Healthcare Management System

1.0.0

Generated by Doxygen 1.15.0

Chapter 1

Directory Hierarchy

1.1 Directories

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Directory Documentation

4.1 include Directory Reference

Files

- file admin.h

  Admin management functions for Healthcare Management System.
- file appointment.h

  Appointment management functions for Healthcare Management System.
- file auth.h

  Authentication functions for Healthcare Management System.
- file doctor.h

  Doctor management functions for Healthcare Management System.
- file doctor_portal.h

  Doctor portal functions for Healthcare Management System.
- file hospital.h

  Core data structures and definitions for Healthcare Management System.
- file patient.h

  Patient management functions for Healthcare Management System.
- file receptionist.h

  Receptionist portal and data management functions.
- file ui.h

  Header file for user interface functions.
- file utils.h

  Utility functions for Healthcare Management System.

4.2 src Directory Reference

Files

- file admin.c

  Admin management implementation for Healthcare Management System.
- file appointment.c

  Appointment management implementation for Healthcare Management System.

- file auth.c

  Authentication implementation for Healthcare Management System.
- file doctor.c

  Doctor management implementation for Healthcare Management System.
- file doctor_portal.c

  Doctor portal implementation for Healthcare Management System.
- file hospital.c

  Global data definitions for Healthcare Management System.
- file patient.c

  Patient management implementation for Healthcare Management System.
- file receptionist.c

  Receptionist portal implementation for Healthcare Management System.
- file ui.c

  User interface functions for Healthcare Management System.
- file utils.c

  Utility functions for Healthcare Management System.

4.3 tests Directory Reference

Files

- file patient_test.c
- file test.c
- file test_phone_validator.c
- file ui_test.c
- file utils_test.c

Chapter 5

Class Documentation

5.1 Appointment Struct Reference

```
#include <hospital.h>
```

Public Attributes

- int id
- int patient_id
- int doctor_id
- char date [15]
- char time_slot [10]
- char reason [100]
- AppointmentStatus status

5.1.1 Detailed Description

Definition at line 138 of file hospital.h.

5.1.2 Member Data Documentation

5.1.2.1 date

```
char Appointment::date[15]
```

Definition at line 142 of file hospital.h.

5.1.2.2 doctor_id

```
int Appointment::doctor_id
```

Definition at line 141 of file hospital.h.

5.1.2.3 id

```
int Appointment::id
```

Definition at line 139 of file hospital.h.

5.1.2.4 patient_id

```
int Appointment::patient_id
```

Definition at line 140 of file hospital.h.

5.1.2.5 reason

```
char Appointment::reason[100]
```

Definition at line 144 of file hospital.h.

5.1.2.6 status

```
AppointmentStatus Appointment::status
```

Definition at line 145 of file hospital.h.

5.1.2.7 time_slot

```
char Appointment::time_slot[10]
```

Definition at line 143 of file hospital.h.

The documentation for this struct was generated from the following file:

- include/hospital.h

5.2 Doctor Struct Reference

```
#include <hospital.h>
```

Public Attributes

- int id
- char name [50]
- char phone [15]
- char email [50]
- char specialization [30]
- int room_number
- bool is_available
- bool is_active

5.2.1 Detailed Description

Definition at line 119 of file hospital.h.

5.2.2 Member Data Documentation

5.2.2.1 email

```
char Doctor::email[50]
```

Definition at line 123 of file hospital.h.

5.2.2.2 id

```
int Doctor::id
```

Definition at line 120 of file hospital.h.

5.2.2.3 is_active

```
bool Doctor::is_active
```

Definition at line 127 of file hospital.h.

5.2.2.4 is_available

```
bool Doctor::is_available
```

Definition at line 126 of file hospital.h.

5.2.2.5 name

```
char Doctor::name[50]
```

Definition at line 121 of file hospital.h.

5.2.2.6 phone

```
char Doctor::phone[15]
```

Definition at line 122 of file hospital.h.

5.2.2.7 room_number

```
int Doctor::room_number
```

Definition at line 125 of file hospital.h.

5.2.2.8 specialization

```
char Doctor::specialization[30]
```

Definition at line 124 of file hospital.h.

The documentation for this struct was generated from the following file:

- include/hospital.h

5.3 Patient Struct Reference

```
#include <hospital.h>
```

Public Attributes

- int id
- char name [50]
- int age
- Gender gender
- char phone [15]
- char address [100]
- char blood_group [5]
- bool is_active

5.3.1 Detailed Description

Definition at line 108 of file hospital.h.

5.3.2 Member Data Documentation

5.3.2.1 address

```
char Patient::address[100]
```

Definition at line 114 of file hospital.h.

5.3.2.2 age

```
int Patient::age
```

Definition at line 111 of file hospital.h.

5.3.2.3 blood_group

```
char Patient::blood_group[5]
```

Definition at line 115 of file hospital.h.

5.3.2.4 gender

```
Gender Patient::gender
```

Definition at line 112 of file hospital.h.

5.3.2.5 id

```
int Patient::id
```

Definition at line 109 of file hospital.h.

5.3.2.6 is_active

```
bool Patient::is_active
```

Definition at line 116 of file hospital.h.

5.3.2.7 name

```
char Patient::name[50]
```

Definition at line 110 of file hospital.h.

5.3.2.8 phone

```
char Patient::phone[15]
```

Definition at line 113 of file hospital.h.

The documentation for this struct was generated from the following file:

- include/hospital.h

5.4 Receptionist Struct Reference

```
#include <hospital.h>
```

Public Attributes

- int id
- char name [50]
- char phone [15]
- char email [50]
- bool is_available
- bool is_active

5.4.1 Detailed Description

Definition at line 148 of file hospital.h.

5.4.2 Member Data Documentation

5.4.2.1 email

```
char Receptionist::email[50]
```

Definition at line 152 of file hospital.h.

5.4.2.2 id

```
int Receptionist::id
```

Definition at line 149 of file hospital.h.

5.4.2.3 is_active

```
bool Receptionist::is_active
```

Definition at line 154 of file hospital.h.

5.4.2.4 is_available

```
bool Receptionist::is_available
```

Definition at line 153 of file hospital.h.

5.4.2.5 name

```
char Receptionist::name[50]
```

Definition at line 150 of file hospital.h.

5.4.2.6 phone

```
char Receptionist::phone[15]
```

Definition at line 151 of file hospital.h.

The documentation for this struct was generated from the following file:

- include/hospital.h

5.5 User Struct Reference

```
#include <hospital.h>
```

Public Attributes

- int id
- char username [30]
- char password [50]
- UserRole role
- bool is_active

5.5.1 Detailed Description

Definition at line 130 of file hospital.h.

5.5.2 Member Data Documentation

5.5.2.1 id

```
int User::id
```

Definition at line 131 of file hospital.h.

5.5.2.2 is_active

```
bool User::is_active
```

Definition at line 135 of file hospital.h.

5.5.2.3 password

```
char User::password[50]
```

Definition at line 133 of file hospital.h.

5.5.2.4 role

```
UserRole User::role
```

Definition at line 134 of file hospital.h.

5.5.2.5 username

```
char User::username[30]
```

Definition at line 132 of file hospital.h.

The documentation for this struct was generated from the following file:

- include/hospital.h

Chapter 6

File Documentation

6.1 include/admin.h File Reference

Admin management functions for Healthcare Management System.

```
#include "hospital.h"
```

Functions

- void admin_view_discharged_patients (void)
- void admin_search_discharged_by_id (void)
- void admin_search_discharged_by_name (void)
- void admin_search_discharged (void)
- void admin_delete_patient (void)
- void admin_view_discharged_doctors (void)
- void admin_delete_doctor (void)
- void admin_patient_menu (void)
- void admin_doctor_menu (void)
- void admin_main_menu (void)
- void admin_receptionist_menu (void)

6.1.1 Detailed Description

Admin management functions for Healthcare Management System.

This header declares admin-specific operations including discharged patient management that receptionists cannot access.

Definition in file admin.h.

6.1.2 Function Documentation

6.1.2.1 admin_delete_doctor()

```
void admin_delete_doctor (
             void )
```

Permanently deletes an inactive doctor from the system.

Definition at line 312 of file admin.c.

6.1.2.2 admin_delete_patient()

```
void admin_delete_patient (
             void )
```

Permanently deletes a discharged patient from the system.

Definition at line 154 of file admin.c.

6.1.2.3 admin_doctor_menu()

```
void admin_doctor_menu (
             void )
```

Admin doctor management menu.

Definition at line 382 of file admin.c.

6.1.2.4 admin_main_menu()

```
void admin_main_menu (
             void )
```

Main admin menu.

Definition at line 485 of file admin.c.

6.1.2.5 admin_patient_menu()

```
void admin_patient_menu (
             void )
```

Admin patient management menu.

Definition at line 224 of file admin.c.

6.1.2.6 admin_receptionist_menu()

```
void admin_receptionist_menu (
             void )
```

Admin receptionist management menu.

Definition at line 435 of file admin.c.

6.1.2.7 admin_search_discharged()

```
void admin_search_discharged (
             void )
```

Handles search choice for discharged patients.

Definition at line 121 of file admin.c.

6.1.2.8 admin_search_discharged_by_id()

```
void admin_search_discharged_by_id (
             void )
```

Searches discharged patients by ID.

Definition at line 46 of file admin.c.

6.1.2.9 admin_search_discharged_by_name()

```
void admin_search_discharged_by_name (
             void )
```

Searches discharged patients by name.

Definition at line 84 of file admin.c.

6.1.2.10 admin_view_discharged_doctors()

```
void admin_view_discharged_doctors (
             void )
```

Views all discharged (inactive) doctors.

Definition at line 292 of file admin.c.

6.1.2.11 admin_view_discharged_patients()

```
void admin_view_discharged_patients (
             void )
```

Views all discharged (inactive) patients.

Definition at line 26 of file admin.c.

6.2 admin.h

Go to the documentation of this file.

```


#ifndef ADMIN_H
#define ADMIN_H

#include "hospital.h"

 void admin_view_discharged_patients(void);

 void admin_search_discharged_by_id(void);

 void admin_search_discharged_by_name(void);

 void admin_search_discharged(void);

 void admin_delete_patient(void);

 void admin_view_discharged_doctors(void);

 void admin_delete_doctor(void);

  void admin_patient_menu(void);

 void admin_doctor_menu(void);

 void admin_main_menu(void);

 void admin_receptionist_menu(void);

#endif
```

6.3 include/appointment.h File Reference

Appointment management functions for Healthcare Management System.

```
#include "hospital.h"
```

Functions

- int appointment_save_to_file (void)
- int appointment_load_from_file (void)
- int appointment_generate_id (void)
- void appointment_create (void)
- void appointment_view_by_doctor (int doctor_id)
- void appointment_update_status (int appt_id, AppointmentStatus status)
- void appointment_cancel (int appt_id)
- int appointment_search_id (int id)
- const char * appointment_status_str (AppointmentStatus status)
- void ui_print_appointment (Appointment appt, int index)

6.3.1 Detailed Description

Appointment management functions for Healthcare Management System.

This header declares appointment-related CRUD operations.

Definition in file appointment.h.

6.3.2 Function Documentation

6.3.2.1 appointment_cancel()

```
void appointment_cancel (
            int appt_id)
```

Cancels an appointment.

Parameters

appt↩ _id	The appointment ID.

Definition at line 305 of file appointment.c.

6.3.2.2 appointment_create()

```
void appointment_create (
            void )
```

Creates a new appointment (called by receptionist).

Definition at line 77 of file appointment.c.

6.3.2.3 appointment_generate_id()

```
int appointment_generate_id (
            void )
```

Generates a unique appointment ID.

Returns

The generated appointment ID.

Definition at line 54 of file appointment.c.

6.3.2.4 appointment_load_from_file()

```
int appointment_load_from_file (
            void )
```

Loads all appointments from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 31 of file appointment.c.

6.3.2.5 appointment_save_to_file()

```
int appointment_save_to_file (
             void )
```

Saves all appointments to binary file.

Returns

0 on success, -1 on failure.

Definition at line 17 of file appointment.c.

6.3.2.6 appointment_search_id()

```
int appointment_search_id (
             int id)
```

Finds appointment by ID.

Parameters

id	The appointment ID.

Returns

Index of appointment, or -1 if not found.

Definition at line 68 of file appointment.c.

6.3.2.7 appointment_status_str()

```
const char * appointment_status_str (
             AppointmentStatus status)
```

Gets status string from enum.

Parameters

status	The appointment status.

Returns

String representation.

Definition at line 58 of file appointment.c.

6.3.2.8 appointment_update_status()

```
void appointment_update_status (
             int appt_id,
             AppointmentStatus status)
```

Updates appointment status.

Parameters

appt↩_id	The appointment ID.
status	The new status.

Definition at line 291 of file appointment.c.

6.3.2.9 appointment_view_by_doctor()

```
void appointment_view_by_doctor (
             int doctor_id)
```

Views all appointments for a specific doctor.

Parameters

doctor↩_id	The doctor's ID.

Definition at line 272 of file appointment.c.

6.3.2.10 ui_print_appointment()

```
void ui_print_appointment (
             Appointment appt,
             int index)
```

Prints an appointment in a formatted box.

Parameters

appt	The appointment to print.
index	The display index.

Definition at line 229 of file appointment.c.

6.4 appointment.h

Go to the documentation of this file.

```


#ifndef APPOINTMENT_H
#define APPOINTMENT_H

#include "hospital.h"

 int appointment_save_to_file(void);

 int appointment_load_from_file(void);

 int appointment_generate_id(void);

 void appointment_create(void);

 void appointment_view_by_doctor(int doctor_id);

 void appointment_update_status(int appt_id, AppointmentStatus status);

 void appointment_cancel(int appt_id);

 int appointment_search_id(int id);

 const char* appointment_status_str(AppointmentStatus status);

void ui_print_appointment(Appointment appt, int index);

#endif
```

6.5 include/auth.h File Reference

Authentication functions for Healthcare Management System.

```
#include "hospital.h"
```

Functions

- int auth_save_to_file (void)
- int auth_load_from_file (void)
- void auth_register_user (void)
- void auth_view_users (void)
- void auth_init_default_admin (void)
- void auth_user_menu (void)
- void auth_role_login (UserRole required_role)
- void login_menu (void)
- void encrypt (char ∗password)
- void decrypt (char ∗password)

6.5.1 Detailed Description

Authentication functions for Healthcare Management System.

Definition in file auth.h.

6.5.2 Function Documentation

6.5.2.1 auth_init_default_admin()

```
void auth_init_default_admin (
             void )
```

Creates default admin if no users exist.

Definition at line 57 of file auth.c.

6.5.2.2 auth_load_from_file()

```
int auth_load_from_file (
             void )
```

Loads all users from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 34 of file auth.c.

6.5.2.3 auth_register_user()

```
void auth_register_user (
             void )
```

Registers a new user (admin only).

Definition at line 70 of file auth.c.

6.5.2.4 auth_role_login()

```
void auth_role_login (
             UserRole required_role)
```

Login with role-based authentication.

Parameters

required_role	The role to authenticate as.

Definition at line 402 of file auth.c.

6.5.2.5 auth_save_to_file()

```
int auth_save_to_file (
             void )
```

Saves all users to binary file.

Returns

0 on success, -1 on failure.

Definition at line 17 of file auth.c.

6.5.2.6 auth_user_menu()

```
void auth_user_menu (
             void )
```

User management menu (admin only).

Definition at line 367 of file auth.c.

6.5.2.7 auth_view_users()

```
void auth_view_users (
             void )
```

Views all users (admin only).

Definition at line 330 of file auth.c.

6.5.2.8 decrypt()

```
void decrypt (
             char * password)
```

Decrypts a password using XOR cipher.

Parameters

password	The password to decrypt.

Definition at line 539 of file auth.c.

6.5.2.9 encrypt()

```
void encrypt (
             char * password)
```

Encrypts a password using XOR cipher.

Parameters

password	The password to encrypt.

Definition at line 533 of file auth.c.

6.5.2.10 login_menu()

```
void login_menu (
              void )
```

Login menu with role selection.

Definition at line 496 of file auth.c.

6.6 auth.h

Go to the documentation of this file.

```

#ifndef AUTH_H
#define AUTH_H

#include "hospital.h"

int auth_save_to_file(void);

int auth_load_from_file(void);

void auth_register_user(void);

void auth_view_users(void);

void auth_init_default_admin(void);

void auth_user_menu(void);

void auth_role_login(UserRole required_role);

void login_menu(void);

void encrypt(char* password);

void decrypt(char* password);

#endif
```

6.7 include/doctor.h File Reference

Doctor management functions for Healthcare Management System.

`#include "hospital.h"`

Functions

- int doctor_save_to_file (void)
- int doctor_load_from_file (void)
- int doctor_generate_id (void)
- void doctor_view_all (void)
- void doctor_view_one (void)
- void doctor_view (void)
- void doctor_search_by_id (void)
- void doctor_search_by_name (void)
- void doctor_search_by_phone (void)
- void doctor_search_by (void)
- int doctor_search_id (int id)
- void doctor_update_using_id (void)
- void doctor_deactivate_account (void)
- void doctor_view_discharged (void)

6.7.1 Detailed Description

Doctor management functions for Healthcare Management System.

This header declares all doctor-related CRUD operations.

Definition in file doctor.h.

6.7.2 Function Documentation

6.7.2.1 doctor_deactivate_account()

```
void doctor_deactivate_account (
             void )
```

Deactivates a doctor by ID (sets is_active to false).

Definition at line 563 of file doctor.c.

6.7.2.2 doctor_generate_id()

```
int doctor_generate_id (
             void )
```

Generates a unique doctor ID.

Returns

The generated doctor ID.

Definition at line 54 of file doctor.c.

6.7.2.3 doctor_load_from_file()

```
int doctor_load_from_file (
             void )
```

Loads all doctors from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 30 of file doctor.c.

6.7.2.4 doctor_save_to_file()

```
int doctor_save_to_file (
             void )
```

Saves all doctors to binary file.

Returns

0 on success, -1 on failure.

Definition at line 15 of file doctor.c.

6.7.2.5 doctor_search_by()

```
void doctor_search_by (
             void )
```

Handles the search choice for doctor.

Definition at line 249 of file doctor.c.

6.7.2.6 doctor_search_by_id()

```
void doctor_search_by_id (
             void )
```

Searches for a doctor by ID.

Definition at line 137 of file doctor.c.

6.7.2.7 doctor_search_by_name()

```
void doctor_search_by_name (
             void )
```

Searches for a doctor by name.

Definition at line 175 of file doctor.c.

6.7.2.8 doctor_search_by_phone()

```
void doctor_search_by_phone (
             void )
```

Searches for a doctor by phone number.

Definition at line 213 of file doctor.c.

6.7.2.9 doctor_search_id()

```
int doctor_search_id (
             int id)
```

Searches doctor by ID and returns index.

Parameters

id	The doctor ID to search for.

Returns

Index of doctor in array, or -1 if not found.

Definition at line 288 of file doctor.c.

6.7.2.10 doctor_update_using_id()

```
void doctor_update_using_id (
             void )
```

Updates a doctor information by ID.

Definition at line 459 of file doctor.c.

6.7.2.11 doctor_view()

```
void doctor_view (
             void )
```

Handles the view choice for doctor.

Definition at line 102 of file doctor.c.

6.7.2.12 doctor_view_all()

```
void doctor_view_all (
             void )
```

Displays all doctors in the system.

Definition at line 58 of file doctor.c.

6.7.2.13 doctor_view_discharged()

```
void doctor_view_discharged (
             void )
```

Displays all inactive doctors in the system.

Definition at line 623 of file doctor.c.

6.7.2.14 doctor_view_one()

```
void doctor_view_one (
             void )
```

Displays doctors one by one.

Definition at line 79 of file doctor.c.

6.8 doctor.h

Go to the documentation of this file.

```
#ifndef DOCTOR_H
#define DOCTOR_H

#include "hospital.h"

 int doctor_save_to_file(void);

 int doctor_load_from_file(void);

 int doctor_generate_id(void);

 void doctor_view_all(void);

 void doctor_view_one(void);

 void doctor_view(void);

 void doctor_search_by_id(void);

 void doctor_search_by_name(void);

 void doctor_search_by_phone(void);

 void doctor_search_by(void);

 int doctor_search_id(int id);

 void doctor_update_using_id(void);

 void doctor_deactivate_account(void);

 void doctor_view_discharged(void);

#endif
```

6.9 include/doctor_portal.h File Reference

Doctor portal functions for Healthcare Management System.

```
#include "hospital.h"
```

Functions

- void doctor_portal_view_appointments (int doctor_id)
- void doctor_portal_view_pending (int doctor_id)
- void doctor_portal_view_today (int doctor_id, const char *today_date)
- void doctor_portal_view_patient (void)
- void doctor_portal_complete_appointment (int doctor_id)
- void doctor_portal_cancel_appointment (int doctor_id)
- void doctor_portal_update_availability (int doctor_id)
- void doctor_portal_view_profile (int doctor_id)
- void doctor_portal_menu (int doctor_id, const char *doctor_name)

6.9.1 Detailed Description

Doctor portal functions for Healthcare Management System.

This header declares doctor-specific portal operations.

Definition in file doctor_portal.h.

6.9.2 Function Documentation

6.9.2.1 doctor_portal_cancel_appointment()

```
void doctor_portal_cancel_appointment (
             int doctor_id)
```

Cancels an appointment.

Parameters

doctor←↩ _id	The doctor's ID.

Definition at line 146 of file doctor_portal.c.

6.9.2.2 doctor_portal_complete_appointment()

```
void doctor_portal_complete_appointment (
             int doctor_id)
```

Marks an appointment as completed.

Parameters

doctor←↩ _id	The doctor's ID.

Definition at line 95 of file doctor_portal.c.

6.9.2.3 doctor_portal_menu()

```
void doctor_portal_menu (
             int doctor_id,
             const char * doctor_name)
```

Main doctor portal menu.

Parameters

doctor_id	The doctor's ID.
doctor_name	The doctor's name for display.

Definition at line 256 of file doctor_portal.c.

6.9.2.4 doctor_portal_update_availability()

```
void doctor_portal_update_availability (
             int doctor_id)
```

Updates doctor's availability status.

Parameters

doctor↵ _id	The doctor's ID.

Definition at line 196 of file doctor_portal.c.

6.9.2.5 doctor_portal_view_appointments()

```
void doctor_portal_view_appointments (
             int doctor_id)
```

Views all appointments for current doctor.

Parameters

doctor↵ _id	The doctor's ID.

Definition at line 19 of file doctor_portal.c.

6.9.2.6 doctor_portal_view_patient()

```
void doctor_portal_view_patient (
             void )
```

Views patient details for a given patient ID.

Definition at line 62 of file doctor_portal.c.

6.9.2.7 doctor_portal_view_pending()

```
void doctor_portal_view_pending (
             int doctor_id)
```

Views pending appointments for current doctor.

Parameters

doctor←↩ _id	The doctor's ID.

Definition at line 23 of file doctor_portal.c.

6.9.2.8 doctor_portal_view_profile()

```
void doctor_portal_view_profile (
             int doctor_id)
```

Views doctor's own profile.

Parameters

doctor←↩ _id	The doctor's ID.

Definition at line 242 of file doctor_portal.c.

6.9.2.9 doctor_portal_view_today()

```
void doctor_portal_view_today (
             int doctor_id,
             const char * today_date)
```

Views today's appointments for current doctor.

Parameters

doctor_id	The doctor's ID.
today_date	Today's date string.

Definition at line 42 of file doctor_portal.c.

6.10 doctor_portal.h

Go to the documentation of this file.

```

#ifndef DOCTOR_PORTAL_H
#define DOCTOR_PORTAL_H

#include "hospital.h"

 void doctor_portal_view_appointments(int doctor_id);

 void doctor_portal_view_pending(int doctor_id);

 void doctor_portal_view_today(int doctor_id, const char* today_date);

void doctor_portal_view_patient(void);

void doctor_portal_complete_appointment(int doctor_id);

void doctor_portal_cancel_appointment(int doctor_id);

void doctor_portal_update_availability(int doctor_id);

void doctor_portal_view_profile(int doctor_id);

void doctor_portal_menu(int doctor_id, const char* doctor_name);

#endif
```

6.11 include/hospital.h File Reference

Core data structures and definitions for Healthcare Management System.

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

Classes

- struct Patient
- struct Doctor
- struct User
- struct Appointment
- struct Receptionist

Macros

- #define UI_SIZE 72
- #define MAX_PATIENTS 100
- #define MAX_DOCTORS 20
- #define MAX_RECEPTIONISTS 20
- #define MAX_USERS 50
- #define MAX_APPOINTMENTS 200
- #define NAME_SIZE 50
- #define PHONE_SIZE 15
- #define EMAIL_SIZE 50
- #define ADDRESS_SIZE 100
- #define SPEC_SIZE 30 /∗ Specialization ∗/
- #define BLOOD_SIZE 5 /∗ Blood group ∗/
- #define AGE_SIZE 5 /∗ Age ∗/
- #define USERNAME_SIZE 30
- #define PASSWORD_SIZE 50
- #define GENDER_SIZE 10
- #define STATUS_SIZE 10
- #define ROOM_SIZE 10
- #define ID_LINE_SIZE 20
- #define NAME_LINE_SIZE NAME_SIZE + 20
- #define PHONE_LINE_SIZE PHONE_SIZE + 20
- #define ADDRESS_LINE_SIZE ADDRESS_SIZE + 20
- #define BLOOD_LINE_SIZE BLOOD_SIZE + 20
- #define GENDER_LINE_SIZE GENDER_SIZE + 20

- #define STATUS_LINE_SIZE STATUS_SIZE + 20
- #define SPEC_LINE_SIZE SPEC_SIZE + 20
- #define ROOM_LINE_SIZE ROOM_SIZE + 20
- #define AGE_LINE_SIZE AGE_SIZE + 20
- #define PHONE_LINE_SIZE PHONE_SIZE + 20
- #define EMAIL_LINE_SIZE EMAIL_SIZE + 20
- #define BLOOD_LINE_SIZE BLOOD_SIZE + 20
- #define GENDER_LINE_SIZE GENDER_SIZE + 20
- #define STATUS_LINE_SIZE STATUS_SIZE + 20
- #define DATA_DIR "data/"
- #define PATIENTS_FILE "data/patients.dat"
- #define DOCTORS_FILE "data/doctors.dat"
- #define RECEPTIONISTS_FILE "data/receptionists.dat"
- #define USERS_FILE "data/users.dat"
- #define APPOINTMENTS_FILE "data/appointments.dat"
- #define PATIENT_ID_START 1001
- #define DOCTOR_ID_START 2001
- #define ADMIN_ID_START 3001
- #define RECEPTIONIST_ID_START 4001
- #define APPOINTMENT_ID_START 5001
- #define REASON_SIZE 100
- #define TIME_SIZE 10
- #define DATE_SIZE 15
- #define VERSION "1.0.0"

Enumerations

- enum UserRole { ROLE_ADMIN , ROLE_DOCTOR , ROLE_RECEPTIONIST , ROLE_PATIENT }
- enum Gender { MALE , FEMALE }
- enum AppointmentStatus { APPT_PENDING , APPT_CONFIRMED , APPT_COMPLETED , APPT_CANCELLED }

Functions

- void hospital_init (void)
- void show_about (void)
- void print_help (const char ∗program_name)
- void print_version (void)
- void ensure_data_dir (void)

Variables

- Patient patients [100]
- Doctor doctors [20]
- Receptionist receptionists [20]
- User users [50]
- Appointment appointments [200]
- int patient_count
- int doctor_count
- int receptionist_count
- int user_count
- int patient_available

- int patient_unavailable
- int doctor_available
- int doctor_unavailable
- int receptionist_available
- int receptionist_unavailable
- int user_available
- int user_unavailable
- int appointment_count
- User ∗ current_user

6.11.1 Detailed Description

Core data structures and definitions for Healthcare Management System.

