 소프트웨어

TouchGFX

UI 설계 도구

TouchGFX Designer

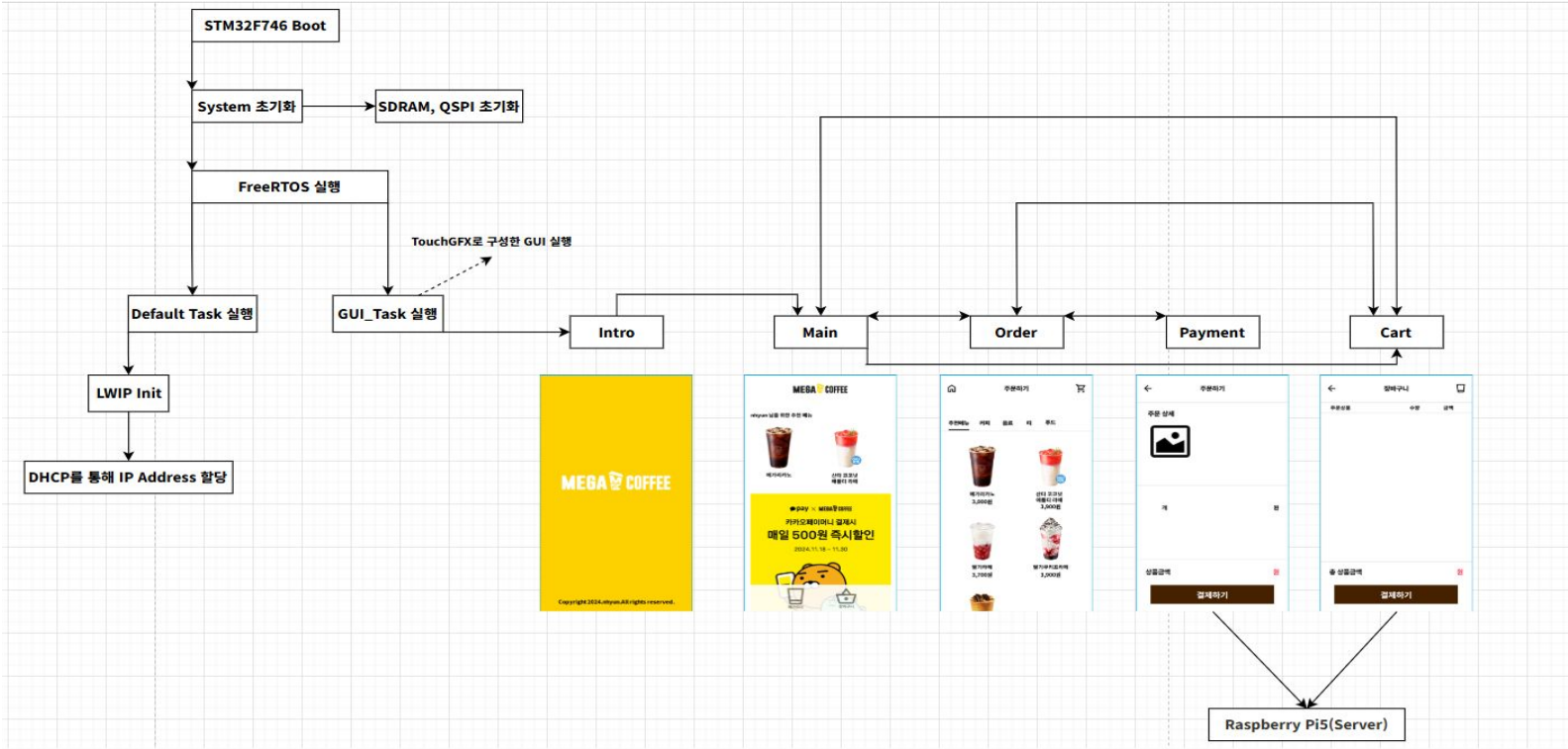
※ 특징
STM32 보드의 GUI 설계를 지원하는 툴
다양한 규격, 종류의 보드에 대응
Cube IDE와 연동하여 자동으로 코드 생성

STM32 CubeIDE

코드 생성 및 통합 개발 환경

C / C++, STM32 Cube IDE

Flow Chart


STM32F746 Boot
System 초기화
SDRAM, QSPI 초기화
FreeRTOS 실행
TouchGFX로 구성한 GUI 실행
Default Task 실행
GUI_Task 실행
Intro
Main
Order
Payment
Cart
LWIP Init
DHCP를 통해 IP Address 할당
MEGA COFFEE
매일 500원 즉시할인
결제하기
Raspberry Pi5(Server)

프로젝트 상세 : 프로젝트 목표

※ 프로젝트 목표

1. STM32 보드에 대한 이해와 STMCubeIde를 이용한 프로그램 설계
   (*하드웨어 이해 및 펌웨어 개발에 대한 경험*)

2. TouchGFX를 통한 UI / UX 설계 경험
   (*실제 UI 설계 및 사용자 경험 최적화*)

3. LWIP, C/C++, Socket API를 통한 네트워크 통신 학습
   (*서버 - 클라이언트 구조 구현*)

4. 위 세 가지 목표를 통합한 프로젝트 결과물 도출 (*통합 시스템 완성*)

프로젝트 상세 : Borad Boot

- STM32F746 Board Boot / main.c

전원 연결을 통해 보드가 부트됨
System 초기화 및 FreeRTOS 실행 → Default Task, GUI_Task 실행

```
MX_FMC_Init();
MX_I2C1_Init();
MX_I2C3_Init();
MX_LTDC_Init();
MX_QUADSPI_Init();
```

