

Healthcare Management System

1.0.0

Generated by Doxygen 1.15.0

Chapter 1

Directory Hierarchy

1.1 Directories

include	??
admin.h	??
appointment.h	??
auth.h	??
doctor.h	??
doctor_portal.h	??
hospital.h	??
patient.h	??
receptionist.h	??
ui.h	??
utils.h	??
src	??
admin.c	??
appointment.c	??
auth.c	??
doctor.c	??
doctor_portal.c	??
hospital.c	??
patient.c	??
receptionist.c	??
ui.c	??
utils.c	??
tests	??
patient_test.c	??
test.c	??
test_phone_validator.c	??
ui_test.c	??
utils_test.c	??

Chapter 2

Class Index

2.1 Class List

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

Appointment	??
Doctor	??
Patient	??
Receptionist	??
User	??

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

include/ admin.h	Admin management functions for Healthcare Management System	??
include/ appointment.h	Appointment management functions for Healthcare Management System	??
include/ auth.h	Authentication functions for Healthcare Management System	??
include/ doctor.h	Doctor management functions for Healthcare Management System	??
include/ doctor_portal.h	Doctor portal functions for Healthcare Management System	??
include/ hospital.h	Core data structures and definitions for Healthcare Management System	??
include/ patient.h	Patient management functions for Healthcare Management System	??
include/ receptionist.h	Receptionist portal and data management functions	??
include/ ui.h	Header file for user interface functions	??
include/ utils.h	Utility functions for Healthcare Management System	??
src/ admin.c	Admin management implementation for Healthcare Management System	??
src/ appointment.c	Appointment management implementation for Healthcare Management System	??
src/ auth.c	Authentication implementation for Healthcare Management System	??
src/ doctor.c	Doctor management implementation for Healthcare Management System	??
src/ doctor_portal.c	Doctor portal implementation for Healthcare Management System	??
src/ hospital.c	Global data definitions for Healthcare Management System	??
src/ patient.c	Patient management implementation for Healthcare Management System	??
src/ receptionist.c	Receptionist portal implementation for Healthcare Management System	??

src/ui.c	
User interface functions for Healthcare Management System	??
src/utls.c	
Utility functions for Healthcare Management System	??
tests/patient_test.c	??
tests/test.c	??
tests/test_phone_validator.c	??
tests/ui_test.c	??
tests/utls_test.c	??

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.](#)

```

00001
00008
00009 #ifndef ADMIN_H
00010 #define ADMIN_H
00011
00012 #include "hospital.h"
00013
00017 void admin_view_discharged_patients(void);
00018
00022 void admin_search_discharged_by_id(void);
00023
00027 void admin_search_discharged_by_name(void);
00028
00032 void admin_search_discharged(void);
00033
00037 void admin_delete_patient(void);
00038
00042 void admin_view_discharged_doctors(void);
00043
00047 void admin_delete_doctor(void);
00048
00052 void admin_patient_menu(void);
00053
00057 void admin_doctor_menu(void);
00058
00062 void admin_main_menu(void);
00063
00067 void admin_receptionist_menu(void);
00068
00069 #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

<i>appt_id</i>	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

<i>id</i>	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

<i>status</i>	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 apt_id,  
    AppointmentStatus status)
```

Updates appointment status.

Parameters

<i>appt</i> ↔ <i>_id</i>	The appointment ID.
<i>status</i>	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

<i>doctor</i> ↔ <i>_id</i>	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

<i>appt</i>	The appointment to print.
<i>index</i>	The display index.

Definition at line 229 of file [appointment.c](#).

6.4 appointment.h

[Go to the documentation of this file.](#)

```
00001
00007
00008 #ifndef APPOINTMENT_H
00009 #define APPOINTMENT_H
00010
00011 #include "hospital.h"
00012
00017 int appointment_save_to_file(void);
00018
00023 int appointment_load_from_file(void);
00024
00029 int appointment_generate_id(void);
```

```

00030
00034 void appointment_create(void);
00035
00040 void appointment_view_by_doctor(int doctor_id);
00041
00047 void appointment_update_status(int appt_id, AppointmentStatus status);
00048
00053 void appointment_cancel(int appt_id);
00054
00060 int appointment_search_id(int id);
00061
00067 const char* appointment_status_str(AppointmentStatus status);
00068
00074 void ui_print_appointment(Appointment appt, int index);
00075
00076 #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

<i>required_role</i>	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

<i>password</i>	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

<i>password</i>	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.](#)

```
00001
00005
00006 #ifndef AUTH_H
00007 #define AUTH_H
00008
00009 #include "hospital.h"
00010
00015 int auth_save_to_file(void);
00016
00021 int auth_load_from_file(void);
00022
00026 void auth_register_user(void);
00027
00031 void auth_view_users(void);
00032
00036 void auth_init_default_admin(void);
00037
00041 void auth_user_menu(void);
00042
00047 void auth_role_login(UserRole required_role);
00048
00052 void login_menu(void);
00053
00058 void encrypt(char* password);
00059
00064 void decrypt(char* password);
00065
00066 #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

