A Appendix: Code

A.1 buttons.h

```
#ifndef BUTTONS_H
#define BUTTONS_H

#define BUTTON1 0x11
#define BUTTON2 0x14
#define BUTTON3 0x50
#define BUTTON4 0x90
#define BUTTON5 0x09
#define BUTTON6 0x0C
#define BUTTON7 0x48
#define BUTTON8 0x88
#define BUTTON9 0x03
#define BUTTON10 0x06
#define BUTTON11 0x42
#define BUTTON12 0x82
#define BUTTON13 0x21
#define BUTTON14 0x24
#define BUTTON15 0x60
#define BUTTON16 0xA0

#define LOGOUT 'A'
#define BACK 'B'
#define CLEAR 'C'
#define CONFIRM 'D'



void buttons_init(void);
void check_buttons(void);
char get_char(void);


#endif
```

A.2 buttons.c

```
#define F_CPU 8000000UL
#include <util/delay.h>
#include <avr/io.h>
#include "buttons.h"

static char ch;
```

```
/*
 * Function:  buttons_init
 * --------------------
 * Initialize buttons
 *
 */
void buttons_init(void) {

         DDRB = 0x00;

        _delay_ms(1);

         PORTB = 0xC5;

         _delay_ms(1);

}

/*
 * Function:  check_buttons
 * --------------------
 * Check which button is pressed, store the corresponding
 *        char in ch
 *
 */
void check_buttons(void) {
        int keypressed = 0;

        if (PINB != 0x3A)//in any of column pins goes high execute the loop
        {
                keypressed = PINB^0x3A;//taking the column value into integer

                _delay_ms(5);

                DDRB ^=0xFF;//making rows as inputs and columns as ouput

                _delay_ms(1);

                PORTB ^= 0xFF;//powering columns

                _delay_ms(1);


                keypressed |= PINB^0x3A;//taking row value and OR ing it to column value
```

```
if (keypressed == BUTTON1)
{
        ch = '1';
}

if (keypressed == BUTTON2)
{
        ch = '2';
}

if (keypressed == BUTTON3)
{
        ch = '3';
}

if (keypressed == BUTTON4)
{
        ch = LOGOUT;
}

if (keypressed == BUTTON5)
{
        ch = '4';
}

if (keypressed == BUTTON6)
{
        ch = '5';
}

if (keypressed == BUTTON7)
{
        ch = '6';
}

if (keypressed == BUTTON8)
{
        ch = BACK;
}

if (keypressed == BUTTON9)
{

        ch = '7';
}
```

```
if (keypressed == BUTTON10)
{
        ch = '8';
}

if (keypressed == BUTTON11)
{
        ch = '9';
}

if (keypressed == BUTTON12)
{
        ch = CLEAR;
}

if (keypressed == BUTTON13)
{
        ch = '*';
}

if (keypressed == BUTTON14)
{
        ch = '0';
}

if (keypressed == BUTTON15)
{
        ch = '#';
}

if (keypressed == BUTTON16)
{
        ch = CONFIRM;
}

keypressed=0;//after showing integer erasing the row column memory

DDRB ^=0xFF;//shifting input and power port

_delay_ms(1);

PORTB ^= 0xFF;//powering row pins of keypad

//_delay_ms(220);
```

```
        }
}

/*
 * Function:  get_char
 * --------------------
 * Get char of the button pressed
 *
 */
char get_char(void) {
        char ret = ch;
        ch = 0;
        return ret;
}
```

A.3 eeprom.h

```
#ifndef EEPROM_H
#define EEPROM_H

void EEPROM_write(unsigned int, unsigned char);
unsigned char EEPROM_read(unsigned char);

#endif
```

A.4 eeprom.c

```
#include "eeprom.h"
#include <avr/io.h>
#include <avr/interrupt.h>


/*
 * Function:  EEPROM_write
 * --------------------
 * Writes data ucData to address uiAddress on the EEPROM
 *
 *  uiAddress: address to be written to
 *        ucData: Data to be written
 */
void EEPROM_write(unsigned int uiAddress, unsigned char ucData) {
        while (EECR & (1 << EEWE));

        EEAR = uiAddress;
```

```c
        EEDR = ucData;

        EECR |= (1 << EEMWE);
        EECR |= (1 << EEWE);

}

/*
 * Function:  EEPROM_write