Default Task → LWIP.Init(); → 보드에 IP 주소 할당
GUI_Task → TouchGFX로 설계한 GUI 실행

```
void StartDefaultTask(void const * argument)
```

```
__weak void TouchGFX_Task(void const * argument)
```

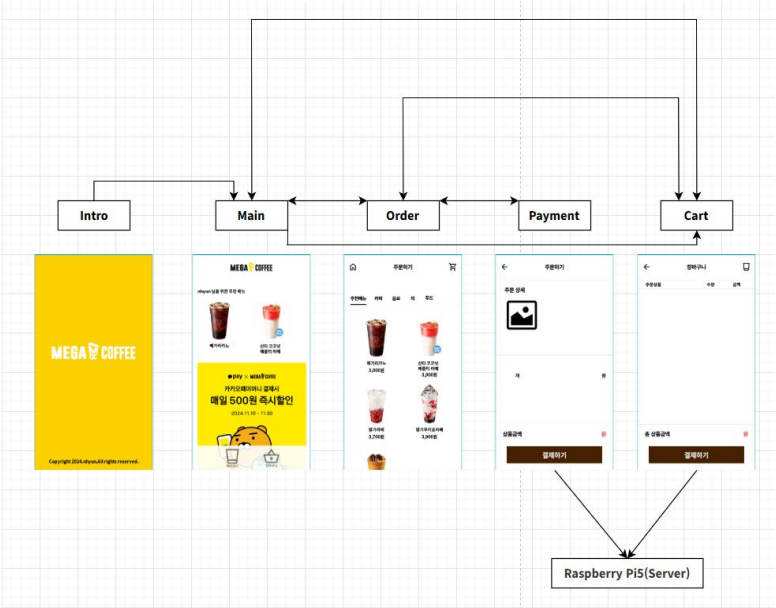
※ LWIP : Lightweight IP, 임베디드 시스템을 위한 경량화된 TCP/IP 네트워크 스택
메모리와 리소스가 제한된 환경에서 인터넷 프로토콜 기능을 제공하도록 설계.

프로젝트 상세 : 스크린 구성

- 스크린 구성

제작한 프로그램은 Intro, Main, Order, NowPurchase, Cart,
총 5개의 스크린으로 구성되어있으며 서로 상호작용한다.

각 스크린은 독립적이며, 스크린이 전환되면
스크린에서 사용한 데이터는 저장되지 않는다.
이를 보완하기 위해 MVP 패턴을 사용할 수 있다.



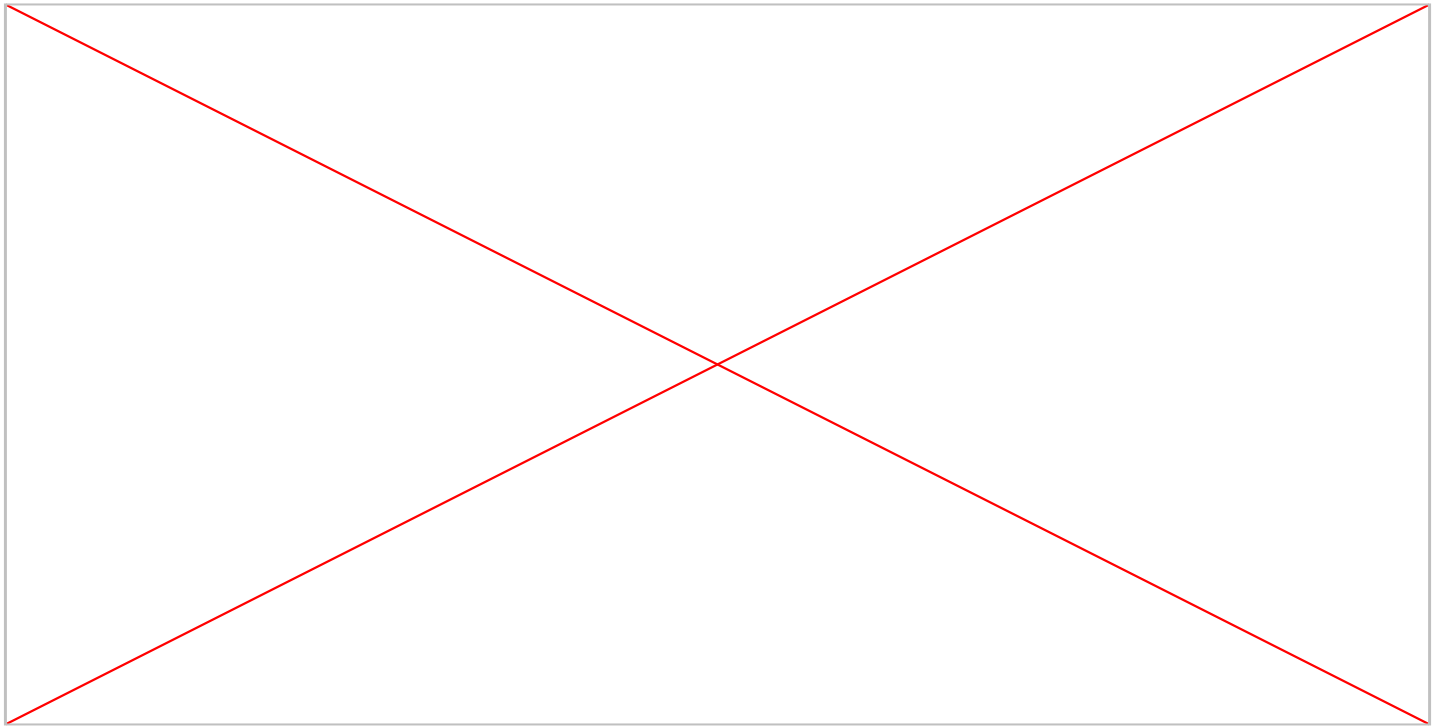
※ MVP : Model - View - Presenter 구조로 이루어진 Design Pattern이다.