This header defines all shared structures, enumerations, and constants used throughout the HMS application.

Definition in file hospital.h.

6.11.2 Macro Definition Documentation

6.11.2.1 ADDRESS_LINE_SIZE

```
#define ADDRESS_LINE_SIZE ADDRESS_SIZE + 20
```

Definition at line 44 of file hospital.h.

6.11.2.2 ADDRESS_SIZE

```
#define ADDRESS_SIZE 100
```

Definition at line 31 of file hospital.h.

6.11.2.3 ADMIN_ID_START

```
#define ADMIN_ID_START 3001
```

Definition at line 66 of file hospital.h.

6.11.2.4 AGE_LINE_SIZE

```
#define AGE_LINE_SIZE AGE_SIZE + 20
```

Definition at line 50 of file hospital.h.

6.11.2.5 AGE_SIZE

```
#define AGE_SIZE 5 /* Age */
```

Definition at line 34 of file hospital.h.

6.11.2.6 APPOINTMENT_ID_START

```
#define APPOINTMENT_ID_START 5001
```

Definition at line 68 of file hospital.h.

6.11.2.7 APPOINTMENTS_FILE

```
#define APPOINTMENTS_FILE "data/appointments.dat"
```

Definition at line 62 of file hospital.h.

6.11.2.8 BLOOD_LINE_SIZE [1/2]

```
#define BLOOD_LINE_SIZE BLOOD_SIZE + 20
```

Definition at line 45 of file hospital.h.

6.11.2.9 BLOOD_LINE_SIZE [2/2]

```
#define BLOOD_LINE_SIZE BLOOD_SIZE + 20
```

Definition at line 45 of file hospital.h.

6.11.2.10 BLOOD_SIZE

```
#define BLOOD_SIZE 5 /* Blood group */
```

Definition at line 33 of file hospital.h.

6.11.2.11 DATA_DIR

```
#define DATA_DIR "data/"
```

Definition at line 57 of file hospital.h.

6.11.2.12 DATE_SIZE

```
#define DATE_SIZE 15
```

Definition at line 72 of file hospital.h.

6.11.2.13 DOCTOR_ID_START

```
#define DOCTOR_ID_START 2001
```

Definition at line 65 of file hospital.h.

6.11.2.14 DOCTORS_FILE

```
#define DOCTORS_FILE "data/doctors.dat"
```

Definition at line 59 of file hospital.h.

6.11.2.15 EMAIL_LINE_SIZE

```
#define EMAIL_LINE_SIZE EMAIL_SIZE + 20
```

Definition at line 52 of file hospital.h.

6.11.2.16 EMAIL_SIZE

```
#define EMAIL_SIZE 50
```

Definition at line 30 of file hospital.h.

6.11.2.17 GENDER_LINE_SIZE [1/2]

```
#define GENDER_LINE_SIZE GENDER_SIZE + 20
```

Definition at line 46 of file hospital.h.

6.11.2.18 GENDER_LINE_SIZE [2/2]

```
#define GENDER_LINE_SIZE GENDER_SIZE + 20
```

Definition at line 46 of file hospital.h.

6.11.2.19 GENDER_SIZE

```
#define GENDER_SIZE 10
```

Definition at line 37 of file hospital.h.

6.11.2.20 ID_LINE_SIZE

```
#define ID_LINE_SIZE 20
```

Definition at line 41 of file hospital.h.

6.11.2.21 MAX_APPOINTMENTS

```
#define MAX_APPOINTMENTS 200
```

Definition at line 26 of file hospital.h.

6.11.2.22 MAX_DOCTORS

```
#define MAX_DOCTORS 20
```

Definition at line 23 of file hospital.h.

6.11.2.23 MAX_PATIENTS

```
#define MAX_PATIENTS 100
```

Definition at line 22 of file hospital.h.

6.11.2.24 MAX_RECEPTIONISTS

```
#define MAX_RECEPTIONISTS 20
```

Definition at line 24 of file hospital.h.

6.11.2.25 MAX_USERS

```
#define MAX_USERS 50
```

Definition at line 25 of file hospital.h.

6.11.2.26 NAME_LINE_SIZE

```
#define NAME_LINE_SIZE NAME_SIZE + 20
```

Definition at line 42 of file hospital.h.

6.11.2.27 NAME_SIZE

```
#define NAME_SIZE 50
```

Definition at line 28 of file hospital.h.

6.11.2.28 PASSWORD_SIZE

```
#define PASSWORD_SIZE 50
```

Definition at line 36 of file hospital.h.

6.11.2.29 PATIENT_ID_START

```
#define PATIENT_ID_START 1001
```

Definition at line 64 of file hospital.h.

6.11.2.30 PATIENTS_FILE

```
#define PATIENTS_FILE "data/patients.dat"
```

Definition at line 58 of file hospital.h.

6.11.2.31 PHONE_LINE_SIZE [1/2]

```
#define PHONE_LINE_SIZE PHONE_SIZE + 20
```

Definition at line 43 of file hospital.h.

6.11.2.32 PHONE_LINE_SIZE [2/2]

```
#define PHONE_LINE_SIZE PHONE_SIZE + 20
```

Definition at line 43 of file hospital.h.

6.11.2.33 PHONE_SIZE

```
#define PHONE_SIZE 15
```

Definition at line 29 of file hospital.h.

6.11.2.34 REASON_SIZE

```
#define REASON_SIZE 100
```

Definition at line 70 of file hospital.h.

6.11.2.35 RECEPTIONIST_ID_START

```
#define RECEPTIONIST_ID_START 4001
```

Definition at line 67 of file hospital.h.

6.11.2.36 RECEPTIONISTS_FILE

```
#define RECEPTIONISTS_FILE "data/receptionists.dat"
```

Definition at line 60 of file hospital.h.

6.11.2.37 ROOM_LINE_SIZE

```
#define ROOM_LINE_SIZE ROOM_SIZE + 20
```

Definition at line 49 of file hospital.h.

6.11.2.38 ROOM_SIZE

```
#define ROOM_SIZE 10
```

Definition at line 39 of file hospital.h.

6.11.2.39 SPEC_LINE_SIZE

```
#define SPEC_LINE_SIZE SPEC_SIZE + 20
```

Definition at line 48 of file hospital.h.

6.11.2.40 SPEC_SIZE

```
#define SPEC_SIZE 30 /* Specialization */
```

Definition at line 32 of file hospital.h.

6.11.2.41 STATUS_LINE_SIZE [1/2]

```
#define STATUS_LINE_SIZE STATUS_SIZE + 20
```

Definition at line 47 of file hospital.h.

6.11.2.42 STATUS_LINE_SIZE [2/2]

```
#define STATUS_LINE_SIZE STATUS_SIZE + 20
```

Definition at line 47 of file hospital.h.

6.11.2.43 STATUS_SIZE

```
#define STATUS_SIZE 10
```

Definition at line 38 of file hospital.h.

6.11.2.44 TIME_SIZE

```
#define TIME_SIZE 10
```

Definition at line 71 of file hospital.h.

6.11.2.45 UI_SIZE

```
#define UI_SIZE 72
```

Definition at line 20 of file hospital.h.

6.11.2.46 USERNAME_SIZE

```
#define USERNAME_SIZE 30
```

Definition at line 35 of file hospital.h.

6.11.2.47 USERS_FILE

```
#define USERS_FILE "data/users.dat"
```

Definition at line 61 of file hospital.h.

6.11.2.48 VERSION

```
#define VERSION "1.0.0"
```

Definition at line 75 of file hospital.h.

6.11.3 Enumeration Type Documentation

6.11.3.1 AppointmentStatus

```
enum AppointmentStatus
```

Enumerator

APPT_PENDING	
APPT_CONFIRMED	
APPT_COMPLETED	
APPT_CANCELLED	

Definition at line 95 of file hospital.h.

6.11.3.2 Gender

```
enum Gender
```

Enumerator

MALE	
FEMALE	

Definition at line 90 of file hospital.h.

6.11.3.3 UserRole

```
enum UserRole
```

Enumerator

ROLE_ADMIN	
ROLE_DOCTOR	
ROLE_RECEPTIONIST	
ROLE_PATIENT	

Definition at line 83 of file hospital.h.

6.11.4 Function Documentation

6.11.4.1 ensure_data_dir()

```
void ensure_data_dir (
             void )
```

Ensure data directory exists.

Definition at line 92 of file hospital.c.

6.11.4.2 hospital_init()

```
void hospital_init (
             void )
```

Initialize the hospital system by loading all data.

Definition at line 47 of file hospital.c.

6.11.4.3 print_help()

```
void print_help (
             const char * program_name)
```

Display help information.

Definition at line 73 of file hospital.c.

6.11.4.4 print_version()

```
void print_version (
            void )
```

Display version information.

Definition at line 86 of file hospital.c.

6.11.4.5 show_about()

```
void show_about (
            void )
```

Display about information.

Definition at line 55 of file hospital.c.

6.11.5 Variable Documentation

6.11.5.1 appointment_count

```
int appointment_count  [extern]
```

Definition at line 43 of file hospital.c.

6.11.5.2 appointments

```
Appointment appointments[200]  [extern]
```

Definition at line 31 of file hospital.c.

6.11.5.3 current_user

```
User* current_user  [extern]
```

Definition at line 45 of file hospital.c.

6.11.5.4 doctor_available

```
int doctor_available  [extern]
```

Definition at line 37 of file hospital.c.

6.11.5.5 doctor_count

```
int doctor_count  [extern]
```

Definition at line 36 of file hospital.c.

6.11.5.6 doctor_unavailable

```
int doctor_unavailable  [extern]
```

Definition at line 38 of file hospital.c.

6.11.5.7 doctors

```
Doctor doctors[20]  [extern]
```

Definition at line 28 of file hospital.c.

6.11.5.8 patient_available

```
int patient_available  [extern]
```

Definition at line 34 of file hospital.c.

6.11.5.9 patient_count

```
int patient_count  [extern]
```

Definition at line 33 of file hospital.c.

6.11.5.10 patient_unavailable

```
int patient_unavailable  [extern]
```

Definition at line 35 of file hospital.c.

6.11.5.11 patients

```
Patient patients[100]  [extern]
```

Definition at line 27 of file hospital.c.

6.11.5.12 receptionist_available

```
int receptionist_available  [extern]
```

Definition at line 40 of file hospital.c.

6.11.5.13 receptionist_count

```
int receptionist_count  [extern]
```

Definition at line 39 of file hospital.c.

6.11.5.14 receptionist_unavailable

```
int receptionist_unavailable  [extern]
```

Definition at line 41 of file hospital.c.

6.11.5.15 receptionists

```
Receptionist receptionists[20]  [extern]
```

Definition at line 29 of file hospital.c.

6.11.5.16 user_available

```
int user_available  [extern]
```

6.11.5.17 user_count

```
int user_count  [extern]
```

Definition at line 42 of file hospital.c.

6.11.5.18 user_unavailable

```
int user_unavailable  [extern]
```

6.11.5.19 users

```
User users[50]  [extern]
```

Definition at line 30 of file hospital.c.

6.12 hospital.h

Go to the documentation of this file.

```

#ifndef HOSPITAL_H
#define HOSPITAL_H

#include <stdbool.h>

/*
 *==========================================================================
 *                              CONSTANTS
 *==========================================================================
 */

#define UI_SIZE         72

#define MAX_PATIENTS    100
#define MAX_DOCTORS     20
#define MAX_RECEPTIONISTS 20
#define MAX_USERS       50
#define MAX_APPOINTMENTS 200

#define NAME_SIZE       50
#define PHONE_SIZE      15
#define EMAIL_SIZE      50
#define ADDRESS_SIZE    100
#define SPEC_SIZE       30       /* Specialization */
#define BLOOD_SIZE      5        /* Blood group */
#define AGE_SIZE        5        /* Age */
#define USERNAME_SIZE   30
#define PASSWORD_SIZE   50
#define GENDER_SIZE     10
#define STATUS_SIZE     10
#define ROOM_SIZE       10

#define ID_LINE_SIZE        20
#define NAME_LINE_SIZE      NAME_SIZE + 20
#define PHONE_LINE_SIZE     PHONE_SIZE + 20
#define ADDRESS_LINE_SIZE   ADDRESS_SIZE + 20
#define BLOOD_LINE_SIZE     BLOOD_SIZE + 20
#define GENDER_LINE_SIZE    GENDER_SIZE + 20
#define STATUS_LINE_SIZE    STATUS_SIZE + 20
#define SPEC_LINE_SIZE      SPEC_SIZE + 20
#define ROOM_LINE_SIZE      ROOM_SIZE + 20
#define AGE_LINE_SIZE       AGE_SIZE + 20
#define PHONE_LINE_SIZE     PHONE_SIZE + 20
#define EMAIL_LINE_SIZE     EMAIL_SIZE + 20
#define BLOOD_LINE_SIZE     BLOOD_SIZE + 20
#define GENDER_LINE_SIZE    GENDER_SIZE + 20
#define STATUS_LINE_SIZE    STATUS_SIZE + 20

#define DATA_DIR            "data/"
#define PATIENTS_FILE       "data/patients.dat"
#define DOCTORS_FILE        "data/doctors.dat"
#define RECEPTIONISTS_FILE  "data/receptionists.dat"
#define USERS_FILE          "data/users.dat"
#define APPOINTMENTS_FILE   "data/appointments.dat"

#define PATIENT_ID_START      1001
#define DOCTOR_ID_START       2001
#define ADMIN_ID_START        3001
#define RECEPTIONIST_ID_START 4001
#define APPOINTMENT_ID_START  5001

#define REASON_SIZE     100
#define TIME_SIZE       10
#define DATE_SIZE       15


#define VERSION "1.0.0"

/*
 *==========================================================================
 *                              ENUMERATIONS
 *==========================================================================
 */

typedef enum {
    ROLE_ADMIN,
    ROLE_DOCTOR,
    ROLE_RECEPTIONIST,
    ROLE_PATIENT
} UserRole;
```

```

typedef enum {
    MALE,
    FEMALE
} Gender;

typedef enum {
    APPT_PENDING,
    APPT_CONFIRMED,
    APPT_COMPLETED,
    APPT_CANCELLED
} AppointmentStatus;

/*
 *==========================================================================
 *                              STRUCTURES
 *==========================================================================
 */

typedef struct {
    int id;
    char name[NAME_SIZE];
    int age;
    Gender gender;
    char phone[PHONE_SIZE];
    char address[ADDRESS_SIZE];
    char blood_group[BLOOD_SIZE];
    bool is_active;
} Patient;

typedef struct {
    int id;
    char name[NAME_SIZE];
    char phone[PHONE_SIZE];
    char email[EMAIL_SIZE];
    char specialization[SPEC_SIZE];
    int room_number;
    bool is_available;
    bool is_active;
} Doctor;

typedef struct {
    int id;
    char username[USERNAME_SIZE];
    char password[PASSWORD_SIZE];
    UserRole role;
    bool is_active;
} User;

typedef struct {
    int id;
    int patient_id;
    int doctor_id;
    char date[DATE_SIZE];           // "DD-MM-YYYY"
    char time_slot[TIME_SIZE];      // "10:00 AM"
    char reason[REASON_SIZE];
    AppointmentStatus status;
} Appointment;

typedef struct {
    int id;
    char name[NAME_SIZE];
    char phone[PHONE_SIZE];
    char email[EMAIL_SIZE];
    bool is_available;
    bool is_active;
} Receptionist;

/*
 *==========================================================================
 *                          GLOBAL DECLARATIONS
 *==========================================================================
 */

/* Global declarations for data arrays */
extern Patient patients[MAX_PATIENTS];
extern Doctor doctors[MAX_DOCTORS];
extern Receptionist receptionists[MAX_RECEPTIONISTS];
extern User users[MAX_USERS];
extern Appointment appointments[MAX_APPOINTMENTS];

/* Count variables */
extern int patient_count;
extern int doctor_count;
extern int receptionist_count;
extern int user_count;
extern int patient_available;
extern int patient_unavailable;
extern int doctor_available;
extern int doctor_unavailable;
extern int receptionist_available;
extern int receptionist_unavailable;
extern int user_available;
extern int user_unavailable;
extern int appointment_count;

/* Current user */
extern User* current_user;

void hospital_init(void);

void show_about(void);

void print_help(const char* program_name);

void print_version(void);

void ensure_data_dir(void);

#endif
```

6.13 include/patient.h File Reference

Patient management functions for Healthcare Management System.

`#include "hospital.h"`

Functions

- int patient_save_to_file (void)
- int patient_load_from_file (void)
- int patient_generate_id (void)
- void patient_add (void)
- void patient_search_by_id (void)
- void patient_search_by_name (void)
- void patient_search_by_phone (void)
- void patient_search_by (void)
- void patient_update_using_id (void)
- void patient_discharge (void)
- void patient_view_all (void)
- void patient_view_discharged (void)
- int patient_search_id (int id)

6.13.1 Detailed Description

Patient management functions for Healthcare Management System.

This header declares all patient-related CRUD operations.

Definition in file patient.h.

6.13.2 Function Documentation

6.13.2.1 patient_add()

```
void patient_add (
             void )
```

Adds a new patient to the system.

Definition at line 58 of file patient.c.

6.13.2.2 patient_discharge()

```
void patient_discharge (
             void )
```

Discharges a patient by ID (sets is_active to false).

Definition at line 693 of file patient.c.

6.13.2.3 patient_generate_id()

```
int patient_generate_id (
             void )
```

Generates a unique patient ID.

Returns

The generated patient ID.

Definition at line 54 of file patient.c.

6.13.2.4 patient_load_from_file()

```
int patient_load_from_file (
             void )
```

Loads all patients from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 30 of file patient.c.

6.13.2.5 patient_save_to_file()

```
int patient_save_to_file (
            void )
```

Saves all patients to binary file.

Returns

0 on success, -1 on failure.

Definition at line 15 of file patient.c.

6.13.2.6 patient_search_by()

```
void patient_search_by (
            void )
```

Handles the search choice for patient.

Definition at line 402 of file patient.c.

6.13.2.7 patient_search_by_id()

```
void patient_search_by_id (
            void )
```

Searches for a patient by ID.

Definition at line 290 of file patient.c.

6.13.2.8 patient_search_by_name()

```
void patient_search_by_name (
            void )
```

Searches for a patient by name.

Definition at line 328 of file patient.c.

6.13.2.9 patient_search_by_phone()

```
void patient_search_by_phone (
            void )
```

Searches for a patient by phone number.

Definition at line 366 of file patient.c.

6.13.2.10 patient_search_id()

```
int patient_search_id (
            int id)
```

Searches patient by ID and returns index.

Parameters

id	The patient ID to search for.

Returns

Index of patient in array, or -1 if not found.

Definition at line 441 of file patient.c.

6.13.2.11 patient_update_using_id()

```
void patient_update_using_id (
             void )
```

Updates a patient information by ID.

Definition at line 594 of file patient.c.

6.13.2.12 patient_view_all()

```
void patient_view_all (
             void )
```

Displays all patients in the system.

Definition at line 211 of file patient.c.

6.13.2.13 patient_view_discharged()

```
void patient_view_discharged (
             void )
```

Displays all discharged patients in the system.

Definition at line 753 of file patient.c.

6.14 patient.h

Go to the documentation of this file.

```
#ifndef PATIENT_H
#define PATIENT_H

#include "hospital.h"

 int patient_save_to_file(void);

 int patient_load_from_file(void);

 int patient_generate_id(void);

 void patient_add(void);

 void patient_search_by_id(void);

 void patient_search_by_name(void);

 void patient_search_by_phone(void);

 void patient_search_by(void);

 void patient_update_using_id(void);

 void patient_discharge(void);

 void patient_view_all(void);

 void patient_view_discharged(void);

 int patient_search_id(int id);

#endif
```

6.15 include/receptionist.h File Reference

Receptionist portal and data management functions.

```
#include "hospital.h"
```

Functions

- int receptionist_save_to_file (void)
- int receptionist_load_from_file (void)
- int receptionist_search_id (int id)
- void receptionist_view_all (void)
- void receptionist_view_discharged (void)
- void receptionist_discharge (void)
- void receptionist_patient_menu (void)
- void receptionist_appointment_menu (void)
- void receptionist_menu (void)

6.15.1 Detailed Description

Receptionist portal and data management functions.

Definition in file receptionist.h.

6.15.2 Function Documentation

6.15.2.1 receptionist_appointment_menu()

```
void receptionist_appointment_menu (
             void )
```

Receptionist appointment menu.

Definition at line 70 of file receptionist.c.

6.15.2.2 receptionist_discharge()

```
void receptionist_discharge (
             void )
```

Deactivates a receptionist.

Definition at line 243 of file receptionist.c.

6.15.2.3 receptionist_load_from_file()

```
int receptionist_load_from_file (
             void )
```

Loads all receptionists from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 170 of file receptionist.c.

6.15.2.4 receptionist_menu()

```
void receptionist_menu (
             void )
```

Main receptionist portal menu.

Definition at line 114 of file receptionist.c.

6.15.2.5 receptionist_patient_menu()

```
void receptionist_patient_menu (
             void )
```

Receptionist patient management menu.

Definition at line 16 of file receptionist.c.

6.15.2.6 receptionist_save_to_file()

```
int receptionist_save_to_file (
             void )
```

Saves all receptionists to binary file.

Returns

0 on success, -1 on failure.

Definition at line 155 of file receptionist.c.

6.15.2.7 receptionist_search_id()

```
int receptionist_search_id (
             int id)
```

Searches for a receptionist by ID.

Parameters

id	The receptionist ID to search for.

Returns

Index of receptionist, or -1 if not found.

Definition at line 194 of file receptionist.c.

6.15.2.8 receptionist_view_all()

```
void receptionist_view_all (
             void )
```

Views all active receptionists.

Definition at line 203 of file receptionist.c.

6.15.2.9 receptionist_view_discharged()

```
void receptionist_view_discharged (
             void )
```

Views all inactive receptionists.

Definition at line 223 of file receptionist.c.

6.16 receptionist.h

Go to the documentation of this file.

```
#ifndef RECEPTIONIST_H
#define RECEPTIONIST_H

#include "hospital.h"

int receptionist_save_to_file(void);

int receptionist_load_from_file(void);

int receptionist_search_id(int id);

void receptionist_view_all(void);

void receptionist_view_discharged(void);

void receptionist_discharge(void);

void receptionist_patient_menu(void);

void receptionist_appointment_menu(void);

void receptionist_menu(void);

#endif
```

6.17 include/ui.h File Reference

Header file for user interface functions.

```
#include "hospital.h"
```

Macros

- #define BLACK "\033[30m"
- #define RED "\033[31m"
- #define GREEN "\033[32m"
- #define BLUE "\033[34m"
- #define YELLOW "\033[33m"
- #define MAGENTA "\033[35m"
- #define CYAN "\033[36m"
- #define WHITE "\033[37m"
- #define BRIGHT_BLACK "\033[90m"
- #define BRIGHT_RED "\033[91m"
- #define BRIGHT_GREEN "\033[92m"
- #define BRIGHT_YELLOW "\033[93m"
- #define BRIGHT_BLUE "\033[94m"
- #define BRIGHT_MAGENTA "\033[95m"
- #define BRIGHT_CYAN "\033[96m"
- #define BRIGHT_WHITE "\033[97m"
- #define LIME "\033[38;5;154m"
- #define SOFT_CYAN "\033[38;5;159m"
- #define SOFT_TEAL "\033[38;5;44m"
- #define SOFT_GREEN "\033[38;5;120m"
- #define SOFT_RED "\033[38;5;124m"

- #define SOFT_BLUE "\033[38;5;75m"
- #define SOFT_GRAY "\033[38;5;250m"
- #define SOFT_YELLOW "\033[38;5;187m"
- #define NEON_TEAL "\033[38;5;51m"
- #define NEON_PINK "\033[38;5;205m"
- #define NEON_PURPLE "\033[38;5;141m"
- #define NEON_ORANGE "\033[38;5;208m"
- #define STATUS_GOOD "\033[38;5;114m"
- #define STATUS_WARN "\033[38;5;214m"
- #define STATUS_BAD "\033[38;5;160m"
- #define STATUS_INFO "\033[38;5;75m"
- #define TEXT_MUTED "\033[38;5;244m"
- #define TEXT_DIM "\033[38;5;240m"
- #define BG_BLACK "\033[40m"
- #define BG_WHITE "\033[47m"
- #define BG_BRIGHT_BLACK "\033[100m"
- #define BG_BRIGHT_RED "\033[101m"
- #define BG_BRIGHT_GREEN "\033[102m"
- #define BG_BRIGHT_YELLOW "\033[103m"
- #define BG_NEON_PURPLE "\033[48;5;57m"
- #define BG_NEON_TEAL "\033[48;5;43m"
- #define RESET "\033[0m"
- #define BOLD "\033[1m"
- #define UNDERLINE "\033[4m"
- #define HIDE_CURSOR "\033[?25l"
- #define SHOW_CURSOR "\033[?25h"

Functions

- void ui_clear_screen (void)
- void ui_pause (void)
- void ui_print_header (const char ∗title)
- void ui_print_success (const char ∗message)
- void ui_print_error (const char ∗message)
- void ui_print_warning (const char ∗message)
- void ui_print_info (const char ∗message)
- void ui_print_banner (void)
- void ui_print_menu (const char ∗title, const char ∗items[ ], int item_count, int box_width)
- void ui_print_patient (Patient patient, int index)
- void ui_print_doctor (Doctor doctor, int index)
- void ui_print_receptionist (Receptionist receptionist, int index)
- void ui_dummy_loading (int time)

6.17.1 Detailed Description

Header file for user interface functions.