 * --------------------
 * Read data from EEPROM
 *
 *  uiAddress: address to be read from
 *
 *        returns: data on address uiAddress
 */
unsigned char EEPROM_read(unsigned char uiAddress) {
        while (EECR & (1 << EEWE));

        EEAR = uiAddress;
        EECR |= (1 << EERE);
        return EEDR;
}
```

A.5 LCD.h

```c
#ifndef LCD_H
#define LCD_H

#define RS 6
#define E  0

void LCD_init(void);
void print_to_LCD_size(char*, int);
void print_to_LCD(char* str);
void send_a_command(unsigned char);
void send_a_character(unsigned char);
void clear_a_character(void);
void clear_LCD(void);
void move_cursor(unsigned char);

#endif
```

A.6 LCD.c

```
#include "LCD.h"

#define F_CPU 8000000UL
#include <util/delay.h>
#include <avr/io.h>

/*
 * Function:  LCD_init
 * --------------------
 * Initialize LCD
 *
 */
void LCD_init(void) {
        send_a_command(0x3C);
        _delay_us(38);
        send_a_command(0x3C);
        _delay_us(38);
        send_a_command(0x0C);
        _delay_us(38);
        send_a_command(0x01);// sending all clear command LCD
        _delay_ms(1.53);
        send_a_command(0x06);
        _delay_ms(1.53);

        print_to_LCD("LCD initialized");
        _delay_ms(2000);
}

/*
 * Function:  print_to_LCD_size
 * --------------------
 * Print the buffer str to LCD
 *
 *        str: buffer to be printed
 *        size: size of buffer
 */
void print_to_LCD_size(char* str, int size) {
        for (int i = 0; i < size; i++)
                send_a_character(str[i]);
}

/*
 * Function:  print_to_LCD
 * --------------------
 * Print str to LCD
 *
 *        str: string to be printed
 */
void print_to_LCD(char* str) {
        while (*str)
                send_a_character(*str++);
}

/*
 * Function:  send_a_command
 * --------------------
 * Send a command to the LCD
 *
 *        command: command to be sent
 */
void send_a_command(unsigned char command)
{
        PORTC &= ~(1<<RS);
        PORTD |= command & 0xF3;
        PORTA |= (command >> 2) & 0x03;
        PORTC |= (1<<E);
        _delay_ms(50);
        PORTC &= ~(1<<E);
        PORTD &= 0x0C;
        PORTA &= 0xFC;
}

/*
 * Function:  send_a_character
 * --------------------
 * Print a character at the cursor
 *
 *        character: character to be printed
 */
void send_a_character(unsigned char character)
{
        PORTC |= (1<<RS);
        PORTD |= character & 0xF3;
        PORTA |= (character >> 2) & 0x03;
        PORTC|= (1<<E);
        _delay_ms(50);
        PORTC &= ~(1<<E);
        PORTD &= 0x0C;
        PORTA &= 0xFC;
}
```

```
/*
 * Function:  clear_a_character
 * --------------------
 * Clear character at the cursor
 *
 */
void clear_a_character(void) {
        send_a_command(0x10);
        _delay_us(38);
        send_a_character(' ');
        send_a_command(0x10);
        _delay_us(38);
}

/*
 * Function:  clear_LCD
 * --------------------
 * Clears the whole LCD of all characters
 *
 */
void clear_LCD(void)
{
        send_a_command(0x01);// sending all clear command LCD
        _delay_ms(1.53);
}

/*
 * Function:  move_cursor
 * --------------------
 * Move cursor to the location to
 *
 *  to: where to move the cursor
 *
 */
void move_cursor(unsigned char to) {
        send_a_command(0x80 | to);
        _delay_ms(1.53);
}
```

A.7 user.h

```
#ifndef USER_H
#define USER_H

#include <stdint.h>
```

```c
#define USR_COUNT 10
#define NAME_SIZE 6
#define PASS_SIZE 4
#define EEPROM_START_ADDRESS 128

/*
 * Struct:  user_t
 * --------------------
 * Struct of a user in the system.