프로젝트 상세 : Intro

- Intro 스크린

보드 부트 후 바로 전환되는 스크린이다.
사용자가 MEGA COFFE에 관련된 어플임을 직관적으로 알 수 있는 스크린이다.
약 3초 후 Main 스크린으로 전환된다.

프로젝트 상세 : Intro 스크린 실행



프로젝트 상세 : Main

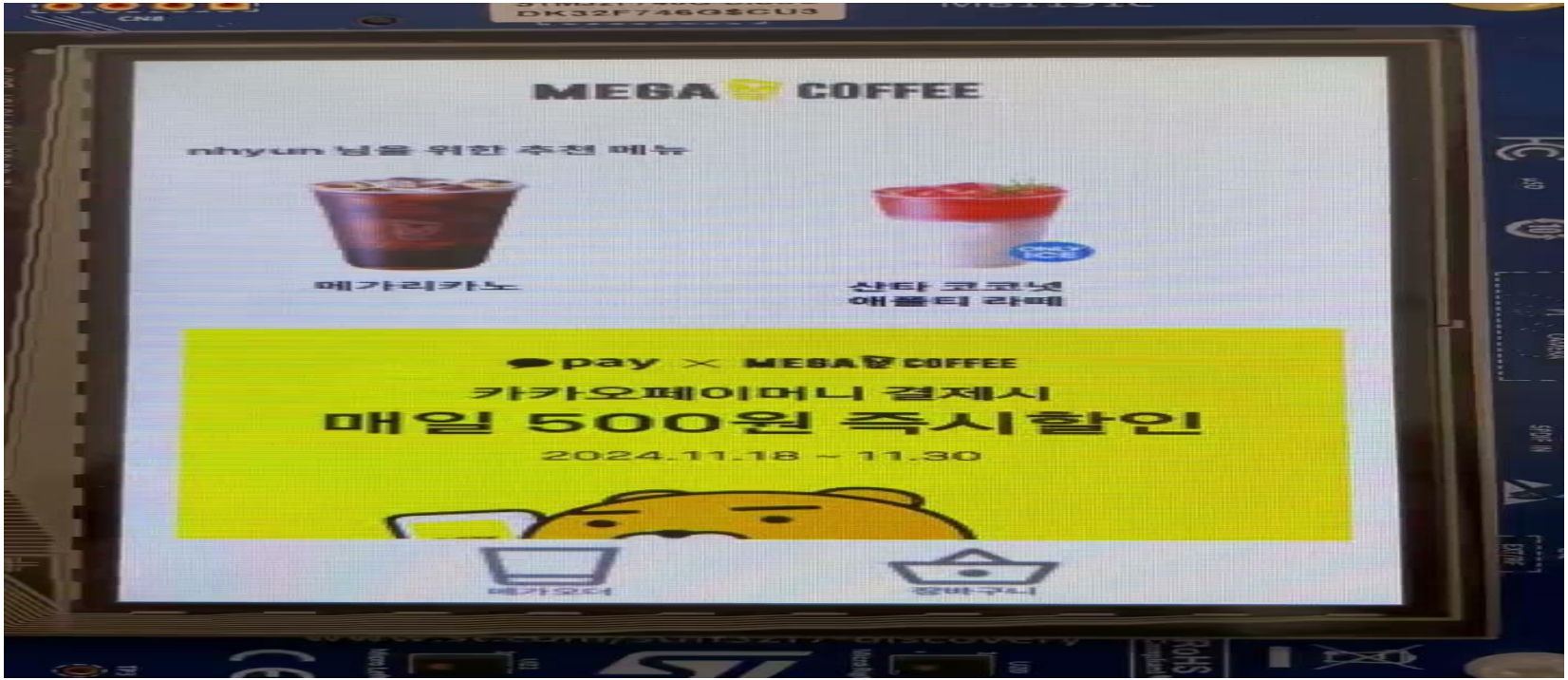
- Main 스크린

추천 메뉴 항목

이벤트 페이지 항목

네비게이션 버튼 항목으로
Order, Cart 스크린으로 전환

프로젝트 상세 : Main 스크린 실행