<i>id</i>	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.](#)

```
00001
00007
00008 #ifndef DOCTOR_H
00009 #define DOCTOR_H
00010
00011 #include "hospital.h"
00012
00017 int doctor_save_to_file(void);
00018
00023 int doctor_load_from_file(void);
00024
00029 int doctor_generate_id(void);
00030
00034 void doctor_view_all(void);
00035
00039 void doctor_view_one(void);
00040
00044 void doctor_view(void);
00045
00049 void doctor_search_by_id(void);
00050
00054 void doctor_search_by_name(void);
00055
00059 void doctor_search_by_phone(void);
00060
00064 void doctor_search_by(void);
00065
00071 int doctor_search_id(int id);
00072
00076 void doctor_update_using_id(void);
00077
00081 void doctor_deactivate_account(void);
00082
00086 void doctor_view_discharged(void);
00087
00088 #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

<i>doctor↔ _id</i>	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

<i>doctor↔ _id</i>	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

<i>doctor_id</i>	The doctor's ID.
<i>doctor_name</i>	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

<i>doctor↔ _id</i>	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

<i>doctor↔ _id</i>	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

<i>doctor↵ _id</i>	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

<i>doctor↵ _id</i>	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

<i>doctor_id</i>	The doctor's ID.
<i>today_date</i>	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.](#)

```
00001
00007
00008 #ifndef DOCTOR_PORTAL_H
00009 #define DOCTOR_PORTAL_H
00010
00011 #include "hospital.h"
00012
00017 void doctor_portal_view_appointments(int doctor_id);
00018
00023 void doctor_portal_view_pending(int doctor_id);
00024
00030 void doctor_portal_view_today(int doctor_id, const char* today_date);
00031
```

```

00035 void doctor_portal_view_patient(void);
00036
00041 void doctor_portal_complete_appointment(int doctor_id);
00042
00047 void doctor_portal_cancel_appointment(int doctor_id);
00048
00053 void doctor_portal_update_availability(int doctor_id);
00054
00059 void doctor_portal_view_profile(int doctor_id);
00060
00066 void doctor_portal_menu(int doctor_id, const char* doctor_name);
00067
00068 #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.](#)

```

00001
00008
00009 #ifndef HOSPITAL_H
00010 #define HOSPITAL_H
00011
00012 #include <stdbool.h>
00013
00014 /*
00015  *=====
00016  *                                     CONSTANTS
00017  *=====
00018  */
00019
00020 #define UI_SIZE          72
00021
00022 #define MAX_PATIENTS     100
00023 #define MAX_DOCTORS      20
00024 #define MAX_RECEPTIONISTS 20
00025 #define MAX_USERS        50
00026 #define MAX_APPOINTMENTS 200
00027
00028 #define NAME_SIZE        50
00029 #define PHONE_SIZE       15
00030 #define EMAIL_SIZE       50
00031 #define ADDRESS_SIZE     100
00032 #define SPEC_SIZE        30      /* Specialization */
00033 #define BLOOD_SIZE       5       /* Blood group */
00034 #define AGE_SIZE         5       /* Age */
00035 #define USERNAME_SIZE    30
00036 #define PASSWORD_SIZE    50
00037 #define GENDER_SIZE      10
00038 #define STATUS_SIZE      10
00039 #define ROOM_SIZE        10
00040
00041 #define ID_LINE_SIZE      20
00042 #define NAME_LINE_SIZE    NAME_SIZE + 20
00043 #define PHONE_LINE_SIZE   PHONE_SIZE + 20
00044 #define ADDRESS_LINE_SIZE ADDRESS_SIZE + 20
00045 #define BLOOD_LINE_SIZE   BLOOD_SIZE + 20
00046 #define GENDER_LINE_SIZE  GENDER_SIZE + 20
00047 #define STATUS_LINE_SIZE  STATUS_SIZE + 20
00048 #define SPEC_LINE_SIZE    SPEC_SIZE + 20
00049 #define ROOM_LINE_SIZE    ROOM_SIZE + 20
00050 #define AGE_LINE_SIZE     AGE_SIZE + 20
00051 #define PHONE_LINE_SIZE   PHONE_SIZE + 20
00052 #define EMAIL_LINE_SIZE   EMAIL_SIZE + 20
00053 #define BLOOD_LINE_SIZE   BLOOD_SIZE + 20
00054 #define GENDER_LINE_SIZE  GENDER_SIZE + 20
00055 #define STATUS_LINE_SIZE  STATUS_SIZE + 20
00056
00057 #define DATA_DIR         "data/"
00058 #define PATIENTS_FILE     "data/patients.dat"
00059 #define DOCTORS_FILE      "data/doctors.dat"
00060 #define RECEPTIONISTS_FILE "data/receptionists.dat"
00061 #define USERS_FILE        "data/users.dat"
00062 #define APPOINTMENTS_FILE "data/appointments.dat"
00063
00064 #define PATIENT_ID_START   1001
00065 #define DOCTOR_ID_START   2001
00066 #define ADMIN_ID_START    3001
00067 #define RECEPTIONIST_ID_START 4001
00068 #define APPOINTMENT_ID_START 5001
00069
00070 #define REASON_SIZE       100
00071 #define TIME_SIZE         10
00072 #define DATE_SIZE        15
00073
00074
00075 #define VERSION "1.0.0"
00076
00077 /*
00078  *=====
00079  *                                     ENUMERATIONS
00080  *=====
00081  */
00082
00083 typedef enum {
00084     ROLE_ADMIN,
00085     ROLE_DOCTOR,
00086     ROLE_RECEPTIONIST,
00087     ROLE_PATIENT
00088 } UserRole;