This header file contains declarations for user interface functions used in the hospital management system.

Definition in file ui.h.

6.17.2 Macro Definition Documentation

6.17.2.1 BG_BLACK

```
#define BG_BLACK "\033[40m"
```

Definition at line 67 of file ui.h.

6.17.2.2 BG_BRIGHT_BLACK

```
#define BG_BRIGHT_BLACK "\033[100m"
```

Definition at line 71 of file ui.h.

6.17.2.3 BG_BRIGHT_GREEN

```
#define BG_BRIGHT_GREEN "\033[102m"
```

Definition at line 73 of file ui.h.

6.17.2.4 BG_BRIGHT_RED

```
#define BG_BRIGHT_RED "\033[101m"
```

Definition at line 72 of file ui.h.

6.17.2.5 BG_BRIGHT_YELLOW

```
#define BG_BRIGHT_YELLOW "\033[103m"
```

Definition at line 74 of file ui.h.

6.17.2.6 BG_NEON_PURPLE

```
#define BG_NEON_PURPLE "\033[48;5;57m"
```

Definition at line 77 of file ui.h.

6.17.2.7 BG_NEON_TEAL

```
#define BG_NEON_TEAL "\033[48;5;43m"
```

Definition at line 78 of file ui.h.

6.17.2.8 BG_WHITE

```
#define BG_WHITE "\033[47m"
```

Definition at line 68 of file ui.h.

6.17.2.9 BLACK

```
#define BLACK "\033[30m"
```

Definition at line 21 of file ui.h.

6.17.2.10 BLUE

```
#define BLUE "\033[34m"
```

Definition at line 24 of file ui.h.

6.17.2.11 BOLD

```
#define BOLD "\033[1m"
```

Definition at line 82 of file ui.h.

6.17.2.12 BRIGHT_BLACK

```
#define BRIGHT_BLACK "\033[90m"
```

Definition at line 31 of file ui.h.

6.17.2.13 BRIGHT_BLUE

```
#define BRIGHT_BLUE "\033[94m"
```

Definition at line 35 of file ui.h.

6.17.2.14 BRIGHT_CYAN

```
#define BRIGHT_CYAN "\033[96m"
```

Definition at line 37 of file ui.h.

6.17.2.15 BRIGHT_GREEN

```
#define BRIGHT_GREEN "\033[92m"
```

Definition at line 33 of file ui.h.

6.17.2.16 BRIGHT_MAGENTA

```
#define BRIGHT_MAGENTA "\033[95m"
```

Definition at line 36 of file ui.h.

6.17.2.17 BRIGHT_RED

```
#define BRIGHT_RED "\033[91m"
```

Definition at line 32 of file ui.h.

6.17.2.18 BRIGHT_WHITE

```
#define BRIGHT_WHITE "\033[97m"
```

Definition at line 38 of file ui.h.

6.17.2.19 BRIGHT_YELLOW

```
#define BRIGHT_YELLOW "\033[93m"
```

Definition at line 34 of file ui.h.

6.17.2.20 CYAN

```
#define CYAN "\033[36m"
```

Definition at line 27 of file ui.h.

6.17.2.21 GREEN

```
#define GREEN "\033[32m"
```

Definition at line 23 of file ui.h.

6.17.2.22 HIDE_CURSOR

```
#define HIDE_CURSOR "\033[?25l"
```

Definition at line 86 of file ui.h.

6.17.2.23 LIME

```
#define LIME "\033[38;5;154m"
```

Definition at line 41 of file ui.h.

6.17.2.24 MAGENTA

```
#define MAGENTA "\033[35m"
```

Definition at line 26 of file ui.h.

6.17.2.25 NEON_ORANGE

```
#define NEON_ORANGE "\033[38;5;208m"
```

Definition at line 54 of file ui.h.

6.17.2.26 NEON_PINK

```
#define NEON_PINK "\033[38;5;205m"
```

Definition at line 52 of file ui.h.

6.17.2.27 NEON_PURPLE

```
#define NEON_PURPLE "\033[38;5;141m"
```

Definition at line 53 of file ui.h.

6.17.2.28 NEON_TEAL

```
#define NEON_TEAL "\033[38;5;51m"
```

Definition at line 51 of file ui.h.

6.17.2.29 RED

```
#define RED "\033[31m"
```

Definition at line 22 of file ui.h.

6.17.2.30 RESET

```
#define RESET "\033[0m"
```

Definition at line 81 of file ui.h.

6.17.2.31 SHOW_CURSOR

```
#define SHOW_CURSOR "\033[?25h"
```

Definition at line 87 of file ui.h.

6.17.2.32 SOFT_BLUE

```
#define SOFT_BLUE "\033[38;5;75m"
```

Definition at line 46 of file ui.h.

6.17.2.33 SOFT_CYAN

```
#define SOFT_CYAN "\033[38;5;159m"
```

Definition at line 42 of file ui.h.

6.17.2.34 SOFT_GRAY

```
#define SOFT_GRAY "\033[38;5;250m"
```

Definition at line 47 of file ui.h.

6.17.2.35 SOFT_GREEN

```
#define SOFT_GREEN "\033[38;5;120m"
```

Definition at line 44 of file ui.h.

6.17.2.36 SOFT_RED

```
#define SOFT_RED "\033[38;5;124m"
```

Definition at line 45 of file ui.h.

6.17.2.37 SOFT_TEAL

```
#define SOFT_TEAL "\033[38;5;44m"
```

Definition at line 43 of file ui.h.

6.17.2.38 SOFT_YELLOW

```
#define SOFT_YELLOW "\033[38;5;187m"
```

Definition at line 48 of file ui.h.

6.17.2.39 STATUS_BAD

```
#define STATUS_BAD "\033[38;5;160m"
```

Definition at line 59 of file ui.h.

6.17.2.40 STATUS_GOOD

```
#define STATUS_GOOD "\033[38;5;114m"
```

Definition at line 57 of file ui.h.

6.17.2.41 STATUS_INFO

```
#define STATUS_INFO "\033[38;5;75m"
```

Definition at line 60 of file ui.h.

6.17.2.42 STATUS_WARN

```
#define STATUS_WARN "\033[38;5;214m"
```

Definition at line 58 of file ui.h.

6.17.2.43 TEXT_DIM

```
#define TEXT_DIM "\033[38;5;240m"
```

Definition at line 64 of file ui.h.

6.17.2.44 TEXT_MUTED

```
#define TEXT_MUTED "\033[38;5;244m"
```

Definition at line 63 of file ui.h.

6.17.2.45 UNDERLINE

```
#define UNDERLINE "\033[4m"
```

Definition at line 83 of file ui.h.

6.17.2.46 WHITE

```
#define WHITE "\033[37m"
```

Definition at line 28 of file ui.h.

6.17.2.47 YELLOW

```
#define YELLOW "\033[33m"
```

Definition at line 25 of file ui.h.

6.17.3 Function Documentation

6.17.3.1 ui_clear_screen()

```
void ui_clear_screen (
             void )
```

Clears the console screen.

Definition at line 22 of file ui.c.

6.17.3.2 ui_dummy_loading()

```
void ui_dummy_loading (
             int time)
```

Prints a dummy loading animation.

Definition at line 268 of file ui.c.

6.17.3.3 ui_pause()

```
void ui_pause (
             void )
```

Pauses the program execution until the user presses a key.

Definition at line 30 of file ui.c.

6.17.3.4 ui_print_banner()

```
void ui_print_banner (
             void )
```

Prints the HMS banner.

Definition at line 56 of file ui.c.

6.17.3.5 ui_print_doctor()

```
void ui_print_doctor (
             Doctor doctor,
             int index)
```

Prints a doctor in a box.

Parameters

doctor	The doctor to print.
index	The display index.

Definition at line 190 of file ui.c.

6.17.3.6 ui_print_error()

```
void ui_print_error (
             const char * message)
```

Prints an error message in red.

Parameters

message	The error message.

Definition at line 43 of file ui.c.

6.17.3.7 ui_print_header()

```
void ui_print_header (
             const char * title)
```

Prints a formatted header with title.

Parameters

title	The title to display.

Definition at line 35 of file ui.c.

6.17.3.8 ui_print_info()

```
void ui_print_info (
             const char * message)
```

Prints an info message in cyan.

Parameters

message	The info message.

Definition at line 51 of file ui.c.

6.17.3.9 ui_print_menu()

```
void ui_print_menu (
             const char * title,
             const char * items[],
             int item_count,
             int box_width)
```

Prints a menu in a box.

Parameters

title	The title of the menu.
items	The array of menu items.
item_count	The number of menu items.
box_width	The width of the box.

Definition at line 95 of file ui.c.

6.17.3.10 ui_print_patient()

```
void ui_print_patient (
            Patient patient,
            int index)
```

Prints a patient in a box.

Parameters

patient	The patient to print.

Definition at line 147 of file ui.c.

6.17.3.11 ui_print_receptionist()

```
void ui_print_receptionist (
            Receptionist receptionist,
            int index)
```

Prints a receptionist in a box.

Parameters

receptionist	The receptionist to print.
index	The display index.

Definition at line 233 of file ui.c.

6.17.3.12 ui_print_success()

```
void ui_print_success (
            const char * message)
```

Prints a success message in green.

Parameters

message	The success message.

Definition at line 39 of file ui.c.

6.17.3.13 ui_print_warning()

```
void ui_print_warning (
              const char * message)
```

Prints a warning message in yellow.

Parameters

message	The warning message.

Definition at line 47 of file ui.c.

6.18 ui.h

Go to the documentation of this file.

```


#ifndef UI_H
#define UI_H

#include "hospital.h"

/*
 *=====================================================================
 *                         ANSI COLOR CODES
 *=====================================================================
 */

/* Basic Foreground Colors */
#define BLACK           "\033[30m"
#define RED             "\033[31m"
#define GREEN           "\033[32m"
#define BLUE            "\033[34m"
#define YELLOW          "\033[33m"
#define MAGENTA         "\033[35m"
#define CYAN            "\033[36m"
#define WHITE           "\033[37m"

/* Bright Foreground Colors */
#define BRIGHT_BLACK    "\033[90m"
#define BRIGHT_RED      "\033[91m"
#define BRIGHT_GREEN    "\033[92m"
#define BRIGHT_YELLOW   "\033[93m"
#define BRIGHT_BLUE     "\033[94m"
#define BRIGHT_MAGENTA  "\033[95m"
#define BRIGHT_CYAN     "\033[96m"
#define BRIGHT_WHITE    "\033[97m"

/* Extended Foreground Colors (256-color palette) */
#define LIME            "\033[38;5;154m"
#define SOFT_CYAN       "\033[38;5;159m"
#define SOFT_TEAL       "\033[38;5;44m"
#define SOFT_GREEN      "\033[38;5;120m"
#define SOFT_RED        "\033[38;5;124m"
#define SOFT_BLUE       "\033[38;5;75m"
#define SOFT_GRAY       "\033[38;5;250m"
#define SOFT_YELLOW     "\033[38;5;187m"

/* Neon Accent Colors */
#define NEON_TEAL       "\033[38;5;51m"
```

```
#define NEON_PINK       "\033[38;5;205m"
#define NEON_PURPLE     "\033[38;5;141m"
#define NEON_ORANGE     "\033[38;5;208m"

/* Status Colors */
#define STATUS_GOOD     "\033[38;5;114m"
#define STATUS_WARN     "\033[38;5;214m"
#define STATUS_BAD      "\033[38;5;160m"
#define STATUS_INFO     "\033[38;5;75m"

/* Muted/Dim Text Colors */
#define TEXT_MUTED      "\033[38;5;244m"
#define TEXT_DIM        "\033[38;5;240m"

/* Basic Background Colors */
#define BG_BLACK        "\033[40m"
#define BG_WHITE        "\033[47m"

/* Bright Background Colors */
#define BG_BRIGHT_BLACK    "\033[100m"
#define BG_BRIGHT_RED      "\033[101m"
#define BG_BRIGHT_GREEN    "\033[102m"
#define BG_BRIGHT_YELLOW   "\033[103m"

/* Extended Background Colors */
#define BG_NEON_PURPLE "\033[48;5;57m"
#define BG_NEON_TEAL    "\033[48;5;43m"

/* Text Styles */
#define RESET           "\033[0m"
#define BOLD            "\033[1m"
#define UNDERLINE       "\033[4m"

/* Cursor */
#define HIDE_CURSOR     "\033[?25l"
#define SHOW_CURSOR     "\033[?25h"

/*
 *=======================================================================
 *                         FUNCTION PROTOTYPES
 *=======================================================================
 */


 void ui_clear_screen(void);


 void ui_pause(void);

 void ui_print_header(const char *title);

 void ui_print_success(const char *message);

 void ui_print_error(const char *message);

 void ui_print_warning(const char *message);

 void ui_print_info(const char *message);

 void ui_print_banner(void);

 void ui_print_menu(const char *title, const char *items[], int item_count, int box_width);

 void ui_print_patient(Patient patient, int index);

 void ui_print_doctor(Doctor doctor, int index);

 void ui_print_receptionist(Receptionist receptionist, int index);

 void ui_dummy_loading(int time);

 #endif
```

6.19 include/utils.h File Reference

Utility functions for Healthcare Management System.

```
#include <stddef.h>
#include <stdbool.h>
#include "hospital.h"
```

Functions

- void utils_clear_input_buffer (void)
- int utils_get_int (void)
- float utils_get_float (void)
- double utils_get_double (void)
- char utils_get_char (void)
- char ∗ utils_get_string (char ∗str, size_t size)
- bool utils_is_valid_phone (const char ∗phone)
- bool utils_is_valid_email (const char ∗email)
- bool utils_is_valid_name (const char ∗name)
- bool utils_is_valid_id (int id, UserRole role)
- bool utils_is_valid_gender (const char ∗gender)
- char ∗ utils_str_to_upper (char ∗str)
- char ∗ utils_fix_name (char ∗name)
- bool utils_is_valid_blood_group (const char ∗blood_group)
- bool utils_is_valid_address (const char ∗address)

6.19.1 Detailed Description

Utility functions for Healthcare Management System.

This header defines all shared utility functions used throughout the HMS application.

Definition in file utils.h.

6.19.2 Function Documentation

6.19.2.1 utils_clear_input_buffer()

```
void utils_clear_input_buffer (
             void )
```

Clears the input buffer to prevent leftover characters.

Definition at line 14 of file utils.c.

6.19.2.2 utils_fix_name()

```
char * utils_fix_name (
             char * name)
```

Fixes name to Title Case (first letter of each word uppercase, rest lowercase). Example: "jOHN dOE" -> "John Doe"

Parameters

name	The name string to fix (modified in-place).

Returns

The fixed name string.

Definition at line 169 of file utils.c.

6.19.2.3 utils_get_char()

```
char utils_get_char (
             void )
```

Scans a char value from the user.

Returns

The valid char entered by the user.

Definition at line 54 of file utils.c.

6.19.2.4 utils_get_double()

```
double utils_get_double (
             void )
```

Scans a double value from the user with validation.

Returns

The valid double entered by the user.

Definition at line 44 of file utils.c.

6.19.2.5 utils_get_float()

```
float utils_get_float (
             void )
```

Scans a float value from the user with validation.

Returns

The valid float entered by the user.

Definition at line 34 of file utils.c.

6.19.2.6 utils_get_int()

```
int utils_get_int (
             void )
```

Scans an integer value from the user with validation.

Returns

The valid integer entered by the user.

Definition at line 24 of file utils.c.

6.19.2.7 utils_get_string()

```
char * utils_get_string (
             char * str,
             size_t size)
```

Scans a string value from the user.

Parameters

str	The buffer to store the string.
size	The maximum number of characters to read.

Returns

Pointer to string on success, NULL on failure.

Definition at line 61 of file utils.c.

6.19.2.8 utils_is_valid_address()

```
bool utils_is_valid_address (
             const char * address)
```

Validates if an address contains only valid characters. Allowed: letters, digits, spaces, commas, periods.

Parameters

address	The address string to validate.

Returns

true if valid, false otherwise.

Definition at line 204 of file utils.c.

6.19.2.9 utils_is_valid_blood_group()

```
bool utils_is_valid_blood_group (
             const char * blood_group)
```

Validates if a blood group is in correct format.

Parameters

blood_group	The blood group string to validate.

Returns

true if valid, false otherwise.

Definition at line 186 of file utils.c.

6.19.2.10 utils_is_valid_email()

```
bool utils_is_valid_email (
             const char * email)
```

Validates if an email contains @ and .

Parameters

email	The email string to validate.

Returns

true if valid, false otherwise.

Definition at line 97 of file utils.c.

6.19.2.11 utils_is_valid_gender()

```
bool utils_is_valid_gender (
             const char * gender)
```

Validates if a gender is in correct format.

Parameters

gender	The gender string to validate.

Returns

true if valid gender, false otherwise.

6.19.2.12 utils_is_valid_id()

```
bool utils_is_valid_id (
             int id,
             UserRole role)
```

Validates if an ID is in correct format.

Parameters

id	The ID to validate.

Returns

true if valid ID, false otherwise.

Definition at line 140 of file utils.c.

6.19.2.13 utils_is_valid_name()

```
bool utils_is_valid_name (
             const char * name)
```

Validates if a name contains only letters and spaces.

Parameters

name	The name to validate.

Returns

true if valid name, false otherwise.

Definition at line 128 of file utils.c.

6.19.2.14 utils_is_valid_phone()

```
bool utils_is_valid_phone (
              const char * phone)
```

6.19.3 Validity Functions

Validates if a phone number is in correct format.

Parameters

phone	The phone number string to validate.

Returns

true if valid, false otherwise.

6.19.4 Validity Functions

Definition at line 81 of file utils.c.

6.19.4.1 utils_str_to_upper()

```
char * utils_str_to_upper (
              char * str)
```

Converts a string to uppercase in-place.

Parameters

str	The string to convert.

Returns

The uppercase string.

Definition at line 161 of file utils.c.

6.20 utils.h

Go to the documentation of this file.

```
#ifndef UTILS_H
#define UTILS_H

#include <stddef.h>
#include <stdbool.h>
#include "hospital.h"

 void utils_clear_input_buffer(void);

 int utils_get_int(void);

  float utils_get_float(void);

 double utils_get_double(void);

 char utils_get_char(void);

 char* utils_get_string(char *str, size_t size);


 bool utils_is_valid_phone(const char *phone);

 bool utils_is_valid_email(const char *email);

 bool utils_is_valid_name(const char *name);

 bool utils_is_valid_id(int id, UserRole role);

 bool utils_is_valid_gender(const char *gender);

 char* utils_str_to_upper(char *str);

 char* utils_fix_name(char *name);

 bool utils_is_valid_blood_group(const char *blood_group);

 bool utils_is_valid_address(const char *address);

#endif
```

6.21 src/admin.c File Reference

Admin management implementation for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include "../include/admin.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/receptionist.h"
#include "../include/auth.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
```

Functions

- void admin_view_discharged_patients (void)
- void admin_search_discharged_by_id (void)
- void admin_search_discharged_by_name (void)
- void admin_search_discharged (void)
- void admin_delete_patient (void)
- void admin_patient_menu (void)
- void admin_view_discharged_doctors (void)
- void admin_delete_doctor (void)
- void admin_doctor_menu (void)

- void admin_receptionist_menu (void)
- void admin_main_menu (void)

6.21.1 Detailed Description

Admin management implementation for Healthcare Management System.
This file contains admin-specific operations including discharged patient/doctor management that receptionists cannot access.
Definition in file admin.c.

6.21.2 Function Documentation

6.21.2.1 admin_delete_doctor()

```
void admin_delete_doctor (
             void )
```
Permanently deletes an inactive doctor from the system.
Definition at line 312 of file admin.c.

6.21.2.2 admin_delete_patient()

```
void admin_delete_patient (
             void )
```
Permanently deletes a discharged patient from the system.
Definition at line 154 of file admin.c.

6.21.2.3 admin_doctor_menu()

```
void admin_doctor_menu (
             void )
```
Admin doctor management menu.
Definition at line 382 of file admin.c.

6.21.2.4 admin_main_menu()

```
void admin_main_menu (
             void )
```
Main admin menu.
Definition at line 485 of file admin.c.

6.21.2.5 admin_patient_menu()

```
void admin_patient_menu (
             void )
```
Admin patient management menu.
Definition at line 224 of file admin.c.

6.21.2.6 admin_receptionist_menu()

```
void admin_receptionist_menu (
             void )
```
Admin receptionist management menu.
Definition at line 435 of file admin.c.

6.21.2.7 admin_search_discharged()

```
void admin_search_discharged (
             void )
```
Handles search choice for discharged patients.
Definition at line 121 of file admin.c.

6.21.2.8 admin_search_discharged_by_id()

```
void admin_search_discharged_by_id (
            void )
```

Searches discharged patients by ID.
Definition at line 46 of file admin.c.

6.21.2.9 admin_search_discharged_by_name()

```
void admin_search_discharged_by_name (
            void )
```

Searches discharged patients by name.
Definition at line 84 of file admin.c.

6.21.2.10 admin_view_discharged_doctors()

```
void admin_view_discharged_doctors (
            void )
```

Views all discharged (inactive) doctors.
Definition at line 292 of file admin.c.

6.21.2.11 admin_view_discharged_patients()

```
void admin_view_discharged_patients (
            void )
```

Views all discharged (inactive) patients.
Definition at line 26 of file admin.c.

6.22 admin.c

Go to the documentation of this file.

```
00001
00008
00009 #include <stdio.h>
00010 #include <string.h>
00011 #include "../include/admin.h"
00012 #include "../include/patient.h"
00013 #include "../include/doctor.h"
00014 #include "../include/receptionist.h"
00015 #include "../include/auth.h"
00016 #include "../include/utils.h"
00017 #include "../include/ui.h"
00018 #include "../include/hospital.h"
00019
00020 /*
00021  *==========================================================================
00022  *                        DISCHARGED PATIENT FUNCTIONS
00023  *==========================================================================
00024  */
00025
00026 void admin_view_discharged_patients(void) {
00027     int count = 0;
00028     ui_clear_screen();
00029     ui_print_banner();
00030
00031     if (patient_unavailable == 0) {
00032         const char* menu_items[] = {"No discharged patients found!"};
00033         ui_print_menu("Discharged Patients", menu_items, 1, UI_SIZE);
00034         ui_pause();
00035         return;
00036     }
00037
00038     for (int i = 0; i < patient_count; i++) {
00039         if (!patients[i].is_active) {
00040             ui_print_patient(patients[i], count++);
00041         }
00042     }
00043     ui_pause();
00044 }
00045
00046 void admin_search_discharged_by_id(void) {
00047     int id;
```

```

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter Patient ID (0 to cancel): ",
            "» "
        };

        ui_print_menu("Search Discharged Patient", menu_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) return;

        if (!utils_is_valid_id(id, ROLE_PATIENT)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }

        for (int i = 0; i < patient_count; i++) {
            if (patients[i].id == id && !patients[i].is_active) {
                ui_clear_screen();
                ui_print_banner();
                ui_print_patient(patients[i], i);
                ui_pause();
                return;
            }
        }
        ui_print_error("Discharged patient not found with that ID!");
        ui_pause();
        return;
    } while (1);
}

void admin_search_discharged_by_name(void) {
    char name[NAME_SIZE];

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter Patient Name: ",
            "» "
        };

        ui_print_menu("Search Discharged Patient", menu_items, 2, UI_SIZE);
        utils_get_string(name, NAME_SIZE);

        if (!utils_is_valid_name(name)) {
            ui_print_error("Invalid name!");
            ui_pause();
            continue;
        }

        utils_fix_name(name);
        for (int i = 0; i < patient_count; i++) {
            if (strcmp(patients[i].name, name) == 0 && !patients[i].is_active) {
                ui_clear_screen();
                ui_print_banner();
                ui_print_patient(patients[i], i);
                ui_pause();
                return;
            }
        }
        ui_print_error("Discharged patient not found with that name!");
        ui_pause();
        return;
    } while (1);
}

void admin_search_discharged(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Search by ID",
            "Search by Name",
            "Back",
            "» "
        };

```

```c
        ui_print_menu("Search Discharged Patients", menu_items, 4, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                admin_search_discharged_by_id();
                break;
            case 2:
                admin_search_discharged_by_name();
                break;
            case 3:
                return;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 3);
}

void admin_delete_patient(void) {
    int id;

    while (1) {
        ui_clear_screen();
        ui_print_banner();

        const char* enter_id_items[] = {
            "Enter patient ID to permanently delete (0 to cancel): ",
            "» "
        };

        ui_print_menu("Delete Patient Permanently", enter_id_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) return;

        if (!utils_is_valid_id(id, ROLE_PATIENT)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }
        break;
    }

    // Find the patient
    int index = -1;
    for (int i = 0; i < patient_count; i++) {
        if (patients[i].id == id && !patients[i].is_active) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        ui_print_error("Discharged patient not found! Only discharged patients can be permanently deleted.");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();
    ui_print_patient(patients[index], index);

    const char* menu[] = {
        "Confirm Permanent Delete",
        "Cancel",
        "» "
    };

    ui_print_menu("Delete Patient Permanently", menu, 3, UI_SIZE);
    int input = utils_get_int();

    if (input == 1) {
        // Shift all patients after this index
        for (int i = index; i < patient_count - 1; i++) {
            patients[i] = patients[i + 1];
        }
        patient_count--;
        patient_unavailable--;

        ui_print_success("Patient permanently deleted from the system!");
        patient_save_to_file();
        ui_pause();
    } else {
        ui_print_info("Deletion cancelled.");
        ui_pause();
    }
}

void admin_patient_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Add Patient",
            "View Active Patients",
            "View Discharged Patients",
            "Search Active Patient",
            "Search Discharged Patient",
            "Update Patient",
            "Discharge Patient",
            "Delete Patient Permanently",
            "Back to Admin Menu",
            "» "
        };

        ui_print_menu("Admin Patient Management", menu_items, 10, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                patient_add();
                patient_save_to_file();
                break;
            case 2:
                patient_view_all();
                break;
            case 3:
                admin_view_discharged_patients();
                break;
            case 4:
                patient_search_by();
                break;
            case 5:
                admin_search_discharged();
                break;
            case 6:
                patient_update_using_id();
                patient_save_to_file();
                break;
            case 7:
                patient_discharge();
                patient_save_to_file();
                break;
            case 8:
                admin_delete_patient();
                break;
            case 9:
                ui_print_info("Returning to admin menu...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 9);
}

