FRI

UNIVERZA V LJUBLJANI

Fakulteta za računalništvo in informatiko

ORGANIZACIJA RAČUNALNIKOV

2. DOMAČA NALOGA

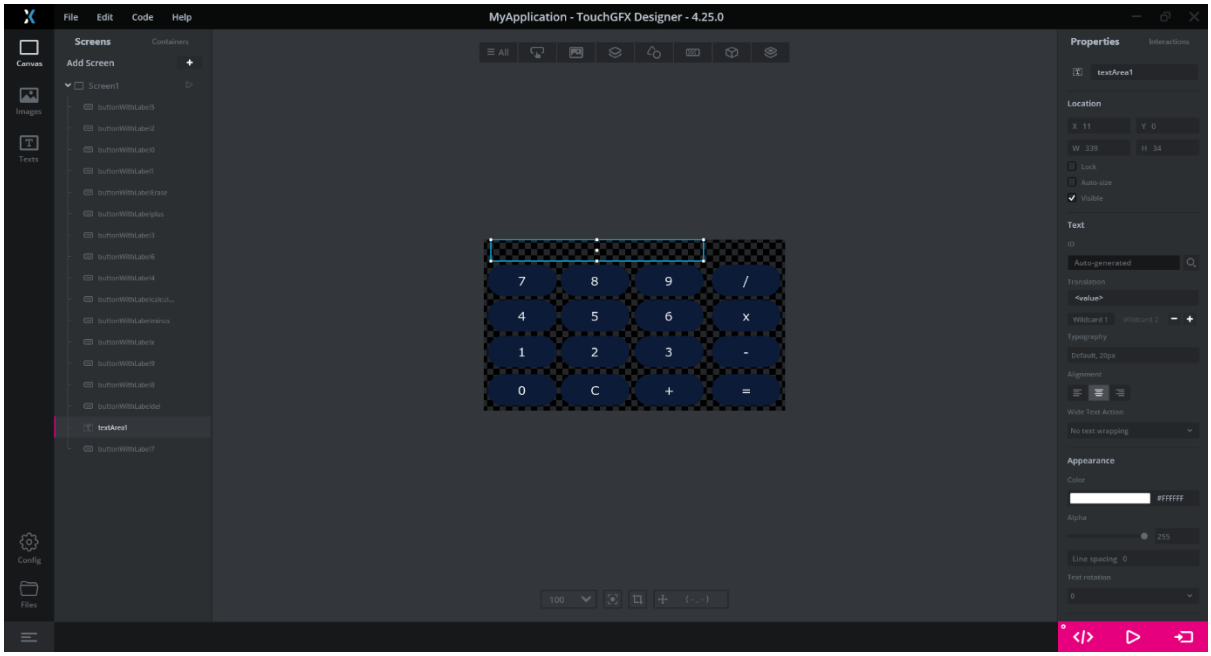
KALKULATOR

DAVID REŠETA

Za domačo nalogo sem izdelal preprost kalkulator na razvojni plošči stm32h750, za vnos podatkov sem uporabil zaslon na dotik. Kalkulator sem izdelal v razvojnem okolju **STM32CubeIDE** in **TouchGFX.** Osnovni TouchGFX projekt sem razširil s funkcijami ki omogočajo preproste matematične operacije.

1. GRAFIČNI VMESNIK

V okolju TouchGFX sem ustvaril osnovni »screen«, na katerega sem dodal gumbe in textArea elemente, vsakemu gumbu sem dodal interakcijo ob kliku, ta sproži klic virtualne funkcije. TextArea pa sem uporabil za izpis računa in rezultata.



Slika 1: TouchGFX

Na zgornji slik lahko vidimo kako izgleda orodje TouchGFX in kalkulator, lahko tudi zaženemo simulator ki nam pokaže kako bo aplikacija izgledala na plošči.

Ko sem zaključil in dodal vse interakcije za gumbe sem zgeneriral kodo, v katero sem pozneje dodal vse funkcije ki so potrebne da kalkulator deluje.

2. PROGRAMIRANJE

Po zaključenemu oblikovanju kalkulatorja sem delo nadaljeval v cubeIDE. Svojo kodo sem dodal v Screen1View.hpp in v Screen1View.cpp.

```hpp
#ifndef SCREEN1VIEW_HPP
#define SCREEN1VIEW_HPP

#include <gui_generated/screen1_screen/Screen1ViewBase.hpp>
#include <gui/screen1_screen/Screen1Presenter.hpp>
#include <string>


class Screen1View : public Screen1ViewBase
{
public:
    Screen1View();
    virtual ~Screen1View() {}
    virtual void setupScreen();
    virtual void tearDownScreen();
    virtual void number7();
    virtual void number8();
    virtual void number9();
    virtual void number4();
    virtual void number5();
    virtual void number6();
    virtual void number3();
    virtual void number2();
    virtual void number1();
    virtual void number0();
    virtual void operatorPlus();
    virtual void operatorCalculate();
    virtual void operatorErase();
    virtual void operatorMultiply();
    virtual void operatorDel();
    virtual void operatorMinus();

private:
    // Shranjujemo trenutni vnos, prejšnji vnos in operator
    std::string currentInput;
    std::string previousInput;
    char currentOperator;

    // Pomožna metoda za posodobitev izpisa na textArea
    void updateDisplay();

protected:
```

Slika 2: Screen1View.hpp

V Screen1View.hpp sem definiral vse funkcije in spremenljivke, ki sem jih pozneje uporabil v Screen1View.cpp.

Za vsak gumb sem ustvaril funkcijo ki predstavlja njegov namen, na primer za vsako številko in operator je funkcija, s katero »pusham« številko k že prej dodani številki. Funkcija updateDisplay pa se kliče ob vsakem pritisku na ekran.

V Screen1View.cpp sem pa dodal logiko vsem funkcijam.

```cpp
// Pretvorimo operandne nize v cela števila
int num1 = std::stoi(operand1);
int num2 = std::stoi(operand2);
int result = 0;

// Uporabimo switch stavek za izvedbo ustrezne operacije
switch (currentOperator)
{
    case '+':
        result = num1 + num2;
        break;
    case '-':
        result = num1 - num2;
        break;
    case 'x':
        result = num1 * num2;
        break;
    case '/':
        if (num2 == 0)
        {
            // Če pride do deljenja z 0, prikažemo napako
            currentInput = "Error";
            updateDisplay();
            return;
        }
        result = num1 / num2;
        break;
    default:
        return;
}
```

```cpp
void Screen1View::operatorMultiply()
{
    // Preverimo, če v izrazu že obstaja operator, če ne, ga dodamo
    if (currentInput.find('x') == std::string::npos)
    {
        currentInput.push_back('x');
        currentOperator = 'x';
    }
    updateDisplay();
}
```

```cpp
void Screen1View::number1()
{
    if (currentInput == "0")
    {
        currentInput = "";
    }
    currentInput.push_back('1');
    updateDisplay();
}
```

KONČNI IZGLED: