```
::::::::::::::
./source/shop/clients.h
::::::::::::::
#pragma once

#include <stdio.h>

#include "utils/list.h"
#include "shop/barecode.h"
#include "model/model.h"

#define CLIENT_FIRST_NAME_SIZE 64
#define CLIENT_LAST_NAME_SIZE 64
#define CLIENT_EMAIL_SIZE 64

typedef struct
{
    BareCode id;

    char firstname[CLIENT_FIRST_NAME_SIZE];
    char lastname[CLIENT_LAST_NAME_SIZE];
    char email[CLIENT_EMAIL_SIZE];

    int points;
} Client;

typedef List ClientsList;

Client *clients_lookup(ClientsList *clients, BareCode id);

int clients_generate_id(ClientsList *clients);

Model clients_model_create(void);
::::::::::::::
./source/shop/users.h
::::::::::::::
#pragma once

#include "utils/list.h"
#include "model/model.h"

typedef List UsersList;

Model users_model_create(void);

User *users_lookup(UsersList *users, const char *login);
::::::::::::::
./source/shop/barecode.h
::::::::::::::
#pragma once

#define EXTRA "\e[31mEXTRA\e[0m"

typedef int BareCode;

void barecode_print(BareCode barecode);

BareCode barecode_input(const char *prompt);
::::::::::::::
./source/shop/basket.h
::::::::::::::
#pragma once

#include "shop/barecode.h"
#include "shop/clients.h"
#include "shop/stocks.h"
#include "utils/list.h"

typedef struct
```

```
{
    BareCode barecode;
    int quantity;
    bool is_consigne;
} BasketItem;

typedef struct
{
    bool pay_with_point;
    StockList *stocks;
    List *items;
    Client *owner;
} Basket;

Basket *basket_create(StockList *stocks, Client *owner);

void basket_destroy(Basket *basket);

void basket_add_item(Basket *this, BareCode barecode, bool is_consigned, int quantity);

Model basket_model_create(void);

float basket_pay(User *user, Basket *this, FILE *fout);

BasketItem *Basket_lookup(Basket *this, BareCode id, bool is_consigned);
::::::::::::::
./source/shop/stocks.h
::::::::::::::
#pragma once

#include <stdbool.h>
#include <stdio.h>

#include "model/model.h"
#include "shop/barecode.h"
#include "utils/list.h"

#define ITEM_CATEGORY_LIST(__ENTRY) \
    __ENTRY(UNDEFINED)              \
    __ENTRY(OTHER)                  \
    __ENTRY(ALCOHOL)                \
    __ENTRY(DRINK)                  \
    __ENTRY(FRESH_PRODUCT)          \
    __ENTRY(COMPUTER)               \
    __ENTRY(ELECTRONIC)             \
    __ENTRY(TOYS)

#define ITEM_ENUM_ENTRY(__x) ITEM_##__x,

typedef enum
{
    ITEM_CATEGORY_LIST(ITEM_ENUM_ENTRY)
        __ITEM_CATEGORY_COUNT
} ItemCategory;

#define ITEM_LABEL_SIZE 64

typedef struct
{
    BareCode id;
    char label[ITEM_LABEL_SIZE];

    int quantity;
    float price;
    int discount; // in pourcent
    ItemCategory category;
    bool isConsigned;
    float consignedValue;
} Item;
```

```
typedef List StockList;

void stocks_display(StockList *stocks);

void stocks_display_consigned(StockList *stocks);

Item *stocks_lookup_item(StockList *stocks, BareCode barecode);

Model stocks_model_create(void);

int stocks_generate_id(StockList *stocks);
::::::::::::::
./source/utils/logger.h
::::::::::::::
#pragma once

#include <stdio.h>
#include <stdarg.h>

enum
{
    LOGGER_TRACE,
    LOGGER_DEBUG,
    LOGGER_INFO,
    LOGGER_WARN,
    LOGGER_ERROR,
    LOGGER_FATAL
};

#define log_trace(...) log_log(LOGGER_TRACE, __VA_ARGS__)
#define log_debug(...) log_log(LOGGER_DEBUG, __VA_ARGS__)
#define log_info(...) log_log(LOGGER_INFO, __VA_ARGS__)
#define log_warn(...) log_log(LOGGER_WARN, __VA_ARGS__)
#define log_error(...) log_log(LOGGER_ERROR, __VA_ARGS__)
#define log_fatal(...) log_log(LOGGER_FATAL, __VA_ARGS__)

void log_log(int level, const char *fmt, ...);
::::::::::::::
./source/utils/assert.h
::::::::::::::
#pragma once

#include <assert.h>

#define ASSERT_NOT_REACHED() assert(!"REACHED");
::::::::::::::
./source/utils/math.h
::::::::::::::
#pragma once

#define max(__a, __b) ((__a) > (__b) ? (__a) : (__b))

#define min(__a, __b) ((__a) < (__b) ? (__a) : (__b))
::::::::::::::
./source/utils/variant.h
::::::::::::::
#pragma once

#define VARIANT_STRING_SIZE 128
#define VARIANT_SERIALIZED_SIZE 256

typedef enum
{
    VARIANT_INT,
    VARIANT_FLOAT,
    VARIANT_STRING,
} VarianType;
```

```
typedef struct
{
    VarianType type;

    union {
        long int as_int;
        float as_float;
    };

    char as_string[VARIANT_STRING_SIZE + 1];
} Variant;

Variant vint(long int value);

Variant vfloat(float value);

Variant vstring(const char *value);

Variant vstringf(const char *fmt, ...);

int variant_cmp(Variant left, Variant right);

Variant variant_deserialize(const char *source);

void variant_serialize(Variant value, char *destination);
::::::::::::::
./source/utils/string.h
::::::::::::::
#pragma once

#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>

typedef uint32_t Codepoint;

void strnapd(char *str, char c, size_t n);

bool str_start_with(const char *pre, const char *str);

bool is_numeric(int c);

bool is_letter(int c);

bool is_white_space(int c);

bool str_is_int(const char *str);

bool str_is_float(const char *str);

size_t utf8len(const char *s);

uint32_t strhash(const uint8_t *str);

int strutf8(uint8_t *out, Codepoint utf);

int utf8str(const uint8_t *in, Codepoint *out);
::::::::::::::
./source/utils/terminal.h
::::::::::::::
#pragma once

int terminal_read_key(void);

void terminal_enter_rawmode(void);

void terminal_exit_rawmode(void);

void terminal_set_cursor_position(int x, int y);
```

```
void terminal_get_size(int *width, int *height);

void terminal_clear(void);

void terminal_save_cursor(void);

void terminal_restore_cursor(void);

void terminal_enable_alternative_screen_buffer(void);

void terminal_disable_alternative_screen_buffer(void);

void terminal_hide_cursor(void);

void terminal_show_cursor(void);
::::::::::::::
./source/utils/renderer.h
::::::::::::::
#pragma once

#include <stdbool.h>
#include <utils/string.h>

typedef enum
{
    COLOR_BLACK,
    COLOR_RED,
    COLOR_GREEN,
    COLOR_ORANGE,
    COLOR_BLUE,
    COLOR_MAGENTA,
    COLOR_CYAN,
    COLOR_WHITE,
    COLOR_BRIGHT_BLACK = 60,
    COLOR_BRIGHT_RED,
    COLOR_BRIGHT_GREEN,
    COLOR_BRIGHT_ORANGE,
    COLOR_BRIGHT_BLUE,
    COLOR_BRIGHT_MAGENTA,
    COLOR_BRIGHT_CYAN,
    COLOR_BRIGHT_WHITE,
} Color;

typedef enum
{
    TEXT_CENTER,
    TEXT_LEFT,
    TEXT_RIGHT,
} TextAlign;

typedef struct
{
    int x;
    int y;
} Point;

typedef struct
{
    int x;
    int y;
    int width;
    int height;
} Region;

typedef struct
{
    bool bold;
    bool underline;
```

```
    Color foreground;
    Color background;

    TextAlign align;
} Style;

#define DEFAULT_STYLE \
    (Style) { false, false, COLOR_WHITE, COLOR_BLACK, TEXT_LEFT }

#define DISABLED_DEFAULT_STYLE \
    (Style) { false, false, COLOR_BRIGHT_BLACK, COLOR_BLACK, TEXT_LEFT }

#define INVERTED_STYLE \
    (Style) { false, false, COLOR_BLACK, COLOR_WHITE, TEXT_LEFT }

#define DISABLED_INVERTED_STYLE \
    (Style) { false, false, COLOR_BLACK, COLOR_BRIGHT_BLACK, TEXT_LEFT }

#define ALTERNATIVE_STYLE \
    (Style) { false, false, COLOR_WHITE, COLOR_BRIGHT_BLACK, TEXT_LEFT }

#define RED_STYLE \
    (Style) { false, false, COLOR_RED, COLOR_BLACK, TEXT_LEFT }

#define BLUE_STYLE \
    (Style) { false, false, COLOR_BLUE, COLOR_BLACK, TEXT_LEFT }

#define WHITE_STYLE \
    (Style) { false, false, COLOR_WHITE, COLOR_BLACK, TEXT_LEFT }

#define BOLD_STYLE \
    (Style) { true, false, COLOR_WHITE, COLOR_BLACK, TEXT_LEFT }

#define UNDERLINE_STYLE \
    (Style) { false, true, COLOR_WHITE, COLOR_BLACK, TEXT_LEFT }

Style style_regular(Style style);

Style style_bold(Style style);

Style style_with_background(Style style, Color background);

Style style_with_foreground(Style style, Color foreground);

Style style_centered(Style style);

Style style_inverted(Style style);

typedef struct
{
    Codepoint codepoint;
    Style style;
} Cell;

typedef struct
{
    int width;
    int height;

    Region clipstack[16];
    int clipstack_top;

    Cell *cells;
    int cells_allocated;
} Surface;

Surface *surface_create(void);
```

```
void surface_destroy(Surface *this);

void surface_update(Surface *this);

void surface_render(Surface *this);

Region surface_region(Surface *this);

int surface_width(Surface *this);

int surface_height(Surface *this);

void surface_push_clip(Surface *this, Region clip);

void surface_pop_clip(Surface *this);

void surface_clear(Surface *this, Style style);

void surface_plot(Surface *this, Codepoint codepoint, int x, int y, Style style);

void surface_fill(Surface *this, Codepoint codepoint, Region region, Style style);

void surface_text(Surface *this, const char *text, int x, int y, int width, Style style);

void surface_plot_line(Surface *this, Codepoint codepoint, int x0, int y0, int x1, int
y1, Style style);
::::::::::::::
./source/utils/list.h
::::::::::::::
#pragma once

#include <stdbool.h>

#define list_foreach(item, list) for (ListItem *item = list->head; item != NULL; item =
item->next)

typedef struct ListItem
{
    void *value;

    struct ListItem *prev;
    struct ListItem *next;
} ListItem;

typedef struct list
{
    int count;

    ListItem *head;
    ListItem *tail;
} List;

typedef bool (*ListComparator)(void *left, void *right);

List *list_create(void);

void list_destroy(List *this);

List *list_clone(List *this);

void list_clear(List *this);

void list_insert_sorted(List *this, void *value, ListComparator comparator);

bool list_peek(List *this, void **value);

bool list_peekback(List *this, void **value);

bool list_peekat(List *this, int index, void **value);
```

```
int list_indexof(List *this, void *value);

void list_push(List *this, void *value);

void list_pushback(List *this, void *value);

bool list_pop(List *this, void **value);

bool list_popback(List *this, void **value);

bool list_contains(List *this, void *value);

bool list_remove(List *this, void *value);

#define list_empty(__list) ((__list)->count == 0)

#define list_any(__list) ((__list)->count != 0)

#define list_count(__list) ((__list)->count)
::::::::::::::
./source/utils/input.h
::::::::::::::
#pragma once

#include <stdbool.h>

#include "model/user.h"

typedef enum
{
    INPUT_INVALID,
    INPUT_VALID,
    INPUT_OK,
} InputValidState;

typedef void (*ListCallback)(const char *user_input, void *args);

void setup_terminal_for_user_input(void);

void restore_terminal_after_user_input(void);

#define YES 1
#define NO 0

bool user_yes_no(const char *prompt, bool default_choice);

int user_select(User *user, const char *prompt, const char *options[]);

void user_input(const char *prompt, const char *format, char *result);

void user_input_password(const char *prompt, char *result, int n);
::::::::::::::
./source/view/views.h
::::::::::::::
#pragma once

#include "shop/basket.h"
#include "shop/clients.h"
#include "shop/stocks.h"
#include "shop/users.h"

void user_login(UsersList *users, StockList *stocks, ClientsList *clients);

void home_select_what_todo(User *user, UsersList *users, StockList *stocks, ClientsList
*clients);

void cashier_select_what_todo(User *user, Basket *basket, StockList *stocks);
```

```
void cashier_scan_items(Basket *basket, StockList *stock);

void cashier_return_consigned_bottles(Basket *basket, StockList *stock);

Client *cashier_input_card_id(ClientsList *clients);
::::::::::::::
./source/model/lexer.h
::::::::::::::
#pragma once

#include <stdio.h>
#include "utils/variant.h"

typedef enum
{
    TOKEN_INVALID,
    TOKEN_BEGIN,
    TOKEN_END,
    TOKEN_KEY,
    TOKEN_VALUE,
    TOKEN_EOF,
} TokenType;

typedef struct
{
    int ln;
    int col;

    TokenType type;
    char literal[VARIANT_SERIALIZED_SIZE];
} Token;

typedef struct
{
    int ln;
    int col;

    FILE *source;
} Lexer;

const char *token_type_string(Token *tok);

const char *token_type_string_type(TokenType type);

Token lexer_next_token(Lexer *lex);
::::::::::::::
./source/model/user.h
::::::::::::::
#pragma once

typedef enum
{
    ACCESS_NONE = -1,
    ACCESS_ADMIN,
    ACCESS_MANAGER,
    ACCESS_CASHIER,
    ACCESS_ALL,
} ModelAccess;

#define USER_FIRST_NAME_SIZE 64
#define USER_LAST_NAME_SIZE 64

typedef struct
{
    char login[8];
    char firstname[USER_FIRST_NAME_SIZE];
    char lastname[USER_LAST_NAME_SIZE];

    long int password;
```

```
    ModelAccess access;
} User;
::::::::::::::
./source/model/view.h
::::::::::::::
#pragma once

#include "model/model.h"
#include "shop/users.h"

typedef struct ModelViewState
{
    bool exited;

    int scroll;
    int slected;

    int width;
    int height;

    int sortby;
    bool sort_accending;
    bool sort_dirty;

    // FIXME: il y a peux etre moyen de faire mieux ^^
    int sorted[10000];
} ModelViewState;

void model_view_title(User *user, Surface *surface, const char *title);

void model_view_status_bar(Surface *surface, ModelViewState *state, Model model, void
*data);

void model_view(User *user, const char *title, Model model, void *data);
::::::::::::::
./source/model/action.h
::::::::::::::
#pragma once

#include "model/user.h"

struct ModelViewState;
struct Model;

typedef void (*ModelActionCallback)(User *user, Surface *surface, struct ModelViewState
*state, struct Model model, void *data, int row);

typedef struct
{
    int key_codepoint;
    ModelActionCallback callback;

    const char *name;
    const char *desciption;
} ModelAction;

void quit_ModelActionCallback(User *user, Surface *surface, struct ModelViewState *state,
struct Model model, void *data, int row);
void help_ModelActionCallback(User *user, Surface *surface, struct ModelViewState *state,
struct Model model, void *data, int row);

void scroll_up_ModelActionCallback(User *user, Surface *surface, struct ModelViewState
*state, struct Model model, void *data, int row);
void scroll_down_ModelActionCallback(User *user, Surface *surface, struct ModelViewState
*state, struct Model model, void *data, int row);
void page_up_ModelActionCallback(User *user, Surface *surface, struct ModelViewState
*state, struct Model model, void *data, int row);
void page_down_ModelActionCallback(User *user, Surface *surface, struct ModelViewState
*state, struct Model model, void *data, int row);
```

```
void home_ModelActionCallback(User *user, Surface *surface, struct ModelViewState *state,
struct Model model, void *data, int row);
void end_ModelActionCallback(User *user, Surface *surface, struct ModelViewState *state,
struct Model model, void *data, int row);

void edit_ModelActionCallback(User *user, Surface *surface, struct ModelViewState *state,
struct Model model, void *data, int row);
void create_ModelActionCallback(User *user, Surface *surface, struct ModelViewState
*state, struct Model model, void *data, int row);
void delete_ModelActionCallback(User *user, Surface *surface, struct ModelViewState
*state, struct Model model, void *data, int row);

#define DEFAULT_MODEL_MOVE_ACTION {'q', quit_ModelActionCallback, "Quitter", "Quitter
l'inpecteur de modèle."},        \
                                  {'h', help_ModelActionCallback, "Aide", "Afficher
l'aide"},                          \
                                  {'k', scroll_up_ModelActionCallback, "Scroll haut",
"Scroller vers le haut."},       \
                                  {'j', scroll_down_ModelActionCallback, "Scroll base",
"Scroller vers le base"},      \
                                  {'K', page_up_ModelActionCallback, "Page haut",
"Scroller une page vers le haut."},  \
                                  {'J', page_down_ModelActionCallback, "Page bas",
"Scroller une page vers le base."}, \
                                  {'g', home_ModelActionCallback, "Début", "Scroller tout
en haut de la liste."},      \
                                  {'G', end_ModelActionCallback, "Fin", "Scroller tout en
bas de la liste."},

#define DEFAULT_MODEL_VIEW_ACTION {'q', quit_ModelActionCallback, "Quitter", "Quitter
l'inpecteur de modèle."},          \
                                  {'h', help_ModelActionCallback, "Aide", "Afficher
l'aide"},                              \
                                  {'k', scroll_up_ModelActionCallback, "Scroll haut",
"Scroller vers le haut."},         \
                                  {'j', scroll_down_ModelActionCallback, "Scroll base",
"Scroller vers le base"},        \
                                  {'K', page_up_ModelActionCallback, "Page haut",
"Scroller une page vers le haut."},      \
                                  {'J', page_down_ModelActionCallback, "Page bas",
"Scroller une page vers le base."},    \
                                  {'g', home_ModelActionCallback, "Début", "Scroller tout
en haut de la liste."},        \
                                  {'G', end_ModelActionCallback, "Fin", "Scroller tout en
bas de la liste."},            \
                                  {'e', edit_ModelActionCallback, "Éditer", "Éditer
l'élément actuelle."},                 \
                                  {'i', create_ModelActionCallback, "Créer", "Créer un
nouvelle élément dans la liste."}, \
                                  {'d', delete_ModelActionCallback, "Supprimer",
"Supprimer l'élément selectioner de la liste"},

#define END_MODEL_VIEW_ACTION {0, NULL, NULL, NULL},::::::::::::::
./source/model/model.h
::::::::::::::
#pragma once

#include <stdio.h>

#include "utils/variant.h"
#include "utils/renderer.h"
#include "model/action.h"
#include "model/user.h"

typedef enum
{
    ROLE_DATA,
    ROLE_DISPLAY,
```

```
    ROLE_EDITOR,
} ModelRole;

typedef ModelAccess (*ModelReadAccess)(void *data, int row, int column, User *user);
typedef ModelAccess (*ModelWriteAccess)(void *data, int row, int column, User *user);

typedef int (*ModelRowCount)(void *data);
typedef int (*ModelRowCreate)(void *data);
typedef void (*ModelRowDelete)(void *data, int index);

typedef int (*ModelColumnCount)(void);
typedef const char *(*ModelColumnName)(int index, ModelRole role);
typedef VarianType (*ModelColumnType)(int index, ModelRole role);
typedef Style (*ModelColumnStyle)(int index);

typedef Variant (*ModelGetData)(void *data, int row, int column, ModelRole role);
typedef void (*ModelSetData)(void *data, int row, int column, Variant value, ModelRole
role);

typedef ModelAction *(*ModelGetActions)(void);

typedef struct Model
{
    ModelReadAccess read_access;
    ModelWriteAccess write_access;

    ModelRowCount row_count;
    ModelRowCreate row_create;
    ModelRowDelete row_delete;

    ModelColumnCount column_count;
    ModelColumnName column_name;
    ModelColumnType column_type;
    ModelColumnStyle column_style;

    ModelGetData get_data;
    ModelSetData set_data;

    ModelGetActions get_actions;
} Model;

int model_get_column(Model model, const char *name);

void model_load(Model model, void *data, FILE *source);

void model_save(Model model, void *data, FILE *destination);

Variant model_get_data_with_access(Model model, void *data, int row, int column, User
*user, ModelRole role);

void model_set_data_with_access(Model model, void *data, int row, int column, Variant
value, User *user);
::::::::::::::
./source/shop/barecode.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>

#include "shop/barecode.h"
#include "utils/input.h"

const char *barecode_char[] = {
    "▐▌",
    "▐▌",
    "▐▌",
    "▐▌",
    "▐▌",
    "▐▌",
    "▐▌",
```

```
    " |",
    "| ",
    "|",
};

void barecode_print(BareCode barecode)
{
    char buffer[5];
    sprintf(buffer, "%04d", barecode);

    printf("\e[47;30m");
    for (int i = 0; i < 4; i++)
    {
        printf("%s", barecode_char[buffer[i] - '0']);
    }

    for (int i = 0; i < 4; i++)
    {
        printf("%s", barecode_char[buffer[i] - '0']);
    }
    printf("\e[0m");
}

BareCode barecode_input(const char *prompt)
{
    char raw_barecode[5];
    user_input(prompt, "####", raw_barecode);
    return (BareCode)atoi(raw_barecode);
}
::::::::::::::
./source/shop/clients.c
::::::::::::::
#include <string.h>
#include <stdlib.h>
#include <time.h>

#include "shop/clients.h"
#include "utils/assert.h"

typedef enum
{
    COL_CLIENTS_BARECODE,
    COL_CLIENTS_LASTNAME,
    COL_CLIENTS_FIRSTNAME,
    COL_CLIENTS_EMAIL,
    COL_CLIENTS_POINTS,

    __COL_CLIENTS_COUNT,
} ClientModelColumn;

int clients_generate_id(ClientsList *clients)
{
    int id;

    srand(time(NULL));

    do
    {
        id = rand() % 9999;
        if (list_count(clients) == 0)
            return id;

    } while (clients_lookup(clients, id));

    return id;
}

Client *clients_lookup(ClientsList *clients, BareCode id)
{
```

```
    list_foreach(item, clients)
    {
        Client *client = (Client *)item->value;

        if (client->id == id)
        {
            return client;
        }
    }

    return NULL;
}

ModelAccess clients_ModelReadAccess(ClientsList *clients, int row, int column, User
*user)
{
    (void)clients;
    (void)row;
    (void)column;
    (void)user;

    return ACCESS_ALL;
}

ModelAccess clients_ModelWriteAccess(ClientsList *clients, int row, int column, User
*user)
{
    (void)clients;
    (void)row;
    (void)user;

    if (column == COL_CLIENTS_BARECODE)
    {
        return ACCESS_ADMIN;
    }
    else if (column == COL_CLIENTS_POINTS)
    {
        return ACCESS_MANAGER;
    }
    else
    {
        return ACCESS_ALL;
    }
}

int clients_ModelRowCount(ClientsList *clients)
{
    return list_count(clients);
}

int clients_ModelRowCreate(ClientsList *clients)
{
    Client *new_client = malloc(sizeof(Client));
    *new_client = (Client){0};
    new_client->id = clients_generate_id(clients);

    list_pushback(clients, new_client);

    return list_count(clients) - 1;
}

void clients_ModelRowDelete(ClientsList *clients, int index)
{
    Client *client_to_remove;
    list_peekat(clients, index, (void **)&client_to_remove);
    list_remove(clients, client_to_remove);
}

int clients_ModelColumnCount(void)
```

```
{
    return __COL_CLIENTS_COUNT;
}

const char *clients_ModelColumnName(int index, ModelRole role)
{
    if (role == ROLE_DATA)
    {
        switch (index)
        {
        case COL_CLIENTS_BARECODE:
            return "BARECODE";

        case COL_CLIENTS_FIRSTNAME:
            return "FIRSTNAME";

        case COL_CLIENTS_LASTNAME:
            return "LASTNAME";

        case COL_CLIENTS_EMAIL:
            return "EMAIL";

        case COL_CLIENTS_POINTS:
            return "POINTS";
        }
    }
    else
    {
        switch (index)
        {
        case COL_CLIENTS_BARECODE:
            return "Code";

        case COL_CLIENTS_FIRSTNAME:
            return "Prénom";

        case COL_CLIENTS_LASTNAME:
            return "Nom";

        case COL_CLIENTS_EMAIL:
            return "E-mail";

        case COL_CLIENTS_POINTS:
            return "Points";
        }
    }

    ASSERT_NOT_REACHED();
}

VarianType clients_ModelColumnType(int index, ModelRole role)
{
    (void)role;

    switch (index)
    {
    case COL_CLIENTS_BARECODE:
        return VARIANT_INT;

    case COL_CLIENTS_FIRSTNAME:
        return VARIANT_STRING;

    case COL_CLIENTS_LASTNAME:
        return VARIANT_STRING;

    case COL_CLIENTS_EMAIL:
        return VARIANT_STRING;

    case COL_CLIENTS_POINTS:
```

```
        return VARIANT_INT;
    }

    ASSERT_NOT_REACHED();
}

Style clients_ModelColumnStyle(int index)
{
    switch (index)
    {
    case COL_CLIENTS_BARECODE:
        return style_centered(DEFAULT_STYLE);

    case COL_CLIENTS_FIRSTNAME:
        return style_centered(DEFAULT_STYLE);

    case COL_CLIENTS_LASTNAME:
        return style_centered(DEFAULT_STYLE);

    case COL_CLIENTS_EMAIL:
        return DEFAULT_STYLE;

    case COL_CLIENTS_POINTS:
        return style_centered(DEFAULT_STYLE);
    }

    ASSERT_NOT_REACHED();
}

Variant clients_ModelGetData(ClientsList *clients, int row, int column, ModelRole role)
{
    (void)role;

    Client *client;
    list_peekat(clients, row, (void **)&client);

    switch (column)
    {
    case COL_CLIENTS_BARECODE:
        if (role == ROLE_DATA)
            return vint(client->id);
        else
            return vstringf("%04d", client->id);
    case COL_CLIENTS_FIRSTNAME:
        return vstring(client->firstname);

    case COL_CLIENTS_LASTNAME:
        return vstring(client->lastname);

    case COL_CLIENTS_EMAIL:
        return vstring(client->email);

    case COL_CLIENTS_POINTS:
        if (role == ROLE_DATA)
        {
            return vint(client->points);
        }
        else
        {
            return vstringf("%4dpts", client->points);
        }
    }

    ASSERT_NOT_REACHED();
}

void clients_ModelSetData(ClientsList *clients, int row, int column, Variant value,
ModelRole role)
{
```

```
    (void)role;

    Client *client;
    list_peekat(clients, row, (void **)&client);
    assert(client);

    switch (column)
    {
    case COL_CLIENTS_BARECODE:
        client->id = value.as_int;
        break;

    case COL_CLIENTS_FIRSTNAME:
        strcpy(client->firstname, value.as_string);
        break;

    case COL_CLIENTS_LASTNAME:
        strcpy(client->lastname, value.as_string);
        break;

    case COL_CLIENTS_EMAIL:
        strcpy(client->email, value.as_string);
        break;

    case COL_CLIENTS_POINTS:
        client->points = value.as_int;
        break;
    default:
        ASSERT_NOT_REACHED();
    }
}

ModelAction clients_actions[] = {DEFAULT_MODEL_VIEW_ACTION END_MODEL_VIEW_ACTION};

ModelAction *clients_ModelGetActions(void)
{
    return clients_actions;
}

Model clients_model_create(void)
{
    return (Model){
        (ModelReadAccess)clients_ModelReadAccess,
        (ModelWriteAccess)clients_ModelWriteAccess,

        (ModelRowCount)clients_ModelRowCount,
        (ModelRowCreate)clients_ModelRowCreate,
        (ModelRowDelete)clients_ModelRowDelete,

        (ModelColumnCount)clients_ModelColumnCount,
        (ModelColumnName)clients_ModelColumnName,
        (ModelColumnType)clients_ModelColumnType,
        (ModelColumnStyle)clients_ModelColumnStyle,

        (ModelGetData)clients_ModelGetData,
        (ModelSetData)clients_ModelSetData,

        (ModelGetActions)clients_ModelGetActions,
    };
}
::::::::::::::
./source/shop/basket.c
::::::::::::::
#include <stdlib.h>
#include <string.h>

#include "model/view.h"
#include "shop/basket.h"
#include "utils/assert.h"
```

```c
#include "utils/math.h"
#include "utils/terminal.h"

Basket *basket_create(StockList *stocks, Client *owner)
{
    Basket *this = malloc(sizeof(Basket));

    this->pay_with_point = false;
    this->items = list_create();
    this->stocks = stocks;
    this->owner = owner;

    return this;
}

void basket_destroy(Basket *this)
{
    list_destroy(this->items);
    free(this);
}

BasketItem *Basket_lookup(Basket *this, BareCode id, bool is_consigned)
{
    list_foreach(item, this->items)
    {
        BasketItem *b = (BasketItem *)item->value;

        if (b->barecode == id && b->is_consigne == is_consigned)
        {
            return b;
        }
    }
    return NULL;
}

void basket_add_item(Basket *this, BareCode barecode, bool is_consigned, int quantity)
{
    BasketItem *existingItem;
    existingItem = Basket_lookup(this, barecode, is_consigned);

    if (existingItem != NULL)
        existingItem->quantity += quantity;
    else
    {
        BasketItem *item = malloc(sizeof(BasketItem));

        item->barecode = barecode;
        item->quantity = quantity;
        item->is_consigne = is_consigned;

        list_pushback(this->items, item);
    }
}

float basket_pay(User *user, Basket *this, FILE *fout)
{
    printf("Votre caissier: %s.\n\n", user->lastname);

    if (fout)
    {
        fprintf(fout, "Voici le contenu du panier : \n\n");
        fprintf(fout, " N°.art | Dénomination               | Vidange | Qte | Prix
unitaire | Montant \n");
        fprintf(fout, "-----------------------------------------------------------------
------------\n");
    }

    float basket_total = 0;
    float basket_discount = 0;
```

```
    list_foreach(item, this->items)
    {
        BasketItem *basket_item = (BasketItem *)item->value;
        Item *stock_item = stocks_lookup_item(this->stocks, basket_item->barecode);

        fprintf(fout, "  %04d  |", basket_item->barecode);
        fprintf(fout, " %-26s |", stock_item->label);

        if (basket_item->is_consigne)
        {
            fprintf(fout, " oui     |");
        }
        else
        {
            fprintf(fout, "         |");
        }

        if (basket_item->quantity)
        {
            fprintf(fout, " x%-2d |", basket_item->quantity);
        }
        else
        {
            fprintf(fout, "      ");
        }

        float item_unit_value = 0.0;
        float item_unit_discount = 0.0;

        if (basket_item->is_consigne)
        {
            item_unit_value = -stock_item->consignedValue;
            fprintf(fout, " %5.2f         |", item_unit_value);
        }
        else
        {
            item_unit_value = stock_item->price;
            item_unit_discount = item_unit_value * (stock_item->discount / 100.0);

            if (stock_item->discount)
            {
                fprintf(fout, " %5.2f  -%-2d%%   |", item_unit_value, stock_item-
>discount);
            }
            else
            {
                fprintf(fout, " %5.2f         |", item_unit_value);
            }
        }

        fprintf(fout, " %7.2f \n", item_unit_value * basket_item->quantity);

        basket_total += item_unit_value * basket_item->quantity;
        basket_discount += item_unit_discount * basket_item->quantity;
    }

    fprintf(fout, "\n-----------------------------------------------------------------------
-----------\n\n");

    float basket_point_discount = 0.0;

    fprintf(fout, "Total hors réduction: %.2f€\n", basket_total);

    if (this->owner && this->pay_with_point)
    {
        int point_used = min((basket_total - basket_discount) * 100, this->owner-
>points);
```

```
        fprintf(fout, "Reduction fidelité: %.2f€ (%2dpts)\n", point_used / 100.0,
 point_used);

        basket_point_discount = (point_used / 100.0);
        this->owner->points -= point_used;
    }

    float basket_final_total = basket_total - basket_discount - basket_point_discount;

    fprintf(fout, "Réduction: %.2f€\n", basket_discount);
    fprintf(fout, "\nTotal à payer: %.2f€\n\n", basket_final_total);

    if (this->owner)
    {
        fprintf(fout, "Vous avez gagnez %dpts\n", (int)basket_final_total / 10);
        this->owner->points += basket_total / 10;
    }

    // Apply stocks changes.

    list_foreach(item, this->items)
    {
        BasketItem *item_in_basket = (BasketItem *)item->value;

        if (!item_in_basket->is_consigne)
        {
            stocks_lookup_item(this->stocks, item_in_basket->barecode)->quantity -=
 item_in_basket->quantity;
        }
    }

    return basket_total;
 }

 typedef enum
 {
    COL_BASKET_BARECODE,
    COL_BASKET_LABEL,
    COL_BASKET_CONSIGNE,
    CAL_BASKET_UNIT_PRICE,
    COL_BASKET_QUANTITY,
    COL_BASKET_REDUCTION,
    COL_BASKET_PRICE,

    __COL_BASKET_COUNT,
 } BasketModelColumn;

 ModelAccess basket_ModelReadAccess(Basket *basket, int row, int column, User *user)
 {
    (void)basket;
    (void)row;
    (void)column;
    (void)user;

    return ACCESS_ALL;
 }

 ModelAccess basket_ModelWriteAccess(Basket *basket, int row, int column, User *user)
 {
    (void)basket;
    (void)row;
    (void)user;

    if (column == COL_BASKET_CONSIGNE)
    {
        BasketItem *item_in_basket = NULL;
        list_peekat(basket->items, row, (void **)&item_in_basket);
        Item *item_in_stock = stocks_lookup_item(basket->stocks, item_in_basket-
 >barecode);
```

```
        if (item_in_stock != NULL && item_in_stock->isConsigned)
        {
            return ACCESS_ALL;
        }

        return ACCESS_NONE;
    }

    if (column == COL_BASKET_BARECODE ||
        column == COL_BASKET_QUANTITY ||
        column == COL_BASKET_CONSIGNE)
    {
        return ACCESS_ALL;
    }
    else
    {
        return ACCESS_NONE;
    }
}

int basket_ModelRowCount(Basket *basket)
{
    return list_count(basket->items);
}

int basket_ModelRowCreate(Basket *basket)
{
    BasketItem *item = malloc(sizeof(BasketItem));

    *item = (BasketItem){0};

    list_pushback(basket->items, item);

    return list_count(basket->items) - 1;
}

void basket_ModelRowDelete(Basket *basket, int index)
{
    BasketItem *item_to_remove;
    list_peekat(basket->items, index, (void **)&item_to_remove);
    list_remove(basket->items, item_to_remove);
}

int basket_ModelColumnCount(void)
{
    return __COL_BASKET_COUNT;
}

const char *basket_ModelColumnName(int index, ModelRole role)
{
    (void)role;

    if (role == ROLE_DISPLAY)
    {
        switch (index)
        {
        case COL_BASKET_BARECODE:
            return "N°.art";

        case COL_BASKET_LABEL:
            return "Dénomination";

        case COL_BASKET_CONSIGNE:
            return "Vidange";

        case CAL_BASKET_UNIT_PRICE:
            return "Prix Unitaire";
```

```
        case COL_BASKET_QUANTITY:
            return "Quantitée";

        case COL_BASKET_REDUCTION:
            return "Réduction";

        case COL_BASKET_PRICE:
            return "Montant";
        }
    }
    else
    {

        switch (index)
        {
        case COL_BASKET_BARECODE:
            return "BARECODE";

        case COL_BASKET_LABEL:
            return "LABEL";

        case COL_BASKET_CONSIGNE:
            return "CONSIGNE";

        case CAL_BASKET_UNIT_PRICE:
            return "UNIT_PRICE";

        case COL_BASKET_QUANTITY:
            return "QUANTITY";

        case COL_BASKET_REDUCTION:
            return "REDUCTION";

        case COL_BASKET_PRICE:
            return "PRICE";
        }
    }

    ASSERT_NOT_REACHED();
}

VarianType basket_ModelColumnType(int index, ModelRole role)
{
    (void)role;

    switch (index)
    {
    case COL_BASKET_BARECODE:
        return VARIANT_INT;

    case COL_BASKET_LABEL:
        return VARIANT_STRING;

    case COL_BASKET_CONSIGNE:
        return VARIANT_INT;

    case CAL_BASKET_UNIT_PRICE:
        return VARIANT_FLOAT;

    case COL_BASKET_QUANTITY:
        return VARIANT_INT;

    case COL_BASKET_REDUCTION:
        return VARIANT_FLOAT;

    case COL_BASKET_PRICE:
        return VARIANT_FLOAT;
    }
```

```
    ASSERT_NOT_REACHED();
}

Style basket_ModelColumnStyle(int index)
{
    (void)index;

    return style_centered(DEFAULT_STYLE);
}

Variant basket_ModelGetData(Basket *basket, int row, int column, ModelRole role)
{
    (void)role;

    BasketItem *basket_item;
    list_peekat(basket->items, row, (void **)&basket_item);
    Item *item = stocks_lookup_item(basket->stocks, basket_item->barecode);

    if (item == NULL)
    {
        return vstring("(null)");
    }
    else
    {
        if (column == COL_BASKET_QUANTITY)
        {
            if (role == ROLE_DATA || role == ROLE_EDITOR)
            {
                return vint(basket_item->quantity);
            }
            else
            {
                return vstringf("x%-3d", basket_item->quantity);
            }
        }

        if (basket_item->is_consigne)
        {
            switch (column)
            {
            case COL_BASKET_BARECODE:
                return vint(basket_item->barecode);

            case COL_BASKET_LABEL:
                return vstring(item->label);

            case COL_BASKET_CONSIGNE:
                if (role == ROLE_DATA || role == ROLE_EDITOR)
                {
                    return vint(1);
                }
                else
                {
                    return vstring("oui");
                }

            case CAL_BASKET_UNIT_PRICE:
                if (role == ROLE_DATA || role == ROLE_EDITOR)
                {
                    return vfloat(item->consignedValue);
                }
                else
                {
                    return vstringf("%5.2f€", item->consignedValue);
                }

            case COL_BASKET_REDUCTION:
                return vstring("-");
```

```
            case COL_BASKET_PRICE:
                if (role == ROLE_DATA || role == ROLE_EDITOR)
                {
                    return vfloat(-(item->consignedValue * basket_item->quantity));
                }
                else
                {
                    return vstringf("-%5.2f€", (item->consignedValue * basket_item-
>quantity));
                }
            }

            ASSERT_NOT_REACHED();
        }
        else
        {
            switch (column)
            {
            case COL_BASKET_BARECODE:
                return vint(basket_item->barecode);

            case COL_BASKET_LABEL:
                return vstring(item->label);

            case COL_BASKET_CONSIGNE:
                if (role == ROLE_DATA || role == ROLE_EDITOR)
                {
                    return vint(0);
                }
                else
                {
                    return vstring("non");
                }

            case CAL_BASKET_UNIT_PRICE:
                if (role == ROLE_DATA || role == ROLE_EDITOR)
                {
                    return vfloat(item->price);
                }
                else
                {
                    return vstringf("%5.2f€", item->price);
                }

            case COL_BASKET_REDUCTION:
                if (role == ROLE_DISPLAY)
                {
                    if (item->discount)
                    {
                        return vstringf("-%2d%%", item->discount);
                    }
                    else
                    {
                        return vstring("-");
                    }
                }
                else
                {
                    return vint(item->discount);
                }

            case COL_BASKET_PRICE:
            {
                float total_value = (item->price * basket_item->quantity);

                if (role == ROLE_DATA || role == ROLE_EDITOR)
                {
                    return vfloat(total_value);
                }
```

```
                else
                {
                    return vstringf("%5.2f€", total_value);
                }
            }
            }

            ASSERT_NOT_REACHED();
        }
    }
}

void basket_ModelSetData(Basket *basket, int row, int column, Variant value, ModelRole
role)
{
    (void)role;

    BasketItem *basket_item;
    list_peekat(basket->items, row, (void **)&basket_item);

    switch (column)
    {
    case COL_BASKET_BARECODE:
        basket_item->barecode = value.as_int;
        break;
    case COL_BASKET_CONSIGNE:
        basket_item->is_consigne = value.as_int;
        break;
    case COL_BASKET_QUANTITY:
        basket_item->quantity = value.as_int;
        break;
    }
}

ModelAction basket_actions[] = {DEFAULT_MODEL_VIEW_ACTION END_MODEL_VIEW_ACTION};

ModelAction *basket_ModelGetActions(void)
{
    return basket_actions;
}

Model basket_model_create(void)
{
    return (Model){
        (ModelReadAccess)basket_ModelReadAccess,
        (ModelWriteAccess)basket_ModelWriteAccess,

        (ModelRowCount)basket_ModelRowCount,
        (ModelRowCreate)basket_ModelRowCreate,
        (ModelRowDelete)basket_ModelRowDelete,

        (ModelColumnCount)basket_ModelColumnCount,
        (ModelColumnName)basket_ModelColumnName,
        (ModelColumnType)basket_ModelColumnType,
        (ModelColumnStyle)basket_ModelColumnStyle,

        (ModelGetData)basket_ModelGetData,
        (ModelSetData)basket_ModelSetData,

        (ModelGetActions)basket_ModelGetActions,
    };
}::::::::::::::
./source/shop/stocks.c
::::::::::::::
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "model/model.h"
```

```c
#include "shop/stocks.h"
#include "utils/assert.h"
#include "utils/logger.h"

typedef enum
{
    COL_ITEM_BARECODE,
    COL_ITEM_LABEL,
    COL_ITEM_PRICE,
    COL_ITEM_QUANTITY,
    COL_ITEM_CONSIGNED,
    COL_ITEM_DISCOUNT,
    COL_ITEM_CATEGORY,

    __COL_ITEM_COUNT,
} StockModelColumn;

#define ITEM_STRING_ENTRY(__x) #__x,
static const char *item_category_string[] = {ITEM_CATEGORY_LIST(ITEM_STRING_ENTRY) NULL};

void stocks_display(StockList *stocks)
{
    list_foreach(item, stocks)
    {
        Item *itemInStocks = (Item *)item->value;
        barecode_print(itemInStocks->id);

        printf(" %04d %16s %16s %5.2f€", itemInStocks->id, itemInStocks->label,
item_category_string[itemInStocks->category], itemInStocks->price);

        if (itemInStocks->discount != 0)
        {
            printf(" \e[103;30m-%d%%\e[0m", itemInStocks->discount);
        }

        printf("\n");
    }
}

void stocks_display_consigned(StockList *stocks)
{
    list_foreach(item, stocks)
    {
        Item *itemInStocks = (Item *)item->value;
        if (itemInStocks->isConsigned)
        {
            printf("%04d %s %s\n", itemInStocks->id, itemInStocks->label,
item_category_string[itemInStocks->category]);
        }
    }
}

Item *stocks_lookup_item(StockList *stocks, BareCode barecode)
{
    list_foreach(item, stocks)
    {
        Item *itemInStocks = (Item *)item->value;

        if (itemInStocks->id == barecode)
        {
            return itemInStocks;
        }
    }

    return NULL;
}

int stocks_generate_id(StockList *stocks)
{
```

```
    int id;

    srand(time(NULL));

    do
    {
        id = rand() % 9999;

        if (list_count(stocks) == 0)
            return id;
    } while (stocks_lookup_item(stocks, id));

    return id;
}

ModelAccess stocks_ModelReadAccess(StockList *stocks, int row, int column, User *user)
{
    (void)stocks;
    (void)row;
    (void)column;
    (void)user;

    return ACCESS_ALL;
}

ModelAccess stocks_ModelWriteAccess(StockList *stocks, int row, int column, User *user)
{
    (void)stocks;
    (void)row;
    (void)user;

    if (column == COL_ITEM_BARECODE)
    {
        return ACCESS_ADMIN;
    }
    else
    {
        return ACCESS_MANAGER;
    }
}

int stocks_ModelRowCount(StockList *stock)
{
    return list_count(stock);
}

int stocks_ModelRowCreate(StockList *stocks)
{
    Item *new_item = malloc(sizeof(Item));
    *new_item = (Item){0};
    new_item->id = stocks_generate_id(stocks);

    list_pushback(stocks, new_item);

    return list_count(stocks) - 1;
}

void stocks_ModelRowDelete(StockList *stocks, int index)
{
    Item *item_to_remove;
    list_peekat(stocks, index, (void **)&item_to_remove);
    list_remove(stocks, item_to_remove);
}

int stocks_ModelColumnCount(void)
{
    return __COL_ITEM_COUNT;
}
```

```
const char *stocks_ModelColumnName(int index, ModelRole role)
{
    if (role == ROLE_DATA)
    {
        switch (index)
        {
        case COL_ITEM_BARECODE:
            return "BARECODE";

        case COL_ITEM_LABEL:
            return "LABEL";

        case COL_ITEM_PRICE:
            return "PRICE";

        case COL_ITEM_QUANTITY:
            return "QUANTITY";

        case COL_ITEM_CONSIGNED:
            return "CONSIGNED";

        case COL_ITEM_DISCOUNT:
            return "DISCOUNT";

        case COL_ITEM_CATEGORY:
            return "CATEGORY";
        }
    }
    else
    {
        switch (index)
        {
        case COL_ITEM_BARECODE:
            return "N°.art";

        case COL_ITEM_LABEL:
            return "Dénomination";

        case COL_ITEM_PRICE:
            return "Prix";

        case COL_ITEM_QUANTITY:
            return "En Stock";

        case COL_ITEM_CONSIGNED:
            return "Consigne";

        case COL_ITEM_DISCOUNT:
            return "Réduction";

        case COL_ITEM_CATEGORY:
            return "Section";
        }
    }

    ASSERT_NOT_REACHED();
}

VarianType stocks_ModelColumnType(int index, ModelRole role)
{
    (void)role;

    switch (index)
    {
    case COL_ITEM_BARECODE:
        return VARIANT_INT;

    case COL_ITEM_LABEL:
```

```
        return VARIANT_STRING;

    case COL_ITEM_PRICE:
        return VARIANT_FLOAT;

    case COL_ITEM_QUANTITY:
        return VARIANT_INT;

    case COL_ITEM_CONSIGNED:
        return VARIANT_FLOAT;

    case COL_ITEM_DISCOUNT:
        return VARIANT_INT;

    case COL_ITEM_CATEGORY:
        return VARIANT_INT;
    }

    ASSERT_NOT_REACHED();
}

Style stocks_ModelColumnStyle(int index)
{
    switch (index)
    {
    case COL_ITEM_BARECODE:
        return style_centered(DEFAULT_STYLE);

    case COL_ITEM_LABEL:
        return DEFAULT_STYLE;

    case COL_ITEM_PRICE:
        return style_centered(DEFAULT_STYLE);

    case COL_ITEM_QUANTITY:
        return style_centered(RED_STYLE);

    case COL_ITEM_CONSIGNED:
        return style_centered(DEFAULT_STYLE);

    case COL_ITEM_DISCOUNT:
        return style_centered(BLUE_STYLE);

    case COL_ITEM_CATEGORY:
        return style_centered(DEFAULT_STYLE);
    }

    ASSERT_NOT_REACHED();
}

Variant stocks_ModelGetData(StockList *stock, int row, int column, ModelRole role)
{
    Item *item;
    assert(list_peekat(stock, row, (void **)&item));

    switch (column)
    {
    case COL_ITEM_BARECODE:
        if (role == ROLE_DATA || role == ROLE_EDITOR)
        {
            return vint(item->id);
        }
        else
        {
            return vstringf("%04d", item->id);
        }

    case COL_ITEM_LABEL:
        return vstring(item->label);
```

```
    case COL_ITEM_PRICE:
        if (role == ROLE_DATA || role == ROLE_EDITOR)
        {
            return vfloat(item->price);
        }
        else
        {
            return vstringf("%5.2f€", item->price);
        }

    case COL_ITEM_QUANTITY:
        if (role == ROLE_DATA || role == ROLE_EDITOR)
        {
            return vint(item->quantity);
        }
        else
        {
            if (item->quantity)
            {
                return vstringf("x%-3d", item->quantity);
            }
            else
            {
                return vstring("VIDE!");
            }
        }

    case COL_ITEM_CONSIGNED:
        if (role == ROLE_DATA || role == ROLE_EDITOR)
        {
            return vfloat(item->consignedValue);
        }
        else
        {
            if (item->consignedValue == 0)
            {
                return vstring("   -  ");
            }
            else
            {
                return vstringf("%5.2f€", item->consignedValue);
            }
        }

    case COL_ITEM_DISCOUNT:
        if (role == ROLE_DATA || role == ROLE_EDITOR)
        {
            return vint(item->discount);
        }
        else
        {
            if (item->discount != 0)
            {
                return vstringf("%3d%%", -item->discount);
            }

            return vstring("");
        }

    case COL_ITEM_CATEGORY:
    {
        if (role == ROLE_DATA || role == ROLE_EDITOR)
        {
            return vint(item->category);
        }
        else
        {
            return vstring(item_category_string[item->category]);
```

```
        }
    }
    }

    ASSERT_NOT_REACHED();
}

void stocks_ModelSetData(StockList *stock, int row, int column, Variant value, ModelRole
role)
{
    (void)role;

    Item *item;
    list_peekat(stock, row, (void **)&item);
    assert(item);

    switch (column)
    {
    case COL_ITEM_BARECODE:
        item->id = value.as_int;
        break;

    case COL_ITEM_LABEL:
        strcpy(item->label, value.as_string);
        break;

    case COL_ITEM_PRICE:
        item->price = value.as_float;
        break;

    case COL_ITEM_QUANTITY:
        item->quantity = value.as_int;
        break;

    case COL_ITEM_CONSIGNED:
        item->consignedValue = value.as_float;
        if (item->consignedValue > 0)
            item->isConsigned = true;
        break;

    case COL_ITEM_DISCOUNT:
        item->discount = value.as_int;
        break;

    case COL_ITEM_CATEGORY:
        if (__ITEM_CATEGORY_COUNT > value.as_int && value.as_int >= 0)
        {
            item->category = value.as_int;
        }
        else
        {
            item->category = 0;
        }

        break;

    default:
        ASSERT_NOT_REACHED();
    }
}

ModelAction stocks_actions[] = {DEFAULT_MODEL_VIEW_ACTION END_MODEL_VIEW_ACTION};

ModelAction *stocks_ModelGetActions(void)
{
    return stocks_actions;
}

Model stocks_model_create(void)
```

```
{
    return (Model){
        (ModelReadAccess)stocks_ModelReadAccess,
        (ModelWriteAccess)stocks_ModelWriteAccess,

        (ModelRowCount)stocks_ModelRowCount,
        (ModelRowCreate)stocks_ModelRowCreate,
        (ModelRowDelete)stocks_ModelRowDelete,

        (ModelColumnCount)stocks_ModelColumnCount,
        (ModelColumnName)stocks_ModelColumnName,
        (ModelColumnType)stocks_ModelColumnType,
        (ModelColumnStyle)stocks_ModelColumnStyle,

        (ModelGetData)stocks_ModelGetData,
        (ModelSetData)stocks_ModelSetData,
        (ModelGetActions)stocks_ModelGetActions,
    };
}
::::::::::::::
./source/shop/users.c
::::::::::::::
#include <stdlib.h>
#include <string.h>

#include "shop/users.h"
#include "utils/assert.h"
#include "utils/string.h"

typedef enum
{
    COL_USERS_LOGIN,
    COL_USERS_LASTNAME,
    COL_USERS_FIRSTNAME,
    COL_USERS_PASSWORD,
    COL_USERS_ACCESS,

    __COL_USERS_COUNT,
} UserModelColumn;

User *users_lookup(UsersList *users, const char *login)
{
    list_foreach(item, users)
    {
        User *user = (User *)item->value;

        if (strcmp(login, user->login) == 0)
        {
            return user;
        }
    }

    return NULL;
}

ModelAccess users_ModelReadAccess(UsersList *users, int row, int column, User *user)
{
    if (list_indexof(users, user) == row)
    {
        return user->access;
    }
    else
    {
        switch (column)
        {
        case COL_USERS_LOGIN:
            return ACCESS_ADMIN;

        case COL_USERS_FIRSTNAME:
```

```
            return ACCESS_ALL;

        case COL_USERS_LASTNAME:
            return ACCESS_ALL;

        case COL_USERS_PASSWORD:
            return ACCESS_ADMIN;

        case COL_USERS_ACCESS:
            return ACCESS_ALL;
        }

        ASSERT_NOT_REACHED();
    }
}

ModelAccess users_ModelWriteAccess(UsersList *users, int row, int column, User *user)
{

    switch (column)
    {
    case COL_USERS_LOGIN:
        return ACCESS_ADMIN;

    case COL_USERS_FIRSTNAME:
        if (list_indexof(users, user) == row)
        {
            return ACCESS_ALL;
        }
        else
        {
            return ACCESS_MANAGER;
        }

    case COL_USERS_LASTNAME:
        if (list_indexof(users, user) == row)
        {
            return ACCESS_ALL;
        }
        else
        {
            return ACCESS_MANAGER;
        }

    case COL_USERS_PASSWORD:
        if (list_indexof(users, user) == row)
        {
            return ACCESS_ALL;
        }
        else
        {
            return ACCESS_MANAGER;
        }

    case COL_USERS_ACCESS:
        return ACCESS_ADMIN;
    }

    ASSERT_NOT_REACHED();
}

int users_ModelRowCount(UsersList *users)
{
    return list_count(users);
}

int users_ModelRowCreate(UsersList *users)
{
    User *new_user = malloc(sizeof(User));
```

```
    *new_user = (User){0};

    list_pushback(users, new_user);

    return list_count(users) - 1;
}

void users_ModelRowDelete(UsersList *users, int index)
{
    User *user_to_remove;
    list_peekat(users, index, (void **)&user_to_remove);
    list_remove(users, user_to_remove);
}

int users_ModelColumnCount(void)
{
    return __COL_USERS_COUNT;
}

const char *users_ModelColumnName(int index, ModelRole role)
{
    if (role == ROLE_DATA)
    {
        switch (index)
        {
        case COL_USERS_LOGIN:
            return "LOGIN";

        case COL_USERS_FIRSTNAME:
            return "FIRSTNAME";

        case COL_USERS_LASTNAME:
            return "LASTNAME";

        case COL_USERS_PASSWORD:
            return "PASSWORD";

        case COL_USERS_ACCESS:
            return "ACCESS";
        }
    }
    else
    {
        switch (index)
        {
        case COL_USERS_LOGIN:
            return "Login";

        case COL_USERS_FIRSTNAME:
            return "Prénom";

        case COL_USERS_LASTNAME:
            return "Nom";

        case COL_USERS_PASSWORD:
            return "Hash";

        case COL_USERS_ACCESS:
            return "Accès";
        }
    }

    ASSERT_NOT_REACHED();
}

VarianType users_ModelColumnType(int index, ModelRole role)
{
    switch (index)
```

```
    {
    case COL_USERS_LOGIN:
        return VARIANT_STRING;

    case COL_USERS_FIRSTNAME:
        return VARIANT_STRING;

    case COL_USERS_LASTNAME:
        return VARIANT_STRING;

    case COL_USERS_PASSWORD:
        if (role == ROLE_EDITOR)
        {
            return VARIANT_STRING;
        }
        else
        {
            return VARIANT_INT;
        }

    case COL_USERS_ACCESS:
        return VARIANT_INT;
    }

    ASSERT_NOT_REACHED();
}

Style users_ModelColumnStyle(int index)
{
    switch (index)
    {
    case COL_USERS_LOGIN:
        return style_centered(DEFAULT_STYLE);

    case COL_USERS_FIRSTNAME:
        return style_centered(DEFAULT_STYLE);

    case COL_USERS_LASTNAME:
        return style_centered(DEFAULT_STYLE);

    case COL_USERS_PASSWORD:
        return style_centered(DEFAULT_STYLE);

    case COL_USERS_ACCESS:
        return style_centered(DEFAULT_STYLE);
    }

    ASSERT_NOT_REACHED();
}

Variant users_ModelGetData(UsersList *users, int row, int column, ModelRole role)
{
    (void)role;

    User *user;
    list_peekat(users, row, (void **)&user);

    switch (column)
    {
    case COL_USERS_LOGIN:
        return vstring(user->login);

    case COL_USERS_FIRSTNAME:
        return vstring(user->firstname);

    case COL_USERS_LASTNAME:
        return vstring(user->lastname);

    case COL_USERS_PASSWORD:
```

```
        if (role == ROLE_EDITOR || role == ROLE_DISPLAY)
        {
            return vstring("*****");
        }
        else
        {
            return vint(user->password);
        }

    case COL_USERS_ACCESS:
        if (role == ROLE_DISPLAY)
        {
            switch (user->access)
            {
            case ACCESS_ADMIN:
                return vstring("Admin");

            case ACCESS_MANAGER:
                return vstring("Manager");

            case ACCESS_CASHIER:
                return vstring("Caissier");

            default:
                ASSERT_NOT_REACHED();
            }
        }
        else
        {
            return vint(user->access);
        }
    }

    ASSERT_NOT_REACHED();
}

void users_ModelSetData(UsersList *users, int row, int column, Variant value, ModelRole
role)
{
    User *user;
    list_peekat(users, row, (void **)&user);
    assert(user);

    switch (column)
    {
    case COL_USERS_LOGIN:
        strcpy(user->login, value.as_string);

        break;

    case COL_USERS_FIRSTNAME:
        strcpy(user->firstname, value.as_string);
        break;

    case COL_USERS_LASTNAME:
        strcpy(user->lastname, value.as_string);
        break;

    case COL_USERS_PASSWORD:
        if (role == ROLE_EDITOR)
        {
            if (strcmp(value.as_string, "*****") != 0)
            {
                user->password = strhash((const uint8_t *)value.as_string);
            }
        }
        else
        {
            user->password = value.as_int;
```

```
        }
        break;

    case COL_USERS_ACCESS:
        user->access = value.as_int;
        break;
    default:
        ASSERT_NOT_REACHED();
    }
}

ModelAction users_actions[] = {DEFAULT_MODEL_VIEW_ACTION END_MODEL_VIEW_ACTION};

ModelAction *users_ModelGetActions(void)
{
    return users_actions;
}

Model users_model_create(void)
{
    return (Model){
        (ModelReadAccess)users_ModelReadAccess,
        (ModelWriteAccess)users_ModelWriteAccess,

        (ModelRowCount)users_ModelRowCount,
        (ModelRowCreate)users_ModelRowCreate,
        (ModelRowDelete)users_ModelRowDelete,

        (ModelColumnCount)users_ModelColumnCount,
        (ModelColumnName)users_ModelColumnName,
        (ModelColumnType)users_ModelColumnType,
        (ModelColumnStyle)users_ModelColumnStyle,

        (ModelGetData)users_ModelGetData,
        (ModelSetData)users_ModelSetData,

        (ModelGetActions)users_ModelGetActions,
    };
}
::::::::::::::
./source/utils/logger.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>

#include "utils/logger.h"

static const char *level_names[] = {
    "TRACE",
    "DEBUG",
    "INFO",
    "ATTENTION",
    "ERREUR",
    "FATAL",
};

static const char *level_colors[] = {
    "\e[94m",
    "\e[36m",
    "\e[32m",
    "\e[33m",
    "\e[31m",
    "\e[35m",
};

void log_log(int level, const char *fmt, ...)