/*
 *===========================================================================
 *                        INACTIVE DOCTOR FUNCTIONS
 *===========================================================================
 */

void admin_view_discharged_doctors(void) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    if (doctor_unavailable == 0) {
        const char* menu_items[] = {"No inactive doctors found!"};
        ui_print_menu("Inactive Doctors", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < doctor_count; i++) {
        if (!doctors[i].is_active) {
            ui_print_doctor(doctors[i], count++);
        }
    }
    ui_pause();
}

void admin_delete_doctor(void) {
    int id;

    while (1) {
        ui_clear_screen();
        ui_print_banner();

        const char* enter_id_items[] = {
            "Enter doctor ID to permanently delete (0 to cancel): ",
            "» "
        };

        ui_print_menu("Delete Doctor Permanently", enter_id_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) return;

        if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }
        break;
    }

    // Find the doctor
    int index = -1;
    for (int i = 0; i < doctor_count; i++) {
        if (doctors[i].id == id && !doctors[i].is_active) {
            index = i;
            break;
        }
    }

    if (index == -1) {
        ui_print_error("Inactive doctor not found! Only inactive doctors can be permanently
deleted.");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();
    ui_print_doctor(doctors[index], index);

    const char* menu[] = {
        "Confirm Permanent Delete",
        "Cancel",
        "» "
    };

    ui_print_menu("Delete Doctor Permanently", menu, 3, UI_SIZE);
    int input = utils_get_int();

    if (input == 1) {
        // Shift all doctors after this index
        for (int i = index; i < doctor_count - 1; i++) {
            doctors[i] = doctors[i + 1];
        }
        doctor_count--;
        doctor_unavailable--;

        ui_print_success("Doctor permanently deleted from the system!");
        doctor_save_to_file();
        ui_pause();
    } else {
        ui_print_info("Deletion cancelled.");
        ui_pause();
    }
}

void admin_doctor_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "View Active Doctors",
            "View Inactive Doctors",
            "Search Doctor",
            "Update Doctor",
            "Deactivate Doctor",
            "Delete Doctor Permanently",
            "Back to Admin Menu",
            "» "
        };

        ui_print_menu("Admin Doctor Management", menu_items, 8, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                doctor_view_all();
                break;
            case 2:
                admin_view_discharged_doctors();
                break;
            case 3:
                doctor_search_by();
                break;
            case 4:
                doctor_update_using_id();
                doctor_save_to_file();
                break;
            case 5:
                doctor_deactivate_account();
                doctor_save_to_file();
                break;
            case 6:
                admin_delete_doctor();
                break;
            case 7:
                ui_print_info("Returning to admin menu...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 7);
}

void admin_receptionist_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "View Active Receptionists",
            "View Inactive Receptionists",
            "Deactivate Receptionist",
            "Delete Receptionist Permanently",
            "Back to Admin Menu",
            "» "
        };

        ui_print_menu("Admin Receptionist Management", menu_items, 6, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                receptionist_view_all();
                break;
            case 2:
                receptionist_view_discharged();
                break;
            case 3:
                receptionist_discharge();
                break;
            case 4:
                ui_print_info("Delete receptionist permanently - Coming soon!");
                ui_pause();
                break;
            case 5:
                ui_print_info("Returning to admin menu...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 4);
}

/*
 *===========================================================================
 *                              MAIN ADMIN MENU
 *==========================================================================
 */

void admin_main_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Register New User",
            "Patient Management",
            "Doctor Management",
            "Receptionist Management",
            "Logout",
            "» "
        };

        ui_print_menu("Admin Portal", menu_items, 6, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                auth_register_user();
                break;
            case 2:
                admin_patient_menu();
                break;
            case 3:
                admin_doctor_menu();
                break;
            case 4:
                admin_receptionist_menu();
                break;
            case 5:
                ui_print_info("Logging out...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 5);
}
```

6.23 src/appointment.c File Reference

Appointment management implementation for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include "../include/appointment.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
```

Functions

- int appointment_save_to_file (void)
- int appointment_load_from_file (void)
- int appointment_generate_id (void)
- const char ∗ appointment_status_str (AppointmentStatus status)
- int appointment_search_id (int id)
- void appointment_create (void)
- void ui_print_appointment (Appointment appt, int index)
- void appointment_view_by_doctor (int doctor_id)
- void appointment_update_status (int appt_id, AppointmentStatus status)
- void appointment_cancel (int appt_id)

6.23.1 Detailed Description

Appointment management implementation for Healthcare Management System.
This file contains the implementation of appointment CRUD operations.
Definition in file appointment.c.

6.23.2 Function Documentation

6.23.2.1 appointment_cancel()

```
void appointment_cancel (
             int appt_id)
```

Cancels an appointment.

Parameters

appt←_id	The appointment ID.

Definition at line 305 of file appointment.c.

6.23.2.2 appointment_create()

```
void appointment_create (
             void )
```

Creates a new appointment (called by receptionist).
Definition at line 77 of file appointment.c.

6.23.2.3 appointment_generate_id()

```
int appointment_generate_id (
             void )
```

Generates a unique appointment ID.

Returns

The generated appointment ID.

Definition at line 54 of file appointment.c.

6.23.2.4 appointment_load_from_file()

```
int appointment_load_from_file (
             void )
```

Loads all appointments from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 31 of file appointment.c.

6.23.2.5 appointment_save_to_file()

```
int appointment_save_to_file (
             void )
```

Saves all appointments to binary file.

Returns

0 on success, -1 on failure.

Definition at line 17 of file appointment.c.

6.23.2.6 appointment_search_id()

```
int appointment_search_id (
            int id)
```

Finds appointment by ID.

Parameters

id	The appointment ID.

Returns

Index of appointment, or -1 if not found.

Definition at line 68 of file appointment.c.

6.23.2.7 appointment_status_str()

```
const char * appointment_status_str (
            AppointmentStatus status)
```

Gets status string from enum.

Parameters

status	The appointment status.

Returns

String representation.

Definition at line 58 of file appointment.c.

6.23.2.8 appointment_update_status()

```
void appointment_update_status (
            int appt_id,
            AppointmentStatus status)
```

Updates appointment status.

Parameters

appt↩_id	The appointment ID.
status	The new status.

Definition at line 291 of file appointment.c.

6.23.2.9 appointment_view_by_doctor()

```
void appointment_view_by_doctor (
            int doctor_id)
```

Views all appointments for a specific doctor.

Parameters

doctor←_id	The doctor's ID.

Definition at line 272 of file appointment.c.

6.23.2.10 ui_print_appointment()

```
void ui_print_appointment (
             Appointment appt,
             int index)
```

Prints an appointment in a formatted box.

Parameters

appt	The appointment to print.
index	The display index.

Definition at line 229 of file appointment.c.

6.24 appointment.c

Go to the documentation of this file.

```

#include <stdio.h>
#include <string.h>
#include "../include/appointment.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"

int appointment_save_to_file(void) {
    FILE* file = fopen(APPOINTMENTS_FILE, "wb");
    if (file == NULL) {
        return -1;
    }
    if (fwrite(&appointment_count, sizeof(int), 1, file) != 1 ||
        fwrite(appointments, sizeof(Appointment), appointment_count, file) !=
(size_t)appointment_count) {
        fclose(file);
        return -1;
    }
    fclose(file);
    return 0;
}

int appointment_load_from_file(void) {
    FILE* file = fopen(APPOINTMENTS_FILE, "rb");
    if (file == NULL) {
        return -1;
    }
    if (fread(&appointment_count, sizeof(int), 1, file) != 1) {
        fclose(file);
        return -1;
    }
    if (appointment_count < 0 || appointment_count > MAX_APPOINTMENTS) {
        fclose(file);
        appointment_count = 0;
        return -1;
    }
    if (fread(appointments, sizeof(Appointment), appointment_count, file) !=
(size_t)appointment_count) {
        fclose(file);
        appointment_count = 0;
        return -1;
    }
    fclose(file);
    return 0;
}

```

```c
int appointment_generate_id(void) {
    return APPOINTMENT_ID_START + appointment_count;
}

const char* appointment_status_str(AppointmentStatus status) {
    switch (status) {
        case APPT_PENDING:   return "Pending";
        case APPT_CONFIRMED: return "Confirmed";
        case APPT_COMPLETED: return "Completed";
        case APPT_CANCELLED: return "Cancelled";
        default:             return "Unknown";
    }
}

int appointment_search_id(int id) {
    for (int i = 0; i < appointment_count; i++) {
        if (appointments[i].id == id) {
            return i;
        }
    }
    return -1;
}

void appointment_create(void) {
    if (appointment_count >= MAX_APPOINTMENTS) {
        ui_print_error("Error: Maximum appointment limit reached!");
        ui_pause();
        return;
    }

    Appointment new_appt;
    new_appt.id = appointment_generate_id();
    new_appt.status = APPT_PENDING;

    char patient_line[70], doctor_line[70], date_line[70], time_line[70];

    // Step 1: Get Patient ID
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step1[] = {"Patient ID:", "» "};
        ui_print_menu("Create Appointment", step1, 2, UI_SIZE);
        new_appt.patient_id = utils_get_int();

        if (!utils_is_valid_id(new_appt.patient_id, ROLE_PATIENT)) {
            ui_print_error("Invalid patient ID!");
            ui_pause();
            continue;
        }

        int idx = patient_search_id(new_appt.patient_id);
        if (idx == -1 || !patients[idx].is_active) {
            ui_print_error("Patient not found or inactive!");
            ui_pause();
            continue;
        }
        snprintf(patient_line, sizeof(patient_line), "Patient: %s (ID: %d)",
                 patients[idx].name, new_appt.patient_id);
        break;
    }

    // Step 2: Get Doctor ID
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step2[] = {patient_line, "Doctor ID:", "» "};
        ui_print_menu("Create Appointment", step2, 3, UI_SIZE);
        new_appt.doctor_id = utils_get_int();

        if (!utils_is_valid_id(new_appt.doctor_id, ROLE_DOCTOR)) {
            ui_print_error("Invalid doctor ID!");
            ui_pause();
            continue;
        }

        int idx = doctor_search_id(new_appt.doctor_id);
        if (idx == -1 || !doctors[idx].is_active) {
            ui_print_error("Doctor not found or inactive!");
            ui_pause();
            continue;
        }
        if (!doctors[idx].is_available) {
            ui_print_warning("Warning: Doctor is marked as unavailable.");
            ui_pause();
        }
        snprintf(doctor_line, sizeof(doctor_line), "Doctor: Dr. %s (ID: %d)",
                 doctors[idx].name, new_appt.doctor_id);
        break;
    }

    // Step 3: Get Date
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step3[] = {patient_line, doctor_line, "Date (DD-MM-YYYY):", "» "};
        ui_print_menu("Create Appointment", step3, 4, UI_SIZE);
        utils_get_string(new_appt.date, DATE_SIZE);

        if (strlen(new_appt.date) > 0) {
            snprintf(date_line, sizeof(date_line), "Date: %s", new_appt.date);
            break;
        }
        ui_print_error("Date cannot be empty!");
        ui_pause();
    }

    // Step 4: Get Time Slot
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step4[] = {patient_line, doctor_line, date_line, "Time (e.g. 10:00 AM):", "» "};
        ui_print_menu("Create Appointment", step4, 5, UI_SIZE);
        utils_get_string(new_appt.time_slot, TIME_SIZE);

        if (strlen(new_appt.time_slot) > 0) {
            snprintf(time_line, sizeof(time_line), "Time: %s", new_appt.time_slot);
            break;
        }
        ui_print_error("Time cannot be empty!");
        ui_pause();
    }

    // Step 5: Get Reason
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step5[] = {patient_line, doctor_line, date_line, time_line, "Reason for visit:",
"» "};
        ui_print_menu("Create Appointment", step5, 6, UI_SIZE);
        utils_get_string(new_appt.reason, REASON_SIZE);

        if (strlen(new_appt.reason) > 0) {
            break;
        }
        ui_print_error("Reason cannot be empty!");
        ui_pause();
    }

    // Step 6: Confirm
    char reason_line[120];
    snprintf(reason_line, sizeof(reason_line), "Reason: %s", new_appt.reason);
    const char* step6[] = {
        patient_line, doctor_line, date_line, time_line, reason_line,
        "Confirm (Y/N):", "» "
    };
    ui_clear_screen();
    ui_print_banner();
    ui_print_menu("Create Appointment", step6, 7, UI_SIZE);
    char confirm = utils_get_char();

    if (confirm != 'Y' && confirm != 'y') {
        ui_print_info("Appointment creation cancelled.");
        ui_pause();
        return;
    }

    // Add to array
    appointments[appointment_count] = new_appt;
    appointment_count++;

    ui_clear_screen();
    ui_print_banner();

    char id_line[70];
    snprintf(id_line, sizeof(id_line), "Appointment ID: %d", new_appt.id);
    const char* success_items[] = {
        id_line, patient_line, doctor_line, date_line, time_line,
        "Status: Pending",
        "Appointment created successfully!"
    };
    ui_print_menu("Appointment Created", success_items, 7, UI_SIZE);
    ui_pause();

    appointment_save_to_file();
```

```
}

void ui_print_appointment(Appointment appt, int index) {
    int p_idx = patient_search_id(appt.patient_id);
    int d_idx = doctor_search_id(appt.doctor_id);

    char id_line[70];
    snprintf(id_line, sizeof(id_line), "Appointment ID: %d", appt.id);

    char patient_line[70];
    if (p_idx != -1) {
        snprintf(patient_line, sizeof(patient_line), "Patient: %s (ID: %d)",
                 patients[p_idx].name, appt.patient_id);
    } else {
        snprintf(patient_line, sizeof(patient_line), "Patient ID: %d", appt.patient_id);
    }

    char doctor_line[70];
    if (d_idx != -1) {
        snprintf(doctor_line, sizeof(doctor_line), "Doctor: Dr. %s", doctors[d_idx].name);
    } else {
        snprintf(doctor_line, sizeof(doctor_line), "Doctor ID: %d", appt.doctor_id);
    }

    char date_line[70];
    snprintf(date_line, sizeof(date_line), "Date: %s", appt.date);

    char time_line[70];
    snprintf(time_line, sizeof(time_line), "Time: %s", appt.time_slot);

    char reason_line[120];
    snprintf(reason_line, sizeof(reason_line), "Reason: %s", appt.reason);

    char status_line[70];
    snprintf(status_line, sizeof(status_line), "Status: %s", appointment_status_str(appt.status));

    const char* items[] = {
        id_line, patient_line, doctor_line, date_line, time_line, reason_line, status_line, ""
    };

    char title[70];
    snprintf(title, sizeof(title), "Appointment %d", index + 1);
    ui_print_menu(title, items, 8, 72);
}

void appointment_view_by_doctor(int doctor_id) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    for (int i = 0; i < appointment_count; i++) {
        if (appointments[i].doctor_id == doctor_id &&
            appointments[i].status != APPT_CANCELLED) {
            ui_print_appointment(appointments[i], count++);
        }
    }

    if (count == 0) {
        const char* menu_items[] = {"No appointments found!"};
        ui_print_menu("My Appointments", menu_items, 1, UI_SIZE);
    }
    ui_pause();
}

void appointment_update_status(int appt_id, AppointmentStatus status) {
    int idx = appointment_search_id(appt_id);
    if (idx == -1) {
        ui_print_error("Appointment not found!");
        ui_pause();
        return;
    }

    appointments[idx].status = status;
    appointment_save_to_file();
    ui_print_success("Appointment status updated!");
    ui_pause();
}

void appointment_cancel(int appt_id) {
    appointment_update_status(appt_id, APPT_CANCELLED);
}
```

6.25 src/auth.c File Reference

Authentication implementation for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include "../include/auth.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
#include "../include/receptionist.h"
#include "../include/admin.h"
#include "../include/doctor_portal.h"
#include "../include/doctor.h"
```

Functions

- int auth_save_to_file (void)
- int auth_load_from_file (void)
- void auth_init_default_admin (void)
- void auth_register_user (void)
- void auth_view_users (void)
- void auth_user_menu (void)
- void auth_role_login (UserRole required_role)
- void login_menu (void)
- void encrypt (char ∗password)
- void decrypt (char ∗password)

6.25.1 Detailed Description

Authentication implementation for Healthcare Management System.
Definition in file auth.c.

6.25.2 Function Documentation

6.25.2.1 auth_init_default_admin()

```
void auth_init_default_admin (
            void )
```

Creates default admin if no users exist.
Definition at line 57 of file auth.c.

6.25.2.2 auth_load_from_file()

```
int auth_load_from_file (
            void )
```

Loads all users from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 34 of file auth.c.

6.25.2.3 auth_register_user()

```
void auth_register_user (
            void )
```

Registers a new user (admin only).
Definition at line 70 of file auth.c.

6.25.2.4 auth_role_login()

```
void auth_role_login (
             UserRole required_role)
```

Login with role-based authentication.

Parameters

required_role	The role to authenticate as.

Definition at line 402 of file auth.c.

6.25.2.5 auth_save_to_file()

```
int auth_save_to_file (
             void )
```

Saves all users to binary file.

Returns

0 on success, -1 on failure.

Definition at line 17 of file auth.c.

6.25.2.6 auth_user_menu()

```
void auth_user_menu (
             void )
```

User management menu (admin only).
Definition at line 367 of file auth.c.

6.25.2.7 auth_view_users()

```
void auth_view_users (
             void )
```

Views all users (admin only).
Definition at line 330 of file auth.c.

6.25.2.8 decrypt()

```
void decrypt (
             char * password)
```

Decrypts a password using XOR cipher.

Parameters

password	The password to decrypt.

Definition at line 539 of file auth.c.

6.25.2.9 encrypt()

```
void encrypt (
             char * password)
```

Encrypts a password using XOR cipher.

Parameters

password	The password to encrypt.

Definition at line 533 of file auth.c.

6.25.2.10 login_menu()

```
void login_menu (
              void )
```

Login menu with role selection.
Definition at line 496 of file auth.c.

6.26 auth.c

Go to the documentation of this file.

```


#include <stdio.h>
#include <string.h>
#include "../include/auth.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
#include "../include/receptionist.h"
#include "../include/admin.h"
#include "../include/doctor_portal.h"
#include "../include/doctor.h"

int auth_save_to_file(void) {
    FILE* file = fopen(USERS_FILE, "wb");
    if (file == NULL) {
        return -1;
    }
    if (fwrite(&user_count, sizeof(int), 1, file) != 1) {
        fclose(file);
        return -1;
    }
    if (fwrite(users, sizeof(User), user_count, file) != (size_t)user_count) {
        fclose(file);
        return -1;
    }
    fclose(file);
    return 0;
}

int auth_load_from_file(void) {
    FILE* file = fopen(USERS_FILE, "rb");
    if (file == NULL) {
        return -1;
    }
    if (fread(&user_count, sizeof(int), 1, file) != 1) {
        fclose(file);
        return -1;
    }
    if (user_count < 0 || user_count > MAX_USERS) {
        fclose(file);
        user_count = 0;
        return -1;
    }
    if (fread(users, sizeof(User), user_count, file) != (size_t)user_count) {
        fclose(file);
        user_count = 0;
        return -1;
    }
    fclose(file);
    return 0;
}

void auth_init_default_admin(void) {
    if (user_count == 0) {
        users[0].id = ADMIN_ID_START;
        strcpy(users[0].username, "admin");
        strcpy(users[0].password, "admin123");
        encrypt(users[0].password);
        users[0].role = ROLE_ADMIN;
        users[0].is_active = true;
        user_count = 1;
        auth_save_to_file();
    }
```

```
}

void auth_register_user(void) {
    if (user_count >= MAX_USERS) {
        ui_print_error("Maximum user limit reached!");
        ui_pause();
        return;
    }

    User new_user;
    Doctor new_doctor;
    char username_line[80], password_line[80], role_line[80];
    char name_line[NAME_LINE_SIZE], phone_line[PHONE_LINE_SIZE], email_line[EMAIL_LINE_SIZE];
    char spec_line[SPEC_LINE_SIZE], room_line[ROOM_LINE_SIZE];
    int selected_role = 0;

    // Step 1: Select role first
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step1[] = {
            "Admin",
            "Receptionist",
            "Doctor",
            "» ",
        };
        ui_print_menu("Register User", step1, 4, UI_SIZE);
        selected_role = utils_get_int();

        switch (selected_role) {
            case 0:
                ui_print_info("Cancelled.");
                ui_pause();
                return;
            case 1:
                new_user.role = ROLE_ADMIN;
                snprintf(role_line, sizeof(role_line), "Role: Admin");
                new_user.id = ADMIN_ID_START + user_count;
                break;
            case 2:
                new_user.role = ROLE_RECEPTIONIST;
                snprintf(role_line, sizeof(role_line), "Role: Receptionist");
                new_user.id = RECEPTIONIST_ID_START + user_count;
                break;
            case 3:
                if (doctor_count >= MAX_DOCTORS) {
                    ui_print_error("Maximum doctor limit reached!");
                    ui_pause();
                    return;
                }
                new_user.role = ROLE_DOCTOR;
                snprintf(role_line, sizeof(role_line), "Role: Doctor");
                new_user.id = DOCTOR_ID_START + doctor_count;
                new_doctor.id = new_user.id;
                break;
            default:
                ui_print_error("Invalid role! Try again.");
                ui_pause();
                continue;
        }
        break;
    }

    // Step 2: Get username
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step2[] = {role_line, "Username:", "» "};
        ui_print_menu("Register User", step2, 3, UI_SIZE);
        utils_get_string(new_user.username, USERNAME_SIZE);

        if (strlen(new_user.username) < 3) {
            ui_print_error("Username must be at least 3 characters!");
            ui_pause();
            continue;
        }

        // Check if username exists
        bool exists = false;
        for (int i = 0; i < user_count; i++) {
            if (strcmp(users[i].username, new_user.username) == 0) {
                exists = true;
                break;
            }
        }
        if (exists) {
            ui_print_error("Username already exists!");
            ui_pause();
            continue;
        }

        snprintf(username_line, sizeof(username_line), "Username: %s", new_user.username);
        break;
    }

    // Step 3: Get password
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step3[] = {role_line, username_line, "Password:", "» "};
        ui_print_menu("Register User", step3, 4, UI_SIZE);
        utils_get_string(new_user.password, PASSWORD_SIZE);

        if (strlen(new_user.password) < 4) {
            ui_print_error("Password must be at least 4 characters!");
            ui_pause();
            continue;
        }
        snprintf(password_line, sizeof(password_line), "Password: ****");
        break;
    }

    // If Doctor, collect additional profile data
    if (selected_role == 3) {
        // Step 4: Get full name
        while (1) {
            ui_clear_screen();
            ui_print_banner();
            const char* step4[] = {role_line, username_line, "Full Name:", "» "};
            ui_print_menu("Register Doctor", step4, 4, UI_SIZE);
            utils_get_string(new_doctor.name, NAME_SIZE);

            if (!utils_is_valid_name(new_doctor.name)) {
                ui_print_error("Invalid name!");
                ui_pause();
                continue;
            }
            utils_fix_name(new_doctor.name);
            snprintf(name_line, sizeof(name_line), "Name: %s", new_doctor.name);
            break;
        }

        // Step 5: Get phone
        while (1) {
            ui_clear_screen();
            ui_print_banner();
            const char* step5[] = {role_line, username_line, name_line, "Phone (11 digits):", "» "};
            ui_print_menu("Register Doctor", step5, 5, UI_SIZE);
            utils_get_string(new_doctor.phone, PHONE_SIZE);

            if (!utils_is_valid_phone(new_doctor.phone)) {
                ui_print_error("Invalid phone!");
                ui_pause();
                continue;
            }
            snprintf(phone_line, sizeof(phone_line), "Phone: %s", new_doctor.phone);
            break;
        }

        // Step 6: Get email
        while (1) {
            ui_clear_screen();
            ui_print_banner();
            const char* step6[] = {role_line, username_line, name_line, phone_line, "Email:", "» "};
            ui_print_menu("Register Doctor", step6, 6, UI_SIZE);
            utils_get_string(new_doctor.email, EMAIL_SIZE);

            if (strlen(new_doctor.email) < 5) {
                ui_print_error("Invalid email!");
                ui_pause();
                continue;
            }
            snprintf(email_line, sizeof(email_line), "Email: %s", new_doctor.email);
            break;
        }

        // Step 7: Get specialization
        while (1) {
            ui_clear_screen();
            ui_print_banner();
            const char* step7[] = {role_line, username_line, name_line, phone_line, email_line,
      "Specialization:", "» "};
            ui_print_menu("Register Doctor", step7, 7, UI_SIZE);
            utils_get_string(new_doctor.specialization, SPEC_SIZE);

            if (strlen(new_doctor.specialization) < 2) {
                ui_print_error("Invalid specialization!");
                ui_pause();
                continue;
            }
            snprintf(spec_line, sizeof(spec_line), "Specialization: %s", new_doctor.specialization);
            break;
        }

        // Step 8: Get room number
        while (1) {
            ui_clear_screen();
            ui_print_banner();
            const char* step8[] = {role_line, name_line, phone_line, spec_line, "Room Number:", "» "};
            ui_print_menu("Register Doctor", step8, 6, UI_SIZE);
            new_doctor.room_number = utils_get_int();

            if (new_doctor.room_number < 1 || new_doctor.room_number > 999) {
                ui_print_error("Invalid room number (1-999)!");
                ui_pause();
                continue;
            }
            snprintf(room_line, sizeof(room_line), "Room: %d", new_doctor.room_number);
            break;
        }

        new_doctor.is_available = true;
        new_doctor.is_active = true;