```

```

00089
00090 typedef enum {
00091     MALE,
00092     FEMALE
00093 } Gender;
00094
00095 typedef enum {
00096     APPT_PENDING,
00097     APPT_CONFIRMED,
00098     APPT_COMPLETED,
00099     APPT_CANCELLED
00100 } AppointmentStatus;
00101
00102 /*
00103  *=====
00104  *                                STRUCTURES
00105  *=====
00106  */
00107
00108 typedef struct {
00109     int id;
00110     char name[NAME_SIZE];
00111     int age;
00112     Gender gender;
00113     char phone[PHONE_SIZE];
00114     char address[ADDRESS_SIZE];
00115     char blood_group[BLOOD_SIZE];
00116     bool is_active;
00117 } Patient;
00118
00119 typedef struct {
00120     int id;
00121     char name[NAME_SIZE];
00122     char phone[PHONE_SIZE];
00123     char email[EMAIL_SIZE];
00124     char specialization[SPEC_SIZE];
00125     int room_number;
00126     bool is_available;
00127     bool is_active;
00128 } Doctor;
00129
00130 typedef struct {
00131     int id;
00132     char username[USERNAME_SIZE];
00133     char password[PASSWORD_SIZE];
00134     UserRole role;
00135     bool is_active;
00136 } User;
00137
00138 typedef struct {
00139     int id;
00140     int patient_id;
00141     int doctor_id;
00142     char date[DATE_SIZE];           // "DD-MM-YYYY"
00143     char time_slot[TIME_SIZE];     // "10:00 AM"
00144     char reason[REASON_SIZE];
00145     AppointmentStatus status;
00146 } Appointment;
00147
00148 typedef struct {
00149     int id;
00150     char name[NAME_SIZE];
00151     char phone[PHONE_SIZE];
00152     char email[EMAIL_SIZE];
00153     bool is_available;
00154     bool is_active;
00155 } Receptionist;
00156
00157 /*
00158  *=====
00159  *                                GLOBAL DECLARATIONS
00160  *=====
00161  */
00162
00163 /* Global declarations for data arrays */
00164 extern Patient patients[MAX_PATIENTS];
00165 extern Doctor doctors[MAX_DOCTORS];
00166 extern Receptionist receptionists[MAX_RECEPTIONISTS];
00167 extern User users[MAX_USERS];
00168 extern Appointment appointments[MAX_APPOINTMENTS];
00169
00170 /* Count variables */
00171 extern int patient_count;
00172 extern int doctor_count;
00173 extern int receptionist_count;
00174 extern int user_count;
00175 extern int patient_available;

```

```
00176 extern int patient_unavailable;
00177 extern int doctor_available;
00178 extern int doctor_unavailable;
00179 extern int receptionist_available;
00180 extern int receptionist_unavailable;
00181 extern int user_available;
00182 extern int user_unavailable;
00183 extern int appointment_count;
00184
00185 /* Current user */
00186 extern User* current_user;
00187
00191 void hospital_init(void);
00192
00196 void show_about(void);
00197
00201 void print_help(const char* program_name);
00202
00206 void print_version(void);
00207
00211 void ensure_data_dir(void);
00212
00213 #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

<i>id</i>	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.](#)

```
00001
00007
00008 #ifndef PATIENT_H
00009 #define PATIENT_H
00010
00011 #include "hospital.h"
00012
00017 int patient_save_to_file(void);
00018
00023 int patient_load_from_file(void);
00024
00029 int patient_generate_id(void);
00030
00034 void patient_add(void);
00035
00039 void patient_search_by_id(void);
00040
00044 void patient_search_by_name(void);
00045
00049 void patient_search_by_phone(void);
00050
00054 void patient_search_by(void);
00055
00059 void patient_update_using_id(void);
00060
00064 void patient_discharge(void);
00065
00069 void patient_view_all(void);
00070
00074 void patient_view_discharged(void);
00075
00081 int patient_search_id(int id);
00082
00083 #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

<i>id</i>	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.](#)

```
00001
00005
00006 #ifndef RECEPTIONIST_H
00007 #define RECEPTIONIST_H
00008
00009 #include "hospital.h"
00010
00015 int receptionist_save_to_file(void);
00016
00021 int receptionist_load_from_file(void);
00022
00028 int receptionist_search_id(int id);
00029
00033 void receptionist_view_all(void);
00034
00038 void receptionist_view_discharged(void);
00039
00043 void receptionist_discharge(void);
00044
00048 void receptionist_patient_menu(void);
00049
00053 void receptionist_appointment_menu(void);
00054
00058 void receptionist_menu(void);
00059
00060 #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[?251"
```

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

<i>doctor</i>	The doctor to print.
<i>index</i>	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

<i>message</i>	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

<i>title</i>	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

<i>message</i>	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

<i>title</i>	The title of the menu.
<i>items</i>	The array of menu items.
<i>item_count</i>	The number of menu items.
<i>box_width</i>	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

<i>patient</i>	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

<i>receptionist</i>	The receptionist to print.
<i>index</i>	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

<i>message</i>	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

<i>message</i>	The warning message.
----------------	----------------------

Definition at line 47 of file [ui.c](#).