```

```
{
    time_t t = time(NULL);
    struct tm *lt = localtime(&t);

    va_list args;
    char buf[16];
    buf[strftime(buf, sizeof(buf), "%H:%M:%S", lt)] = '\0';

    fprintf(
        stderr, "%s %s%s \e[0m",
        buf, level_colors[level], level_names[level]);

    va_start(args, fmt);
    vfprintf(stderr, fmt, args);
    va_end(args);
    fprintf(stderr, "\n");
    fflush(stderr);
}:::::::::::::
./source/utils/variant.c
:::::::::::::
#include <stdarg.h>
#include <stdio.h>
#include <string.h>

#include "utils/assert.h"
#include "utils/logger.h"
#include "utils/string.h"
#include "utils/variant.h"

Variant vint(long int value)
{
    Variant v = (Variant){
        .type = VARIANT_INT,
        .as_int = value,
    };

    snprintf(v.as_string, VARIANT_STRING_SIZE, "%ld", value);

    return v;
}

Variant vfloat(float value)
{
    Variant v = (Variant){
        .type = VARIANT_FLOAT,
        .as_float = value,
    };

    snprintf(v.as_string, VARIANT_STRING_SIZE, "%.2f", value);

    return v;
}

Variant vstring(const char *value)
{
    Variant v = (Variant){.type = VARIANT_STRING};

    assert(strlen(value) < VARIANT_STRING_SIZE);

    strncpy(v.as_string, value, VARIANT_STRING_SIZE);

    return v;
}

Variant vstringf(const char *fmt, ...)
{
    Variant v = (Variant){.type = VARIANT_STRING};

    va_list args;
```

```
    va_start(args, fmt);
    vsnprintf(v.as_string, VARIANT_STRING_SIZE, fmt, args);
    va_end(args);

    return v;
}

int variant_cmp(Variant left, Variant right)
{
    if (left.type == VARIANT_INT && right.type == VARIANT_INT)
    {
        return left.as_int - right.as_int;
    }
    else if (left.type == VARIANT_FLOAT && right.type == VARIANT_FLOAT)
    {
        return left.as_float - right.as_float;
    }
    else
    {
        return strcmp(left.as_string, right.as_string);
    }
}

Variant variant_deserialize(const char *source)
{
    Variant value = vint(-69420);

    if (source[0] == '"')
    {
        char buffer[VARIANT_STRING_SIZE] = {0};

        bool escaped = true;

        for (int i = 1; source[i]; i++)
        {
            char c = source[i];

            if (c == '\\' && !escaped)
            {
                escaped = true;
            }
            else if (escaped || (c != '\\' && c != '"'))
            {
                strnapd(buffer, c, VARIANT_STRING_SIZE);

                escaped = false;
            }
        }

        value = vstring(buffer);
    }
    else if (str_is_int(source))
    {
        long int v;
        sscanf(source, "%ld", &v);
        value = vint(v);
    }
    else if (str_is_float(source))
    {
        float v;
        sscanf(source, "%f", &v);
        value = vfloat(v);
    }

    return value;
}

void variant_serialize(Variant value, char *destination)
{
```

```
    destination[0] = '\0';

    switch (value.type)
    {
    case VARIANT_INT:
        sprintf(destination, "%ld", value.as_int);
        break;

    case VARIANT_FLOAT:
        sprintf(destination, "%f", value.as_float);
        break;

    case VARIANT_STRING:
        strnapd(destination, '"', VARIANT_SERIALIZED_SIZE);

        for (int i = 0; value.as_string[i]; i++)
        {
            char c = value.as_string[i];

            if (c == '"' || c == '\\')
            {
                strnapd(destination, '\\', VARIANT_SERIALIZED_SIZE);
            }

            strnapd(destination, c, VARIANT_SERIALIZED_SIZE);
        }

        strnapd(destination, '"', VARIANT_SERIALIZED_SIZE);
        break;

    default:
        ASSERT_NOT_REACHED();
    }
}:::::::::::::::
./source/utils/string.c
:::::::::::::::
#include <string.h>

#include "utils/string.h"

void strnapd(char *str, char c, size_t n)
{
    for (size_t i = 0; i < (n - 1); i++)
    {
        if (str[i] == '\0')
        {
            str[i] = c;
            str[i + 1] = '\0';
            return;
        }
    }
}

bool str_start_with(const char *pre, const char *str)
{
    int lenpre = strlen(pre),
        lenstr = strlen(str);

    return lenstr < lenpre ? false : memcmp(pre, str, lenpre) == 0;
}

bool is_numeric(int c)
{
    return c == '.' || (c >= '0' && c <= '9');
}

bool is_letter(int c)
{
    return c == '_' || (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
```

```
}

bool is_white_space(int c)
{
    return c == ' ' || c == '\n' || c == '\r' || c == '\t';
}

bool str_is_int(const char *str)
{
    for (int i = 0; str[i]; i++)
    {
        if (!is_numeric(str[i]) || str[i] == '.')
        {
            return false;
        }
    }

    return true;
}

bool str_is_float(const char *str)
{
    for (int i = 0; str[i]; i++)
    {
        if (!is_numeric(str[i]))
        {
            return false;
        }
    }

    return true;
}

size_t utf8len(const char *s)
{
    size_t count = 0;
    while (*s)
    {
        count += (*s++ & 0xC0) != 0x80;
    }
    return count;
}

uint32_t strhash(const unsigned char *str)
{
    unsigned long hash = 5381;
    int c;

    while ((c = *str++))
        hash = ((hash << 5) + hash) + c;

    return hash;
}

int strutf8(uint8_t *out, Codepoint utf)
{
    if (utf <= 0x7F)
    {
        out[0] = (uint8_t)utf;
        out[1] = 0;

        return 1;
    }
    else if (utf <= 0x07FF)
    {
        out[0] = (uint8_t)(((utf >> 6) & 0x1F) | 0xC0);
        out[1] = (uint8_t)(((utf >> 0) & 0x3F) | 0x80);
        out[2] = 0;
```

```
        return 2;
    }
    else if (utf <= 0xFFFF)
    {
        out[0] = (uint8_t)(((utf >> 12) & 0x0F) | 0xE0);
        out[1] = (uint8_t)(((utf >> 6) & 0x3F) | 0x80);
        out[2] = (uint8_t)(((utf >> 0) & 0x3F) | 0x80);
        out[3] = 0;

        return 3;
    }
    else if (utf <= 0x10FFFF)
    {
        out[0] = (uint8_t)(((utf >> 18) & 0x07) | 0xF0);
        out[1] = (uint8_t)(((utf >> 12) & 0x3F) | 0x80);
        out[2] = (uint8_t)(((utf >> 6) & 0x3F) | 0x80);
        out[3] = (uint8_t)(((utf >> 0) & 0x3F) | 0x80);
        out[4] = 0;

        return 4;
    }
    else
    {
        out[0] = (uint8_t)0xEF;
        out[1] = (uint8_t)0xBF;
        out[2] = (uint8_t)0xBD;
        out[3] = 0;

        return 0;
    }
}

int utf8str(const uint8_t *in, Codepoint *out)
{
    if ((in[0] & 0xf8) == 0xf0)
    {
        *out = ((0x07 & in[0]) << 18) |
               ((0x3f & in[1]) << 12) |
               ((0x3f & in[2]) << 6) |
               ((0x3f & in[3]));

        return 4;
    }
    else if ((in[0] & 0xf0) == 0xe0)
    {
        *out = ((0x0f & in[0]) << 12) |
               ((0x3f & in[1]) << 6) |
               ((0x3f & in[2]));

        return 3;
    }
    else if ((in[0] & 0xe0) == 0xc0)
    {
        *out = ((0x1f & in[0]) << 6) |
               ((0x3f & in[1]));

        return 2;
    }
    else
    {
        *out = in[0];

        return 1;
    }

    return 0;
}::::::::::::::
./source/utils/renderer.c
::::::::::::::
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "utils/assert.h"
#include "utils/math.h"
#include "utils/renderer.h"
#include "utils/string.h"
#include "utils/terminal.h"

Style style_regular(Style style)
{
    style.bold = false;
    style.underline = false;

    return style;
}

Style style_bold(Style style)
{
    style.bold = true;
    style.underline = false;

    return style;
}

Style style_centered(Style style)
{
    style.align = TEXT_CENTER;

    return style;
}

Style style_with_background(Style style, Color background)
{
    style.background = background;

    return style;
}

Style style_with_foreground(Style style, Color foreground)
{
    style.foreground = foreground;

    return style;
}

Style style_inverted(Style style)
{
    Color tmp = style.foreground;
    style.foreground = style.background;
    style.background = tmp;

    return style;
}

Surface *surface_create(void)
{
    Surface *this = malloc(sizeof(Surface));

    terminal_get_size(&this->width, &this->height);

    this->cells = calloc(this->width * this->height, sizeof(Cell));
    this->cells_allocated = this->width * this->height;

    this->clipstack_top = 0;

    return this;
}
```

```
void surface_destroy(Surface *this)
{
    free(this->cells);
    free(this);
}

void surface_update(Surface *this)
{
    assert(this->clipstack_top == 0);

    terminal_get_size(&this->width, &this->height);

    if (this->width * this->height > this->cells_allocated)
    {
        this->cells = realloc(this->cells, sizeof(Cell) * this->width * this->height);
        this->cells_allocated = this->width * this->height;
    }
}

static int style_cmp(Style a, Style b)
{
    return memcmp(&a, &b, sizeof(Style));
}

static void style_use(Style style)
{
    printf("\e[0m");

    if (style.bold)
    {
        printf("\e[1m");
    }

    if (style.underline)
    {
        printf("\e[4m");
    }

    printf("\e[3%d;4%dm", style.foreground, style.background);
}

void surface_render(Surface *this)
{
    terminal_set_cursor_position(0, 0);
    terminal_hide_cursor();

    Style current_style = DEFAULT_STYLE;
    style_use(current_style);

    for (int y = 0; y < this->height; y++)
    {
        for (int x = 0; x < this->width; x++)
        {
            Cell c = this->cells[x + y * this->width];

            if (style_cmp(current_style, c.style) != 0)
            {
                style_use(c.style);

                current_style = c.style;
            }

            if (c.codepoint == 0)
            {
                printf(" ");
            }
            else
            {
```

```
                uint8_t utf8[5];
                strutf8(utf8, c.codepoint);
                printf("%s", utf8);
            }
        }
    }

    terminal_show_cursor();
}

Region surface_clip(Surface *this)
{
    if (this->clipstack_top == 0)
    {
        return (Region){0, 0, this->width, this->height};
    }
    else
    {
        return this->clipstack[this->clipstack_top - 1];
    }
}

int surface_width(Surface *this)
{
    return surface_clip(this).width;
}

int surface_height(Surface *this)
{
    return surface_clip(this).height;
}

Region surface_region(Surface *this)
{
    return (Region){0, 0, surface_width(this), surface_height(this)};
}

void surface_push_clip(Surface *this, Region clip)
{
    clip.x += surface_clip(this).x;
    clip.y += surface_clip(this).y;

    this->clipstack[this->clipstack_top] = clip;
    this->clipstack_top++;
}

void surface_pop_clip(Surface *this)
{
    assert(this->clipstack_top > 0);
    this->clipstack_top--;
}

void surface_clear(Surface *this, Style style)
{
    surface_fill(this, ' ', surface_region(this), style);
}

void surface_plot(Surface *this, Codepoint codepoint, int x, int y, Style style)
{
    Region r = surface_clip(this);

    // FIXME: this condition look sily...
    if (x >= 0 && x < r.width && x + r.x >= 0 && x + r.x < this->width &&
        y >= 0 && y < r.height && y + r.y >= 0 && y + r.y < this->height)
    {
        x += r.x;
        y += r.y;

        this->cells[x + y * this->width] = (Cell){codepoint, style};
```

```
    }
}

void surface_fill(Surface *this, Codepoint codepoint, Region region, Style style)
{
    for (int x = 0; x < region.width; x++)
    {
        for (int y = 0; y < region.height; y++)
        {
            surface_plot(this, codepoint, region.x + x, region.y + y, style);
        }
    }
}

void surface_text(Surface *this, const char *text, int x, int y, int width, Style style)
{
    int textlenght = utf8len(text);

    int offset = 0;
    int content = min(textlenght, width);
    int padding = max(0, width - textlenght);

    if (style.align == TEXT_CENTER)
    {
        padding /= 2;
    }

    if (style.align == TEXT_RIGHT || style.align == TEXT_CENTER)
    {
        for (int i = 0; i < padding; i++)
        {
            surface_plot(this, ' ', x + i, y, style_regular(style));
        }

        offset += padding;
    }

    //FIXME: iterate over code point...