        // Confirm Doctor registration
        ui_clear_screen();
        ui_print_banner();
        const char* confirm_doc[] = {
            username_line,
            name_line,
            phone_line,
            email_line,
            spec_line,
            room_line,
            "Confirm registration? (Y/N):",
            "» "
        };
        ui_print_menu("Confirm Doctor Registration", confirm_doc, 8, UI_SIZE);
        char confirm = utils_get_char();

        if (confirm != 'Y' && confirm != 'y') {
            ui_print_info("Registration cancelled.");
            ui_pause();
            return;
        }

        // Save doctor
        doctors[doctor_count] = new_doctor;
        doctor_count++;
        doctor_available++;
        doctor_save_to_file();

    } else {
        // Confirm Admin/Receptionist registration
        ui_clear_screen();
        ui_print_banner();
        const char* confirm_items[] = {
            username_line,
            role_line,
            "Confirm registration? (Y/N):",
            "» "
        };
        ui_print_menu("Register User", confirm_items, 4, UI_SIZE);
        char confirm = utils_get_char();

        if (confirm != 'Y' && confirm != 'y') {
            ui_print_info("Registration cancelled.");
            ui_pause();
            return;
        }
    }

    // Save user credentials
    new_user.is_active = true;
    encrypt(new_user.password);  // Encryption
    users[user_count] = new_user;
    user_count++;
    auth_save_to_file();

    ui_print_success("User registered successfully!");
    ui_pause();
```

```
}

void auth_view_users(void) {
    ui_clear_screen();
    ui_print_banner();

    if (user_count == 0) {
        const char* empty[] = {"No users found!"};
        ui_print_menu("All Users", empty, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < user_count; i++) {
        char id_line[50], user_line[80], role_line[50], status_line[50];

        snprintf(id_line, sizeof(id_line), "User ID: %d", users[i].id);
        snprintf(user_line, sizeof(user_line), "Username: %s", users[i].username);

        const char* role_str;
        switch (users[i].role) {
            case ROLE_ADMIN: role_str = "Admin"; break;
            case ROLE_RECEPTIONIST: role_str = "Receptionist"; break;
            case ROLE_DOCTOR: role_str = "Doctor"; break;
            default: role_str = "Unknown";
        }
        snprintf(role_line, sizeof(role_line), "Role: %s", role_str);
        snprintf(status_line, sizeof(status_line), "Status: %s",
                 users[i].is_active ? "Active" : "Inactive");

        const char* items[] = {id_line, user_line, role_line, status_line, ""};

        char title[50];
        snprintf(title, sizeof(title), "User %d", i + 1);
        ui_print_menu(title, items, 5, UI_SIZE);
    }
    ui_pause();
}

void auth_user_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Register New User",
            "View All Users",
            "Back",
            "» "
        };

        ui_print_menu("User Management", menu_items, 4, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                auth_register_user();
                break;
            case 2:
                auth_view_users();
                break;
            case 3:
                ui_print_info("Returning...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 3);
}

void auth_role_login(UserRole required_role) {
    char username[USERNAME_SIZE];
    char password[PASSWORD_SIZE];

    // Get role name for display
    const char* role_name;
    switch (required_role) {
        case ROLE_ADMIN:
            role_name = "Admin";
            break;
        case ROLE_RECEPTIONIST:
            if (receptionist_count == 0) {
                ui_print_error("No receptionist account found!");
                ui_pause();
                return;
            }
            role_name = "Receptionist";
            break;
        case ROLE_DOCTOR:
            if (doctor_count == 0) {
                ui_print_error("No doctor account found!");
                ui_pause();
                return;
            }
            role_name = "Doctor";
            break;
        default:
            role_name = "User";
    }

    char title[50];
    snprintf(title, sizeof(title), "%s Login", role_name);

    // Get username
    ui_clear_screen();
    ui_print_banner();

    const char* menu_items[] = {
        "Username:",
        "» "
    };
    ui_print_menu(title, menu_items, 2, UI_SIZE);
    utils_get_string(username, USERNAME_SIZE);

    // Get password
    ui_clear_screen();
    ui_print_banner();

    char user_line[80];
    snprintf(user_line, sizeof(user_line), "Username: %s", username);
    const char* pass_items[] = {
        user_line,
        "Password:",
        "» "
    };
    ui_print_menu(title, pass_items, 3, UI_SIZE);
    utils_get_string(password, PASSWORD_SIZE);
    encrypt(password);  // Encryption for comparison

    // Find user with matching role
    for (int i = 0; i < user_count; i++) {
        if (strcmp(users[i].username, username) == 0 &&
            strcmp(users[i].password, password) == 0 &&
            users[i].is_active &&
            users[i].role == required_role) {

            current_user = &users[i];

            // Route to appropriate portal
            switch (required_role) {
                case ROLE_ADMIN:
                    admin_main_menu();
                    break;
                case ROLE_RECEPTIONIST:
                    receptionist_menu();
                    break;
                case ROLE_DOCTOR:
                    doctor_portal_menu(users[i].id, username);
                    break;
                default:
                    break;
            }

            current_user = NULL;
            return;
        }
    }

    char error_msg[80];
    snprintf(error_msg, sizeof(error_msg), "Invalid credentials or not a %s!", role_name);
    ui_print_error(error_msg);
    ui_pause();
}

void login_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Admin Login",
            "Receptionist Login",
            "Doctor Login",
            "Back",
            "» "
        };

        ui_print_menu("Login", menu_items, 5, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                auth_role_login(ROLE_ADMIN);
                break;
            case 2:
                auth_role_login(ROLE_RECEPTIONIST);
                break;
            case 3:
                auth_role_login(ROLE_DOCTOR);
                break;
            case 4:
                return;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 4);
}

void encrypt(char* password) {
    for (int i = 0; password[i] != '\0'; i++) {
        password[i] = password[i] ^ 1;
    }
}

void decrypt(char* password) {
    encrypt(password);
}
```

6.27 src/doctor.c File Reference

Doctor management implementation for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include "../include/doctor.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
```

Functions

- int doctor_save_to_file (void)
- int doctor_load_from_file (void)
- int doctor_generate_id (void)
- void doctor_view_all ()
- void doctor_view_one ()
- void doctor_view ()
- void doctor_search_by_id (void)
- void doctor_search_by_name (void)
- void doctor_search_by_phone (void)
- void doctor_search_by (void)
- int doctor_search_id (int id)
- void doctor_update_name (const char ∗menu_items[ ], int index)
- void doctor_update_phone (const char ∗menu_items[ ], int index)
- void doctor_update_email (const char ∗menu_items[ ], int index)
- void doctor_update_specialization (const char ∗menu_items[ ], int index)
- void doctor_update_room (const char ∗menu_items[ ], int index)

- void doctor_update_availability (const char ∗menu_items[ ], int index)
- void doctor_update_status (const char ∗menu_items[ ], int index)
- void doctor_update_using_id ()
- void doctor_deactivate_account ()
- void doctor_view_discharged (void)

6.27.1 Detailed Description

Doctor management implementation for Healthcare Management System.
This file contains the implementation of all doctor CRUD operations.
Definition in file doctor.c.

6.27.2 Function Documentation

6.27.2.1 doctor_deactivate_account()

```
void doctor_deactivate_account (
             void )
```

Deactivates a doctor by ID (sets is_active to false).
Definition at line 563 of file doctor.c.

6.27.2.2 doctor_generate_id()

```
int doctor_generate_id (
             void )
```

Generates a unique doctor ID.

Returns

The generated doctor ID.

Definition at line 54 of file doctor.c.

6.27.2.3 doctor_load_from_file()

```
int doctor_load_from_file (
             void )
```

Loads all doctors from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 30 of file doctor.c.

6.27.2.4 doctor_save_to_file()

```
int doctor_save_to_file (
             void )
```

Saves all doctors to binary file.

Returns

0 on success, -1 on failure.

Definition at line 15 of file doctor.c.

6.27.2.5 doctor_search_by()

```
void doctor_search_by (
             void )
```

Handles the search choice for doctor.
Definition at line 249 of file doctor.c.

6.27.2.6 doctor_search_by_id()

```
void doctor_search_by_id (
             void )
```

Searches for a doctor by ID.
Definition at line 137 of file doctor.c.

6.27.2.7 doctor_search_by_name()

```
void doctor_search_by_name (
             void )
```

Searches for a doctor by name.
Definition at line 175 of file doctor.c.

6.27.2.8 doctor_search_by_phone()

```
void doctor_search_by_phone (
             void )
```

Searches for a doctor by phone number.
Definition at line 213 of file doctor.c.

6.27.2.9 doctor_search_id()

```
int doctor_search_id (
             int id)
```

Searches doctor by ID and returns index.

Parameters

id	The doctor ID to search for.

Returns

Index of doctor in array, or -1 if not found.

Definition at line 288 of file doctor.c.

6.27.2.10 doctor_update_availability()

```
void doctor_update_availability (
             const char * menu_items[],
             int index)
```

Definition at line 412 of file doctor.c.

6.27.2.11 doctor_update_email()

```
void doctor_update_email (
             const char * menu_items[],
             int index)
```

Definition at line 344 of file doctor.c.

6.27.2.12 doctor_update_name()

```
void doctor_update_name (
             const char * menu_items[],
             int index)
```

Definition at line 297 of file doctor.c.

6.27.2.13 doctor_update_phone()

```
void doctor_update_phone (
            const char * menu_items[ ],
            int index)
```

Definition at line 321 of file doctor.c.

6.27.2.14 doctor_update_room()

```
void doctor_update_room (
            const char * menu_items[ ],
            int index)
```

Definition at line 391 of file doctor.c.

6.27.2.15 doctor_update_specialization()

```
void doctor_update_specialization (
            const char * menu_items[ ],
            int index)
```

Definition at line 367 of file doctor.c.

6.27.2.16 doctor_update_status()

```
void doctor_update_status (
            const char * menu_items[ ],
            int index)
```

Definition at line 436 of file doctor.c.

6.27.2.17 doctor_update_using_id()

```
void doctor_update_using_id (
            void )
```

Updates a doctor information by ID.
Definition at line 459 of file doctor.c.

6.27.2.18 doctor_view()

```
void doctor_view (
            void )
```

Handles the view choice for doctor.
Definition at line 102 of file doctor.c.

6.27.2.19 doctor_view_all()

```
void doctor_view_all (
            void )
```

Displays all doctors in the system.
Definition at line 58 of file doctor.c.

6.27.2.20 doctor_view_discharged()

```
void doctor_view_discharged (
            void )
```

Displays all inactive doctors in the system.
Definition at line 623 of file doctor.c.

6.27.2.21 doctor_view_one()

```
void doctor_view_one (
            void )
```

Displays doctors one by one.
Definition at line 79 of file doctor.c.

6.28 doctor.c

Go to the documentation of this file.

```
00001
00007
00008 #include <stdio.h>
00009 #include <string.h>
00010 #include "../include/doctor.h"
00011 #include "../include/utils.h"
00012 #include "../include/ui.h"
00013 #include "../include/hospital.h"
00014
00015 int doctor_save_to_file(void) {
00016     FILE* file = fopen(DOCTORS_FILE, "wb");
00017     if (file == NULL) {
00018         return -1;
00019     }
00020     if (fwrite(&doctor_count, sizeof(int), 1, file) != 1 ||
00021         fwrite(&doctor_available, sizeof(int), 1, file) != 1 ||
00022         fwrite(doctors, sizeof(Doctor), doctor_count, file) != (size_t)doctor_count) {
00023         fclose(file);
00024         return -1;
00025     }
00026     fclose(file);
00027     return 0;
00028 }
00029
00030 int doctor_load_from_file(void) {
00031     FILE* file = fopen(DOCTORS_FILE, "rb");
00032     if (file == NULL) {
00033         return -1;
00034     }
00035     if (fread(&doctor_count, sizeof(int), 1, file) != 1 ||
00036         fread(&doctor_available, sizeof(int), 1, file) != 1) {
00037         fclose(file);
00038         return -1;
00039     }
00040     if (doctor_count < 0 || doctor_count > MAX_DOCTORS) {
00041         fclose(file);
00042         doctor_count = 0;
00043         return -1;
00044     }
00045     if (fread(doctors, sizeof(Doctor), doctor_count, file) != (size_t)doctor_count) {
00046         fclose(file);
00047         doctor_count = 0;
00048         return -1;
00049     }
00050     fclose(file);
00051     return 0;
00052 }
00053
00054 int doctor_generate_id(void) {
00055     return DOCTOR_ID_START + doctor_count;
00056 }
00057
00058 void doctor_view_all() {
00059
00060     int count = 0;
00061     ui_clear_screen();
00062     ui_print_banner();
00063
00064     if (doctor_available == 0 || doctor_count == 0) {
00065         const char* menu_items[] = {"No doctors found!"};
00066         ui_print_menu("View All Doctors", menu_items, 1, UI_SIZE);
00067         ui_pause();
00068         return;
00069     }
00070
00071     for (int i = 0; i < doctor_count; i++) {
00072         if (doctors[i].is_active) {
00073             ui_print_doctor(doctors[i], count++);
00074         }
00075     }
00076     ui_pause();
00077 }
00078
00079 void doctor_view_one() {
00080
00081     int count = 0;
00082
```

```
    if (doctor_available == 0) {
        ui_clear_screen();
        ui_print_banner();
        const char* menu_items[] = {"No doctors found!"};
        ui_print_menu("View All Doctors", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < doctor_count; i++) {
        ui_clear_screen();
        ui_print_banner();
        if (doctors[i].is_active) {
            ui_print_doctor(doctors[i], count++);
            ui_pause();
        }
    }
}

void doctor_view() {

    const char* menu_items[] =
    {
        "All at once",
        "One after another",
        "Back to Doctor Menu",
        "» "
    };

    int choice;
    do {
        ui_clear_screen();
        ui_print_banner();
        ui_print_menu("View Doctors", menu_items, 4, UI_SIZE);
        choice = utils_get_int();
        switch (choice) {
            case 1:
                doctor_view_all();
                break;
            case 2:
                doctor_view_one();
                break;
            case 3:
                ui_print_info("Returning to doctor menu...");
                ui_pause();
                return;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
                break;
        }
    } while (choice != 3);
}

void doctor_search_by_id(void) {

    int id;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter ID: ",
            "» "
        };

        ui_print_menu("Search Doctor", menu_items, 2, UI_SIZE);
        id = utils_get_int();

        if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }
        for (int i = 0; i < doctor_count; i++) {
            if (doctors[i].id == id) {
                ui_clear_screen();
                ui_print_banner();

                ui_print_doctor(doctors[i], (doctors[i].id - DOCTOR_ID_START));
                ui_pause();
                return;
            }
        }
        ui_print_error("Doctor not found!");
        ui_pause();
        return;
    } while (1);

}

void doctor_search_by_name(void) {
    char name[NAME_SIZE];

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter name: ",
            "» "
        };

        ui_print_menu("Search Doctor", menu_items, 2, UI_SIZE);
        utils_get_string(name, NAME_SIZE);

        if (!utils_is_valid_name(name)) {
            ui_print_error("Invalid name!");
            ui_pause();
            continue;
        }

        utils_fix_name(name);
        for (int i = 0; i < doctor_count; i++) {
            if (strcmp(doctors[i].name, name) == 0) {
                ui_clear_screen();
                ui_print_banner();

                ui_print_doctor(doctors[i], (doctors[i].id - DOCTOR_ID_START));
                ui_pause();
                return;
            }
        }
        ui_print_error("Doctor not found!");
        ui_pause();
        return;
    } while (1);
}

void doctor_search_by_phone(void) {
    char phone[PHONE_SIZE];

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter phone: ",
            "» "
        };

        ui_print_menu("Search Doctor", menu_items, 2, UI_SIZE);
        utils_get_string(phone, PHONE_SIZE);

        if (!utils_is_valid_phone(phone)) {
            ui_print_error("Invalid phone!");
            ui_pause();
            continue;
        }
        for (int i = 0; i < doctor_count; i++) {
            if (strcmp(doctors[i].phone, phone) == 0) {
                ui_clear_screen();
                ui_print_banner();

                ui_print_doctor(doctors[i], (doctors[i].id - DOCTOR_ID_START));
                ui_pause();
                return;
            }
        }
        ui_print_error("Doctor not found!");
        ui_pause();
        return;
    } while (1);
}

void doctor_search_by(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Search by doctor ID",
            "Search by doctor name",
            "Search by doctor phone",
            "Back to Doctor Menu",
            "» "
        };

        ui_print_menu("Search Doctor", menu_items, 5, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                doctor_search_by_id();
                break;
            case 2:
                doctor_search_by_name();
                break;
            case 3:
                doctor_search_by_phone();
                break;
            case 4:
                ui_print_info("Returning to doctor menu...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice! Please try again.");
                ui_pause();
        }
    } while (choice != 4);
}

int doctor_search_id (int id) {
    for (int i = 0; i < doctor_count; i++) {
        if (doctors[i].id == id) {
            return i;
        }
    }
    return -1;
}

void doctor_update_name(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[0],
        "Enter new name: ",
        "» "
    };

    char name[NAME_SIZE];
    ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
    utils_get_string(name, NAME_SIZE);
    utils_fix_name(name);

    if (!utils_is_valid_name(name)) {
        ui_print_error("Invalid name! Could not update.");
        ui_pause();
        return;
    }
    doctors[index].name[0] = '\0';
    strncpy(doctors[index].name, name, NAME_SIZE);
}

void doctor_update_phone(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[1],
        "Enter new phone: ",
        "» "
    };

    char phone[PHONE_SIZE];
    ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
    utils_get_string(phone, PHONE_SIZE);

    if (!utils_is_valid_phone(phone)) {
        ui_print_error("Invalid phone! Could not update.");
        ui_pause();
        return;
    }
    doctors[index].phone[0] = '\0';
    strncpy(doctors[index].phone, phone, PHONE_SIZE);
}

```

```
void doctor_update_email(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[2],
        "Enter new email: ",
        "» "
    };

    char email[EMAIL_SIZE];
    ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
    utils_get_string(email, EMAIL_SIZE);

    if (!utils_is_valid_email(email)) {
        ui_print_error("Invalid email! Could not update.");
        ui_pause();
        return;
    }
    doctors[index].email[0] = '\0';
    strncpy(doctors[index].email, email, EMAIL_SIZE);
}

void doctor_update_specialization(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[3],
        "Enter new specialization: ",
        "» "
    };

    char spec[SPEC_SIZE];
    ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
    utils_get_string(spec, SPEC_SIZE);

    if (strlen(spec) == 0) {
        ui_print_error("Specialization cannot be empty! Could not update.");
        ui_pause();
        return;
    }
    utils_fix_name(spec);
    doctors[index].specialization[0] = '\0';
    strncpy(doctors[index].specialization, spec, SPEC_SIZE);
}

void doctor_update_room(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[4],
        "Enter new room number: ",
        "» "
    };

    ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
    int room = utils_get_int();

    if (room <= 0 || room >= 1000) {
        ui_print_error("Invalid room number! Could not update.");
        ui_pause();
        return;
    }
    doctors[index].room_number = room;
}

void doctor_update_availability(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        "Set availability (1 for available, 2 for unavailable): ",
        menu_items[5],
        "» "
    };

    int input;
    ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
    input = utils_get_int();

    if (input == 1) {
        doctors[index].is_available = true;
    } else if (input == 2) {
        doctors[index].is_available = false;
    } else {
        ui_print_error("Invalid input! Could not update.");
        ui_pause();
    }
}

void doctor_update_status(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[6],
        "Enter new status: (1 for active, 2 for inactive) ",
        "» "
    };

    bool status;
    int input;
    ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
    input = utils_get_int();

    if (input == 1) {
        status = true;
    } else if (input == 2) {
        status = false;
    }
    doctors[index].is_active = status;
}

void doctor_update_using_id() {
    ui_clear_screen();
    ui_print_banner();

    int id;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* enter_id_items[] = {
            "Enter doctor ID: (0 to go back) ",
            "» "
        };

        ui_print_menu("Update Doctor", enter_id_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) return;

        if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
            ui_print_error("Invalid ID!");
            ui_pause();
        } else {
            ui_print_info("Doctor found!");
            ui_pause();
            break;
        }
    } while (1);

    int index = doctor_search_id(id);
    if (index == -1) return;

    char name_line[NAME_LINE_SIZE], phone_line[PHONE_LINE_SIZE], email_line[70], spec_line[70],
room_line[70], avail_line[70], status_line[STATUS_LINE_SIZE];

    snprintf(name_line, NAME_LINE_SIZE, "Name: %s", doctors[index].name);
    snprintf(phone_line, PHONE_LINE_SIZE, "Phone: %s", doctors[index].phone);
    snprintf(email_line, sizeof(email_line), "Email: %s", doctors[index].email);
    snprintf(spec_line, sizeof(spec_line), "Specialization: %s", doctors[index].specialization);
    snprintf(room_line, sizeof(room_line), "Room Number: %d", doctors[index].room_number);

    int doctor_id = doctors[index].id;

    if (doctors[index].is_available) {
        snprintf(avail_line, sizeof(avail_line), "Availability: Available");
    } else {
        snprintf(avail_line, sizeof(avail_line), "Availability: Unavailable");
    }
    if (doctors[index].is_active) {
        snprintf(status_line, STATUS_LINE_SIZE, "Status: Active");
    } else {
        snprintf(status_line, STATUS_LINE_SIZE, "Status: Inactive");
    }

    const char* menu_items[] = {
        name_line,
        phone_line,
        email_line,
        spec_line,
        room_line,
        avail_line,
        "Go back",
        "» "
    };

    char title[50];
    snprintf(title, 50, "Update Doctor | ID: %d", doctor_id);

    ui_clear_screen();
    ui_print_banner();
    ui_print_menu(title, menu_items, 8, UI_SIZE);

    int choice = utils_get_int();

    switch (choice) {
        case 1:
            doctor_update_name(menu_items, index);
            break;
        case 2:
            doctor_update_phone(menu_items, index);
            break;
        case 3:
            doctor_update_email(menu_items, index);
            break;
        case 4:
            doctor_update_specialization(menu_items, index);
            break;
        case 5:
            doctor_update_room(menu_items, index);
            break;
        case 6:
            doctor_update_availability(menu_items, index);
            break;
        case 7:
            ui_print_info("Returning to doctor menu...");
            ui_pause();
            break;
        default:
            ui_print_error("Invalid choice! Please try again.");
            ui_pause();
            break;
    }
}

void doctor_deactivate_account() {
    int id;
    while (1) {
        ui_clear_screen();
        ui_print_banner();

        const char* enter_id_items[] = {
            "Enter doctor ID (0 to cancel): ",
            "» "
        };

        ui_print_menu("Deactivate Doctor", enter_id_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) {
            ui_print_info("User Pressed 0. \nCanceling deactivation...");
            ui_pause();
            return;
        }

        if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }
        break;
    }

    int index = doctor_search_id(id);
    if (index == -1) {
        ui_print_error("Doctor not found!");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();
    ui_print_doctor(doctors[index], index);

    const char* menu[] = {
        "Confirm Deactivation",
        "Cancel",
        "» "
    };

    ui_print_menu("Deactivate Doctor", menu, 3, UI_SIZE);
    int input = utils_get_int();

    if (input == 1) {
        doctors[index].is_active = false;
        ui_print_success("Doctor deactivated successfully!");
        doctor_available--;
        doctor_unavailable++;
        ui_pause();
    } else {
        ui_print_info("Deactivation cancelled.");
        ui_pause();
    }
}

void doctor_view_discharged(void) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    if (doctor_unavailable == 0) {
        const char* menu_items[] = {"No inactive doctors found!"};
        ui_print_menu("Inactive Doctors", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < doctor_count; i++) {
        if (!doctors[i].is_active) {
            ui_print_doctor(doctors[i], count++);
        }
    }
    ui_pause();
}
```

6.29 src/doctor_portal.c File Reference

Doctor portal implementation for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include "../include/doctor_portal.h"
#include "../include/appointment.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
```

Functions

- void doctor_portal_view_appointments (int doctor_id)
- void doctor_portal_view_pending (int doctor_id)
- void doctor_portal_view_today (int doctor_id, const char *today_date)
- void doctor_portal_view_patient (void)
- void doctor_portal_complete_appointment (int doctor_id)
- void doctor_portal_cancel_appointment (int doctor_id)
- void doctor_portal_update_availability (int doctor_id)
- void doctor_portal_view_profile (int doctor_id)
- void doctor_portal_menu (int doctor_id, const char *doctor_name)

6.29.1 Detailed Description

Doctor portal implementation for Healthcare Management System.

This file contains the doctor's personal portal with appointment management and profile features.

Definition in file doctor_portal.c.

6.29.2 Function Documentation

6.29.2.1 doctor_portal_cancel_appointment()

```
void doctor_portal_cancel_appointment (
             int doctor_id)
```

Cancels an appointment.

Parameters

doctor←↩ _id	The doctor's ID.

Definition at line 146 of file doctor_portal.c.

6.29.2.2 doctor_portal_complete_appointment()

```
void doctor_portal_complete_appointment (
             int doctor_id)
```

Marks an appointment as completed.

Parameters

doctor←↩ _id	The doctor's ID.

Definition at line 95 of file doctor_portal.c.

6.29.2.3 doctor_portal_menu()

```
void doctor_portal_menu (
             int doctor_id,
             const char * doctor_name)
```

Main doctor portal menu.

Parameters

doctor_id	The doctor's ID.
doctor_name	The doctor's name for display.

Definition at line 256 of file doctor_portal.c.

6.29.2.4 doctor_portal_update_availability()

```
void doctor_portal_update_availability (
             int doctor_id)
```

Updates doctor's availability status.

Parameters

doctor←↩ _id	The doctor's ID.

Definition at line 196 of file doctor_portal.c.

6.29.2.5 doctor_portal_view_appointments()

```
void doctor_portal_view_appointments (
              int doctor_id)
```

Views all appointments for current doctor.

Parameters

doctor↩_id	The doctor's ID.

Definition at line 19 of file doctor_portal.c.

6.29.2.6 doctor_portal_view_patient()

```
void doctor_portal_view_patient (
              void )
```

Views patient details for a given patient ID.
Definition at line 62 of file doctor_portal.c.

6.29.2.7 doctor_portal_view_pending()

```
void doctor_portal_view_pending (
              int doctor_id)
```

Views pending appointments for current doctor.

Parameters

doctor↩_id	The doctor's ID.

Definition at line 23 of file doctor_portal.c.

6.29.2.8 doctor_portal_view_profile()

```
void doctor_portal_view_profile (
              int doctor_id)
```

Views doctor's own profile.

Parameters

doctor↩_id	The doctor's ID.

Definition at line 242 of file doctor_portal.c.

6.29.2.9 doctor_portal_view_today()

```
void doctor_portal_view_today (
              int doctor_id,
              const char * today_date)
```

Views today's appointments for current doctor.

Parameters

doctor_id	The doctor's ID.
today_date	Today's date string.

Definition at line 42 of file doctor_portal.c.

6.30 doctor_portal.c

Go to the documentation of this file.