 *        counter is used to limit how
 *        much alcohol the user can order
 *        from the system
 *
 */
typedef struct {
        char name[NAME_SIZE];
        char password[PASS_SIZE];
        uint32_t counter;
} user_t;

extern user_t user[USR_COUNT];
extern unsigned char nbr_of_users;
extern int current_user;

int add_user(char*, char*);
void user_array_init(void);
int add_user(char*, char*);
int remove_user(char*);
void dec_users(void);
int login(char*, char*);
void logout(void);
int is_admin(void);
void save_users(void);
void load_users(void);
void reset_user_array(void);
void print_users(void);

#endif
```

A.8 user.c

```c
#include <string.h>
#include "user.h"
#include "eeprom.h"
```

```c
#define F_CPU 8000000UL

#include <util/delay.h>
#include "LCD.h"

user_t user[USR_COUNT];
unsigned char nbr_of_users;
int current_user = -1;

/*
 * Function:  user_array_init
 * --------------------
 * Add some random users to the system
 *
 */
void user_array_init(void) {
        add_user("111114", "1111");
        add_user("222223", "2222");
        add_user("333332", "3333");
        add_user("444441", "4444");
}

/*
 * Function:  username_take
 * --------------------
 * Check if a username already is taken
 *
 *        name: the username to be checked
 *
 *        returns: 1 if username is taken and 0 if username is not
 */
static int username_taken(char* name) {
        for (int i = 0; i < nbr_of_users; i++)
                if (strncmp(user[i].name, name, NAME_SIZE) == 0)
                        return 1;
        return 0;
}


/*
 * Function:  add_user
 * --------------------
 * Add a user with username and password
 *
 *        name: username for the user
 *        password: password for the user
 *
 *        returns: 1 if username was taken,
 *                         2 if maximum number of users has been reached
 *                         and 0 if the user was added successfully
 */
int add_user(char* name, char* password) {
        if (username_taken(name))
                return 1;
        if (nbr_of_users == USR_COUNT)
                return 2;
        strncpy(user[nbr_of_users].name, name, NAME_SIZE);
        strncpy(user[nbr_of_users].password, password, PASS_SIZE);
        nbr_of_users++;
        return 0;
}

/*
 * Function:  remove_user
 * --------------------
 * Remove a user from the system
 *
 *        name: username of the user to be removed
 *
 *        returns: 1 if user could not be found
 *                         and 0 if the user was removed successfully
 */
int remove_user(char* name) {
        for (int i = 1; i < nbr_of_users; i++)
                if (strncmp(user[i].name, name, NAME_SIZE) == 0) {
                        nbr_of_users--;
                        memcpy(user + i, user + i + 1, (nbr_of_users - i)*sizeof(user_t));
                        memset(user + nbr_of_users, 0, sizeof(user_t));
                        return 0;
                }
        return 1;
}

/*
 * Function:  dec_users
 * --------------------
 * Decrement counter for each user.
 *        If counter is 0 it is not decremented
 *
 */
void dec_users(void) {
        for (int i = 0; i < nbr_of_users; i++)
```

```
                user[i].counter = user[i].counter > 0 ? user[i].counter - 1 : 0;
}

/*
 * Function:  login
 * --------------------
 * Tries to login as user with name and password
 *
 *        name: username of the user to login as
 *        password: password to be tested
 *
 *        returns: 1 if login was successful
 *                        and 0 if login was unsuccessful
 */
int login(char* name, char* password) {
        for (int i = 0; i < nbr_of_users; i++) {
                if (strncmp(user[i].name, name, NAME_SIZE) == 0
                    && strncmp(user[i].password, password, PASS_SIZE) == 0) {
                        current_user = i;
                        return 1;
                }
        }
        return 0;
}

/*
 * Function:  logout
 * --------------------
 * Log out
 *
 */
void logout(void) {
        current_user = -1;
}

/*
 * Function:  is_admin
 * --------------------
 * Checks if the current user is admin
 *