6.18 ui.h

[Go to the documentation of this file.](#)

```
00001
00008
00009 #ifndef UI_H
00010 #define UI_H
00011
00012 #include "hospital.h"
00013
00014 /*
00015  *=====
00016  *                      ANSI COLOR CODES
00017  *=====
00018  */
00019
00020 /* Basic Foreground Colors */
00021 #define BLACK      "\033[30m"
00022 #define RED        "\033[31m"
00023 #define GREEN      "\033[32m"
00024 #define BLUE       "\033[34m"
00025 #define YELLOW     "\033[33m"
00026 #define MAGENTA    "\033[35m"
00027 #define CYAN       "\033[36m"
00028 #define WHITE      "\033[37m"
00029
00030 /* Bright Foreground Colors */
00031 #define BRIGHT_BLACK "\033[90m"
00032 #define BRIGHT_RED   "\033[91m"
00033 #define BRIGHT_GREEN "\033[92m"
00034 #define BRIGHT_YELLOW "\033[93m"
00035 #define BRIGHT_BLUE  "\033[94m"
00036 #define BRIGHT_MAGENTA "\033[95m"
00037 #define BRIGHT_CYAN  "\033[96m"
00038 #define BRIGHT_WHITE "\033[97m"
00039
00040 /* Extended Foreground Colors (256-color palette) */
00041 #define LIME          "\033[38;5;154m"
00042 #define SOFT_CYAN     "\033[38;5;159m"
00043 #define SOFT_TEAL     "\033[38;5;44m"
00044 #define SOFT_GREEN    "\033[38;5;120m"
00045 #define SOFT_RED      "\033[38;5;124m"
00046 #define SOFT_BLUE     "\033[38;5;75m"
00047 #define SOFT_GRAY     "\033[38;5;250m"
00048 #define SOFT_YELLOW   "\033[38;5;187m"
00049
00050 /* Neon Accent Colors */
00051 #define NEON_TEAL     "\033[38;5;51m"
```

```

00052 #define NEON_PINK      "\033[38;5;205m"
00053 #define NEON_PURPLE    "\033[38;5;141m"
00054 #define NEON_ORANGE    "\033[38;5;208m"
00055
00056 /* Status Colors */
00057 #define STATUS_GOOD     "\033[38;5;114m"
00058 #define STATUS_WARN     "\033[38;5;214m"
00059 #define STATUS_BAD      "\033[38;5;160m"
00060 #define STATUS_INFO     "\033[38;5;75m"
00061
00062 /* Muted/Dim Text Colors */
00063 #define TEXT_MUTED      "\033[38;5;244m"
00064 #define TEXT_DIM        "\033[38;5;240m"
00065
00066 /* Basic Background Colors */
00067 #define BG_BLACK        "\033[40m"
00068 #define BG_WHITE        "\033[47m"
00069
00070 /* Bright Background Colors */
00071 #define BG_BRIGHT_BLACK "\033[100m"
00072 #define BG_BRIGHT_RED   "\033[101m"
00073 #define BG_BRIGHT_GREEN "\033[102m"
00074 #define BG_BRIGHT_YELLOW "\033[103m"
00075
00076 /* Extended Background Colors */
00077 #define BG_NEON_PURPLE  "\033[48;5;57m"
00078 #define BG_NEON_TEAL    "\033[48;5;43m"
00079
00080 /* Text Styles */
00081 #define RESET           "\033[0m"
00082 #define BOLD            "\033[1m"
00083 #define UNDERLINE       "\033[4m"
00084
00085 /* Cursor */
00086 #define HIDE_CURSOR     "\033[?25l"
00087 #define SHOW_CURSOR     "\033[?25h"
00088
00089 /*
00090 *=====
00091 *                               FUNCTION PROTOTYPES
00092 *=====
00093 */
00094
00095 void ui_clear_screen(void);
00099
00100 void ui_pause(void);
00105
00106 void ui_print_header(const char *title);
00112
00113 void ui_print_success(const char *message);
00119
00120 void ui_print_error(const char *message);
00126
00127 void ui_print_warning(const char *message);
00133
00134 void ui_print_info(const char *message);
00140
00141 void ui_print_banner(void);
00145
00146 void ui_print_menu(const char *title, const char *items[], int item_count, int box_width);
00155
00156 void ui_print_patient(Patient patient, int index);
00162
00163 void ui_print_doctor(Doctor doctor, int index);
00170
00171 void ui_print_receptionist(Receptionist receptionist, int index);
00178
00179 void ui_dummy_loading(int time);
00183
00184 #endif
00185

```

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

<i>name</i>	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

<i>str</i>	The buffer to store the string.
<i>size</i>	The maximum number of characters to read.

Returns

Pointer to string on success, NULL on failure.

Definition at line 61 of file [utls.c](#).

6.19.2.8 utls_is_valid_address()

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

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

Parameters

<i>address</i>	The address string to validate.
----------------	---------------------------------

Returns

true if valid, false otherwise.

Definition at line 204 of file [utls.c](#).

6.19.2.9 utls_is_valid_blood_group()

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

Validates if a blood group is in correct format.

Parameters

<i>blood_group</i>	The blood group string to validate.
--------------------	-------------------------------------

Returns

true if valid, false otherwise.

Definition at line 186 of file [utls.c](#).