```


#include <stdio.h>
#include <string.h>
#include "../include/doctor_portal.h"
#include "../include/appointment.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"

void doctor_portal_view_appointments(int doctor_id) {
    appointment_view_by_doctor(doctor_id);
}

void doctor_portal_view_pending(int doctor_id) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    for (int i = 0; i < appointment_count; i++) {
        if (appointments[i].doctor_id == doctor_id &&
            appointments[i].status == APPT_PENDING) {
            ui_print_appointment(appointments[i], count++);
        }
    }

    if (count == 0) {
        const char* menu_items[] = {"No pending appointments!"};
        ui_print_menu("Pending Appointments", menu_items, 1, UI_SIZE);
    }
    ui_pause();
}

void doctor_portal_view_today(int doctor_id, const char* today_date) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    for (int i = 0; i < appointment_count; i++) {
        if (appointments[i].doctor_id == doctor_id &&
            strcmp(appointments[i].date, today_date) == 0 &&
            appointments[i].status != APPT_CANCELLED) {
            ui_print_appointment(appointments[i], count++);
        }
    }

    if (count == 0) {
        const char* menu_items[] = {"No appointments for today!"};
        ui_print_menu("Today's Appointments", menu_items, 1, UI_SIZE);
    }
    ui_pause();
}

void doctor_portal_view_patient(void) {
    int patient_id;

    ui_clear_screen();
    ui_print_banner();

    const char* menu_items[] = {
        "Enter Patient ID:",
        "» "
    };

    ui_print_menu("View Patient Details", menu_items, 2, UI_SIZE);
    patient_id = utils_get_int();

    if (!utils_is_valid_id(patient_id, ROLE_PATIENT)) {
        ui_print_error("Invalid patient ID!");
        ui_pause();
        return;
    }

    int idx = patient_search_id(patient_id);
    if (idx == -1) {
        ui_print_error("Patient not found!");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();
    ui_print_patient(patients[idx], idx);
    ui_pause();
}

void doctor_portal_complete_appointment(int doctor_id) {
    int appt_id;

    // First show pending appointments
    ui_clear_screen();
    ui_print_banner();

    int count = 0;
    for (int i = 0; i < appointment_count; i++) {
        if (appointments[i].doctor_id == doctor_id &&
            (appointments[i].status == APPT_PENDING ||
             appointments[i].status == APPT_CONFIRMED)) {
            ui_print_appointment(appointments[i], count++);
        }
    }

    if (count == 0) {
        const char* menu_items[] = {"No appointments to complete!"};
        ui_print_menu("Complete Appointment", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    const char* input_items[] = {
        "Enter Appointment ID to mark as completed (0 to cancel):",
        "» "
    };
    ui_print_menu("Complete Appointment", input_items, 2, UI_SIZE);
    appt_id = utils_get_int();

    if (appt_id == 0) return;

    int idx = appointment_search_id(appt_id);
    if (idx == -1) {
        ui_print_error("Appointment not found!");
        ui_pause();
        return;
    }

    if (appointments[idx].doctor_id != doctor_id) {
        ui_print_error("This appointment is not assigned to you!");
        ui_pause();
        return;
    }

    appointments[idx].status = APPT_COMPLETED;
    appointment_save_to_file();
    ui_print_success("Appointment marked as completed!");
    ui_pause();
}

void doctor_portal_cancel_appointment(int doctor_id) {
    int appt_id;

    // First show pending appointments
    ui_clear_screen();
    ui_print_banner();

    int count = 0;
    for (int i = 0; i < appointment_count; i++) {
        if (appointments[i].doctor_id == doctor_id &&
            appointments[i].status == APPT_PENDING) {
            ui_print_appointment(appointments[i], count++);
        }
    }

    if (count == 0) {
        const char* menu_items[] = {"No appointments to cancel!"};
        ui_print_menu("Cancel Appointment", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    const char* input_items[] = {
        "Enter Appointment ID to cancel (0 to go back):",
        "» "
    };
    ui_print_menu("Cancel Appointment", input_items, 2, UI_SIZE);
    appt_id = utils_get_int();

    if (appt_id == 0) return;

    int idx = appointment_search_id(appt_id);
    if (idx == -1) {
        ui_print_error("Appointment not found!");
        ui_pause();
        return;
    }

    if (appointments[idx].doctor_id != doctor_id) {
        ui_print_error("This appointment is not assigned to you!");
        ui_pause();
        return;
    }

    appointments[idx].status = APPT_CANCELLED;
    appointment_save_to_file();
    ui_print_success("Appointment cancelled!");
    ui_pause();
}

void doctor_portal_update_availability(int doctor_id) {
    int idx = doctor_search_id(doctor_id);
    if (idx == -1) {
        ui_print_error("Doctor not found!");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();

    char current_status[50];
    snprintf(current_status, sizeof(current_status), "Current Status: %s",
             doctors[idx].is_available ? "Available" : "Unavailable");

    const char* menu_items[] = {
        "Set availability (1 for available, 2 for unavailable): ",
        current_status,
        "Back",
        "» "
    };

    ui_print_menu("Update Availability", menu_items, 4, UI_SIZE);
    int choice = utils_get_int();

    switch (choice) {
        case 1:
            doctors[idx].is_available = true;
            doctor_save_to_file();
            ui_print_success("Status set to Available!");
            ui_pause();
            break;
        case 2:
            doctors[idx].is_available = false;
            doctor_save_to_file();
            ui_print_success("Status set to Unavailable!");
            ui_pause();
            break;
        case 3:
            return;
        default:
            ui_print_error("Invalid choice!");
            ui_pause();
    }
}

void doctor_portal_view_profile(int doctor_id) {
    int idx = doctor_search_id(doctor_id);
    if (idx == -1) {
        ui_print_error("Doctor not found!");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();
    ui_print_doctor(doctors[idx], idx);
    ui_pause();
}

void doctor_portal_menu(int doctor_id, const char* doctor_name) {
    int choice;

    // Build title with doctor's name
    char title[80];
    snprintf(title, sizeof(title), "Doctor Portal | Dr. %s", doctor_name);

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "View All My Appointments",
            "View Pending Appointments",
            "View Patient Details",
            "Complete Appointment",
            "Cancel Appointment",
            "Update My Availability",
            "View My Profile",
            "Logout",
            "» "
        };

        ui_print_menu(title, menu_items, 9, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                doctor_portal_view_appointments(doctor_id);
                break;
            case 2:
                doctor_portal_view_pending(doctor_id);
                break;
            case 3:
                doctor_portal_view_patient();
                break;
            case 4:
                doctor_portal_complete_appointment(doctor_id);
                break;
            case 5:
                doctor_portal_cancel_appointment(doctor_id);
                break;
            case 6:
                doctor_portal_update_availability(doctor_id);
                break;
            case 7:
                doctor_portal_view_profile(doctor_id);
                break;
            case 8:
                ui_print_info("Logging out...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 8);
}
```

6.31 src/hospital.c File Reference

Global data definitions for Healthcare Management System.

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include "../include/hospital.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/appointment.h"
#include "../include/receptionist.h"
#include "../include/auth.h"
#include "../include/ui.h"
#include "../include/utils.h"
```

Functions

- void hospital_init (void)
- void show_about (void)
- void print_help (const char ∗program_name)
- void print_version (void)
- void ensure_data_dir (void)

Variables

- Patient patients [100]
- Doctor doctors [20]
- Receptionist receptionists [20]
- User users [50]
- Appointment appointments [200]
- int patient_count = 0
- int patient_available = 0
- int patient_unavailable = 0
- int doctor_count = 0
- int doctor_available = 0
- int doctor_unavailable = 0
- int receptionist_count = 0
- int receptionist_available = 0
- int receptionist_unavailable = 0
- int user_count = 0
- int appointment_count = 0
- User ∗ current_user = NULL

6.31.1 Detailed Description

Global data definitions for Healthcare Management System.
This file contains the actual definitions of global variables declared with extern in hospital.h.
Definition in file hospital.c.

6.31.2 Function Documentation

6.31.2.1 ensure_data_dir()

```
void ensure_data_dir (
             void )
```

Ensure data directory exists.
Definition at line 92 of file hospital.c.

6.31.2.2 hospital_init()

```
void hospital_init (
             void )
```

Initialize the hospital system by loading all data.
Definition at line 47 of file hospital.c.

6.31.2.3 print_help()

```
void print_help (
             const char * program_name)
```

Display help information.
Definition at line 73 of file hospital.c.

6.31.2.4 print_version()

```
void print_version (
            void )
```
Display version information.
Definition at line 86 of file hospital.c.

6.31.2.5 show_about()

```
void show_about (
            void )
```
Display about information.
Definition at line 55 of file hospital.c.

6.31.3 Variable Documentation

6.31.3.1 appointment_count

```
int appointment_count = 0
```
Definition at line 43 of file hospital.c.

6.31.3.2 appointments

```
Appointment appointments[200]
```
Definition at line 31 of file hospital.c.

6.31.3.3 current_user

```
User* current_user = NULL
```
Definition at line 45 of file hospital.c.

6.31.3.4 doctor_available

```
int doctor_available = 0
```
Definition at line 37 of file hospital.c.

6.31.3.5 doctor_count

```
int doctor_count = 0
```
Definition at line 36 of file hospital.c.

6.31.3.6 doctor_unavailable

```
int doctor_unavailable = 0
```
Definition at line 38 of file hospital.c.

6.31.3.7 doctors

```
Doctor doctors[20]
```
Definition at line 28 of file hospital.c.

6.31.3.8 patient_available

```
int patient_available = 0
```
Definition at line 34 of file hospital.c.

6.31.3.9 patient_count

```
int patient_count = 0
```
Definition at line 33 of file hospital.c.

6.31.3.10 patient_unavailable

`int patient_unavailable = 0`

Definition at line 35 of file hospital.c.

6.31.3.11 patients

`Patient patients[100]`

Definition at line 27 of file hospital.c.

6.31.3.12 receptionist_available

`int receptionist_available = 0`

Definition at line 40 of file hospital.c.

6.31.3.13 receptionist_count

`int receptionist_count = 0`

Definition at line 39 of file hospital.c.

6.31.3.14 receptionist_unavailable

`int receptionist_unavailable = 0`

Definition at line 41 of file hospital.c.

6.31.3.15 receptionists

`Receptionist receptionists[20]`

Definition at line 29 of file hospital.c.

6.31.3.16 user_count

`int user_count = 0`

Definition at line 42 of file hospital.c.

6.31.3.17 users

`User users[50]`

Definition at line 30 of file hospital.c.

6.32 hospital.c

Go to the documentation of this file.

```
00001
00008 #include <stddef.h>
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011
00012 #ifdef _WIN32
00013     #include <direct.h>
00014 #else
00015     #include <sys/stat.h>
00016 #endif
00017
00018 #include "../include/hospital.h"
00019 #include "../include/patient.h"
00020 #include "../include/doctor.h"
00021 #include "../include/appointment.h"
00022 #include "../include/receptionist.h"
00023 #include "../include/auth.h"
00024 #include "../include/ui.h"
00025 #include "../include/utils.h"
00026
00027 Patient patients[MAX_PATIENTS];
00028 Doctor doctors[MAX_DOCTORS];
00029 Receptionist receptionists[MAX_RECEPTIONISTS];
00030 User users[MAX_USERS];
```

```
Appointment appointments[MAX_APPOINTMENTS];

int patient_count = 0;
int patient_available = 0;
int patient_unavailable = 0;
int doctor_count = 0;
int doctor_available = 0;
int doctor_unavailable = 0;
int receptionist_count = 0;
int receptionist_available = 0;
int receptionist_unavailable = 0;
int user_count = 0;
int appointment_count = 0;

User* current_user = NULL;

void hospital_init(void) {
    patient_load_from_file();
    doctor_load_from_file();
    appointment_load_from_file();
    auth_load_from_file();
    auth_init_default_admin();
}

void show_about(void) {
    ui_clear_screen();
    ui_print_banner();

    const char* about_items[] = {
        "Healthcare Management System (HMS)",
        "Version: 1.0",
        "Developed for efficient hospital management",
        "including patient, doctor, and appointment",
        "tracking with role-based access control.",
        "Roles: Admin, Receptionist, Doctor",
        ""
    };

    ui_print_menu("About HMS", about_items, 7, UI_SIZE);
    ui_pause();
}

void print_help(const char* program_name) {
    printf("\n" BOLD BRIGHT_RED "Healthcare Management System" RESET BOLD SOFT_GREEN " v%s" RESET
"\n\n", VERSION);
    printf(SOFT_YELLOW "Usage:" RESET " %s [option] or .\\hms.exe [option] for windows\n",
program_name);
    printf(SOFT_YELLOW "Usage:" RESET " ./hms.out [option] for linux\n\n");
    printf(SOFT_YELLOW "Options:" RESET "\n");
    printf("  -h, --help      Show this help message\n");
    printf("  -v, --version   Show version information\n");
    printf("  -a, --about     Show about information\n");
    printf("  -l, --login     Go directly to login menu\n");
    printf("\n");
    printf("If no options are provided, the interactive menu will start.\n\n");
}

void print_version(void) {
    printf("\n" BOLD "Healthcare Management System" RESET "\n");
    printf("Version: " SOFT_GREEN "%s" RESET "\n", VERSION);
    printf("Built with C\n\n");
}

void ensure_data_dir(void) {
    #ifdef _WIN32
        _mkdir("data");
    #else
        mkdir("data", 0755);
    #endif
}
```

6.33 src/patient.c File Reference

Patient management implementation for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include "../include/patient.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
```

Functions

- int patient_save_to_file (void)
- int patient_load_from_file (void)
- int patient_generate_id (void)
- void patient_add (void)
- void patient_view_all ()
- void patient_view_one ()
- void patient_view ()
- void patient_search_by_id (void)
- void patient_search_by_name (void)
- void patient_search_by_phone (void)
- void patient_search_by (void)
- int patient_search_id (int id)
- void patient_update_name (const char ∗menu_items[ ], int index)
- void patient_update_phone (const char ∗menu_items[ ], int index)
- void patient_update_address (const char ∗menu_items[ ], int index)
- void patient_update_blood_group (const char ∗menu_items[ ], int index)
- void patient_update_gender (const char ∗menu_items[ ], int index)
- void patient_update_status (const char ∗menu_items[ ], int index)
- void patient_update_using_id ()
- void patient_discharge ()
- void patient_view_discharged (void)

6.33.1 Detailed Description

Patient management implementation for Healthcare Management System.
This file contains the implementation of all patient CRUD operations.
Definition in file patient.c.

6.33.2 Function Documentation

6.33.2.1 patient_add()

```
void patient_add (
             void )
```

Adds a new patient to the system.
Definition at line 58 of file patient.c.

6.33.2.2 patient_discharge()

```
void patient_discharge (
             void )
```

Discharges a patient by ID (sets is_active to false).
Definition at line 693 of file patient.c.

6.33.2.3 patient_generate_id()

```
int patient_generate_id (
             void )
```

Generates a unique patient ID.

Returns

The generated patient ID.

Definition at line 54 of file patient.c.

6.33.2.4 patient_load_from_file()

```
int patient_load_from_file (
            void )
```

Loads all patients from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 30 of file patient.c.

6.33.2.5 patient_save_to_file()

```
int patient_save_to_file (
            void )
```

Saves all patients to binary file.

Returns

0 on success, -1 on failure.

Definition at line 15 of file patient.c.

6.33.2.6 patient_search_by()

```
void patient_search_by (
            void )
```

Handles the search choice for patient.
Definition at line 402 of file patient.c.

6.33.2.7 patient_search_by_id()

```
void patient_search_by_id (
            void )
```

Searches for a patient by ID.
Definition at line 290 of file patient.c.

6.33.2.8 patient_search_by_name()

```
void patient_search_by_name (
            void )
```

Searches for a patient by name.
Definition at line 328 of file patient.c.

6.33.2.9 patient_search_by_phone()

```
void patient_search_by_phone (
            void )
```

Searches for a patient by phone number.
Definition at line 366 of file patient.c.

6.33.2.10 patient_search_id()

```
int patient_search_id (
            int id)
```

Searches patient by ID and returns index.

Parameters

id	The patient ID to search for.

Returns

Index of patient in array, or -1 if not found.

Definition at line 441 of file patient.c.

6.33.2.11 patient_update_address()

```
void patient_update_address (
            const char * menu_items[ ],
            int index)
```

Definition at line 497 of file patient.c.

6.33.2.12 patient_update_blood_group()

```
void patient_update_blood_group (
            const char * menu_items[ ],
            int index)
```

Definition at line 520 of file patient.c.

6.33.2.13 patient_update_gender()

```
void patient_update_gender (
            const char * menu_items[ ],
            int index)
```

Definition at line 544 of file patient.c.

6.33.2.14 patient_update_name()

```
void patient_update_name (
            const char * menu_items[ ],
            int index)
```

Definition at line 450 of file patient.c.

6.33.2.15 patient_update_phone()

```
void patient_update_phone (
            const char * menu_items[ ],
            int index)
```

Definition at line 474 of file patient.c.

6.33.2.16 patient_update_status()

```
void patient_update_status (
            const char * menu_items[ ],
            int index)
```

Definition at line 571 of file patient.c.

6.33.2.17 patient_update_using_id()

```
void patient_update_using_id (
            void )
```

Updates a patient information by ID.
Definition at line 594 of file patient.c.

6.33.2.18 patient_view()

```
void patient_view ()
```

Definition at line 255 of file patient.c.

6.33.2.19 patient_view_all()

```
void patient_view_all (
              void )
```

Displays all patients in the system.

Definition at line 211 of file patient.c.

6.33.2.20 patient_view_discharged()

```
void patient_view_discharged (
              void )
```

Displays all discharged patients in the system.

Definition at line 753 of file patient.c.

6.33.2.21 patient_view_one()

```
void patient_view_one ()
```

Definition at line 232 of file patient.c.

6.34 patient.c

Go to the documentation of this file.

```


#include <stdio.h>
#include <string.h>
#include "../include/patient.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"

int patient_save_to_file(void) {
    FILE* file = fopen(PATIENTS_FILE, "wb");
    if (file == NULL) {
        return -1;
    }
    if (fwrite(&patient_count, sizeof(int), 1, file) != 1 ||
        fwrite(&patient_available, sizeof(int), 1, file) != 1 ||
        fwrite(patients, sizeof(Patient), patient_count, file) != (size_t)patient_count) {
        fclose(file);
        return -1;
    }
    fclose(file);
    return 0;
}

int patient_load_from_file(void) {
    FILE* file = fopen(PATIENTS_FILE, "rb");
    if (file == NULL) {
        return -1;
    }
    if (fread(&patient_count, sizeof(int), 1, file) != 1 ||
        fread(&patient_available, sizeof(int), 1, file) != 1) {
        fclose(file);
        return -1;
    }
    if (patient_count < 0 || patient_count > MAX_PATIENTS) {
        fclose(file);
        patient_count = 0;
        return -1;
    }
    if (fread(patients, sizeof(Patient), patient_count, file) != (size_t)patient_count) {
        fclose(file);
        patient_count = 0;
        return -1;
    }
    fclose(file);
    return 0;
}

int patient_generate_id(void) {
    return PATIENT_ID_START + patient_count;
}

void patient_add(void) {
    if (patient_count >= MAX_PATIENTS) {
        ui_print_error("Error: Maximum patient limit reached!");
        ui_pause();
        return;
    }

    // Create new patient
    Patient new_patient;
    new_patient.id = patient_generate_id();
    new_patient.is_active = true;

    // lines for menu
    char name_line[NAME_LINE_SIZE], age_line[AGE_LINE_SIZE], gender_line[GENDER_LINE_SIZE],
  phone_line[PHONE_LINE_SIZE], address_line[ADDRESS_LINE_SIZE], blood_group_line[BLOOD_LINE_SIZE];

    // Step 1: Get Name
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        const char* step1[] = {"Name:", "» "};
        ui_print_menu("Add Patient", step1, 2, UI_SIZE);
        utils_get_string(new_patient.name, NAME_SIZE);
        if (utils_is_valid_name(new_patient.name)) {
            utils_fix_name(new_patient.name);
            break;
        }
        ui_print_error("Invalid name! Please try again.");
        ui_pause();
    }

    // Step 2: Get Age
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        snprintf(name_line, sizeof(name_line), "Name: %s", new_patient.name);
        const char* step2[] = {name_line, "Age:", "» "};
        ui_print_menu("Add Patient", step2, 3, UI_SIZE);
        new_patient.age = utils_get_int();
        if (new_patient.age > 0 && new_patient.age < 120) {
            break;
        }
        ui_print_error("Invalid age! Please try again.");
        ui_pause();
    }

    // Step 3: Get Gender
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        snprintf(age_line, sizeof(age_line), "Age: %d", new_patient.age);
        const char* step3[] = {name_line, age_line, "Gender (M/F):", "» "};
        ui_print_menu("Add Patient", step3, 4, UI_SIZE);
        char gender_input = utils_get_char();
        if (gender_input == 'M' || gender_input == 'm') {
            new_patient.gender = MALE;
            break;
        } else if (gender_input == 'F' || gender_input == 'f') {
            new_patient.gender = FEMALE;
            break;
        }
        ui_print_error("Invalid gender! Please try again.");
        ui_pause();
    }

    // Step 4: Get Phone
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        snprintf(gender_line, sizeof(gender_line), "Gender: %s", (new_patient.gender == MALE) ? "Male"
  : "Female");
        const char* step4[] = {name_line, age_line, gender_line, "Phone (11 digits):", "» "};
        ui_print_menu("Add Patient", step4, 5, UI_SIZE);
        utils_get_string(new_patient.phone, PHONE_SIZE);
        if (utils_is_valid_phone(new_patient.phone)) {
            break;
        }
        ui_print_error("Invalid phone! Please try again.");
        ui_pause();
    }

    // Step 5: Get Address
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        snprintf(phone_line, sizeof(phone_line), "Phone: %s", new_patient.phone);
        const char* step5[] = {name_line, age_line, gender_line, phone_line, "Address:", "» "};
        ui_print_menu("Add Patient", step5, 6, UI_SIZE);
        utils_get_string(new_patient.address, ADDRESS_SIZE);
        if (utils_is_valid_address(new_patient.address)) {
            break;
        }
        ui_print_error("Invalid address! Please try again.");
        ui_pause();
    }

    // Step 6: Get Blood Group
    while (1) {
        ui_clear_screen();
        ui_print_banner();
        snprintf(address_line, sizeof(address_line), "Address: %s", new_patient.address);
        const char* step6[] =
        {
            name_line, age_line, gender_line, phone_line, address_line,
            "Blood Group (e.g. A+, O-, or U for Unknown):", "» "
        };
        ui_print_menu("Add Patient", step6, 7, UI_SIZE);
        utils_get_string(new_patient.blood_group, BLOOD_SIZE);
        if (utils_is_valid_blood_group(new_patient.blood_group)) {
            utils_str_to_upper(new_patient.blood_group);
            break;
        }
        ui_print_error("Invalid blood group! Please try again.");
        ui_pause();
    }

    // Step 7: Confirm
    snprintf(blood_group_line, sizeof(blood_group_line), "Blood Group: %s", new_patient.blood_group);
    const char* step7[] =
    {
        name_line, age_line, gender_line, phone_line, address_line, blood_group_line,
        "Confirm (Y/N):", "» "
    };
    ui_clear_screen();
    ui_print_banner();
    ui_print_menu("Add Patient", step7, 8, UI_SIZE);
    char confirm = utils_get_char();
    if (confirm != 'Y' && confirm != 'y') {
        ui_print_info("Patient addition cancelled.");
        ui_pause();
        return;
    }

    // Add to array
    patients[patient_count] = new_patient;
    patient_count++;
    patient_available++;

    ui_clear_screen();
    ui_print_banner();

    char id_line[70];
    snprintf(id_line, sizeof(id_line), "Patient ID: %d", new_patient.id);
    const char* success_items[] =
    {
        id_line, name_line, age_line,
        gender_line, phone_line,
        address_line, blood_group_line,
        "Patient added successfully!"
    };
    ui_print_menu("Patient Added", success_items, 8, UI_SIZE);
    ui_pause();

}

void patient_view_all() {

    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    if (patient_available == 0 || patient_count == 0) {
        const char* menu_items[] = {"No patients found!"};
        ui_print_menu("View All Patients", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < patient_count; i++) {
        if (patients[i].is_active) {
            ui_print_patient(patients[i], count++);
        }
    }
    ui_pause();
}

```

```c
void patient_view_one() {

    int count = 0;

    if (patient_available == 0) {
        ui_clear_screen();
        ui_print_banner();
        const char* menu_items[] = {"No patients found!"};
        ui_print_menu("View All Patients", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < patient_count; i++) {
        ui_clear_screen();
        ui_print_banner();
        if (patients[i].is_active) {
            ui_print_patient(patients[i], count++);
            ui_pause();
        }
    }
}

void patient_view() {

    const char* menu_items[] =
    {
        "All at once",
        "One after another",
        "Back to Patient Menu",
        "» "
    };

    int choice;
    do {
        ui_clear_screen();
        ui_print_banner();
        ui_print_menu("View Patients", menu_items, 4, UI_SIZE);
        choice = utils_get_int();
        switch (choice) {
            case 1:
                patient_view_all();
                break;
            case 2:
                patient_view_one();
                break;
            case 3:
                ui_print_info("Returning to receptionist menu...");
                ui_pause();
                return;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
                break;
        }
    } while (choice != 3);
}

void patient_search_by_id(void) {

    int id;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter ID: ",
            "» "
        };

        ui_print_menu("Search Patient", menu_items, 2, UI_SIZE);
        id = utils_get_int();

        if (!utils_is_valid_id(id, ROLE_PATIENT)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }
        for (int i = 0; i < patient_count; i++) {
            if (patients[i].id == id) {
                ui_clear_screen();
                ui_print_banner();

                ui_print_patient(patients[i], (patients[i].id - 1001));
                ui_pause();
                return;
            }
        }
        ui_print_error("Patient not found!");
        ui_pause();
        return;
    } while (1);