 *        returns: 1 if current user is admin
 *                        and 0 if current user is not admin
 */
int is_admin(void) {
        return !current_user;
}
```

```
/*
 * Function:  save_users
 * --------------------
 * Save the users to the EEPROM
 *
 */
void save_users(void) {
        int address = EEPROM_START_ADDRESS;
        EEPROM_write(address++, nbr_of_users);
        for (int i = 0; i < nbr_of_users; i++) {
                for (int j = 0; j < NAME_SIZE; j++)
                        EEPROM_write(address++, user[i].name[j]);
                for (int j = 0; j < PASS_SIZE; j++)
                        EEPROM_write(address++, user[i].password[j]);
                for (int j = 0; j < sizeof(uint32_t)*8; j += 8)
                        EEPROM_write(address++, user[i].counter >> j & 0xFF);
        }
}

/*
 * Function:  load_users
 * --------------------
 * Load users from the EEPROM
 *
 */
void load_users(void) {
        int address = EEPROM_START_ADDRESS;
        nbr_of_users = EEPROM_read(address++);
        for (int i = 0; i < nbr_of_users; i++) {
                for (int j = 0; j < NAME_SIZE; j++)
                        user[i].name[j] = EEPROM_read(address++);
                for (int j = 0; j < PASS_SIZE; j++)
                        user[i].password[j] = EEPROM_read(address++);
                for (int j = 0; j < sizeof(uint32_t)*8; j += 8)
                        user[i].counter |= (uint32_t)EEPROM_read(address++) << j;
        }
}

/*
 * Function:  reset_user_array
 * --------------------
 * Reset the user array(remove all users)
 *
 */
```

```c
void reset_user_array(void) {
        memset(user, 0, USR_COUNT*sizeof(user_t));
        nbr_of_users = 0;
}

/*
 * Function:  print_users
 * --------------------
 * Print username and password of all users
 *
 */
void print_users(void) {
        clear_LCD();
        int temp = nbr_of_users;
        if (temp) {
                send_a_character((char) (temp % 10 + 48));
                temp /= 10;
        }
        else
                send_a_character('0');
        _delay_ms(2000);
        for (int i = 0; i < nbr_of_users; i++) {
                clear_LCD();
                print_to_LCD_size(user[i].name, NAME_SIZE);
                move_cursor(20);
                print_to_LCD_size(user[i].password, PASS_SIZE);
                _delay_ms(200);
        }
}
```

A.9 main.c

```c
#define F_CPU 8000000UL

#include <string.h>
#include <stdint.h>
#include <util/delay.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "user.h"
#include "buttons.h"
#include "LCD.h"

#define ALCO '1'
#define SODA '2'
```

```
#define SODA1 '1'
#define SODA2 '2'

#define ADD_USER '1'
#define REMOVE_USER '2'

#define MINUTE 60*SECOND
#define SECOND 32
#define NBR_OF_PULSES0                6
#define NBR_OF_PULSES1                18
#define flowMeterInterruptEnable0 GICR |= 1<<INT0
#define flowMeterInterruptDisable0 GICR &= ~(1<<INT0)
#define flowMeterInterruptEnable1 GICR |= 1<<INT1
#define flowMeterInterruptDisable1 GICR &= ~(1<<INT1)
#define startTap1 PORTA |= 0x10
#define startTap2 PORTA |= 0x20
#define stopTap1 PORTA &= ~0x10
#define stopTap2 PORTA &= ~0x20

typedef void (*state_handler)(void);

void menu(void);
void soda_menu(void);
void pour_alcohol(void);
void message(void);
void new_name(void);
void new_password(void);
void set_name(void);
void set_password(void);
void admin_menu(void);
void remove_state(void);

void print_soda_menu(void);
void print_menu(void);
void flowmeter_init(void);
void flowmeter_init2(void);
void timer0_init(void);
int drink_allowed(void);
int handle_keyboard(int, char*);
void handle_state_change(void);
void disconnect(void);
void clear_input(void);
void int32_to_string(uint32_t, char*);
void print_counter(void);

state_handler state = &set_name;
state_handler prev_state = &set_name;
state_handler next_state = &set_name;

int chr_offset;

char current_name[NAME_SIZE];
char current_pass[PASS_SIZE];

volatile uint32_t time;

volatile uint8_t tot_overflow;

volatile int last_updated;

volatile int pulses;

/*
 * Macro:  ISR(INT0_vect)
 * --------------------
 * Code which is executed when flow meter