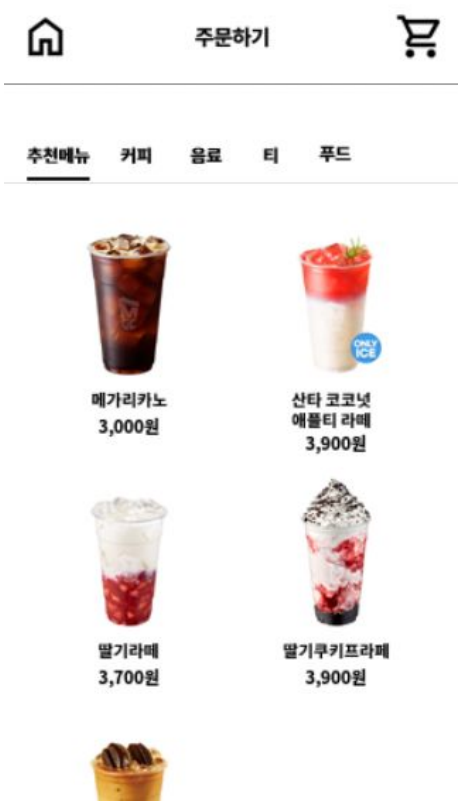
프로젝트 상세 : Order

- **Order 스크린**

Main, Cart 스크린 전환 버튼 항목

카테고리 선택 버튼 항목 - 카테고리별 메뉴버튼 전환

메뉴 버튼 항목 - Order Detail 컨테이너 활성화



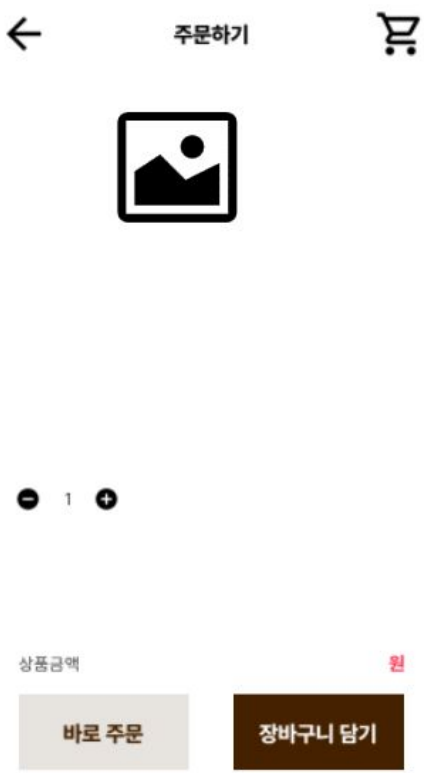
프로젝트 상세 : Order

- **Order Detail**

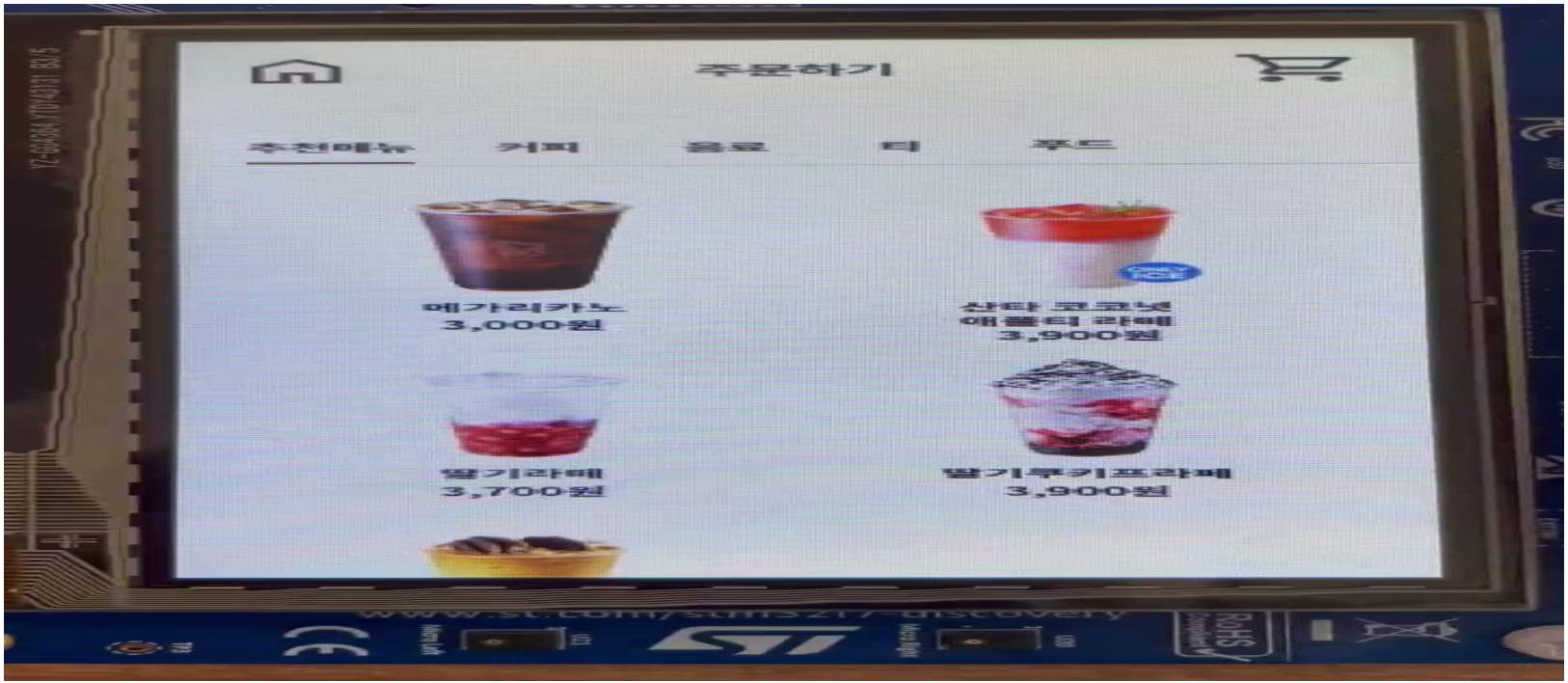
메뉴 버튼 누를 시 해당하는 메뉴의 사진, 설명, 금액으로
화면이 갱신되어 활성화