6.19.2.10 utls_is_valid_email()

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

Validates if an email contains @ and .

Parameters

<i>email</i>	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

<i>gender</i>	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

<i>id</i>	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

<i>name</i>	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

<i>phone</i>	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

<i>str</i>	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.](#)

```

00001
00007
00008 #ifndef UTILS_H
00009 #define UTILS_H
00010
00011 #include <stddef.h>
00012 #include <stdbool.h>
00013 #include "hospital.h"
00014
00018 void utils_clear_input_buffer(void);
00019
00025 int utils_get_int(void);
00026
00032 float utils_get_float(void);
00033
00039 double utils_get_double(void);
00040
00046 char utils_get_char(void);
00047
00056 char* utils_get_string(char *str, size_t size);
00057
00063
00071 bool utils_is_valid_phone(const char *phone);
00072
00080 bool utils_is_valid_email(const char *email);
00081
00089 bool utils_is_valid_name(const char *name);
00090
00098 bool utils_is_valid_id(int id, UserRole role);
00099
00107 bool utils_is_valid_gender(const char *gender);
00108
00116 char* utils_str_to_upper(char *str);
00117
00126 char* utils_fix_name(char *name);
00127
00135 bool utils_is_valid_blood_group(const char *blood_group);
00136
00145 bool utils_is_valid_address(const char *address);
00146
00147 #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
00002
00003 #include <stdio.h>
00004 #include <string.h>
00005 #include "../include/admin.h"
00006 #include "../include/patient.h"
00007 #include "../include/doctor.h"
00008 #include "../include/receptionist.h"
00009 #include "../include/auth.h"
00010 #include "../include/utils.h"
00011 #include "../include/ui.h"
00012 #include "../include/hospital.h"
00013
00014 /*
00015 *=====
00016 *                               DISCHARGED PATIENT FUNCTIONS
00017 *=====
00018 */
00019
00020 void admin_view_discharged_patients(void) {
00021     int count = 0;
00022     ui_clear_screen();
00023     ui_print_banner();
00024
00025     if (patient_unavailable == 0) {
00026         const char* menu_items[] = {"No discharged patients found!"};
00027         ui_print_menu("Discharged Patients", menu_items, 1, UI_SIZE);
00028         ui_pause();
00029         return;
00030     }
00031
00032     for (int i = 0; i < patient_count; i++) {
00033         if (!patients[i].is_active) {
00034             ui_print_patient(patients[i], count++);
00035         }
00036     }
00037     ui_pause();
00038 }
00039
00040 void admin_search_discharged_by_id(void) {
00041     int id;
```



```

00048
00049     do {
00050         ui_clear_screen();
00051         ui_print_banner();
00052
00053         const char* menu_items[] = {
00054             "Enter Patient ID (0 to cancel): ",
00055             "» "
00056         };
00057
00058         ui_print_menu("Search Discharged Patient", menu_items, 2, UI_SIZE);
00059         id = utils_get_int();
00060
00061         if (id == 0) return;
00062
00063         if (!utils_is_valid_id(id, ROLE_PATIENT)) {
00064             ui_print_error("Invalid ID!");
00065             ui_pause();
00066             continue;
00067         }
00068
00069         for (int i = 0; i < patient_count; i++) {
00070             if (patients[i].id == id && !patients[i].is_active) {
00071                 ui_clear_screen();
00072                 ui_print_banner();
00073                 ui_print_patient(patients[i], i);
00074                 ui_pause();
00075                 return;
00076             }
00077         }
00078         ui_print_error("Discharged patient not found with that ID!");
00079         ui_pause();
00080         return;
00081     } while (1);
00082 }
00083
00084 void admin_search_discharged_by_name(void) {
00085     char name[NAME_SIZE];
00086
00087     do {
00088         ui_clear_screen();
00089         ui_print_banner();
00090
00091         const char* menu_items[] = {
00092             "Enter Patient Name: ",
00093             "» "
00094         };
00095
00096         ui_print_menu("Search Discharged Patient", menu_items, 2, UI_SIZE);
00097         utils_get_string(name, NAME_SIZE);
00098
00099         if (!utils_is_valid_name(name)) {
00100             ui_print_error("Invalid name!");
00101             ui_pause();
00102             continue;
00103         }
00104
00105         utils_fix_name(name);
00106         for (int i = 0; i < patient_count; i++) {
00107             if (strcmp(patients[i].name, name) == 0 && !patients[i].is_active) {
00108                 ui_clear_screen();
00109                 ui_print_banner();
00110                 ui_print_patient(patients[i], i);
00111                 ui_pause();
00112                 return;
00113             }
00114         }
00115         ui_print_error("Discharged patient not found with that name!");
00116         ui_pause();
00117         return;
00118     } while (1);
00119 }
00120
00121 void admin_search_discharged(void) {
00122     int choice;
00123
00124     do {
00125         ui_clear_screen();
00126         ui_print_banner();
00127
00128         const char* menu_items[] = {
00129             "Search by ID",
00130             "Search by Name",
00131             "Back",
00132             "» "
00133         };
00134

```

```

00135         ui_print_menu("Search Discharged Patients", menu_items, 4, UI_SIZE);
00136         choice = utils_get_int();
00137
00138         switch (choice) {
00139             case 1:
00140                 admin_search_discharged_by_id();
00141                 break;
00142             case 2:
00143                 admin_search_discharged_by_name();
00144                 break;
00145             case 3:
00146                 return;
00147             default:
00148                 ui_print_error("Invalid choice!");
00149                 ui_pause();
00150         }
00151     } while (choice != 3);
00152 }
00153
00154 void admin_delete_patient(void) {
00155     int id;
00156
00157     while (1) {
00158         ui_clear_screen();
00159         ui_print_banner();
00160
00161         const char* enter_id_items[] = {
00162             "Enter patient ID to permanently delete (0 to cancel): ",
00163             "» "
00164         };
00165
00166         ui_print_menu("Delete Patient Permanently", enter_id_items, 2, UI_SIZE);
00167         id = utils_get_int();
00168
00169         if (id == 0) return;
00170
00171         if (!utils_is_valid_id(id, ROLE_PATIENT)) {
00172             ui_print_error("Invalid ID!");
00173             ui_pause();
00174             continue;
00175         }
00176         break;
00177     }
00178
00179     // Find the patient
00180     int index = -1;
00181     for (int i = 0; i < patient_count; i++) {
00182         if (patients[i].id == id && !patients[i].is_active) {
00183             index = i;
00184             break;
00185         }
00186     }
00187
00188     if (index == -1) {
00189         ui_print_error("Discharged patient not found! Only discharged patients can be permanently
00190         deleted.");
00191         ui_pause();
00192         return;
00193     }
00194
00195     ui_clear_screen();
00196     ui_print_banner();
00197     ui_print_patient(patients[index], index);
00198
00199     const char* menu[] = {
00200         "Confirm Permanent Delete",
00201         "Cancel",
00202         "» "
00203     };
00204
00205     ui_print_menu("Delete Patient Permanently", menu, 3, UI_SIZE);
00206     int input = utils_get_int();
00207
00208     if (input == 1) {
00209         // Shift all patients after this index
00210         for (int i = index; i < patient_count - 1; i++) {
00211             patients[i] = patients[i + 1];
00212         }
00213         patient_count--;
00214         patient_unavailable--;
00215
00216         ui_print_success("Patient permanently deleted from the system!");
00217         patient_save_to_file();
00218         ui_pause();
00219     } else {
00220         ui_print_info("Deletion cancelled.");
00221         ui_pause();
00222     }

```

```

00221     }
00222 }
00223
00224 void admin_patient_menu(void) {
00225     int choice;
00226
00227     do {
00228         ui_clear_screen();
00229         ui_print_banner();
00230
00231         const char* menu_items[] = {
00232             "Add Patient",
00233             "View Active Patients",
00234             "View Discharged Patients",
00235             "Search Active Patient",
00236             "Search Discharged Patient",
00237             "Update Patient",
00238             "Discharge Patient",
00239             "Delete Patient Permanently",
00240             "Back to Admin Menu",
00241             "» "
00242         };
00243
00244         ui_print_menu("Admin Patient Management", menu_items, 10, UI_SIZE);
00245         choice = utils_get_int();
00246
00247         switch (choice) {
00248             case 1:
00249                 patient_add();
00250                 patient_save_to_file();
00251                 break;
00252             case 2:
00253                 patient_view_all();
00254                 break;
00255             case 3:
00256                 admin_view_discharged_patients();
00257                 break;
00258             case 4:
00259                 patient_search_by();
00260                 break;
00261             case 5:
00262                 admin_search_discharged();
00263                 break;
00264             case 6:
00265                 patient_update_using_id();
00266                 patient_save_to_file();
00267                 break;
00268             case 7:
00269                 patient_discharge();
00270                 patient_save_to_file();
00271                 break;
00272             case 8:
00273                 admin_delete_patient();
00274                 break;
00275             case 9:
00276                 ui_print_info("Returning to admin menu...");
00277                 ui_pause();
00278                 break;
00279             default:
00280                 ui_print_error("Invalid choice!");
00281                 ui_pause();
00282         }
00283     } while (choice != 9);
00284 }
00285
00286 /*
00287 *=====
00288 *                               INACTIVE DOCTOR FUNCTIONS
00289 *=====
00290 */
00291
00292 void admin_view_discharged_doctors(void) {
00293     int count = 0;
00294     ui_clear_screen();
00295     ui_print_banner();
00296
00297     if (doctor_unavailable == 0) {
00298         const char* menu_items[] = {"No inactive doctors found!"};
00299         ui_print_menu("Inactive Doctors", menu_items, 1, UI_SIZE);
00300         ui_pause();
00301         return;
00302     }
00303
00304     for (int i = 0; i < doctor_count; i++) {
00305         if (!doctors[i].is_active) {
00306             ui_print_doctor(doctors[i], count++);
00307         }
00308     }