 *        creates a external interrupt
 *
 */
ISR(INT0_vect)
{
        pulses++;
        if (pulses >= NBR_OF_PULSES0) {
                stopTap1;
                flowMeterInterruptDisable0;
        }
        PORTA ^= 0x04;
}

/*
 * Macro:  ISR(INT1_vect)
 * --------------------
 * Code which is executed when flow meter
 *        creates a external interrupt
 *
 */
ISR(INT1_vect)
{
        pulses++;
        if (pulses >= NBR_OF_PULSES1) {
                stopTap2;
                flowMeterInterruptDisable1;
        }
        PORTA ^= 0x08;
}

/*
 * Macro:  ISR(TIMER0_OVF_vect)
 * --------------------
 * Code which is executed when timer 0 overflow
 *
 */
ISR(TIMER0_OVF_vect)
{
        time++;
        last_updated++;
        tot_overflow++;
        dec_users();

}

/*
 * Function:  system_init
 * --------------------
 * Initialize the system
 *
 */
void system_init(void) {
        DDRA |= 0x7F;//activate 7 first LED/valves
        DDRC |= 0x80;//activate 8th

        DDRD = 0xF3;//activate pin out to LCD
        DDRC = 0x45;
        _delay_ms(50);

        LCD_init();
        buttons_init();
        MCUCR = 0;
        flowmeter_init();
        flowmeter_init2();
        timer0_init();
}


int main(void)
{
        load_users();
        system_init();
        handle_state_change();
        sei();
        while (1) {
                check_buttons();
                if (tot_overflow >= 6) {
                        tot_overflow = 0;
                        state();
                }
                if (last_updated >= 15*SECOND) {
                        last_updated = 0;
                        cli();
                        save_users();
                        sei();
                }
        }

}

/*
 * Function:  menu
 * --------------------
 * Handle interaction when in the menu state
 *
 */
void menu(void) {
        char ch;
        print_counter();
        if ((ch = get_char())) {
                if (ch == LOGOUT) {
                        disconnect();
                }
                else if (ch == BACK) {
                        clear_input();
                        state = prev_state;
                        handle_state_change();
                }
                else if (ch == ALCO) {
                        if (drink_allowed()) {
                                prev_state = state = &pour_alcohol;
                                user[current_user].counter = 30*SECOND;
                                cli();
                                save_users();
                                sei();
                                handle_state_change();
                        }
                        else {
                                next_state = &menu;
                                state = &message;
                                clear_LCD();
                                print_to_LCD("You have drinked to much!");

                        }
                }
                else if (ch == SODA) {
                        prev_state = state;
                        state = &soda_menu;
                        handle_state_change();
                }
                else {

                }
        }
}

/*
 * Function:  soda_menu
 * --------------------
 * Handle interaction when in the soda menu state
 *
 */
void soda_menu(void) {
        char ch;
        if ((ch = get_char())) {
                if (ch == LOGOUT) {
                        disconnect();
                }
                else if (ch == BACK) {
                        clear_input();
                        state = prev_state;
                        handle_state_change();
                }
                else if (ch == SODA1) {
                        startTap1;
                }
                else if (ch == SODA2) {
                        startTap2;
                }
                else {
                        stopTap1;
                        stopTap2;

                }
        }
        else {
                stopTap1;
                stopTap2;
        }
}

/*
 * Function:  pour_alcohol
 * --------------------
 * A state where the alcohol is poured.