}

void patient_search_by_name(void) {
    char name[NAME_SIZE];

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter name: ",
            "» "
        };

        ui_print_menu("Search Patient", menu_items, 2, UI_SIZE);
        utils_get_string(name, NAME_SIZE);

        if (!utils_is_valid_name(name)) {
            ui_print_error("Invalid name!");
            ui_pause();
            continue;
        }

        utils_fix_name(name);
        for (int i = 0; i < patient_count; i++) {
            if (strcmp(patients[i].name, name) == 0) {
                ui_clear_screen();
                ui_print_banner();

                ui_print_patient(patients[i], (patients[i].id - 1001));
                ui_pause();
                return;
            }
        }
        ui_print_error("Patient not found!");
        ui_pause();
        return;
    } while (1);
}

void patient_search_by_phone(void) {
    char phone[PHONE_SIZE];

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Enter phone: ",
            "» "
        };

        ui_print_menu("Search Patient", menu_items, 2, UI_SIZE);
        utils_get_string(phone, PHONE_SIZE);

        if (!utils_is_valid_phone(phone)) {
            ui_print_error("Invalid phone!");
            ui_pause();
            continue;
        }
        for (int i = 0; i < patient_count; i++) {
            if (strcmp(patients[i].phone, phone) == 0) {
                ui_clear_screen();
                ui_print_banner();

                ui_print_patient(patients[i], (patients[i].id - 1001));
                ui_pause();
                return;
            }
        }
        ui_print_error("Patient not found!");
        ui_pause();
        return;
    } while (1);
}

void patient_search_by(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Search by patient ID",
            "Search by patient name",
            "Search by patient phone",
            "Back to Patient Menu",
            "» "
        };

        ui_print_menu("Search Patient", menu_items, 5, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                patient_search_by_id();
                break;
            case 2:
                patient_search_by_name();
                break;
            case 3:
                patient_search_by_phone();
                break;
            case 4:
                ui_print_info("Returning to receptionist menu...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice! Please try again.");
                ui_pause();
        }
    } while (choice != 4);
}

int patient_search_id (int id) {
    for (int i = 0; i < patient_count; i++) {
        if (patients[i].id == id) {
            return i;
        }
    }
    return -1;
}

void patient_update_name(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[0],
        "Enter new name: ",
        "» "
    };

    char name[NAME_SIZE];
    ui_print_menu("Update Patient", menu, 3, UI_SIZE);
    utils_get_string(name, NAME_SIZE);
    utils_fix_name(name);

    if (!utils_is_valid_name(name)) {
        ui_print_error("Invalid name! Could not update.");
        ui_pause();
        return;
    }
    patients[index].name[0] = '\0';
    strncpy(patients[index].name, name, NAME_SIZE);
}

void patient_update_phone(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[1],
        "Enter new phone: ",
        "» "
    };

    char phone[PHONE_SIZE];
    ui_print_menu("Update Patient", menu, 3, UI_SIZE);
    utils_get_string(phone, PHONE_SIZE);

    if (!utils_is_valid_phone(phone)) {
        ui_print_error("Invalid phone! Could not update.");
        ui_pause();
        return;
    }
    patients[index].phone[0] = '\0';
    strncpy(patients[index].phone, phone, PHONE_SIZE);
}

void patient_update_address(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[2],
        "Enter new address: ",
        "» "
    };

    char address[ADDRESS_SIZE];
    ui_print_menu("Update Patient", menu, 3, UI_SIZE);
    utils_get_string(address, ADDRESS_SIZE);

    if (!utils_is_valid_address(address)) {
        ui_print_error("Invalid address! Could not update.");
        ui_pause();
        return;
    }
    patients[index].address[0] = '\0';
    strncpy(patients[index].address, address, ADDRESS_SIZE);
}

void patient_update_blood_group(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[3],
        "Enter new blood group: ",
        "» "
    };

    char blood_group[BLOOD_SIZE];
    ui_print_menu("Update Patient", menu, 3, UI_SIZE);
    utils_get_string(blood_group, BLOOD_SIZE);

    if (!utils_is_valid_blood_group(blood_group)) {
        ui_print_error("Invalid blood group! Could not update.");
        ui_pause();
        return;
    }
    utils_str_to_upper(blood_group);
    patients[index].blood_group[0] = '\0';
    strncpy(patients[index].blood_group, blood_group, BLOOD_SIZE);
}

void patient_update_gender(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[4],
        "Enter new gender: ",
        "» "
    };

    char gender_input;
    ui_print_menu("Update Patient", menu, 3, UI_SIZE);
    while (1) {
        gender_input = utils_get_char();
        if (gender_input == 'M' || gender_input == 'm' || gender_input == 'F' || gender_input == 'f')
{
            break;
        }
        ui_print_error("Invalid gender!");
        ui_pause();
    }
    if (gender_input == 'M' || gender_input == 'm') {
        patients[index].gender = MALE;
    } else if (gender_input == 'F' || gender_input == 'f') {
        patients[index].gender = FEMALE;
    }
}

void patient_update_status(const char* menu_items[], int index) {
    ui_clear_screen();
    ui_print_banner();

    const char* menu[] = {
        menu_items[5],
        "Enter new status: (1 for active, 2 for inactive) ",
        "» "
```

```c
    };

    bool status;
    int input;
    ui_print_menu("Update Patient", menu, 3, UI_SIZE);
    input = utils_get_int();

    if (input == 1) {
        status = true;
    } else if (input == 2) {
        status = false;
    }
    patients[index].is_active = status;
}

void patient_update_using_id() {
    ui_clear_screen();
    ui_print_banner();

    int id;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* enter_id_items[] = {
            "Enter patient ID: (0 to go back) ",
            "» "
        };

        ui_print_menu("Update Patient", enter_id_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) return;

        if (!utils_is_valid_id(id, ROLE_PATIENT)) {
            ui_print_error("Invalid ID!");
            ui_pause();
        } else {
            ui_print_info("Patient found!");
            ui_pause();
            break;
        }
    } while (1);

    int index = patient_search_id(id);
    if (index == -1) return;

    char name_line[NAME_LINE_SIZE], phone_line[PHONE_LINE_SIZE], address_line[ADDRESS_LINE_SIZE],
blood_group_line[BLOOD_LINE_SIZE], gender_line[GENDER_LINE_SIZE], status_line[STATUS_LINE_SIZE];

    snprintf(name_line, NAME_LINE_SIZE, "Name: %s",patients[index].name);
    snprintf(phone_line, PHONE_LINE_SIZE, "Phone: %s",patients[index].phone);
    snprintf(address_line, ADDRESS_LINE_SIZE, "Address: %s",patients[index].address);
    snprintf(blood_group_line, BLOOD_LINE_SIZE, "Blood Group: %s",patients[index].blood_group);

    int patient_id = patients[index].id;

    if (patients[index].gender == MALE) {
        snprintf(gender_line, GENDER_LINE_SIZE, "Gender: Male");
    } else {
        snprintf(gender_line, GENDER_LINE_SIZE, "Gender: Female");
    }
    if (patients[index].is_active) {
        snprintf(status_line, STATUS_LINE_SIZE, "Status: Active");
    } else {
        snprintf(status_line, STATUS_LINE_SIZE, "Status: Discharged");
    }

    const char* menu_items[] = {
        name_line,
        phone_line,
        address_line,
        blood_group_line,
        gender_line,
        "Go back",
        "» "
    };

    char title[50];
    snprintf(title, 50, "Update Patient | ID: %d", patient_id);

    ui_clear_screen();
    ui_print_banner();
    ui_print_menu(title, menu_items, 7, UI_SIZE);

    int choice = utils_get_int();
    switch (choice) {
        case 1:
            patient_update_name(menu_items, index);
            break;
        case 2:
            patient_update_phone(menu_items, index);
            break;
        case 3:
            patient_update_address(menu_items, index);
            break;
        case 4:
            patient_update_blood_group(menu_items, index);
            break;
        case 5:
            patient_update_gender(menu_items, index);
            break;
        case 6:
            ui_print_info("Returning to receptionist menu...");
            ui_pause();
            break;
        default:
            ui_print_error("Invalid choice! Please try again.");
            ui_pause();
            break;
    }
}

void patient_discharge() {
    int id;
    while (1) {
        ui_clear_screen();
        ui_print_banner();

        const char* enter_id_items[] = {
            "Enter patient ID (0 to cancel): ",
            "» "
        };

        ui_print_menu("Discharge Patient", enter_id_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) {
            ui_print_info("User Pressed 0. \nCanceling deletion...");
            ui_pause();
            return;
        }

        if (!utils_is_valid_id(id, ROLE_PATIENT)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }
        break;
    }

    int index = patient_search_id(id);
    if (index == -1) {
        ui_print_error("Patient not found!");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();
    ui_print_patient(patients[index], index);

    const char* menu[] = {
        "Confirm Discharge",
        "Cancel",
        "» "
    };

    ui_print_menu("Discharge Patient", menu, 3, UI_SIZE);
    int input = utils_get_int();

    if (input == 1) {
        patients[index].is_active = false;
        ui_print_success("Patient discharged successfully!");
        patient_available--;
        patient_unavailable++;
        ui_pause();
    } else {
        ui_print_info("Discharge cancelled.");
        ui_pause();
    }
}
```

```

void patient_view_discharged(void) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    if (patient_unavailable == 0) {
        const char* menu_items[] = {"No discharged patients found!"};
        ui_print_menu("Discharged Patients", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < patient_count; i++) {
        if (!patients[i].is_active) {
            ui_print_patient(patients[i], count++);
        }
    }
    ui_pause();
}
```

6.35 src/receptionist.c File Reference

Receptionist portal implementation for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include "../include/receptionist.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/appointment.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"
```

Functions

- void receptionist_patient_menu (void)
- void receptionist_appointment_menu (void)
- void receptionist_menu (void)
- int receptionist_save_to_file (void)
- int receptionist_load_from_file (void)
- int receptionist_search_id (int id)
- void receptionist_view_all (void)
- void receptionist_view_discharged (void)
- void receptionist_discharge (void)

6.35.1 Detailed Description

Receptionist portal implementation for Healthcare Management System.
Definition in file receptionist.c.

6.35.2 Function Documentation

6.35.2.1 receptionist_appointment_menu()

```
void receptionist_appointment_menu (
                void )
```

Receptionist appointment menu.
Definition at line 70 of file receptionist.c.

6.35.2.2 receptionist_discharge()

```
void receptionist_discharge (
             void )
```

Deactivates a receptionist.
Definition at line 243 of file receptionist.c.

6.35.2.3 receptionist_load_from_file()

```
int receptionist_load_from_file (
             void )
```

Loads all receptionists from binary file.

Returns

0 on success, -1 if file doesn't exist.

Definition at line 170 of file receptionist.c.

6.35.2.4 receptionist_menu()

```
void receptionist_menu (
             void )
```

Main receptionist portal menu.
Definition at line 114 of file receptionist.c.

6.35.2.5 receptionist_patient_menu()

```
void receptionist_patient_menu (
             void )
```

Receptionist patient management menu.
Definition at line 16 of file receptionist.c.

6.35.2.6 receptionist_save_to_file()

```
int receptionist_save_to_file (
             void )
```

Saves all receptionists to binary file.

Returns

0 on success, -1 on failure.

Definition at line 155 of file receptionist.c.

6.35.2.7 receptionist_search_id()

```
int receptionist_search_id (
             int id)
```

Searches for a receptionist by ID.

Parameters

id	The receptionist ID to search for.

Returns

Index of receptionist, or -1 if not found.

Definition at line 194 of file receptionist.c.

6.35.2.8 receptionist_view_all()

```
void receptionist_view_all (
                void )
```

Views all active receptionists.
Definition at line 203 of file receptionist.c.

6.35.2.9 receptionist_view_discharged()

```
void receptionist_view_discharged (
                void )
```

Views all inactive receptionists.
Definition at line 223 of file receptionist.c.

6.36 receptionist.c

Go to the documentation of this file.

```
#include <stdio.h>
#include <string.h>
#include "../include/receptionist.h"
#include "../include/patient.h"
#include "../include/doctor.h"
#include "../include/appointment.h"
#include "../include/utils.h"
#include "../include/ui.h"
#include "../include/hospital.h"

void receptionist_patient_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Add Patient",
            "View Active Patients",
            "View Discharged Patients",
            "Search Patient",
            "Update Patient",
            "Discharge Patient",
            "Back",
            "» "
        };

        ui_print_menu("Patient Management", menu_items, 8, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                patient_add();
                patient_save_to_file();
                break;
            case 2:
                patient_view_all();
                break;
            case 3:
                patient_view_discharged();
                break;
            case 4:
                patient_search_by();
                break;
            case 5:
                patient_update_using_id();
                patient_save_to_file();
                break;
            case 6:
                patient_discharge();
                patient_save_to_file();
                break;
            case 7:
                ui_print_info("Returning to receptionist menu...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
            }
    } while (choice != 7);
}

void receptionist_appointment_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Create Appointment",
            "View All Appointments",
            "Back",
            "» "
        };

        ui_print_menu("Appointment Management", menu_items, 4, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                appointment_create();
                break;
            case 2:
                ui_clear_screen();
                ui_print_banner();
                for (int i = 0; i < appointment_count; i++) {
                    ui_print_appointment(appointments[i], i);
                }
                if (appointment_count == 0) {
                    const char* no_appt[] = {"No appointments found!"};
                    ui_print_menu("All Appointments", no_appt, 1, UI_SIZE);
                }
                ui_pause();
                break;
            case 3:
                ui_print_info("Returning to receptionist menu...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 3);
}

void receptionist_menu(void) {
    int choice;

    do {
        ui_clear_screen();
        ui_print_banner();

        const char* menu_items[] = {
            "Patient Management",
            "Appointment Management",
            "Logout",
            "» "
        };

        ui_print_menu("Receptionist Portal", menu_items, 4, UI_SIZE);
        choice = utils_get_int();

        switch (choice) {
            case 1:
                receptionist_patient_menu();
                break;
            case 2:
                receptionist_appointment_menu();
                break;
            case 3:
                ui_print_info("Logging out...");
                ui_pause();
                break;
            default:
                ui_print_error("Invalid choice!");
                ui_pause();
        }
    } while (choice != 3);
}

/*
 *=========================================================================
 *                         RECEPTIONIST DATA FUNCTIONS
```

```c
 *==========================================================================
 */

int receptionist_save_to_file(void) {
    FILE* file = fopen(RECEPTIONISTS_FILE, "wb");
    if (file == NULL) {
        return -1;
    }
    if (fwrite(&receptionist_count, sizeof(int), 1, file) != 1 ||
        fwrite(&receptionist_available, sizeof(int), 1, file) != 1 ||
        fwrite(receptionists, sizeof(Receptionist), receptionist_count, file) !=
(size_t)receptionist_count) {
        fclose(file);
        return -1;
    }
    fclose(file);
    return 0;
}

int receptionist_load_from_file(void) {
    FILE* file = fopen(RECEPTIONISTS_FILE, "rb");
    if (file == NULL) {
        return -1;
    }
    if (fread(&receptionist_count, sizeof(int), 1, file) != 1 ||
        fread(&receptionist_available, sizeof(int), 1, file) != 1) {
        fclose(file);
        return -1;
    }
    if (receptionist_count < 0 || receptionist_count > MAX_RECEPTIONISTS) {
        fclose(file);
        receptionist_count = 0;
        return -1;
    }
    if (fread(receptionists, sizeof(Receptionist), receptionist_count, file) !=
(size_t)receptionist_count) {
        fclose(file);
        receptionist_count = 0;
        return -1;
    }
    fclose(file);
    return 0;
}

int receptionist_search_id(int id) {
    for (int i = 0; i < receptionist_count; i++) {
        if (receptionists[i].id == id) {
            return i;
        }
    }
    return -1;
}

void receptionist_view_all(void) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    if (receptionist_available == 0 || receptionist_count == 0) {
        const char* menu_items[] = {"No receptionists found!"};
        ui_print_menu("View All Receptionists", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < receptionist_count; i++) {
        if (receptionists[i].is_active) {
            ui_print_receptionist(receptionists[i], count++);
        }
    }
    ui_pause();
}

void receptionist_view_discharged(void) {
    int count = 0;
    ui_clear_screen();
    ui_print_banner();

    if (receptionist_unavailable == 0) {
        const char* menu_items[] = {"No inactive receptionists found!"};
        ui_print_menu("Inactive Receptionists", menu_items, 1, UI_SIZE);
        ui_pause();
        return;
    }

    for (int i = 0; i < receptionist_count; i++) {
        if (!receptionists[i].is_active) {
            ui_print_receptionist(receptionists[i], count++);
        }
    }
    ui_pause();
}

void receptionist_discharge(void) {
    int id;
    while (1) {
        ui_clear_screen();
        ui_print_banner();

        const char* enter_id_items[] = {
            "Enter receptionist ID (0 to cancel): ",
            "» "
        };

        ui_print_menu("Deactivate Receptionist", enter_id_items, 2, UI_SIZE);
        id = utils_get_int();

        if (id == 0) {
            ui_print_info("Cancelled.");
            ui_pause();
            return;
        }

        if (!utils_is_valid_id(id, ROLE_RECEPTIONIST)) {
            ui_print_error("Invalid ID!");
            ui_pause();
            continue;
        }
        break;
    }

    int index = receptionist_search_id(id);
    if (index == -1) {
        ui_print_error("Receptionist not found!");
        ui_pause();
        return;
    }

    ui_clear_screen();
    ui_print_banner();
    ui_print_receptionist(receptionists[index], index);

    const char* menu[] = {
        "Confirm Deactivate",
        "Cancel",
        "» "
    };

    ui_print_menu("Deactivate Receptionist", menu, 3, UI_SIZE);
    int input = utils_get_int();

    if (input == 1) {
        receptionists[index].is_active = false;
        ui_print_success("Receptionist deactivated successfully!");
        receptionist_available--;
        receptionist_unavailable++;
        receptionist_save_to_file();
        ui_pause();
    } else {
        ui_print_info("Deactivation cancelled.");
        ui_pause();
    }
}
```

6.37 src/ui.c File Reference

User interface functions for Healthcare Management System.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "../include/ui.h"
#include "../include/utils.h"
#include <unistd.h>
```

Functions

- void ui_clear_screen (void)
- void ui_pause (void)
- void ui_print_header (const char ∗title)
- void ui_print_success (const char ∗message)
- void ui_print_error (const char ∗message)
- void ui_print_warning (const char ∗message)
- void ui_print_info (const char ∗message)
- void ui_print_banner (void)
- void ui_print_menu (const char ∗title, const char ∗items[ ], int item_count, int box_width)
- void ui_print_patient (Patient patient, int index)
- void ui_print_doctor (Doctor doctor, int index)
- void ui_print_receptionist (Receptionist receptionist, int index)
- void ui_dummy_loading (int time)

6.37.1 Detailed Description

User interface functions for Healthcare Management System.
This file contains the implementation of user interface functions used throughout the HMS application.
Definition in file ui.c.

6.37.2 Function Documentation

6.37.2.1 ui_clear_screen()

```
void ui_clear_screen (
             void )
```

Clears the console screen.
Definition at line 22 of file ui.c.

6.37.2.2 ui_dummy_loading()

```
void ui_dummy_loading (
             int time)
```

Prints a dummy loading animation.
Definition at line 268 of file ui.c.

6.37.2.3 ui_pause()

```
void ui_pause (
             void )
```

Pauses the program execution until the user presses a key.
Definition at line 30 of file ui.c.

6.37.2.4 ui_print_banner()

```
void ui_print_banner (
             void )
```

Prints the HMS banner.
Definition at line 56 of file ui.c.

6.37.2.5 ui_print_doctor()

```
void ui_print_doctor (
             Doctor doctor,
             int index)
```

Prints a doctor in a box.

Parameters

doctor	The doctor to print.
index	The display index.

Definition at line 190 of file ui.c.

6.37.2.6 ui_print_error()

```
void ui_print_error (
             const char * message)
```

Prints an error message in red.

Parameters

message	The error message.

Definition at line 43 of file ui.c.

6.37.2.7 ui_print_header()

```
void ui_print_header (
             const char * title)
```

Prints a formatted header with title.

Parameters

title	The title to display.

Definition at line 35 of file ui.c.

6.37.2.8 ui_print_info()

```
void ui_print_info (
             const char * message)
```

Prints an info message in cyan.

Parameters

message	The info message.

Definition at line 51 of file ui.c.

6.37.2.9 ui_print_menu()

```
void ui_print_menu (
             const char * title,
             const char * items[ ],
             int item_count,
             int box_width)
```

Prints a menu in a box.

Parameters

title	The title of the menu.

items	The array of menu items.
item_count	The number of menu items.
box_width	The width of the box.

Definition at line 95 of file ui.c.

6.37.2.10 ui_print_patient()

```
void ui_print_patient (
             Patient patient,
             int index)
```

Prints a patient in a box.

Parameters

patient	The patient to print.

Definition at line 147 of file ui.c.

6.37.2.11 ui_print_receptionist()

```
void ui_print_receptionist (
             Receptionist receptionist,
             int index)
```

Prints a receptionist in a box.

Parameters

receptionist	The receptionist to print.
index	The display index.

Definition at line 233 of file ui.c.

6.37.2.12 ui_print_success()

```
void ui_print_success (
             const char * message)
```

Prints a success message in green.

Parameters

message	The success message.

Definition at line 39 of file ui.c.

6.37.2.13 ui_print_warning()

```
void ui_print_warning (
             const char * message)
```

Prints a warning message in yellow.

Parameters

message	The warning message.

Definition at line 47 of file ui.c.

6.38 ui.c

Go to the documentation of this file.

```
00001
00008
00009 #include <stdio.h>
00010 #include <stdlib.h>
00011 #include <ctype.h>
00012 #include <string.h>
00013 #include "../include/ui.h"
00014 #include "../include/utils.h"
00015
00016 #ifdef _WIN32
00017     #include <windows.h>
00018 #else
00019     #include <unistd.h>
00020 #endif
00021
00022 void ui_clear_screen(void) {
00023     #ifdef _WIN32
00024         system("cls");
00025         #else
00026         system("clear");
00027     #endif
00028 }
00029
00030 void ui_pause(void) {
00031     printf("\nPress enter to continue...");
00032     getchar();
00033 }
00034
00035 void ui_print_header(const char *title) {
00036     printf(BG_NEON_PURPLE BOLD "  %s  " RESET, title);
00037 }
00038
00039 void ui_print_success(const char *message) {
00040     printf(SOFT_GREEN "\n%s" RESET , message);
00041 }
00042
00043 void ui_print_error(const char *message) {
00044     printf(SOFT_RED "\n%s" RESET , message);
00045 }
00046
00047 void ui_print_warning(const char *message) {
00048     printf(SOFT_YELLOW "\n%s" RESET , message);
00049 }
00050
00051 void ui_print_info(const char *message) {
00052     printf(SOFT_BLUE "\n%s" RESET , message);
00053 }
00054
00055
00056 void ui_print_banner(void){
00057     unsigned char b = 219;  //  Full block character
00058
00059     char* hms =
00060     "#####################################\n"
00061     "#                                   #\n"
00062     "#        X  X  X   X  XXXX          #\n"
00063     "#        X  X  XX XX  X             #\n"
00064     "#        XXXX  X X X  XXXX          #\n"
00065     "#        X  X  X   X     X          #\n"
00066     "#        X  X  X   X  XXXX          #\n"
00067     "#                                   #\n"
00068     "#    HEALTHCARE MANAGEMENT SYSTEM   #\n"
00069     "#                                   #\n"
00070     "#####################################\n";
00071
00072     printf("\n");
00073
00074     while (*hms) {
00075         if (*hms == 'X') {
00076             printf(BRIGHT_RED "%c%c" RESET, b, b);
00077         }
00078         else if (*hms == '#') {
00079             printf(SOFT_GRAY "%c%c" RESET, b, b);
00080         }
00081         else if (*hms == '\n') {
00082             printf("\n");
00083         }
00084         else if (isalpha(*hms)){
00085             printf(BOLD CYAN "%c" RESET " ", *hms);
```

```
        }
        else {
            printf("  ");
        }
        hms++;
    }
    printf("\n");
}

void ui_print_menu
    (
    const char *title,
    const char *items[],
    int item_count,
    int box_width
    ) {
    unsigned char h = 205;  //
    unsigned char v = 186;  //
    unsigned char tl = 201; //
    unsigned char tr = 187; //
    unsigned char bl = 200; //
    unsigned char br = 188; //

    printf(BRIGHT_BLACK "%c", tl);
    for (int i = 0; i < box_width; i++) printf("%c", h);
    printf("%c" RESET "\n", tr);

    char title_upper[100];
    strncpy(title_upper, title, sizeof(title_upper) - 1);
    title_upper[sizeof(title_upper) - 1] = '\0';
    utils_str_to_upper(title_upper);

    int title_len = strlen(title_upper) + 4;
    int title_padding = (box_width - title_len) / 2;
    printf(BRIGHT_BLACK "%c" RESET, v);
    for (int i = 0; i < title_padding; i++) printf(" ");
    ui_print_header(title_upper);
    for (int i = 0; i < box_width - title_padding - title_len; i++) printf(" ");
    printf(BRIGHT_BLACK "%c" RESET "\n", v);

    printf(BRIGHT_BLACK "%c" RESET, v);
    for (int i = 0; i < box_width; i++) printf(" ");
    printf(BRIGHT_BLACK "%c" RESET "\n", v);

    for (int i = 0; i < item_count - 1; i++) {
        int item_len = strlen(items[i]);
        printf(BRIGHT_BLACK "%c" RESET "   " SOFT_YELLOW BOLD "%d. %s" RESET, v, i + 1, items[i]);
        for (int j = 0; j < box_width - item_len - 5; j++) printf(" ");
        printf(BRIGHT_BLACK "%c" RESET "\n", v);
    }

    printf(BRIGHT_BLACK "%c" RESET, v);
    for (int i = 0; i < box_width; i++) printf(" ");
    printf(BRIGHT_BLACK "%c" RESET "\n", v);
    printf(BRIGHT_BLACK "%c", bl);
    for (int i = 0; i < box_width; i++) printf("%c", h);
    printf("%c" RESET "\n\n", br);

    printf(BOLD SOFT_GREEN "%s" RESET, items[item_count - 1]);
}

void ui_print_patient(Patient patient, int index) {

    char id_line[ID_LINE_SIZE];
    snprintf(id_line, sizeof(id_line), "Patient ID: %d", patient.id);

    char name_line[NAME_LINE_SIZE];
    snprintf(name_line, sizeof(name_line), "Name: %s", patient.name);

    char age_line[AGE_LINE_SIZE];
    snprintf(age_line, sizeof(age_line), "Age: %d", patient.age);

    char gender_line[GENDER_LINE_SIZE];
    snprintf(gender_line, sizeof(gender_line), "Gender: %s", patient.gender == MALE ? "Male" : "Female");

    char phone_line[PHONE_LINE_SIZE];
    snprintf(phone_line, sizeof(phone_line), "Phone: %s", patient.phone);

    char address_line[ADDRESS_LINE_SIZE];
    snprintf(address_line, sizeof(address_line), "Address: %s", patient.address);

    char blood_group_line[BLOOD_LINE_SIZE];
    snprintf(blood_group_line, sizeof(blood_group_line), "Blood Group: %s", patient.blood_group);

    char status_line[STATUS_LINE_SIZE];
    snprintf(status_line, sizeof(status_line), "Status: %s", patient.is_active ? "Active" :
```