```

```

00308     }
00309     ui_pause();
00310 }
00311
00312 void admin_delete_doctor(void) {
00313     int id;
00314
00315     while (1) {
00316         ui_clear_screen();
00317         ui_print_banner();
00318
00319         const char* enter_id_items[] = {
00320             "Enter doctor ID to permanently delete (0 to cancel): ",
00321             "» "
00322         };
00323
00324         ui_print_menu("Delete Doctor Permanently", enter_id_items, 2, UI_SIZE);
00325         id = utils_get_int();
00326
00327         if (id == 0) return;
00328
00329         if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
00330             ui_print_error("Invalid ID!");
00331             ui_pause();
00332             continue;
00333         }
00334         break;
00335     }
00336
00337     // Find the doctor
00338     int index = -1;
00339     for (int i = 0; i < doctor_count; i++) {
00340         if (doctors[i].id == id && !doctors[i].is_active) {
00341             index = i;
00342             break;
00343         }
00344     }
00345
00346     if (index == -1) {
00347         ui_print_error("Inactive doctor not found! Only inactive doctors can be permanently
00348         deleted.");
00349         ui_pause();
00350         return;
00351     }
00352
00353     ui_clear_screen();
00354     ui_print_banner();
00355     ui_print_doctor(doctors[index], index);
00356
00357     const char* menu[] = {
00358         "Confirm Permanent Delete",
00359         "Cancel",
00360         "» "
00361     };
00362
00363     ui_print_menu("Delete Doctor Permanently", menu, 3, UI_SIZE);
00364     int input = utils_get_int();
00365
00366     if (input == 1) {
00367         // Shift all doctors after this index
00368         for (int i = index; i < doctor_count - 1; i++) {
00369             doctors[i] = doctors[i + 1];
00370         }
00371         doctor_count--;
00372         doctor_unavailable--;
00373
00374         ui_print_success("Doctor permanently deleted from the system!");
00375         doctor_save_to_file();
00376         ui_pause();
00377     } else {
00378         ui_print_info("Deletion cancelled.");
00379         ui_pause();
00380     }
00381 }
00382 void admin_doctor_menu(void) {
00383     int choice;
00384
00385     do {
00386         ui_clear_screen();
00387         ui_print_banner();
00388
00389         const char* menu_items[] = {
00390             "View Active Doctors",
00391             "View Inactive Doctors",
00392             "Search Doctor",
00393             "Update Doctor",

```

```

00394         "Deactivate Doctor",
00395         "Delete Doctor Permanently",
00396         "Back to Admin Menu",
00397         "» "
00398     };
00399
00400     ui_print_menu("Admin Doctor Management", menu_items, 8, UI_SIZE);
00401     choice = utils_get_int();
00402
00403     switch (choice) {
00404     case 1:
00405         doctor_view_all();
00406         break;
00407     case 2:
00408         admin_view_discharged_doctors();
00409         break;
00410     case 3:
00411         doctor_search_by();
00412         break;
00413     case 4:
00414         doctor_update_using_id();
00415         doctor_save_to_file();
00416         break;
00417     case 5:
00418         doctor_deactivate_account();
00419         doctor_save_to_file();
00420         break;
00421     case 6:
00422         admin_delete_doctor();
00423         break;
00424     case 7:
00425         ui_print_info("Returning to admin menu...");
00426         ui_pause();
00427         break;
00428     default:
00429         ui_print_error("Invalid choice!");
00430         ui_pause();
00431     }
00432     } while (choice != 7);
00433 }
00434
00435 void admin_receptionist_menu(void) {
00436     int choice;
00437
00438     do {
00439         ui_clear_screen();
00440         ui_print_banner();
00441
00442         const char* menu_items[] = {
00443             "View Active Receptionists",
00444             "View Inactive Receptionists",
00445             "Deactivate Receptionist",
00446             "Delete Receptionist Permanently",
00447             "Back to Admin Menu",
00448             "» "
00449         };
00450
00451         ui_print_menu("Admin Receptionist Management", menu_items, 6, UI_SIZE);
00452         choice = utils_get_int();
00453
00454         switch (choice) {
00455         case 1:
00456             receptionist_view_all();
00457             break;
00458         case 2:
00459             receptionist_view_discharged();
00460             break;
00461         case 3:
00462             receptionist_discharge();
00463             break;
00464         case 4:
00465             ui_print_info("Delete receptionist permanently - Coming soon!");
00466             ui_pause();
00467             break;
00468         case 5:
00469             ui_print_info("Returning to admin menu...");
00470             ui_pause();
00471             break;
00472         default:
00473             ui_print_error("Invalid choice!");
00474             ui_pause();
00475         }
00476     } while (choice != 4);
00477 }
00478
00479 /*
00480  *=====

```

```

00481  *                                     MAIN ADMIN MENU
00482  *=====
00483  */
00484
00485 void admin_main_menu(void) {
00486     int choice;
00487
00488     do {
00489         ui_clear_screen();
00490         ui_print_banner();
00491
00492         const char* menu_items[] = {
00493             "Register New User",
00494             "Patient Management",
00495             "Doctor Management",
00496             "Receptionist Management",
00497             "Logout",
00498             "» "
00499         };
00500
00501         ui_print_menu("Admin Portal", menu_items, 6, UI_SIZE);
00502         choice = utils_get_int();
00503
00504         switch (choice) {
00505             case 1:
00506                 auth_register_user();
00507                 break;
00508             case 2:
00509                 admin_patient_menu();
00510                 break;
00511             case 3:
00512                 admin_doctor_menu();
00513                 break;
00514             case 4:
00515                 admin_receptionist_menu();
00516                 break;
00517             case 5:
00518                 ui_print_info("Logging out...");
00519                 ui_pause();
00520                 break;
00521             default:
00522                 ui_print_error("Invalid choice!");
00523                 ui_pause();
00524         }
00525     } while (choice != 5);
00526 }

```

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

<code>appt_id</code>	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

<i>id</i>	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

<i>status</i>	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

<i>appt_id</i>	The appointment ID.
<i>status</i>	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

<i>doctor</i> ↔ <i>_id</i>	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

<i>appt</i>	The appointment to print.
<i>index</i>	The display index.

