## Layered Architecture with Corresponding Pseudocode & Files

#### 1. Hardware Layer

Purpose: Interact directly with physical devices (GPIO, ADC, UART, SPI, RTC, VFD).

##### Header Files:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| gpio.h | Initialize GPIO for heating rods, pump, direction control TurnOn(heatingRodRelay), TurnOff(), pumpRelay control |
| adc.h | ReadADC(channel) |
| comms.h | Initialize SPI, UART, RS-485 |
| motordriver.h | SetDirection(), VFD.setSpeed(), StartMotor(), StopMotor() |
| rtc.h | RTC.getTime() |

##### Layer Pseudocode:

Initialize GPIO for heating rods, pump, direction control

Initialize ADC for temperature sensor input

Initialize communication interfaces (SPI, UART, RS-485)

Initialize motor drivers and VFD interface

Initialize RTC (Real-Time Clock) for scheduling

adcValue ← ReadADC(channel)

#### 2. Mid/Control Level Layer (control/)

Purpose: Translate raw hardware signals into usable data and apply device-level logic.

##### Header:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| TempControl.h | ConvertADCtoTemp(adcValue) Heating ON/OFF logic |
| MotorControl.h | Logic for agitatorSpeed, agitatorDirection Manual override check |
| PumpControl.h | ON/OFF logic for pumpRelay |
| SensorInterface.h | GUI.updateTemperature ← currentTemperature |

##### Layer Pseudocode:

currentTemperature ← ConvertADCtoTemp(adcValue)

GUI.updateTemperature ← (currentTemperature)

If (currentTemperature < targetTemperature):

TurnOn(heatingRodRelay)

heatingStatus ← ON

Else:

TurnOff(heatingRodRelay)

heatingStatus ← OFF

If (modeSelection in ["Pasteurization", "Cheese", "Cooling"]):

If (manualOverride is FALSE):

agitatorSpeed ← agitatorSettings[modeSelection].speed

agitatorDirection ← agitatorSettings[modeSelection].direction

VFD.setSpeed(agitatorSpeed)

SetDirection(agitatorDirection)

StartMotor()

Else:

StopMotor()

If (modeSelection in ["Cooling", "Drain"]):

TurnOn(pumpRelay)

pumpStatus ← ON

Else:

TurnOff(pumpRelay)

pumpStatus ← OFF

#### 3. High-Level Process Layer (process/)

Purpose: Handle process flows like Pasteurization, Scheduling, Auto logic, and Safety.

##### Header Files:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| ProcessSelect.h | Mode logic (Manual/Auto/Scheduled) |
| Scheduler.h | Load scheduled task logic |
| Config.h | Check limits, Apply the process logic based on phase |
| ProcessLogic.h | Heat and Stir profile Logic |

##### Layer Pseudocode:

Set minimum temperature Threshold

Set maximum temperature Threshold

Define agitator settings:

Pasteurization: speed = x RPM, direction = FORWARD

Cheese: speed = y RPM, direction = FORWARD

Cooling: speed = z RPM, direction = REVERSE

Load scheduled Tasks ← get data from GUI or saved memory

Set current Mode ← GUI input or default (Manual/Auto/Scheduled)

If (currentMode == SCHEDULED):

currentTime ← RTC.getTime()

For each task in scheduledTasks:

If (task.time == currentTime AND task.notCompleted):

currentMode ← task.process

task.markAsCompleted()

Log("Scheduled process started:", task.process)

If (currentMode == AUTO):

ApplyProcessLogicBasedOnPhase()

AdjustHeatAndStirBasedOnPhase(currentTemperature, elapsedTime)

CheckSafetyLimits()

UpdateTimers()

#### 4. User Interface Layer (GUI)

Purpose: Provide interactive display and input system for users.

##### Headers:

|  |  |
| --- | --- |
| **File** | **Pseudocode Responsibility** |
| gui\_interface.h | GUI.getMode(), GUI.getStartStopCommand(), GUI.getTargetTemperature() |
| gui\_update.h | GUI.updateStatus(), GUI.updateTemperature() |

##### Layer Pseudocode:

Start the main loop

While (true):

modeSelection ← GUI.getMode()

startCommand ← GUI.getStartStopCommand()

manualOverride ← GUI.getManualOverrideStatus()

targetTemperature ← GUI.getTargetTemperature()

agitatorSpeed ← GUI.getManualMotorSpeed()

agitatorDirection ← GUI.getManualMotorDirection()

GUI.updateStatus({

"Temperature": currentTemperature,

"Motor Speed": agitatorSpeed,

"Motor Direction": agitatorDirection,

"Heating": heatingStatus,

"Pump": pumpStatus,

"Mode": currentMode,

"Alarms": CheckAlarms()

})

## Pseudocode for Pasteurization

Initialize 🡪 Start motor and Heater 🡪 Heat to Target Temperature 🡪 Hold Temperature for assigned time 🡪 End of the process

**gui\_interface.h**

**gui\_update.h**

**config.h**

TempControl.h

MotorControl.h

PumpControl.h

### Initialize the parameters

targetTemperature ← 72°C

holdingTime ← 15 minutes

agitatorSpeed ← 50 RPM

agitatorDirection ← FORWARD

heatingStatus ← OFF

motorStatus ← OFF

holdingTimer ← 0

processStatus ← "Heating Phase"

### **2.** Start Pasteurization Process

//GUI.display("Starting Pasteurization.")