상단 Back 버튼으로 비활성화, 카트 버튼으로 Cart 스크린으로 전환

바로 주문 버튼으로 NowPurchase 스크린으로 전환
장바구니 담기 버튼으로 Model로 메뉴 데이터 전송



프로젝트 상세 : Order 스크린 실행



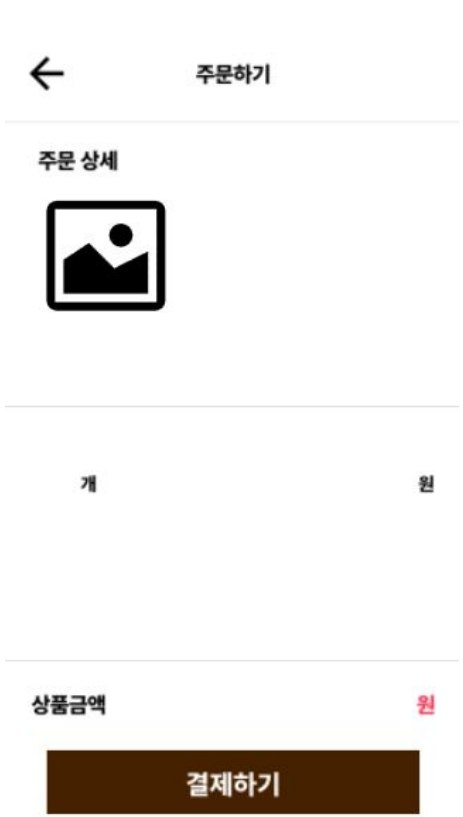
프로젝트 상세 : NowPurchase

- NowPurchase 스크린

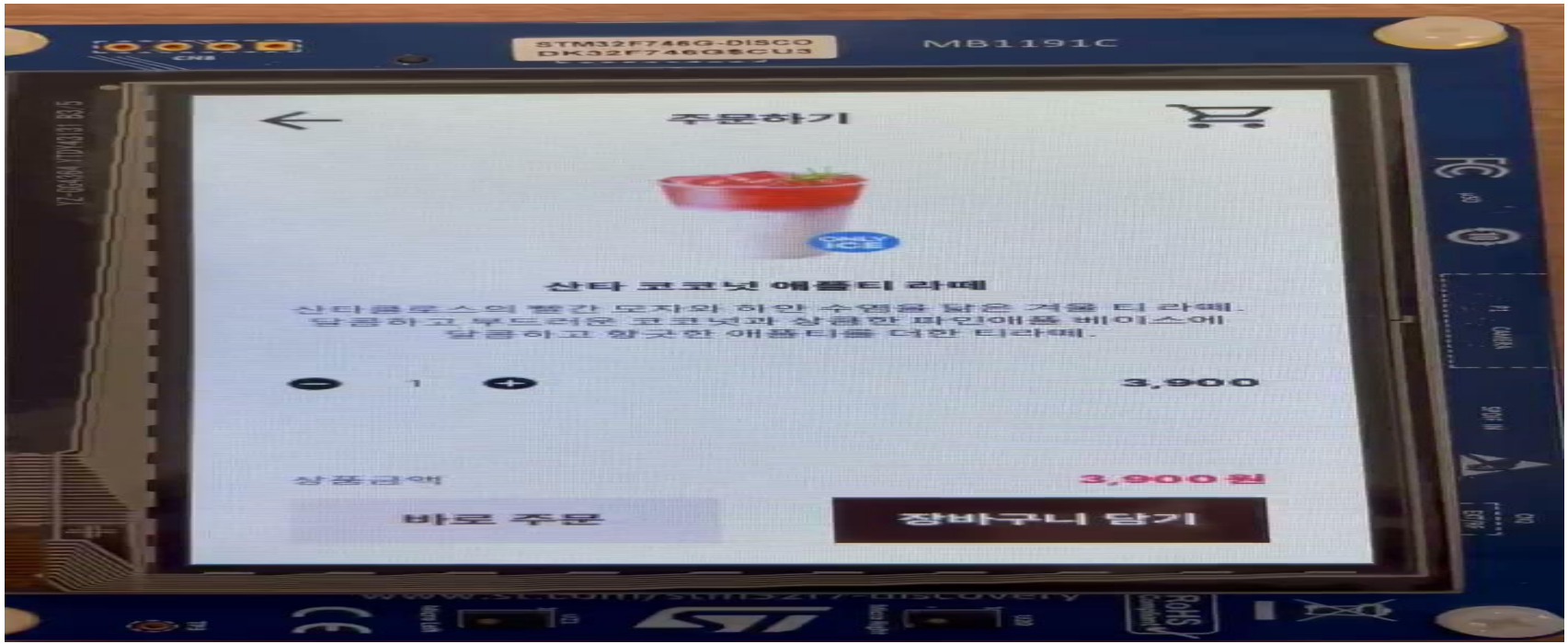
Back버튼으로 Order 스크린으로 전환

이전 과정에 선택한 메뉴버튼에 해당하는 데이터로 갱신됨

결제하기 버튼 클릭 시 결제 여부 활성화
“예” 버튼 클릭 시 서버로 주문 내역 전송



프로젝트 상세 : NowPurchase 스크린 실행



프로젝트 상세 : NowPurchase - Server

프로젝트 상세 : Cart

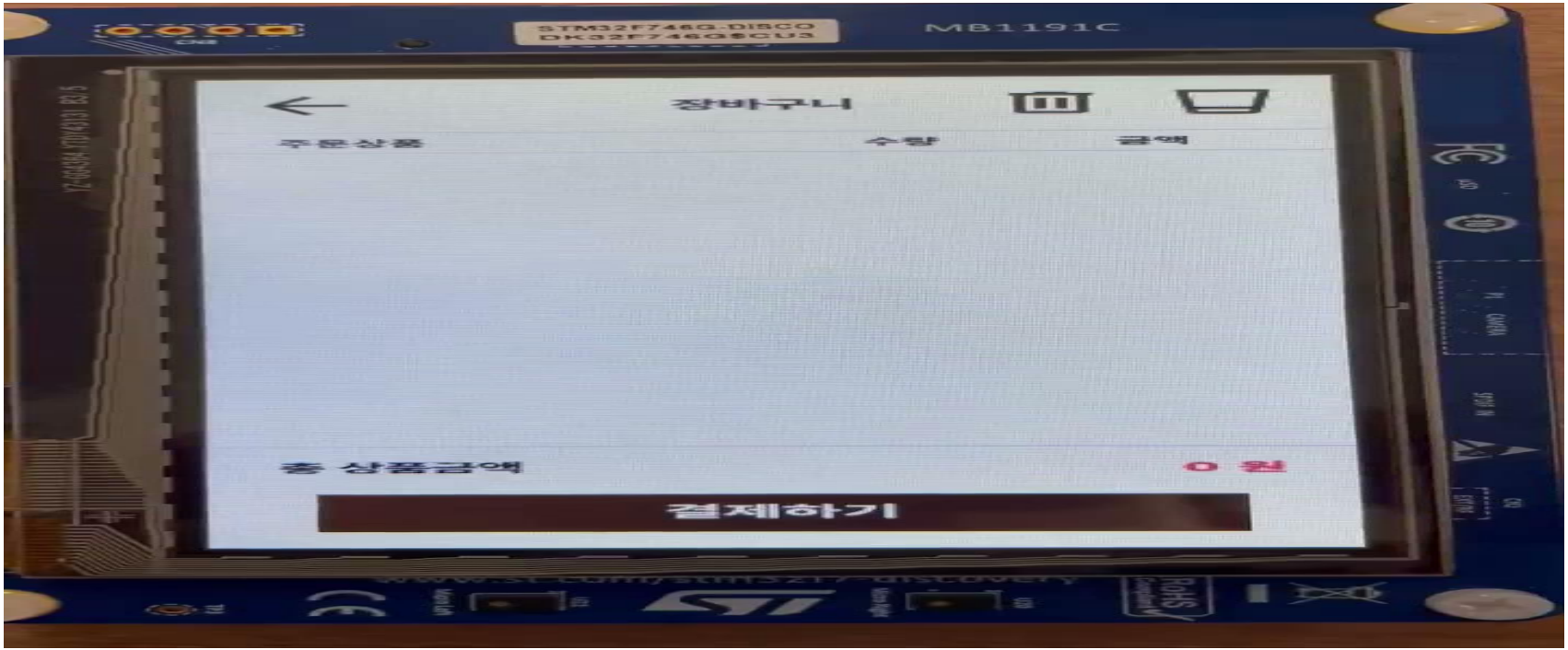
- Cart 스크린

Back버튼으로 Menu 스크린으로 전환
Delete 버튼으로 장바구니 데이터 초기화
Order 버튼으로 Order 스크린으로 전환

이전 과정에 선택한 메뉴버튼에 해당하는 데이터로 갱신됨

결제하기 버튼 클릭 시 결제 여부 활성화
“예” 버튼 클릭 시 서버로 주문 내역 전송

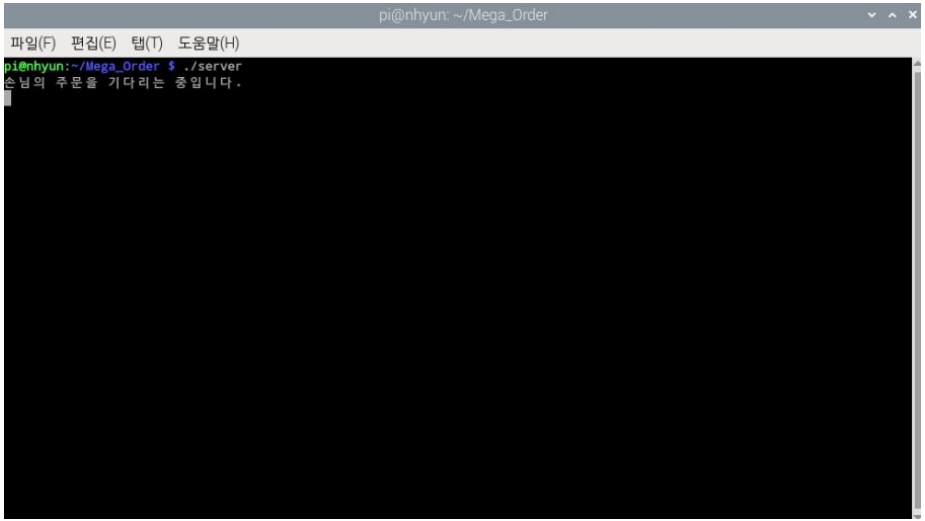
프로젝트 상세 : Cart 스크린 실행



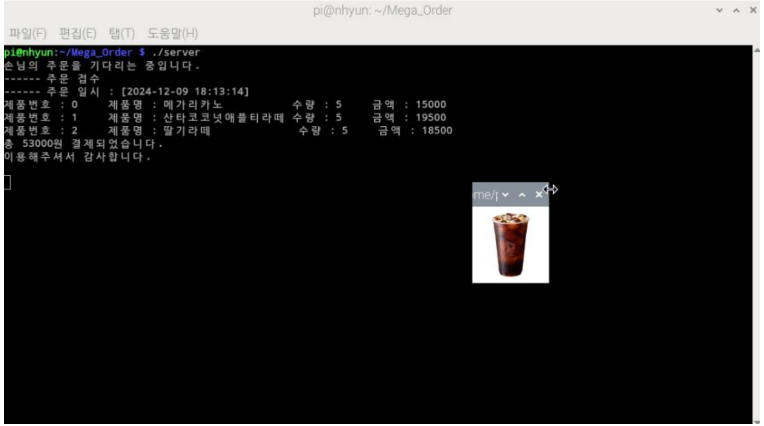
프로젝트 상세 : Cart - Server

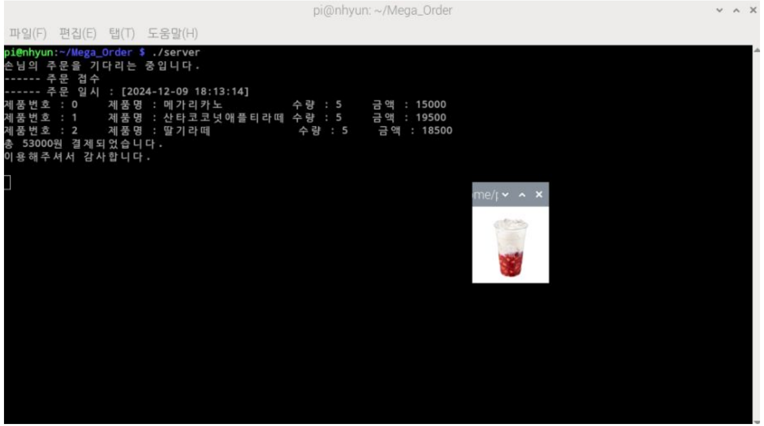
프로젝트 상세 : Raspberry Pi 5와 상호작용

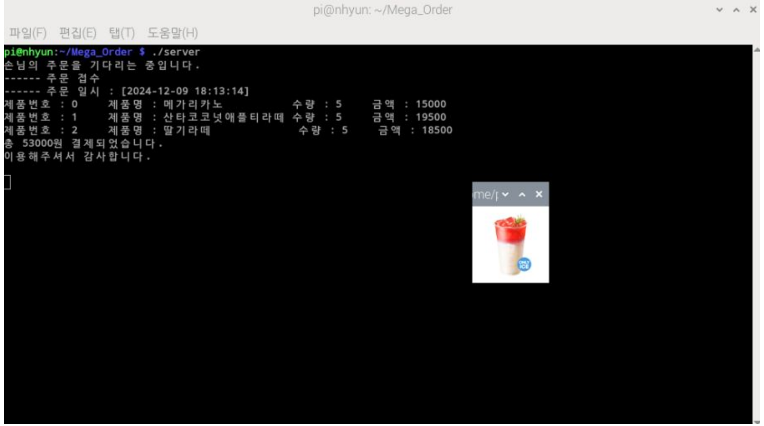
Raspberry Pi 5(이하 파이라고 한다.)는 보드에서 전송된 주문 정보를 받아들이는 서버의 역할을 한다.
주문이 전송되기 전까지 파이는 대기 상태이며, 주문이 왔을 시 주문에 대한 내역을 출력하고,
해당 주문에 대한 정보를 텍스트 파일로 저장한다.
또한, 실물 제품이 나왔다는 것을 빗대어 해당하는 메뉴의 이미지 파일을 오픈한다.



프로젝트 상세 : Raspberry Pi 5와 상호작용







문제점 및 개선과정

※ 메모리 문제

초기 구현에서 GUI에 필요한 폰트, 이미지 등의 데이터가 보드의 내부 Flash에 저장됨. 그러나 STM32F746의
내부 Flash의 용량은 1024KB로 GUI 데이터의 용량을 감당할 수 없음.
따라서 프레임버퍼는 SDRAM으로, 폰트, 이미지 등의 데이터는 QSPI로 옮기는 작업을 수행하여 메모리 문제를 해결함.

```
/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 320K
  FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 1024K
  SDRAM_BUFFER(xrw) : ORIGIN = 0xC0000000, LENGTH = 16M
  QUADSPI (r)    : ORIGIN = 0x90000000, LENGTH = 16M
}
```

링커 스크립트에 SDRAM과 QSPI 메모리에 대한 정의를 추가

문제점 및 개선과정

```
FramebufferSection (NOLOAD):
{
  *(TouchGFX_Framebuffer TouchGFX_Framebuffer.*)
  *(.gnu.linkonce.r.*)
  . = ALIGN(0x4);
} >SDRAM_BUFFER

 ExtFlashSection :
 {
   *(ExtFlashSection ExtFlashSection.*)
   *(.gnu.linkonce.r.*)
   . = ALIGN(0x4);
 } >QUADSPI

 FontFlashSection :
 {
   *(FontFlashSection FontFlashSection.*)
   *(.gnu.linkonce.r.*)
   . = ALIGN(0x4);
 } >QUADSPI

 TextFlashSection :
 {
   *(TextFlashSection TextFlashSection.*)
   *(.gnu.linkonce.r.*)
   . = ALIGN(0x4);
 } >QUADSPI
}
```