```
"Inactive");

    const char* items[] = {
        id_line,
        name_line,
        age_line,
        gender_line,
        phone_line,
        address_line,
        blood_group_line,
        status_line,
        ""
    };

    char title[70];
    snprintf(title, sizeof(title), "Patient %d", index + 1);
    ui_print_menu(title, items, 9, 72);
}

void ui_print_doctor(Doctor doctor, int index) {

    char id_line[70];
    snprintf(id_line, sizeof(id_line), "Doctor ID: %d", doctor.id);

    char name_line[70];
    snprintf(name_line, sizeof(name_line), "Name: %s", doctor.name);

    char phone_line[70];
    snprintf(phone_line, sizeof(phone_line), "Phone: %s", doctor.phone);

    char email_line[70];
    snprintf(email_line, sizeof(email_line), "Email: %s", doctor.email);

    char spec_line[70];
    snprintf(spec_line, sizeof(spec_line), "Specialization: %s", doctor.specialization);

    char room_line[70];
    snprintf(room_line, sizeof(room_line), "Room Number: %d", doctor.room_number);

    char avail_line[70];
    snprintf(avail_line, sizeof(avail_line), "Available: %s", doctor.is_available ? "Yes" : "No");

    char status_line[70];
    snprintf(status_line, sizeof(status_line), "Status: %s", doctor.is_active ? "Active" :
"Inactive");

    const char* items[] = {
        id_line,
        name_line,
        phone_line,
        email_line,
        spec_line,
        room_line,
        avail_line,
        status_line,
        ""
    };

    char title[70];
    snprintf(title, sizeof(title), "Doctor %d", index + 1);
    ui_print_menu(title, items, 9, 72);
}

void ui_print_receptionist(Receptionist receptionist, int index) {

    char id_line[70];
    snprintf(id_line, sizeof(id_line), "Receptionist ID: %d", receptionist.id);

    char name_line[70];
    snprintf(name_line, sizeof(name_line), "Name: %s", receptionist.name);

    char phone_line[70];
    snprintf(phone_line, sizeof(phone_line), "Phone: %s", receptionist.phone);

    char email_line[70];
    snprintf(email_line, sizeof(email_line), "Email: %s", receptionist.email);

    char avail_line[70];
    snprintf(avail_line, sizeof(avail_line), "Available: %s", receptionist.is_available ? "Yes" :
"No");

    char status_line[70];
    snprintf(status_line, sizeof(status_line), "Status: %s", receptionist.is_active ? "Active" :
"Inactive");

    const char* items[] = {
        id_line,
        name_line,
        phone_line,
        email_line,
        avail_line,
        status_line,
        ""
    };

    char title[70];
    snprintf(title, sizeof(title), "Receptionist %d", index + 1);
    ui_print_menu(title, items, 7, 72);
}

void ui_dummy_loading(int time) {
    ui_clear_screen();
    ui_print_banner();

    const char arr[] = {'|', '/', '-', '\\'};
    int i = 0;
    int loading_time = 0;
    printf("\n\n\n");
    while (1) {
        printf(BOLD BRIGHT_RED "\rLoading data from files... " RESET "%c", arr[i]);
        fflush(stdout);
        i = (i + 1) % 4;

        #ifdef _WIN32
            Sleep(100);
        #else
            usleep(100000);
        #endif

        loading_time++;
        if (loading_time == time) break;
    }

    ui_print_success("Successfully loaded data from files!\n");

    ui_pause();
}
```

6.39 src/utils.c File Reference

Utility functions for Healthcare Management System.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "../include/utils.h"
```

Functions

- void utils_clear_input_buffer (void)
- void utils_pause (void)
- int utils_get_int (void)
- float utils_get_float (void)
- double utils_get_double (void)
- char utils_get_char (void)
- char ∗ utils_get_string (char ∗str, size_t size)
- bool utils_is_valid_phone (const char ∗phone)
- bool utils_is_valid_email (const char ∗email)
- bool utils_is_valid_name (const char ∗name)
- bool utils_is_valid_id (int id, UserRole role)
- char ∗ utils_str_to_upper (char ∗str)
- char ∗ utils_fix_name (char ∗name)
- bool utils_is_valid_blood_group (const char ∗blood_group)
- bool utils_is_valid_address (const char ∗address)

6.39.1 Detailed Description

Utility functions for Healthcare Management System.
This file contains the implementation of utility functions used throughout the HMS application.
Definition in file utils.c.

6.39.2 Function Documentation

6.39.2.1 utils_clear_input_buffer()

```
void utils_clear_input_buffer (
             void )
```

Clears the input buffer to prevent leftover characters.
Definition at line 14 of file utils.c.

6.39.2.2 utils_fix_name()

```
char * utils_fix_name (
             char * name)
```

Fixes name to Title Case (first letter of each word uppercase, rest lowercase). Example: "jOHN dOE" -> "John Doe"

Parameters

name	The name string to fix (modified in-place).

Returns

The fixed name string.

Definition at line 169 of file utils.c.

6.39.2.3 utils_get_char()

```
char utils_get_char (
             void )
```

Scans a char value from the user.

Returns

The valid char entered by the user.

Definition at line 54 of file utils.c.

6.39.2.4 utils_get_double()

```
double utils_get_double (
             void )
```

Scans a double value from the user with validation.

Returns

The valid double entered by the user.

Definition at line 44 of file utils.c.

6.39.2.5 utils_get_float()

```
float utils_get_float (
             void )
```

Scans a float value from the user with validation.

Returns

The valid float entered by the user.

Definition at line 34 of file utils.c.

6.39.2.6 utils_get_int()

```
int utils_get_int (
             void )
```

Scans an integer value from the user with validation.

Returns

The valid integer entered by the user.

Definition at line 24 of file utils.c.

6.39.2.7 utils_get_string()

```
char * utils_get_string (
             char * str,
             size_t size)
```

Scans a string value from the user.

Parameters

str	The buffer to store the string.
size	The maximum number of characters to read.

Returns

Pointer to string on success, NULL on failure.

Definition at line 61 of file utils.c.

6.39.2.8 utils_is_valid_address()

```
bool utils_is_valid_address (
             const char * address)
```

Validates if an address contains only valid characters. Allowed: letters, digits, spaces, commas, periods.

Parameters

address	The address string to validate.

Returns

true if valid, false otherwise.

Definition at line 204 of file utils.c.

6.39.2.9 utils_is_valid_blood_group()

```
bool utils_is_valid_blood_group (
             const char * blood_group)
```

Validates if a blood group is in correct format.

Parameters

blood_group	The blood group string to validate.

Returns

true if valid, false otherwise.

Definition at line 186 of file utils.c.

6.39.2.10 utils_is_valid_email()

```
bool utils_is_valid_email (
             const char * email)
```

Validates if an email contains @ and .

Parameters

email	The email string to validate.

Returns

true if valid, false otherwise.

Definition at line 97 of file utils.c.

6.39.2.11 utils_is_valid_id()

```
bool utils_is_valid_id (
             int id,
             UserRole role)
```

Validates if an ID is in correct format.

Parameters

id	The ID to validate.

Returns

true if valid ID, false otherwise.

Definition at line 140 of file utils.c.

6.39.2.12 utils_is_valid_name()

```
bool utils_is_valid_name (
             const char * name)
```

Validates if a name contains only letters and spaces.

Parameters

name	The name to validate.

Returns

true if valid name, false otherwise.

Definition at line 128 of file utils.c.

6.39.2.13 utils_is_valid_phone()

```
bool utils_is_valid_phone (
              const char * phone)
```

6.39.3 Validity Functions

Definition at line 81 of file utils.c.

6.39.3.1 utils_pause()

```
void utils_pause (
              void )
```

Definition at line 19 of file utils.c.

6.39.3.2 utils_str_to_upper()

```
char * utils_str_to_upper (
              char * str)
```

Converts a string to uppercase in-place.

Parameters

str	The string to convert.

Returns

The uppercase string.

Definition at line 161 of file utils.c.

6.40 utils.c

Go to the documentation of this file.

```
00001
00008
00009 #include <stdio.h>
00010 #include <string.h>
00011 #include <ctype.h>
00012 #include "../include/utils.h"
00013
00014 void utils_clear_input_buffer(void) {
00015     int c;
00016     while ((c = getchar()) != '\n' && c != EOF);
00017 }
00018
00019 void utils_pause(void) {
00020     printf("\nPress any key to continue...");
00021     getchar();
00022 }
00023
00024 int utils_get_int(void) {
00025     int num;
00026     while (scanf("%d", &num) != 1) {
00027         printf("Invalid input. Please enter an integer: ");
00028         utils_clear_input_buffer();
00029     }
00030     utils_clear_input_buffer();
00031     return num;
00032 }
00033
00034 float utils_get_float(void) {
00035     float num;
00036     while (scanf("%f", &num) != 1) {
00037         printf("Invalid input. Please enter a float: ");
00038         utils_clear_input_buffer();
00039     }
00040     utils_clear_input_buffer();
00041     return num;
```

```c
}

double utils_get_double(void) {
    double num;
    while (scanf("%lf", &num) != 1) {
        printf("Invalid input. Please enter a double: ");
        utils_clear_input_buffer();
    }
    utils_clear_input_buffer();
    return num;
}

char utils_get_char(void) {
    char ch;
    scanf(" %c", &ch);
    utils_clear_input_buffer();
    return ch;
}

char* utils_get_string(char *str, size_t size) {
    while (1) {
        if (fgets(str, size, stdin) != NULL) {
            str[strcspn(str, "\n")] = '\0';
            if (str[0] != '\0') {
                return str;
            }
            printf("Input cannot be empty. Try again: ");
        } else {
            return NULL;
        }
    }
}


bool utils_is_valid_phone(const char *phone) {
    if (phone == NULL || phone[0] == '\0') return false;

    int digit_count = 0;
    for (size_t i = 0; phone[i] != '\0'; i++) {
        if (!isdigit(phone[i])) return false;
        digit_count++;
    }

    if (digit_count != 11) return false;
    if (phone[0] != '0' || phone[1] != '1') return false;
    if (phone[2] < '3' || phone[2] > '9') return false;

    return true;
}

bool utils_is_valid_email(const char *email) {
    if (email == NULL || email[0] == '\0') return false;

    int at_count = 0;
    int at_position = -1;
    int last_dot_position = -1;
    int length = 0;

    for (size_t i = 0; email[i] != '\0'; i++) {
        char c = email[i];

        if (c != '@' && !(isdigit(c)) && !(isalpha(c)) && c != '.') return false;

        if (c == '@') {
            at_count++;
            at_position = i;
        }
        if (c == '.') {
            last_dot_position = i;
        }
        length++;
    }

    if (at_count != 1) return false;
    if (at_position == 0) return false;
    if (last_dot_position <= at_position) return false;
    if (last_dot_position == length - 1) return false;

    return true;
}

bool utils_is_valid_name(const char *name) {
    if (name == NULL || name[0] == '\0') return false;

    for (size_t i = 0; name[i] != '\0'; i++) {
        if (!isalpha(name[i]) && name[i] != ' ') {
            return false;
        }
    }

    return true;
}

bool utils_is_valid_id(int id, UserRole role) {
    if (id < 0) return false;

    if (role == ROLE_PATIENT) {
        if (id < 1001) return false;
        if (id >= 2001) return false;
    } else if (role == ROLE_DOCTOR) {
        if (id < 2001) return false;
        if (id >= 3001) return false;
    } else if (role == ROLE_ADMIN) {
        if (id < 3001) return false;
        if (id >= 4001) return false;
    } else if (role == ROLE_RECEPTIONIST) {
        if (id < 4001) return false;
        if (id >= 5001) return false;
    }

    return true;
}


char* utils_str_to_upper(char *str) {
    if (str == NULL) return NULL;
    for (int i = 0; str[i] != '\0'; i++) {
        str[i] = toupper(str[i]);
    }
    return str;
}

char* utils_fix_name(char *name) {
    if (name == NULL || name[0] == '\0') return name;

    bool new_word = true;
    for (int i = 0; name[i] != '\0'; i++) {
        if (name[i] == ' ') {
            new_word = true;
        } else if (new_word) {
            name[i] = toupper(name[i]);
            new_word = false;
        } else {
            name[i] = tolower(name[i]);
        }
    }
    return name;
}

bool utils_is_valid_blood_group(const char *blood_group) {
    if (blood_group == NULL || blood_group[0] == '\0') return false;

    const char* valid_blood_groups[] = {
        "A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-", "U",
        "a+", "a-", "b+", "b-", "ab+", "ab-", "o+", "o-", "u"
    };
    int count = sizeof(valid_blood_groups) / sizeof(valid_blood_groups[0]);

    for (int i = 0; i < count; i++) {
        if (strcmp(blood_group, valid_blood_groups[i]) == 0) {
            return true;
        }
    }

    return false;
}

bool utils_is_valid_address(const char *address) {
    if (address == NULL || address[0] == '\0') return false;

    for (int i = 0; address[i] != '\0'; i++) {
        char c = address[i];
        if (!isalnum(c) && c != ' ' && c != ',' && c != '.') {
            return false;
        }
    }
    return true;
}
```

6.41 tests/patient_test.c File Reference

```
#include <stdio.h>
#include "../include/patient.h"
#include "../include/hospital.h"
#include "../include/ui.h"
```

Functions

- void hospital_init (void)
- int main ()

6.41.1 Function Documentation

6.41.1.1 hospital_init()

```
void hospital_init (
            void )
```

Initialize the hospital system by loading all data.
Definition at line 6 of file patient_test.c.

6.41.1.2 main()

```
int main ()
```

Definition at line 10 of file patient_test.c.

6.42 patient_test.c

Go to the documentation of this file.

```
00001 #include <stdio.h>
00002 #include "../include/patient.h"
00003 #include "../include/hospital.h"
00004 #include "../include/ui.h"
00005
00006 void hospital_init(void) {
00007     patient_load_from_file();
00008 }
00009
00010 int main() {
00011     hospital_init();
00012     patient_receptionist_menu();
00013     return 0;
00014 }
```

6.43 tests/test.c File Reference

```
#include <stdio.h>
#include "../include/utils.h"
#include <stdbool.h>
```

Functions

- int main ()

6.43.1 Function Documentation

6.43.1.1 main()

```
int main ()
```

Definition at line 5 of file test.c.

6.44 test.c

Go to the documentation of this file.

```c
#include <stdio.h>
#include "../include/utils.h"
#include <stdbool.h>

int main() {
    char phone[12];
    utils_get_string(phone, 12);
    if(utils_is_valid_phone(phone)) {
        printf("Valid phone number\n");
    } else {
        printf("Invalid phone number\n");
    }
    if (utils_is_valid_email("noice.nice@gmail.com")) {
        printf("Valid email\n");
    } else {
        printf("Invalid email\n");
    }
    utils_pause();
    return 0;
}
```

6.45 tests/test_phone_validator.c File Reference

```
#include <stdio.h>
#include "../include/utils.h"
#include <stdbool.h>
```

Functions

- int test_validate_validphone ()
- int main ()

6.45.1 Function Documentation

6.45.1.1 main()

`int main ()`

Definition at line 16 of file test_phone_validator.c.

6.45.1.2 test_validate_validphone()

`int test_validate_validphone ()`

Definition at line 5 of file test_phone_validator.c.

6.46 test_phone_validator.c

Go to the documentation of this file.

```c
00001 #include <stdio.h>
00002 #include "../include/utils.h"
00003 #include <stdbool.h>
00004
00005 int test_validate_validphone() {
00006     char phone[]="01312381636";
00007    int expected=1;
00008    int actual=utils_is_valid_phone(phone);
00009     int succes= actual==expected;
00010     printf("actual: %d, expected: %d,succes:%d",actual,expected,actual==expected);
00011
00012 }
00013
00014
00015
00016 int main()
00017 {
00018     test_validate_validphone();
00019 }
```

6.47 tests/ui_test.c File Reference

```
#include <stdio.h>
#include "../include/ui.h"
#include "../include/utils.h"
#include "../include/patient.h"
#include "../include/receptionist.h"
```

Functions

- int main ()

6.47.1 Function Documentation

6.47.1.1 main()

```
int main ()
```

Definition at line 7 of file ui_test.c.

6.48 ui_test.c

Go to the documentation of this file.

```
#include <stdio.h>
#include "../include/ui.h"
#include "../include/utils.h"
#include "../include/patient.h"
#include "../include/receptionist.h"

int main() {
    // ui_clear_screen();
    // ui_print_banner();
    // printf("\n\n\n");
    // // ui_print_start_menu();
    // const char* menu[] = {"Login", "Register", "Exit"};
    // ui_print_menu("Main Menu", menu, 3, 72);
    // ui_pause();

    receptionist_menu();
    return 0;
}
```

6.49 tests/utils_test.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "../include/utils.h"
```

Functions

- void test_utils_get_int ()
- void test_utils_get_char ()
- void test_utils_get_float ()
- void test_utils_get_double ()
- void test_utils_get_string ()
- void test_utils_is_valid_phone ()
- void test_utils_is_valid_email ()
- void test_utils_str_to_upper ()
- void test_utils_is_valid_blood_group ()
- void test_utils_is_valid_address ()
- int main ()

6.49.1 Function Documentation

6.49.1.1 main()

`int main ()`
Definition at line 164 of file utils_test.c.

6.49.1.2 test_utils_get_char()

`void test_utils_get_char ()`
Definition at line 15 of file utils_test.c.

6.49.1.3 test_utils_get_double()

`void test_utils_get_double ()`
Definition at line 35 of file utils_test.c.

6.49.1.4 test_utils_get_float()

`void test_utils_get_float ()`
Definition at line 25 of file utils_test.c.

6.49.1.5 test_utils_get_int()

`void test_utils_get_int ()`
Definition at line 5 of file utils_test.c.

6.49.1.6 test_utils_get_string()

`void test_utils_get_string ()`
Definition at line 45 of file utils_test.c.

6.49.1.7 test_utils_is_valid_address()

`void test_utils_is_valid_address ()`
Definition at line 139 of file utils_test.c.

6.49.1.8 test_utils_is_valid_blood_group()

`void test_utils_is_valid_blood_group ()`
Definition at line 114 of file utils_test.c.

6.49.1.9 test_utils_is_valid_email()

`void test_utils_is_valid_email ()`
Definition at line 74 of file utils_test.c.

6.49.1.10 test_utils_is_valid_phone()

`void test_utils_is_valid_phone ()`
Definition at line 56 of file utils_test.c.

6.49.1.11 test_utils_str_to_upper()

`void test_utils_str_to_upper ()`
Definition at line 92 of file utils_test.c.

6.50 utils_test.c

Go to the documentation of this file.

```
#include <stdio.h>
#include <string.h>
#include "../include/utils.h"

void test_utils_get_int() {
    printf("Enter an integer: ");
    int expected = 5;
    int actual = utils_get_int();
    int success = (actual == expected);
    printf("  Expected: %d\n", expected);
    printf("  Actual:   %d\n", actual);
    printf("  Success:  %s\n\n", success ? "Yes" : "No");
}

void test_utils_get_char() {
    printf("Enter a character: ");
    char expected = 'c';
    char actual = utils_get_char();
    int success = (actual == expected);
    printf("  Expected: %c\n", expected);
    printf("  Actual:   %c\n", actual);
    printf("  Success:  %s\n\n", success ? "Yes" : "No");
}

void test_utils_get_float() {
    printf("Enter a float: ");
    float expected = 3.14;
    float actual = utils_get_float();
    int success = (actual >= 3.13 && actual <= 3.15);
    printf("  Expected: %.2f\n", expected);
    printf("  Actual:   %.2f\n", actual);
    printf("  Success:  %s\n\n", success ? "Yes" : "No");
}

void test_utils_get_double() {
    printf("Enter a double: ");
    double expected = 3.141593;
    double actual = utils_get_double();
    int success = (actual >= 3.141592 && actual <= 3.141594);
    printf("  Expected: %.6f\n", expected);
    printf("  Actual:   %.6f\n", actual);
    printf("  Success:  %s\n\n", success ? "Yes" : "No");
}

void test_utils_get_string() {
    printf("Enter a string: ");
    char expected[] = "happy birthday";
    char actual[50];
    utils_get_string(actual, 50);
    int success = (strcmp(actual, expected) == 0);
    printf("  Expected: %s\n", expected);
    printf("  Actual:   %s\n", actual);
    printf("  Success:  %s\n\n", success ? "Yes" : "No");
}

void test_utils_is_valid_phone() {
    printf("Testing utils_is_valid_phone():\n\n");

    char *phone1 = "01315648613";
    int expected1 = 1, actual1 = utils_is_valid_phone(phone1);
    printf("  Input:    %s\n", phone1);
    printf("  Expected: %s\n", expected1 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual1 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected1 == actual1) ? "Yes" : "No");

    char *phone2 = "1234567890";
    int expected2 = 0, actual2 = utils_is_valid_phone(phone2);
    printf("  Input:    %s\n", phone2);
    printf("  Expected: %s\n", expected2 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual2 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected2 == actual2) ? "Yes" : "No");
}

void test_utils_is_valid_email() {
    printf("Testing utils_is_valid_email():\n\n");

    char *email1 = "this.is.an.email@gmail.com";
    int expected1 = 1, actual1 = utils_is_valid_email(email1);
    printf("  Input:    %s\n", email1);
    printf("  Expected: %s\n", expected1 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual1 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected1 == actual1) ? "Yes" : "No");

```

```c
    char *email2 = "noatsign.com";
    int expected2 = 0, actual2 = utils_is_valid_email(email2);
    printf("  Input:    %s\n", email2);
    printf("  Expected: %s\n", expected2 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual2 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected2 == actual2) ? "Yes" : "No");
}

void test_utils_str_to_upper() {
    printf("Testing utils_str_to_upper():\n\n");

    char input1[] = "hello world";
    char expected1[] = "HELLO WORLD";
    char *actual1 = utils_str_to_upper(input1);
    int success1 = (strcmp(actual1, expected1) == 0);
    printf("  Input:    hello world\n");
    printf("  Expected: %s\n", expected1);
    printf("  Actual:   %s\n", actual1);
    printf("  Success:  %s\n\n", success1 ? "Yes" : "No");

    char input2[] = "Test123";
    char expected2[] = "TEST123";
    char *actual2 = utils_str_to_upper(input2);
    int success2 = (strcmp(actual2, expected2) == 0);
    printf("  Input:    Test123\n");
    printf("  Expected: %s\n", expected2);
    printf("  Actual:   %s\n", actual2);
    printf("  Success:  %s\n\n", success2 ? "Yes" : "No");
}

void test_utils_is_valid_blood_group() {
    printf("Testing utils_is_valid_blood_group():\n\n");

    char *blood1 = "A+";
    int expected1 = 1, actual1 = utils_is_valid_blood_group(blood1);
    printf("  Input:    %s\n", blood1);
    printf("  Expected: %s\n", expected1 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual1 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected1 == actual1) ? "Yes" : "No");

    char *blood2 = "AB-";
    int expected2 = 1, actual2 = utils_is_valid_blood_group(blood2);
    printf("  Input:    %s\n", blood2);
    printf("  Expected: %s\n", expected2 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual2 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected2 == actual2) ? "Yes" : "No");

    char *blood3 = "X+";
    int expected3 = 0, actual3 = utils_is_valid_blood_group(blood3);
    printf("  Input:    %s\n", blood3);
    printf("  Expected: %s\n", expected3 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual3 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected3 == actual3) ? "Yes" : "No");
}

void test_utils_is_valid_address() {
    printf("Testing utils_is_valid_address():\n\n");

    char *addr1 = "123 Main Street, Dhaka";
    int expected1 = 1, actual1 = utils_is_valid_address(addr1);
    printf("  Input:    %s\n", addr1);
    printf("  Expected: %s\n", expected1 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual1 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected1 == actual1) ? "Yes" : "No");

    char *addr2 = "Apt. 5, Block B";
    int expected2 = 1, actual2 = utils_is_valid_address(addr2);
    printf("  Input:    %s\n", addr2);
    printf("  Expected: %s\n", expected2 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual2 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected2 == actual2) ? "Yes" : "No");

    char *addr3 = "Invalid@Address#123";
    int expected3 = 0, actual3 = utils_is_valid_address(addr3);
    printf("  Input:    %s\n", addr3);
    printf("  Expected: %s\n", expected3 ? "Valid" : "Invalid");
    printf("  Actual:   %s\n", actual3 ? "Valid" : "Invalid");
    printf("  Success:  %s\n\n", (expected3 == actual3) ? "Yes" : "No");
}

int main() {
    printf("=== UTILITY FUNCTIONS TEST ===\n\n");

    // test_utils_get_int();
    // test_utils_get_char();
    // test_utils_get_float();
    // test_utils_get_double();
    // test_utils_get_string();
    test_utils_is_valid_phone();
    test_utils_is_valid_email();
    test_utils_str_to_upper();
    test_utils_is_valid_blood_group();
    test_utils_is_valid_address();

    return 0;
}
```