    int i = 0;
    while (i < content)
    {
        Codepoint cp;
        text += utf8str((uint8_t *)text, &cp);
        surface_plot(this, cp, x + i + offset, y, style);

        i++;
    }

    if (content < textlenght)
    {
        surface_plot(this, u'…', x + content + offset - 1, y, style);
    }

    offset += content;

    if (style.align == TEXT_LEFT || style.align == TEXT_CENTER)
    {
        for (int i = offset; i < width; i++)
        {
            surface_plot(this, ' ', x + i, y, style_regular(style));
        }
    }
}

void surface_plot_line_x_aligned(Surface *this, Codepoint codepoint, int x, int start,
int end, Style style)
{
    for (int i = start; i < end; i++)
```

```
    {
        surface_plot(this, codepoint, x, i, style);
    }
}

void surface_plot_line_y_aligned(Surface *this, Codepoint codepoint, int y, int start,
int end, Style style)
{
    for (int i = start; i < end; i++)
    {
        surface_plot(this, codepoint, i, y, style);
    }
}

void surface_plot_line_not_aligned(Surface *this, Codepoint codepoint, int x0, int y0,
int x1, int y1, Style style)
{
    int dx = abs(x1 - x0), sx = x0 < x1 ? 1 : -1;
    int dy = abs(y1 - y0), sy = y0 < y1 ? 1 : -1;
    int err = (dx > dy ? dx : -dy) / 2, e2;

    for (;;)
    {
        surface_plot(this, codepoint, x0, y0, style);

        if (x0 == x1 && y0 == y1)
            break;

        e2 = err;
        if (e2 > -dx)
        {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dy)
        {
            err += dx;
            y0 += sy;
        }
    }
}

void surface_plot_line(Surface *this, Codepoint codepoint, int x0, int y0, int x1, int
y1, Style style)
{
    if (x0 == x1)
    {
        surface_plot_line_x_aligned(this, codepoint, x0, min(y0, y1), max(y0, y1),
style);
    }
    else if (y0 == y1)
    {
        surface_plot_line_y_aligned(this, codepoint, y0, min(x0, x1), max(x0, x1),
style);
    }
    else
    {
        surface_plot_line_not_aligned(this, codepoint, x0, y0, x1, y1, style);
    }
}::::::::::::::
./source/utils/terminal.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <termios.h>

#include "utils/terminal.h"
```

```
#include "utils/assert.h"

void terminal_enter_rawmode(void)
{
    struct termios info;

    tcgetattr(0, &info);
    info.c_lflag &= ~(ECHO | ICANON);
    tcsetattr(0, TCSANOW, &info);
}

void terminal_exit_rawmode(void)
{
    struct termios info;

    tcgetattr(0, &info);
    info.c_lflag |= (ECHO | ICANON);
    tcsetattr(0, TCSAFLUSH, &info);
}

void terminal_set_cursor_position(int x, int y)
{
    printf("\e[%d;%dH", y + 1, x + 1);
}

void terminal_get_size(int *width, int *height)
{
    struct winsize w;
    ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);

    *width = w.ws_col;
    *height = w.ws_row;
}

void terminal_clear(void)
{
    printf("\e[J");
}

int terminal_read_key(void)
{
    fflush(stdout);
    terminal_enter_rawmode();

    char c;
    assert(read(STDIN_FILENO, &c, 1) == 1);

    terminal_exit_rawmode();

    return c;
}

void terminal_save_cursor(void)
{
}

void terminal_restore_cursor(void)
{
}

void terminal_enable_alternative_screen_buffer(void)
{
    printf("\e[?1049h");
}

void terminal_disable_alternative_screen_buffer(void)
{
    printf("\e[?1049l");
}
```

```
void terminal_hide_cursor(void)
{
    printf("\e[?25l");
}

void terminal_show_cursor(void)
{
    printf("\033[?25h");
}::::::::::::::
./source/utils/list.c
::::::::::::::
#include <stdlib.h>
#include "list.h"

List *list_create(void)
{
    List *this = malloc(sizeof(List));

    this->count = 0;
    this->head = NULL;
    this->tail = NULL;

    return this;
}

void list_destroy(List *this)
{
    list_clear(this);
    free(this);
}

List *list_clone(List *this)
{
    List *copy = list_create();

    list_foreach(i, this)
    {
        list_pushback(copy, i->value);
    }

    return copy;
}

void list_clear(List *this)
{
    ListItem *current = this->head;

    while (current)
    {
        ListItem *next = current->next;

        free(current->value);
        free(current);

        current = next;
    }

    this->count = 0;
    this->head = NULL;
    this->tail = NULL;
}

void list_insert_sorted(List *this, void *value, ListComparator comparator)
{
    if (this->head == NULL || comparator(value, this->head->value))
    {
        list_push(this, value);
    }
```

```
    else
    {
        ListItem *current = this->head;

        while (current->next != NULL && comparator(current->next->value, value))
        {
            current = current->next;
        }

        ListItem *item = malloc(sizeof(ListItem));

        item->prev = current;
        item->next = current->next;
        item->value = value;

        if (current->next == NULL)
        {
            this->tail = item;
        }
        else
        {
            current->next->prev = item;
        }

        current->next = item;

        this->count++;
    }
}

bool list_peek(List *this, void **value)
{
    if (this->head != NULL)
    {
        *value = this->head->value;

        return true;
    }
    else
    {
        *value = NULL;

        return false;
    }
}

bool list_peekback(List *this, void **value)
{
    if (this->tail != NULL)
    {
        *value = this->tail->value;

        return true;
    }
    else
    {
        return false;
    }
}

static void list_peekat_from_head(List *this, int index, void **value)
{
    ListItem *current = this->head;

    for (int i = 0; i < index; i++)
    {
        current = current->next;
    }
```

```
    *value = current->value;
}

static void list_peekat_from_back(List *this, int index, void **value)
{
    ListItem *current = this->tail;

    for (int i = 0; i < (this->count - index - 1); i++)
    {
        current = current->prev;
    }

    *value = current->value;
}

bool list_peekat(List *this, int index, void **value)
{
    if (this->count >= 1 && index >= 0 && index < this->count)
    {
        if (index < this->count / 2)
        {
            list_peekat_from_head(this, index, value);
        }
        else
        {
            list_peekat_from_back(this, index, value);
        }

        return true;
    }
    else
    {
        return false;
    }
}

int list_indexof(List *this, void *value)
{
    int index = 0;

    list_foreach(item, this)
    {
        if (item->value == value)
        {
            return index;
        }

        index++;
    }

    return -1;
}

void list_push(List *this, void *value)
{
    ListItem *item = malloc(sizeof(ListItem));

    item->prev = NULL;
    item->next = NULL;
    item->value = value;

    this->count++;

    if (this->head == NULL)
    {
        this->head = item;
        this->tail = item;
    }
    else
```

```
    {
        this->head->prev = item;
        item->next = this->head;
        this->head = item;
    }
}

bool list_pop(List *this, void **value)
{
    ListItem *item = this->head;

    if (this->count == 0)
    {
        return false;
    }
    else if (this->count == 1)
    {
        this->count = 0;
        this->head = NULL;
        this->tail = NULL;
    }
    else if (this->count > 1)
    {
        item->next->prev = NULL;
        this->head = item->next;

        this->count--;
    }

    if (value != NULL)
    {
        *value = item->value;
    }

    return true;
}

void list_pushback(List *this, void *value)
{
    ListItem *item = malloc(sizeof(ListItem));

    item->prev = NULL;
    item->next = NULL;
    item->value = value;

    this->count++;

    if (this->tail == NULL)
    {
        this->tail = item;
        this->head = item;
    }
    else
    {
        this->tail->next = item;
        item->prev = this->tail;
        this->tail = item;
    }
}

bool list_popback(List *this, void **value)
{
    ListItem *item = this->tail;

    if (this->count == 0)
    {
        return NULL;
    }
    else if (this->count == 1)
```

```
    {
        this->count = 0;
        this->head = NULL;
        this->tail = NULL;
    }
    else if (this->count > 1)
    {
        item->prev->next = NULL;
        this->tail = item->prev;

        this->count--;
    }

    if (value != NULL)
    {
        *value = item->value;
    }

    return true;
}

bool list_remove(List *this, void *value)
{
    list_foreach(item, this)
    {
        if (item->value == value)
        {
            if (item->prev != NULL)
            {
                item->prev->next = item->next;
            }
            else
            {
                this->head = item->next;
            }

            if (item->next != NULL)
            {
                item->next->prev = item->prev;
            }
            else
            {
                this->tail = item->prev;
            }

            this->count--;
            free(item->value);
            free(item);

            return true;
        }
    }

    return false;
}

bool list_contains(List *this, void *value)
{
    list_foreach(item, this)
    {
        if (item->value == value)
        {
            return true;
        }
    }

    return false;
}::::::::::::::
./source/utils/input.c
```

```
::::::::::::::
#include <assert.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "model/view.h"
#include "utils/input.h"
#include "utils/renderer.h"
#include "utils/string.h"
#include "utils/terminal.h"

InputValidState user_input_valid(const char *format, const char *input)
{
    for (int i = 0; input[i]; i++)
    {
        char in_c = input[i];
        if (format[i] == '_' && !(isalpha(in_c) || isdigit(in_c)))
            return INPUT_INVALID;

        if (format[i] == '.' && !isalpha(in_c))
            return INPUT_INVALID;

        if (format[i] == '#' && !isdigit(in_c))
            return INPUT_INVALID;
    }

    return INPUT_VALID;
}

bool user_yes_no(const char *prompt, bool default_choice)
{

    if (default_choice == NO)
    {
        printf("%s [Oui/\e[1mNon\e[0m]", prompt);
    }
    else
    {
        printf("%s [\e[1mOui\e[0m/Non]", prompt);
    }

    char c = terminal_read_key();

    if (default_choice == NO)
    {

        if (c == 'Y' || c == 'O' || c == 'y' || c == 'o')
        {
            printf(" Oui\n");

            return true;
        }
        else
        {
            printf(" Non\n");

            return false;
        }
    }
    else
    {
        if (c == 'N' || c == 'n')
        {
            printf(" Non\n");

            return false;
```

```
        }
        else
        {
            printf(" Oui\n");

            return true;
        }
    }
}

int user_select(User *user, const char *prompt, const char *options[])
{
    terminal_enable_alternative_screen_buffer();

    Surface *surface = surface_create();

    bool stop = false;
    int selected = 0;

    while (!stop)
    {
        surface_clear(surface, DEFAULT_STYLE);
        model_view_title(user, surface, prompt);

        for (int i = 0; options[i]; i++)
        {
            if (i == selected)
            {
                surface_text(surface, vstringf("> %s <", options[i]).as_string, 0, i,
surface_width(surface), style_centered(style_bold(DEFAULT_STYLE)));
            }
            else
            {
                surface_text(surface, options[i], 0, i, surface_width(surface),
style_centered(DEFAULT_STYLE));
            }
        }

        surface_text(surface, " [K] up / [J] down / [ENTER] select", 0,
surface_height(surface) - 1, surface_width(surface), style_inverted(DEFAULT_STYLE));

        surface_pop_clip(surface);

        surface_render(surface);
        surface_update(surface);

        char c = terminal_read_key();

        if (c == 'k')
        {
            if (selected > 0)
                selected--;
        }
        else if (c == 'j')
        {
            if (options[selected + 1] != NULL)
                selected++;
        }
        else if (c == '\n')
        {
            stop = true;
        }
    }

    surface_destroy(surface);

    terminal_disable_alternative_screen_buffer();

    return selected;
```

```
}

void user_input_password(const char *prompt, char *result, int n)
{
    terminal_enter_rawmode();

    char c;
    int index = 0;

    result[0] = '\0';

    printf("%s: ", prompt);

    do
    {
        c = terminal_read_key();

        if (c == 127 && index > 0)
        {
            index--;
            result[index] = '\0';
        }
        else if (iscntrl(c))
        {
            // do nothing with it
        }
        else if (index < n)
        {
            result[index] = c;
            result[index + 1] = '\0';
            index++;
        }
    } while (c != '\n');

    printf("\n");

    terminal_exit_rawmode();
}

void user_input(const char *prompt, const char *format, char *result)
{
    terminal_enter_rawmode();

    char c;
    int index = 0;

    result[0] = '\0';

    printf("\e[?25l%s: ", prompt);

    do
    {
        printf("\e[s\e[37m%s\e[0m\e[u", format);
        printf("\e[s%s%s\e[0m\e[u", user_input_valid(format, result) ? "\e[32m" :
"\e[31m", result);

        c = terminal_read_key();

        if (c == 127 && index > 0)
        {
            index--;
            result[index] = '\0';
        }
        else if (iscntrl(c))
        {
            // do nothing with it
        }
        else if (index < (int)strlen(format))
        {
```

```
            result[index] = c;
            result[index + 1] = '\0';
            index++;
        }
    } while (c != '\n' || !user_input_valid(format, result));

    printf("\033[?25h\n");

    terminal_exit_rawmode();
}::::::::::::::
./source/view/return_consigned_bottles.c
::::::::::::::
#include <stdlib.h>

#include "utils/input.h"
#include "utils/string.h"
#include "view/views.h"

void cashier_return_consigned_bottles(Basket *basket, StockList *stock)
{
    float totValue = 0.;
    BareCode bottle_barecode;
    char bottle_raw_barecode[5];
    char bottle_raw_count[5];
    int bottle_count;
    Item *bottle;

    do
    {
        stocks_display_consigned(stock);

        user_input("Inserez le codebarre de la bouteille à rendre", "####",
bottle_raw_barecode);
        bottle_barecode = atoi(bottle_raw_barecode);

        bottle = stocks_lookup_item(stock, bottle_barecode);

        if (bottle && bottle->isConsigned)
        {
            user_input("Entrez le nombre de bouteilles à rendre", "####",
bottle_raw_count);
            bottle_count = atoi(bottle_raw_count);

            totValue = bottle->consignedValue * bottle_count;
            basket_add_item(basket, bottle_barecode, true, bottle_count);
            printf("Vous allez recuperer %5.2f€\n", totValue);
        }
        else
        {
            printf("Erreur, le codebarre entré ne correpond pas à un article
consigne\n");
        }
    } while (user_yes_no("Voulez-vous continuer ?", YES) == YES);
}::::::::::::::
./source/view/cashier_input_card_id.c
::::::::::::::
#include <stdlib.h>
#include <unistd.h>

#include "utils/input.h"
#include "utils/logger.h"
#include "view/views.h"

#include "utils/terminal.h"

static Client *login_client(ClientsList *clients)
{
    BareCode extra_barecode;
    Client *client = NULL;
```

```
    do
    {
        extra_barecode = barecode_input("Inserez votre code " EXTRA);
        client = clients_lookup(clients, extra_barecode);

        log_info("Connection avec " EXTRA "#%04d...", extra_barecode);

        if (client == NULL)
        {
            log_error("Ce compte n'existe pas");

            if (user_yes_no("Erreur, identifiant incorrect, voulez-vous réessayer ? ",
YES) == NO)
            {
                return NULL;
            }
        }
    } while (client == NULL);

    return client;
}

static Client *new_client(ClientsList *clients)
{
    BareCode nouveau_client_id = clients_generate_id(clients);
    Client *nouveau_client = (Client *)malloc(sizeof(Client));

    terminal_enable_alternative_screen_buffer();

    do
    {
        user_input("Inserez votre nom ", ".....................", nouveau_client-
>lastname);
        user_input("Inserez votre prénom ", ".....................", nouveau_client-
>firstname);
        user_input("Inserez votre email ", "********************************",
nouveau_client->email);

        printf("\nVos informations :\n"

               "\tId: %d\n"
               "\tNom: %s\n"
               "\tPrenom: %s\n"
               "\tEmail: %s\n\n",

               nouveau_client_id,
               nouveau_client->firstname,
               nouveau_client->lastname,
               nouveau_client->email);

    } while (user_yes_no("Confirmer ?", NO) == NO);

    terminal_disable_alternative_screen_buffer();

    nouveau_client->id = nouveau_client_id;
    nouveau_client->points = 0;

    list_pushback(clients, nouveau_client);

    return nouveau_client;
}

Client *cashier_input_card_id(ClientsList *clients)
{
    const char *prompt = "Cher client, vous pouvez profiter des points extra grace au
compte EXTRA Colruyt";

    const char *choices[] = {
```

```
        "S'authentifier",
        "Créer un compte client Colruyt",
        "Continuer sans compte",
        NULL,
    };

    switch (user_select(NULL, prompt, choices))
    {
    case 0:
        return login_client(clients);

    case 1:
        if (user_yes_no("Acceptez-vous les conditions d'utilisation?", NO) == YES)
            return new_client(clients);

        break;
    case 2:
        log_info("Vous continuez sans compte...");
        break;
    }

    return NULL;
}::::::::::::::
./source/view/home_select_what_todo.c
::::::::::::::
#include <string.h>

#include "model/view.h"
#include "utils/input.h"
#include "utils/logger.h"
#include "utils/terminal.h"
#include "view/views.h"

void home_select_what_todo(User *user, UsersList *users, StockList *stock, ClientsList
*clients)
{
    const char *choices[] = {
        "Interface caissier",

        "Liste des produits",
        "Liste des clients",
        "Liste des employés",

        "  Sortir du programme",

        NULL,
    };

    do
    {
        switch (user_select(user, "Selectionnez une interface", choices))
        {
        case 0:
        {
            Basket *basket = basket_create(stock, cashier_input_card_id(clients));

            log_info("Bonjour et bienvenue chez Colruyt");

            cashier_select_what_todo(user, basket, stock);

            basket_destroy(basket);
            break;
        }

        case 1:
            model_view(user, "Liste des produits", stocks_model_create(), stock);
            break;

        case 2:
```

```
            model_view(user, "Liste des clients", clients_model_create(), clients);
            break;

        case 3:
            model_view(user, "Liste des employés", users_model_create(), users);
            break;

        default:
            log_info("Bye bye :)");
            return;
        }
    } while (1);
}:::::::::::::::
./source/view/cashier_scan_item.c
:::::::::::::::
#include <stdlib.h>
#include <string.h>

#include "utils/input.h"
#include "utils/logger.h"
#include "view/views.h"

void cashier_scan_items(Basket *basket, StockList *stock)
{
    Item *item;

    BareCode item_barecode = -1;
    char item_raw_barecode[5];

    int item_quantity = -1;
    char item_raw_quantity[5];

    bool exited = false;

    do
    {
        stocks_display(stock);
        user_input("Inserez le codebarre de l'article", "####", item_raw_barecode);

        if (strlen(item_raw_barecode) == 0)
        {
            if (user_yes_no("Voulez-vous continuer a ajouter des articles au panier ?",
YES) == NO)
            {
                exited = true;
            }
        }
        else
        {
            item_barecode = atoi(item_raw_barecode);
            item = stocks_lookup_item(stock, item_barecode);

            if (item == NULL)
            {
                log_error("Code bare %04d incorrect!", item_barecode);
                continue;
            }

            user_input("Entrez la quatite que vous souhaitez acheter", "####",
item_raw_quantity);
            item_quantity = atoi(item_raw_quantity);

            if (item_quantity == 0)
            {
                log_warn("Achat annulé!");
                continue;
            }

            BasketItem *item_in_basket = Basket_lookup(basket, item_barecode, false);
```

```
            int total_asked_quantity = (item_in_basket ? item_in_basket->quantity : 0) +
 item_quantity;

            if (total_asked_quantity > item->quantity)
            {
                log_error("Stock insufisants (%d en stock mais %d demander)", item-
 >quantity, total_asked_quantity);
                continue;
            }

            basket_add_item(basket, item_barecode, false, item_quantity);
        }
    } while (!exited);
 }::::::::::::::
 ./source/view/user_login.c
 ::::::::::::::
 #include <unistd.h>

 #include "utils/input.h"
 #include "view/views.h"

 void user_login(UsersList *users, StockList *stocks, ClientsList *clients)
 {
     do
     {
         char user_login[17];
         user_input("\n\tLogin", "****************", user_login);

         char user_password[17];
         user_input_password("\tPassword", user_password, 17);
         long int hash = strhash((uint8_t *)user_password);

         User *user = users_lookup(users, user_login);

         if (user != NULL && hash == user->password)
         {
             home_select_what_todo(user, users, stocks, clients);
             return;
         }
         else
         {
             sleep(3);
             printf("\n\e[35mMot de passe ou nom d'utilisateur incorrect!\e[0m\n");
         }
     } while (true);
 }
 ::::::::::::::
 ./source/view/cashier_select_what_todo.c
 ::::::::::::::
 #include <stdlib.h>
 #include <string.h>

 #include "model/view.h"
 #include "shop/basket.h"
 #include "shop/clients.h"
 #include "utils/input.h"
 #include "utils/logger.h"
 #include "utils/string.h"
 #include "utils/terminal.h"
 #include "view/views.h"

 void cashier_select_what_todo(User *user, Basket *basket, StockList *stocks)
 {
     const char *choices[] = {
         "Effectuer un achat",
         "Rendre des bouteilles consignées",
         "Afficher le panier",
         "$ Payer",
         "  Annuler",
```

```
        NULL,
    };

    char greeting[200];
    if (basket->owner != NULL)
    {
        sprintf(greeting, "Bonjour %s %s, veuillez faire un choix", basket->owner-
>firstname, basket->owner->lastname);
    }
    else
    {
        sprintf(greeting, "Bonjour veuillez faire un choix");
    }

    do
    {
        switch (user_select(user, greeting, choices))
        {
        case 0:
            log_info("Vous avez choisi d'effectuer un achat");
            cashier_scan_items(basket, stocks);
            break;

        case 1:
            log_info("Vous avez choisi de rendre des bouteilles consignées");
            cashier_return_consigned_bottles(basket, stocks);
            break;

        case 2:
        {
            model_view(user, "Panier", basket_model_create(), basket);
            break;
        }
        case 3:
        {
            if (basket->items->count == 0)
            {
                if (user_yes_no("Votre panier est vide, voulez-vous quitter ?", NO) ==
YES)
                {
                    break;
                }
            }
            else
            {

                if (basket->owner)
                {
                    if (user_yes_no("Voulez-vous payer avec vos points fidelitée?", YES))
                    {
                        basket->pay_with_point = true;
                    }
                }

                basket_pay(user, basket, stdout);

                terminal_read_key();
            }

            return;
        }
        default:
            return;
        };
    } while (true);
}
::::::::::::::
./source/model/lexer.c
::::::::::::::
```

```c
#include <stdbool.h>
#include <string.h>

#include "utils/string.h"
#include "utils/logger.h"
#include "utils/assert.h"
#include "model/lexer.h"

static const char *type_name[] = {
    "INVALID",
    "BEGIN",
    "END",
    "KEY",
    "VALUE",
    "EOF",
};

const char *token_type_string(Token *tok)
{
    return type_name[tok->type];
}

const char *token_type_string_type(TokenType type)
{
    return type_name[type];
}

int lexer_peek_char(Lexer *lex)
{
    int c;

    c = fgetc(lex->source);
    ungetc(c, lex->source);

    return c;
}

int lexer_next_char(Lexer *lex)
{
    int c = fgetc(lex->source);

    if (c == '\n')
    {
        lex->ln++;
        lex->col = 0;
    }
    else
    {
        lex->col++;
    }

    return c;
}

void lexer_eat_white_space(Lexer *lex)
{
    int c = lexer_peek_char(lex);

    while (is_white_space(c))
    {
        lexer_next_char(lex);
        c = lexer_peek_char(lex);
    }
}

bool lexer_read_string(Lexer *lex, Token *tok)
{
    int c = lexer_peek_char(lex);
```

```
    if (c != '"')
    {
        return false;
    }

    bool escaped = true;

    while ((c != '"' || escaped) && c != EOF)
    {
        if (c == '\\' && !escaped)
        {
            escaped = true;
        }
        else
        {
            strnapd(tok->literal, lexer_next_char(lex), VARIANT_SERIALIZED_SIZE);
            escaped = false;
        }

        c = lexer_peek_char(lex);
    }

    strnapd(tok->literal, lexer_next_char(lex), VARIANT_SERIALIZED_SIZE);
    tok->type = TOKEN_VALUE;

    return true;
}

bool lexer_read_numeric(Lexer *lex, Token *tok)
{
    int c = lexer_peek_char(lex);

    if (!is_numeric(c))
    {
        return false;
    }

    while (is_numeric(c))
    {
        strnapd(tok->literal, lexer_next_char(lex), VARIANT_SERIALIZED_SIZE);
        c = lexer_peek_char(lex);
    }

    tok->type = TOKEN_VALUE;

    return true;
}

bool lexer_read_keyword_or_key(Lexer *lex, Token *tok)
{
    int c = lexer_peek_char(lex);

    if (!is_letter(c))
    {
        return false;
    }

    while (is_letter(c))
    {
        strnapd(tok->literal, lexer_next_char(lex), VARIANT_SERIALIZED_SIZE);
        c = lexer_peek_char(lex);
    }

    if (strcmp(tok->literal, "BEGIN") == 0)
    {
        tok->type = TOKEN_BEGIN;
    }
    else if (strcmp(tok->literal, "END") == 0)
    {
```

```
        tok->type = TOKEN_END;
    }
    else
    {
        tok->type = TOKEN_KEY;
    }

    return true;
}

bool lexer_read_eof(Lexer *lex, Token *tok)
{
    int c = lexer_peek_char(lex);

    if (c == EOF)
    {
        tok->type = TOKEN_EOF;

        return true;
    }

    return false;
}

Token lexer_next_token(Lexer *lex)
{
    lexer_eat_white_space(lex);

    Token tok = (Token){
        lex->ln,
        lex->col,

        TOKEN_INVALID,
        "",
    };

    if (!(lexer_read_string(lex, &tok) ||
          lexer_read_numeric(lex, &tok) ||
          lexer_read_keyword_or_key(lex, &tok) ||
          lexer_read_eof(lex, &tok)))
    {
        log_error("Lexer: ln%d, col%d: Unexpected codepoint %d '%c'", lex->ln, lex->col,
lexer_peek_char(lex), lexer_peek_char(lex));
        lexer_next_char(lex);
    }

    return tok;
}::::::::::::::
./source/model/model.c
::::::::::::::
#include <string.h>
#include <stdbool.h>

#include "utils/logger.h"
#include "utils/assert.h"
#include "model/lexer.h"
#include "model/model.h"

int model_get_column(Model model, const char *name)
{
    for (int i = 0; i < model.column_count(); i++)
    {
        if (strcmp(name, model.column_name(i, ROLE_DATA)) == 0)
        {
            return i;
        }
    }

    return -1;
```

```
}

typedef enum
{
    MODEL_LOAD_BEGIN,
    MODEL_LOAD_KEY,
    MODEL_LOAD_VALUE,
} ModelLoadState;

bool model_load_expect(Token *tok, TokenType expected)
{
    if (tok->type != expected)
    {
        log_error("Parser: ln%d, col%d: Expected token %s got token %s '%s'",
                  tok->ln,
                  tok->col,
                  token_type_string_type(expected),
                  token_type_string(tok),
                  tok->literal);

        return false;
    }

    return true;
}

void model_load(Model model, void *data, FILE *source)
{
    ModelLoadState state = MODEL_LOAD_BEGIN;
    int row = -1;
    int column = -1;

    Lexer lex = {0};
    lex.ln = 1;
    lex.source = source;

    Token tok = lexer_next_token(&lex);
    do
    {
        if (state == MODEL_LOAD_BEGIN)
        {
            if (model_load_expect(&tok, TOKEN_BEGIN))
            {
                row = model.row_create(data);
                state = MODEL_LOAD_KEY;
            }
        }
        else if (state == MODEL_LOAD_KEY)
        {
            if (tok.type == TOKEN_KEY)
            {
                column = model_get_column(model, tok.literal);

                if (column == -1)
                {
                    log_error("Loader: ln%d, col%d: Le modele ne contient pas la colonne
%s!", tok.ln, tok.col, tok.literal);
                }

                state = MODEL_LOAD_VALUE;
            }
            else if (tok.type == TOKEN_END)
            {
                state = MODEL_LOAD_BEGIN;
            }
            else
            {
                model_load_expect(&tok, TOKEN_KEY);
            }
```

```
        }
        else if (state == MODEL_LOAD_VALUE)
        {
            if (model_load_expect(&tok, TOKEN_VALUE))
            {
                Variant value = variant_deserialize(tok.literal);

                if (value.type == model.column_type(column, ROLE_DATA))
                {
                    model.set_data(data, row, column, value, ROLE_DATA);
                }
                else
                {
                    log_error("Loader: ln%d, col%d: Le type de la colonne(%s) dans ne
corespond pas avec le type colonne du model! (%d!=%d)",
                              tok.ln,
                              tok.col,
                              model.column_name(column, ROLE_DATA),
                              value.type,
                              model.column_type(column, ROLE_DATA));
                }

                state = MODEL_LOAD_KEY;
            }
        }
        else
        {
            ASSERT_NOT_REACHED();
        }

        tok = lexer_next_token(&lex);
    } while (tok.type != TOKEN_EOF);
}

void model_save(Model model, void *data, FILE *destination)
{
    for (int row = 0; row < model.row_count(data); row++)
    {
        fprintf(destination, "BEGIN\n");

        for (int column = 0; column < model.column_count(); column++)
        {
            char serialied_value[VARIANT_SERIALIZED_SIZE];

            variant_serialize(model.get_data(data, row, column, ROLE_DATA),
serialied_value);

            fprintf(destination, "%s %s\n", model.column_name(column, ROLE_DATA),
serialied_value);
        }

        fprintf(destination, "END\n\n");
    }
}

Variant model_get_data_with_access(Model model, void *data, int row, int column, User
*user, ModelRole role)
{
    if (user->access <= model.read_access(data, row, column, user))
    {
        return model.get_data(data, row, column, role);
    }
    else
    {
        return vstring("####");
    }
}

void model_set_data_with_access(Model model, void *data, int row, int column, Variant
```

```
value, User *user)
{
    if (user->access <= model.read_access(data, row, column, user))
    {
        return model.set_data(data, row, column, value, ROLE_DATA);
    }
}
::::::::::::::
./source/model/action.c
::::::::::::::
#include <ctype.h>
#include <stdlib.h>
#include <string.h>

#include "model/view.h"
#include "utils/assert.h"
#include "utils/math.h"
#include "utils/terminal.h"

void quit_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)model, (void)data, (void)row;

    state->exited = true;
}

void help_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    surface_clear(surface, DEFAULT_STYLE);

    model_view_title(NULL, surface, "Rubrique d'aide");

    for (int i = 0; model.get_actions()[i].key_codepoint != 0; i++)
    {
        ModelAction action = model.get_actions()[i];
        char buffer[256];

        snprintf(buffer, 256, "• [%c] %s - %s", action.key_codepoint, action.name,
action.desciption);
        surface_text(surface, buffer, 2, i, surface_width(surface), DEFAULT_STYLE);
    }

    surface_pop_clip(surface);

    surface_render(surface);

    terminal_read_key();
}

void scroll_up_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
```

```
    int row)
{
    (void)user, (void)surface, (void)model, (void)data, (void)row;

    state->slected--;
}

void scroll_down_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model, void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    state->slected++;
}

void page_up_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    state->slected -= 10;
}

void page_down_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    state->slected += 10;
}

void home_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    state->slected = 0;
}

void end_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    state->slected = model.row_count(data);
```

```
}

void edit_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)surface, (void)state, (void)model, (void)data, (void)row;

    bool exited = false;
    int selected = 0;

    bool editing = false;
    char edited_value[VARIANT_STRING_SIZE] = {0};
    int edited_offset = 0;

    do
    {
        surface_clear(surface, DEFAULT_STYLE);
        model_view_title(NULL, surface, "Editer le modèle");
        model_view_status_bar(surface, state, model, data);

        for (int i = 0; i < model.column_count(); i++)
        {

            if (editing)
            {
                if (i == selected)
                {
                    char buffer[256];
                    snprintf(buffer, 256, "%16s: %s▂", model.column_name(i,
ROLE_DISPLAY), edited_value);
                    surface_text(surface, buffer, 0, i, surface_width(surface),
DEFAULT_STYLE);
                }
                else
                {
                    char buffer[256];
                    snprintf(buffer, 256, "%16s: %s", model.column_name(i, ROLE_DISPLAY),
model_get_data_with_access(model, data, row, i, user, ROLE_DISPLAY).as_string);
                    surface_text(surface, buffer, 0, i, surface_width(surface),
DISABLED_DEFAULT_STYLE);
                }
            }
            else
            {
                char buffer[256];
                snprintf(buffer, 256, "%16s: %-16s", model.column_name(i, ROLE_DISPLAY),
model_get_data_with_access(model, data, row, i, user, ROLE_DISPLAY).as_string);

                if (user->access <= model.write_access(data, row, i, user))
                {
                    if (i == selected)
                    {
                        surface_text(surface, buffer, 0, i, surface_width(surface),
INVERTED_STYLE);
                    }
                    else
                    {
                        surface_text(surface, buffer, 0, i, surface_width(surface),
DEFAULT_STYLE);
                    }
                }
                else
                {
                    if (i == selected)
```

```
                    {
                        surface_text(surface, buffer, 0, i, surface_width(surface),
 DISABLED_INVERTED_STYLE);
                    }
                    else
                    {
                        surface_text(surface, buffer, 0, i, surface_width(surface),
 DISABLED_DEFAULT_STYLE);
                    }
                }
            }
        }

        surface_pop_clip(surface);
        surface_pop_clip(surface);
        surface_render(surface);
        surface_update(surface);

        int key = terminal_read_key();

        if (editing)
        {
            if (key == '\n')
            {
                editing = false;
                if (model.column_type(selected, ROLE_EDITOR) == VARIANT_INT)
                {
                    model.set_data(data, row, selected, vint(atoi(edited_value)),
 ROLE_EDITOR);
                }
                else if (model.column_type(selected, ROLE_EDITOR) == VARIANT_FLOAT)
                {
                    float value = 0;
                    sscanf(edited_value, "%f", &value);
                    model.set_data(data, row, selected, vfloat(value), ROLE_EDITOR);
                }
                else
                {
                    model.set_data(data, row, selected, vstring(edited_value),
 ROLE_EDITOR);
                }
            }
            else if (key == 127)
            {
                if (edited_offset > 0)
                {
                    edited_offset--;
                    edited_value[edited_offset] = '\0';
                }
            }
            else if (iscntrl(key))
            {
                // do nothing with it
            }
            else if (edited_offset < VARIANT_STRING_SIZE - 1)
            {
                switch (model.column_type(selected, ROLE_EDITOR))
                {
                case VARIANT_INT:
                    if (key >= '0' && key <= '9')
                    {
                        edited_value[edited_offset] = key;
                        edited_value[edited_offset + 1] = '\0';

                        edited_offset++;
                    }
                    break;

                case VARIANT_FLOAT:
```

```
                    if ((key >= '0' && key <= '9') || key == '.')