MotorControl.startMotor(agitatorSpeed, agitatorDirection)

motorStatus ← ON

TempControl.turnOnHeater()

heatingStatus ← ON

### 3. Heating Phase

While (currentTemperature < targetTemperature):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < targetTemperature):

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

//delay if needed

### 4. Holding Phase

holdingTimer.start()

While (holdingTimer.elapsedTime() < holdingTime):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < (targetTemperature - 1°C)): // if the difference b/w current and target is 1°C

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

GUI.updateHoldingTimer(holdingTime - holdingTimer.elapsedTime())

Delay(short\_interval)

### 5. End Pasteurization

TempControl.turnOffHeater()

MotorControl.stopMotor()

heatingStatus ← OFF

motorStatus ← OFF

GUI.display("Pasteurization Complete!")

//ProcessLogic.markProcessComplete("Pasteurization") //if want to update in this in logs

Cheese Making Process **Initialize → Start motor and Heater → Heat to Target Temperature → Hold Temperature for assigned time → End of the process**

**gui\_interface.h**

**gui\_update.h**

**config.h**

TempControl.h

MotorControl.h

PumpControl.h

### 1. Initialize the parameters

targetTemperature ← 38°C

holdingTime ← 30 minutes

agitatorSpeed ← 30 RPM

agitatorDirection ← FORWARD

heatingStatus ← OFF

motorStatus ← OFF

holdingTimer ← 0

processStatus ← "Heating Phase"

### 2. Start Cheese Making Process

// GUI.display("Starting Cheese Making.")

MotorControl.startMotor(agitatorSpeed, agitatorDirection)

motorStatus ← ON

TempControl.turnOnHeater()

heatingStatus ← ON

### 3. Heating Phase

While (currentTemperature < targetTemperature):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < targetTemperature):

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

// delay if needed

### 4. Holding Phase

holdingTimer.start()

While (holdingTimer.elapsedTime() < holdingTime):

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

GUI.updateTemperature(currentTemperature)

If (currentTemperature < (targetTemperature - 1°C)):

TempControl.turnOnHeater()

heatingStatus ← ON

Else:

TempControl.turnOffHeater()

heatingStatus ← OFF

GUI.updateHeatingStatus(heatingStatus)

GUI.updateHoldingTimer(holdingTime - holdingTimer.elapsedTime())

Delay(short\_interval)

### 5. End Cheese Making

TempControl.turnOffHeater()

MotorControl.stopMotor()

heatingStatus ← OFF

motorStatus ← OFF

GUI.display("Cheese Making Process Complete!")

// ProcessLogic.markProcessComplete("Cheese") // if want to update in logs

## ****Cooling Process Pseudocode:****

**gui\_interface.h**

**gui\_update.h**

**config.h**

TempControl.h

MotorControl.h

PumpControl.h

#### ****1.** Initialize Parameters:**

// Set target temperature for cooling (user-defined)

targetTemperature ← 4°C // Cooling target temperature

// Initialize motor and pump status

// motorStatus ← OFF

// pumpStatus ← OFF

currentTemperature ← 0°C // Placeholder for current temperature

processStatus ← "Cooling Phase" // Track process status

#### ****2.** Start Cooling Process:**

// Start the cooling process

PumpControl.turnOnPump() // Turn on the pump for cooling

pumpStatus ← ON

// Optionally start the motor (if needed for agitation during cooling)

MotorControl.startMotor(agitatorSpeed, agitatorDirection)

motorStatus ← ON

#### ****3.** Monitoring Temperature:**

// continuously check the temperature

While (currentTemperature > targetTemperature):

// Read the current temperature from the sensor

adcValue ← SensorInterface.readTemperatureSensor()

currentTemperature ← TempControl.convertADCtoTemperature(adcValue)

// Update GUI with current temperature

GUI.updateTemperature(currentTemperature)

// Check if cooling target temperature is achieved

If (currentTemperature <= targetTemperature):

// Cooling process is complete

PumpControl.turnOffPump() // Turn off the cooling pump

pumpStatus ← OFF

// Optionally stop the motor if not required after cooling is done

MotorControl.stopMotor()

motorStatus ← OFF

// Display the process completion on the GUI

GUI.display("Cooling Complete!")

//delay