 * No user interaction is possible in this state
 *
 */
void pour_alcohol(void) {
        _delay_ms(5000);
        pulses = 0;
        flowMeterInterruptEnable0;
        startTap1;
        while(pulses < NBR_OF_PULSES0);
        _delay_ms(5000);

        pulses = 0;
        flowMeterInterruptEnable1;
        startTap2;
        while(pulses < NBR_OF_PULSES1);

        disconnect();

}

/*
 * Function:  message
 * --------------------
 * State with delay. This state is used
 *        after a important message
 *
 */
void message(void) {
        _delay_ms(2000);
        state = next_state;
        handle_state_change();
}

/*
 * Function:  new_name
 * ---------------------
 * State when a username of a new user should be typed
 *
 */
void new_name(void) {
        if (handle_keyboard(NAME_SIZE, current_name)) {
                prev_state = state;
                state = &new_password;
                handle_state_change();
        }
}

/*
 * Function:  new_password
 * ---------------------
 * State when a password of a new user should be typed
 *
 */
void new_password(void) {
        if (handle_keyboard(PASS_SIZE, current_pass)) {
                clear_LCD();
                int error_code;
                if ((error_code = add_user(current_name, current_pass)))
                        if (error_code == 1)
                                print_to_LCD("Username already taken");
                        else
                                print_to_LCD("Userlist is Full");
                else {
                        cli();
                        save_users();
                        sei();
                        print_to_LCD("A User was added");
                }
                prev_state = &new_name;
                next_state = &new_name;
                state = &message;
                clear_input();
        }

}

/*
 * Function:  set_name
 * ---------------------
 * State when a name of a user should be typed(login)
 *
 */
void set_name(void) {
        if (handle_keyboard(NAME_SIZE, current_name)) {
                prev_state = state;
                state = &set_password;
                handle_state_change();
        }
}

/*
 * Function:  set_password
 * --------------------
 * State when a password of a user should be typed(login)
 *
 */
void set_password(void) {
        if (handle_keyboard(PASS_SIZE, current_pass)) {
                if (login(current_name, current_pass)) {
                        clear_input();
                        if (is_admin()) {
                                state = &admin_menu;
                                handle_state_change();
                        }
                        else {
                                state = &menu;
                                handle_state_change();
                        }
                        prev_state = state;
                }
                else {
                        clear_LCD();
                        print_to_LCD("Username or password was incorrect");
                        prev_state = &set_name;
                        next_state = &set_name;
                        state = &message;
                }
                clear_input();
        }

}

/*
 * Function:  admin_menu
 * --------------------
 * Handle interactions from admin.
 *       At this state you will have
 *        the possibility to go to the
 *        remove user and add user states
 *
 */
void admin_menu(void) {
        char ch;
        if ((ch = get_char())) {
                if (ch == LOGOUT) {
                        disconnect();
                }
                else if (ch == BACK) {
                        clear_input();
                        state = prev_state;
                        handle_state_change();
                }
                else if (ch == ADD_USER) {
                        prev_state = state;
                        state = &new_name;
                        handle_state_change();
                }
                else if (ch == REMOVE_USER) {
                        prev_state = state;
                        state = &remove_state;
                        handle_state_change();
                }
        }
}

/*
 * Function:  remove_state
 * --------------------
 * State where you can type username of user to remove
 *
 */
void remove_state(void) {
        if (handle_keyboard(NAME_SIZE, current_name)) {
                clear_LCD();
                int error_code;
                if ((error_code = remove_user(current_name))) {
                        if (error_code == 1)
                                print_to_LCD("Username was not found");
                }
                else {
                        cli();
                        save_users();
                        sei();
```

```
                                print_to_LCD("A User was removed");
                        }
                        prev_state = &remove_state;
                        next_state = &remove_state;
                        state = &message;
                        clear_input();
                }
}

/*
 * Function:  print_soda_menu
 * --------------------
 * Prints the soda menu
 *
 */
void print_soda_menu(void) {
        clear_LCD();
        print_to_LCD("1: Cola             2: Zingo");
}

/*
 * Function:  print_menu
 * --------------------
 * Prints the menu
 *
 */
void print_menu(void) {
        clear_LCD();
        print_to_LCD("1: Alcohol");
        move_cursor(20);
        print_to_LCD("2: Soda");
}

/*
 * Function:  flowmeter_init
 * --------------------
 * Initialize flowmeter
 *
 */