링커 스크립트에서 SDRAM과 QSPI 메모리로 GUI에 관련된 데이터를 이동

문제점 및 개선과정

```
/* USER CODE BEGIN 2 */
BSP_SDRAM_Init();
/* USER CODE END 2 */
```

보드 초기화 과정에 SDRAM 초기화 함수 추가

```
/* USER CODE BEGIN QUADSPI_Init 2 */
BSP_QSPI_Init();

BSP_QSPI_MemoryMappedMode();
HAL_NVIC_DisableIRQ(QUADSPI_IRQn);

MPU_Region_InitTypeDef MPU_InitStruct;
MPU_InitStruct.Enable = MPU_REGION_ENABLE;
MPU_InitStruct.BaseAddress = 0x90000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_256MB;
MPU_InitStruct.AccessPermission = MPU_REGION_FULL_ACCESS;
MPU_InitStruct.IsBufferable = MPU_ACCESS_NOT_BUFFERABLE;
MPU_InitStruct.IsCacheable = MPU_ACCESS_NOT_CACHEABLE;
MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER2;
MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL0;
MPU_InitStruct.SubRegionDisable = 0x00;
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;
HAL_MPU_ConfigRegion(&MPU_InitStruct);

MPU_InitStruct.Enable = MPU_REGION_ENABLE;
MPU_InitStruct.BaseAddress = 0x90000000;
MPU_InitStruct.Size = MPU_REGION_SIZE_16MB;
MPU_InitStruct.AccessPermission = MPU_REGION_FULL_ACCESS;
MPU_InitStruct.IsBufferable = MPU_ACCESS_NOT_BUFFERABLE;
MPU_InitStruct.IsCacheable = MPU_ACCESS_CACHEABLE;
MPU_InitStruct.IsShareable = MPU_ACCESS_NOT_SHAREABLE;
MPU_InitStruct.Number = MPU_REGION_NUMBER3;
MPU_InitStruct.TypeExtField = MPU_TEX_LEVEL0;
MPU_InitStruct.SubRegionDisable = 0x00;
MPU_InitStruct.DisableExec = MPU_INSTRUCTION_ACCESS_ENABLE;

HAL_MPU_ConfigRegion(&MPU_InitStruct);
HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
/* USER CODE END QUADSPI_Init 2 */
```

QSPI 초기화 과정에
필요한 과정 추가

고찰

※ 빠른 결정과 유연한 문제 해결

프로젝트 초기, 아이패드 - 라즈베리파이로 프로젝트를 진행하려고 하였으나 기술적 제약과 통신의 한계로 원하는 결과를 도출할 수 없음을 깨달음.
이로 인해 기존 아이디어를 바탕으로 해결 방안을 고민했으나, 적절한 대안을 찾기까지 약 일주일이라는 시간을 소모하게 됨.
이 경험으로 프로젝트 진행 시 초기 계획에만 매몰되지 않고, 빠르게 문제를 판단하고 유연하게 대처하는 능력이 절실하게 필요함을 느낌.

※ 효율적인 학습의 필요성

LWIP를 활용한 네트워크 통신 구현과 같은 특정 기능을 익히는 데 지나치게 많은 시간을 투자했음.
약 일주일을 소모하며 관련 기술을 깊이 학습했지만, 결과적으로 프로젝트 전반의 일정이 지연되는 결과를 초래함.
이 경험을 통해 모든 기능을 완벽히 이해하려는 접근보다는, 필요한 수준에서 학습을 마치고 빠르게 다음 단계로 넘어가는 것이 중요하다는 것을 깨달았음.
이후에는 주요 목표를 기준으로 학습 시간을 제한하고, 프로젝트 전체의 흐름을 놓치지 않도록 개선함.

고찰

※ 효율적 협업의 가능성

이번 프로젝트를 개인으로 진행하면서 팀과의 효율적인 협업에 대해 생각할 수 있는 기회가 되었음.
핵심 알고리즘, UI/UX, 네트워크 구성, 하드웨어 제어 등을 혼자 동시에 진행하려다 보니 일정의 지연이나 문제 해결에 대해 비효율적이었음.
개인으로 프로젝트를 진행함으로써 모든 과정을 스스로 경험하며 성장할 수 있는 계기가 되었지만, 팀으로 진행했다면 목표를 효율적으로 달성하고 더 높은 퀄리티의 결과물을 기대할 수 있었을 거라 생각함. 추후에는 개인의 역량 발전도 좋지만 팀 프로젝트에서 각자의 강점을 극대화할 수 있는 협업 방식을 배워 적용하기 위해 노력할 것임.

※ 새로운 가능성과 사용자 경험의 확장

보통 STM32 보드 등을 이용한 임베디드 환경에서는 UI/UX를 상세히 표현하기보다 기능 구현에 초점이 맞춰지는 경우가 많음.
그러나 이번 프로젝트를 진행하면서, 이러한 임베디드 환경에서도 세련된 UI/UX를 제공할 수 있으며 이를 통해 사용자의 경험을 크게 향상시킬 수 있는 가능성을 확인했음. 앞으로도 이런 가능성을 기반으로 사용자 중심의 시스템 설계를 좀 더 경험해보고 싶음.

감사합니다