                    {
                        edited_value[edited_offset] = key;
                        edited_value[edited_offset + 1] = '\0';

                        edited_offset++;
                    }
                    break;

                case VARIANT_STRING:
                    edited_value[edited_offset] = key;
                    edited_value[edited_offset + 1] = '\0';

                    edited_offset++;
                    break;

                default:
                    ASSERT_NOT_REACHED();
                }
            }
        }
        else
        {
            if (key == 'j')
            {
                selected = min(selected + 1, model.column_count() - 1);
            }
            else if (key == 'k')
            {
                selected = max(selected - 1, 0);
            }
            else if (key == 'e')
            {
                if (user->access <= model.write_access(data, row, selected, user))
                {
                    editing = true;
                    strcpy(edited_value, model.get_data(data, row, selected,
ROLE_EDITOR).as_string);
                    edited_offset = strlen(edited_value);
                }
            }
            else if (key == 'q')
            {
                exited = true;
            }
        }
    } while (!exited);

    state->sort_dirty = true;
}

void create_ModelActionCallback(
    User *user,
    Surface *surface,
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    int new_row = model.row_create(data);
    edit_ModelActionCallback(user, surface, state, model, data, new_row);
    state->sort_dirty = true;
}

void delete_ModelActionCallback(
    User *user,
    Surface *surface,
```

```
    ModelViewState *state,
    Model model,
    void *data,
    int row)
{
    (void)user, (void)surface, (void)state, (void)model, (void)data, (void)row;

    model.row_delete(data, state->sorted[state->slected]);
    state->sort_dirty = true;
}:::::::::::::::
./source/model/view.c
:::::::::::::::
#include "model/view.h"
#include "utils/math.h"
#include "utils/renderer.h"
#include "utils/terminal.h"

void model_view_title(User *user, Surface *surface, const char *title)
{
    if (user)
    {
        switch (user->access)
        {
        case ACCESS_ADMIN:
            surface_text(surface, title, 0, 1, surface_width(surface),
style_bold(style_centered(RED_STYLE)));
            surface_text(surface, "ADMIN", 2, 2, 16,
style_inverted(style_centered(RED_STYLE)));
            surface_plot_line(surface, u' ', 0, 3, surface_width(surface), 3, RED_STYLE);

            break;

        case ACCESS_MANAGER:
            surface_text(surface, title, 0, 1, surface_width(surface),
style_bold(style_centered(BLUE_STYLE)));
            surface_text(surface, "MANAGER", 2, 2, 16,
style_inverted(style_centered(BLUE_STYLE)));
            surface_plot_line(surface, u' ', 0, 3, surface_width(surface), 3,
BLUE_STYLE);

            break;

        case ACCESS_CASHIER:
            surface_text(surface, title, 0, 1, surface_width(surface),
style_bold(style_centered(WHITE_STYLE)));
            surface_text(surface, "CAISSIER", 2, 2, 16,
style_inverted(style_centered(WHITE_STYLE)));
            surface_plot_line(surface, u' ', 0, 3, surface_width(surface), 3,
WHITE_STYLE);

            break;

        default:
            break;
        }

        char buffer[128];
        snprintf(buffer, 128, "%s %s", user->lastname, user->firstname);
        surface_text(surface, buffer, 20, 2, 36, DEFAULT_STYLE);
    }
    else
    {
        surface_text(surface, title, 0, 1, surface_width(surface),
style_bold(style_centered(WHITE_STYLE)));
        surface_plot_line(surface, u' ', 0, 3, surface_width(surface), 3, WHITE_STYLE);
    }

    surface_push_clip(surface, (Region){
                                    0,
```

```
                                    4,
                                    surface_width(surface),
                                    surface_height(surface) - 4,
                                });
}

void model_view_scrollbar(Surface *surface, ModelViewState *state, Model model, void
*data)
{
    if (model.row_count(data) > surface_height(surface))
    {
        float viewport_height = surface_height(surface);
        float content_height = model.row_count(data);

        float viewable_ratio = viewport_height / content_height;
        float scroll_bar_area = viewport_height;
        float thump_pos = (state->scroll / (float)model.row_count(data)) *
viewport_height;
        float thumb_height = scroll_bar_area * viewable_ratio;

        for (int i = 0; i < surface_height(surface); i++)
        {
            surface_plot(surface, ' ', surface_width(surface) - 1, i, DEFAULT_STYLE);
        }

        for (int i = 0; i < thumb_height; i++)
        {
            surface_plot(surface, u'▐', surface_width(surface) - 1, thump_pos + i,
BLUE_STYLE);
        }
    }

    surface_push_clip(surface, (Region){0, 0, surface_width(surface) - 1,
surface_height(surface)});
}

void model_view_headerbar(Surface *surface, ModelViewState *state, Model model)
{
    int column_width = surface_width(surface) / model.column_count();

    for (int column = 0; column < model.column_count(); column++)
    {
        if (state->sortby == column)
        {
            surface_text(surface, model.column_name(column, ROLE_DISPLAY), column *
column_width, 0, column_width, style_centered(BOLD_STYLE));

            surface_plot(surface, state->sort_accending ? u'⌄' : u'^', column *
column_width + 1, 0, DEFAULT_STYLE);
        }
        else
        {
            surface_text(surface, model.column_name(column, ROLE_DISPLAY), column *
column_width, 0, column_width, style_centered(DEFAULT_STYLE));
        }
    }

    surface_plot_line(surface, u'-', 0, 1, surface_width(surface), 1, DEFAULT_STYLE);
    surface_push_clip(surface, (Region){0, 2, surface_width(surface),
surface_height(surface) - 2});
}

void model_view_status_bar(Surface *surface, ModelViewState *state, Model model, void
*data)
{
    char buffer[128];
    snprintf(buffer, 128, " %d éléments - ligne %d - [?] Appuyer sur 'h' pour afficher
l'aide", model.row_count(data), state->slected + 1);
    surface_text(surface, buffer, 0, surface_height(surface) - 1, surface_width(surface),
```

```
DEFAULT_STYLE);

    surface_push_clip(surface, (Region){0, 0, surface_width(surface),
surface_height(surface) - 1});
}

void model_view_update_scroll(Surface *surface, ModelViewState *state, Model model, void
*data)
{
    state->slected = min(model.row_count(data) - 1, max(0, state->slected));

    if (state->slected < state->scroll)
    {
        state->scroll = state->slected;
    }

    if (state->slected >= state->scroll + surface_height(surface))
    {
        state->scroll = state->slected - surface_height(surface) + 1;
    }

    state->scroll = max(0, min(state->scroll, model.row_count(data) - 1));
}

void model_view_list(User *user, Surface *surface, ModelViewState *state, Model model,
void *data)
{
    model_view_scrollbar(surface, state, model, data);

    model_view_headerbar(surface, state, model);

    model_view_update_scroll(surface, state, model, data);

    int column_width = surface_width(surface) / model.column_count();

    for (int row = state->scroll; row < min(state->scroll + surface_height(surface),
model.row_count(data)); row++)
    {
        for (int column = 0; column < model.column_count(); column++)
        {
            Variant value = model_get_data_with_access(model, data, state->sorted[row],
column, user, ROLE_DISPLAY);

            if (row == state->slected)
            {
                surface_text(surface, value.as_string, column * column_width, row -
state->scroll, column_width,
style_with_foreground(style_with_background(model.column_style(column), COLOR_WHITE),
COLOR_BLACK));
            }
            else
            {
                surface_text(surface, value.as_string, column * column_width, row -
state->scroll, column_width, (row % 2) ?
style_with_background(model.column_style(column), COLOR_BRIGHT_BLACK) :
model.column_style(column));
            }
        }
    }

    surface_pop_clip(surface);

    surface_pop_clip(surface);
}

static void reverse_array(int array[], int size)
{
    for (int i = 0; i < size / 2; i++)
    {
```

```
        int tmp = array[i];
        array[i] = array[size - i - 1];
        array[size - i - 1] = tmp;
    }
}

void model_view(User *user, const char *title, Model model, void *data)
{
    (void)title;
    ModelViewState state = {0};
    state.sort_dirty = true;

    terminal_enable_alternative_screen_buffer();

    Surface *surface = surface_create();

    do
    {
        surface_clear(surface, DEFAULT_STYLE);
        surface_update(surface);

        if (state.sort_dirty)
        {
            for (int i = 0; i < model.row_count(data); i++)
            {
                state.sorted[i] = i;
            }

            for (int i = 0; i < model.row_count(data) - 1; i++)
            {
                Variant idata = model.get_data(data, state.sorted[i], state.sortby,
ROLE_DATA);

                for (int j = i + 1; j < model.row_count(data); j++)
                {
                    Variant jdata = model.get_data(data, state.sorted[j], state.sortby,
ROLE_DATA);

                    int cmp = variant_cmp(idata, jdata);

                    if ((cmp > 0 && !state.sort_accending) || (cmp < 0 &&
state.sort_accending))
                    {
                        int tmp = state.sorted[i];
                        state.sorted[i] = state.sorted[j];
                        state.sorted[j] = tmp;

                        idata = jdata;
                    }
                }
            }

            state.sort_dirty = false;
        }

        terminal_enter_rawmode();

        model_view_title(user, surface, title);
        model_view_status_bar(surface, &state, model, data);

        model_view_list(user, surface, &state, model, data);

        surface_pop_clip(surface);
        surface_pop_clip(surface);

        surface_render(surface);

        terminal_exit_rawmode();
```

```
        int codepoint = terminal_read_key();

        for (int i = 0; model.get_actions()[i].key_codepoint != 0; i++)
        {
            ModelAction action = model.get_actions()[i];

            if (action.key_codepoint == codepoint)
            {
                action.callback(user, surface, &state, model, data,
 state.sorted[state.slected]);
            }
        }

        if (codepoint > '0' && codepoint <= '9')
        {
            int new_sort_by = codepoint - '0' - 1;

            if (new_sort_by < model.column_count())
            {
                if (state.sortby == new_sort_by)
                {
                    state.sort_accending = !state.sort_accending;
                    reverse_array(state.sorted, model.row_count(data));
                }
                else
                {
                    state.sortby = new_sort_by;
                    state.sort_accending = false;

                    state.sort_dirty = true;
                }
            }
        }

    } while (!state.exited);

    surface_destroy(surface);

    terminal_disable_alternative_screen_buffer();
}::::::::::::::
./source/main.c
::::::::::::::
#include "view/views.h"

void *load_data(Model model, void *data, const char *path)
{
    char path_with_sufix[256];
    snprintf(path_with_sufix, 256, "%s.saved", path);

    FILE *fdat = fopen(path_with_sufix, "r");
    if (!fdat)
    {
        fdat = fopen(path, "r");
    }

    model_load(model, data, fdat);
    fclose(fdat);

    return data;
}

void save_data(Model model, void *data, const char *path)
{
    char path_with_sufix[256];
    snprintf(path_with_sufix, 256, "%s.saved", path);

    FILE *stocks_save_file = fopen(path_with_sufix, "w");
    model_save(model, data, stocks_save_file);
    list_destroy(data);
```

```
    fclose(stocks_save_file);
}

int main(int argc, char const *argv[])
{
    (void)argc;
    (void)argv;

    printf("\n\e[91m");

    printf("\t   ___________ __    ____  __  _____  _________\n");
    printf("\t  / ____/ __ \\/ /   / __ \\/ / /\\ \\/ /_  __/\n");
    printf("\t / /   / / / / /   / /_/ / / /  \\  / / /   \n");
    printf("\t/ /___/ /_/ / /___/ _, _/ /_/ /   / / / /    \n");
    printf("\t\\____/\\____/_____/_/ |_|\\____/   /_/ /_/     \n");

    printf("\n\e[0;1m\t\tprix • qualité\n\n\e[0m");

    UsersList *users = load_data(users_model_create(), list_create(), "data/user.dat");
    ClientsList *clients = load_data(clients_model_create(), list_create(),
"data/client.dat");
    StockList *stocks = load_data(stocks_model_create(), list_create(),
"data/stock.dat");

    user_login(users, stocks, clients);

    save_data(clients_model_create(), clients, "data/client.dat");
    save_data(stocks_model_create(), stocks, "data/stock.dat");
    save_data(users_model_create(), users, "data/user.dat");

    return 0;
}
::::::::::::::
./tools/hash.c
::::::::::::::
#include <stdio.h>
#include <stdint.h>

uint32_t strhash(const unsigned char *str)
{
    unsigned long hash = 5381;
    int c;

    while ((c = *str++))
        hash = ((hash << 5) + hash) + c;

    return hash;
}

int main(int argc, char const *argv[])
{
    printf("%u\n", strhash((const uint8_t *)argv[1]));
    return 0;
}
```