void flowmeter_init() {

        DDRD  &= ~(1<<PD2);                // Set PD2 as input (Using for interupt INT0)
        PORTD &= ~(1<<PD2);                // Enable PD2 pull-up resistor

        MCUCR |= (1<<ISC01 | 1<<ISC00);        // Trigger INT0 on rising edge
}
```

```
/*
 * Function:  flowmeter_init2
 * --------------------
 * Initialize flowmeter
 *
 */
void flowmeter_init2() {

        DDRD  &= ~(1<<PD3);                 // Set PD2 as input (Using for interupt INT0)
        PORTD &= ~(1<<PD3);                 // Enable PD2 pull-up resistor

        MCUCR |= (1<<ISC11 | 1<<ISC10);        // Trigger INT0 on rising edge
}

/*
 * Function:  timer0_init
 * --------------------
 * Initialize timer0
 *
 */
void timer0_init() {

        // set up timer with prescaler = 256
        TCCR0 |= (1 << CS02) | (1 << CS00);

        // initialize counter
        TCNT0 = 0;

        // enable overflow interrupt
        TIMSK |= (1 << TOIE0);


}

/*
 * Function:  drink_allowed
 * --------------------
 * Checks if the user logged in are allowed to order a alcoholic beverage
 *
 */
int drink_allowed(void) {
        return !user[current_user].counter;
}
```

```c
/*
 * Function:  handle_keyboard
 * --------------------
 * Handle common user interactions from the user
 *        such as going back, logout, clear a character
 *        and confirm a input
 *
 */
int handle_keyboard(int input_size, char* str) {
        char ch;
        if ((ch = get_char())) {
                if (ch == CONFIRM) {
                        chr_offset = 0;
                        return 1;
                }
                else if (ch == CLEAR) {
                        if (chr_offset > 0) {
                                str[--chr_offset] = 0;
                                clear_a_character();
                        }
                }
                else if (ch == LOGOUT) {
                        chr_offset = 0;
                        disconnect();
                }
                else if (ch == BACK) {
                        chr_offset = 0;
                        clear_input();
                        state = prev_state;
                        handle_state_change();
                }
                else {
                        if (chr_offset < input_size) {
                                str[chr_offset++] = ch;
                                if (state == &new_password || state == &set_password)
                                        send_a_character('*');
                                else
                                        send_a_character(ch);
                        }
                }
        }
        return 0;
}

/*
 * Function:  handle_state_change
 * --------------------
 *        Print message for each state change
 *
 */
void handle_state_change(void) {
        if (state == &set_name) {
                clear_LCD();
                print_to_LCD("Type username:");
        }
        else if (state == &set_password) {
                clear_LCD();
                print_to_LCD("Type password:");
        }
        else if (state == &new_name) {
                clear_LCD();
                print_to_LCD("Type new username:");
        }
        else if (state == &new_password) {
                clear_LCD();
                print_to_LCD("Type new password:");
        }
        else if (state == &menu) {
                print_menu();
        }
        else if (state == &soda_menu) {
                print_soda_menu();
        }
        else if (state == &pour_alcohol) {
                clear_LCD();
                print_to_LCD("Pour Alcohol");
        }
        else if (state == &message) {

        }
        else if (state == &admin_menu) {
                clear_LCD();
                print_to_LCD("1: Add user");
                move_cursor(20);
                print_to_LCD("2: Remove user");
        }
        else if (state == &remove_state) {
                clear_LCD();
                print_to_LCD("Type username of user to remove:");
        }
}
```

```
/*
 * Function:  disconnect
 * --------------------
 *        Handle disconnect of a user(logout)
 *
 */
void disconnect(void) {
        clear_input();
        logout();
        prev_state = state = &set_name;
        handle_state_change();
}

/*
 * Function:  clear_input
 * --------------------
 *        Clear name and password buffer
 *
 */
void clear_input(void) {
        memset(current_name, 0, NAME_SIZE);
        memset(current_pass, 0, PASS_SIZE);
}

/*
 * Function:  int32_to_string
 * --------------------
 *        Put a 32 bit integer into a string
 *
 * n: the integer
 * str: the buffer or string to put the integer
 *
 */
void int32_to_string(uint32_t n, char* str) {
        char temp[10];
        int i = 0;
        while (n > 0) {
                temp[i++] = (n % 10) + 48;
                n /= 10;
        }
        for (int i2 = 0; i2 < i; i2++)
                str[i2] = temp[i - i2 - 1];
        if (i == 0) {
                str[0] = 48;
                str[1] = '\0';
        }
        else
                str[i] = '\0';
}

/*
 * Function:  print_counter
 * --------------------
 *        Print counter of the current user
 *
 */
void print_counter(void) {
        if (time >= SECOND) {
                time = 0;
                move_cursor(17);
                char str[4];
                int32_to_string(user[current_user].counter/SECOND, str);
                int padding = 3 - strlen(str);
                while (padding--)
                        send_a_character(' ');
                print_to_LCD(str);
        }
}
```