Definition at line 229 of file [appointment.c](#).

6.24 appointment.c

[Go to the documentation of this file.](#)

```
00001
00007
00008 #include <stdio.h>
00009 #include <string.h>
00010 #include "../include/appointment.h"
00011 #include "../include/patient.h"
00012 #include "../include/doctor.h"
00013 #include "../include/utils.h"
00014 #include "../include/ui.h"
00015 #include "../include/hospital.h"
00016
00017 int appointment_save_to_file(void) {
00018     FILE* file = fopen(APPOINTMENTS_FILE, "wb");
00019     if (file == NULL) {
00020         return -1;
00021     }
00022     if (fwrite(&appointment_count, sizeof(int), 1, file) != 1 ||
00023         fwrite(appointments, sizeof(Appointment), appointment_count, file) !=
00024         (size_t)appointment_count) {
00025         fclose(file);
00026         return -1;
00027     }
00028     fclose(file);
00029     return 0;
00030 }
00031
00031 int appointment_load_from_file(void) {
00032     FILE* file = fopen(APPOINTMENTS_FILE, "rb");
00033     if (file == NULL) {
00034         return -1;
00035     }
00036     if (fread(&appointment_count, sizeof(int), 1, file) != 1) {
00037         fclose(file);
00038         return -1;
00039     }
00040     if (appointment_count < 0 || appointment_count > MAX_APPOINTMENTS) {
00041         fclose(file);
00042         appointment_count = 0;
00043         return -1;
00044     }
00045     if (fread(appointments, sizeof(Appointment), appointment_count, file) !=
00046         (size_t)appointment_count) {
00047         fclose(file);
00048         appointment_count = 0;
00049         return -1;
00050     }
00051     fclose(file);
00052     return 0;
00053 }
```

```

00054 int appointment_generate_id(void) {
00055     return APPOINTMENT_ID_START + appointment_count;
00056 }
00057
00058 const char* appointment_status_str(AppointmentStatus status) {
00059     switch (status) {
00060         case APPT_PENDING: return "Pending";
00061         case APPT_CONFIRMED: return "Confirmed";
00062         case APPT_COMPLETED: return "Completed";
00063         case APPT_CANCELLED: return "Cancelled";
00064         default: return "Unknown";
00065     }
00066 }
00067
00068 int appointment_search_id(int id) {
00069     for (int i = 0; i < appointment_count; i++) {
00070         if (appointments[i].id == id) {
00071             return i;
00072         }
00073     }
00074     return -1;
00075 }
00076
00077 void appointment_create(void) {
00078     if (appointment_count >= MAX_APPOINTMENTS) {
00079         ui_print_error("Error: Maximum appointment limit reached!");
00080         ui_pause();
00081         return;
00082     }
00083
00084     Appointment new_appt;
00085     new_appt.id = appointment_generate_id();
00086     new_appt.status = APPT_PENDING;
00087
00088     char patient_line[70], doctor_line[70], date_line[70], time_line[70];
00089
00090     // Step 1: Get Patient ID
00091     while (1) {
00092         ui_clear_screen();
00093         ui_print_banner();
00094         const char* step1[] = {"Patient ID:", "» "};
00095         ui_print_menu("Create Appointment", step1, 2, UI_SIZE);
00096         new_appt.patient_id = utils_get_int();
00097
00098         if (!utils_is_valid_id(new_appt.patient_id, ROLE_PATIENT)) {
00099             ui_print_error("Invalid patient ID!");
00100             ui_pause();
00101             continue;
00102         }
00103
00104         int idx = patient_search_id(new_appt.patient_id);
00105         if (idx == -1 || !patients[idx].is_active) {
00106             ui_print_error("Patient not found or inactive!");
00107             ui_pause();
00108             continue;
00109         }
00110         snprintf(patient_line, sizeof(patient_line), "Patient: %s (ID: %d)",
00111                 patients[idx].name, new_appt.patient_id);
00112         break;
00113     }
00114
00115     // Step 2: Get Doctor ID
00116     while (1) {
00117         ui_clear_screen();
00118         ui_print_banner();
00119         const char* step2[] = {patient_line, "Doctor ID:", "» "};
00120         ui_print_menu("Create Appointment", step2, 3, UI_SIZE);
00121         new_appt.doctor_id = utils_get_int();
00122
00123         if (!utils_is_valid_id(new_appt.doctor_id, ROLE_DOCTOR)) {
00124             ui_print_error("Invalid doctor ID!");
00125             ui_pause();
00126             continue;
00127         }
00128
00129         int idx = doctor_search_id(new_appt.doctor_id);
00130         if (idx == -1 || !doctors[idx].is_active) {
00131             ui_print_error("Doctor not found or inactive!");
00132             ui_pause();
00133             continue;
00134         }
00135         if (!doctors[idx].is_available) {
00136             ui_print_warning("Warning: Doctor is marked as unavailable.");
00137             ui_pause();
00138         }
00139         snprintf(doctor_line, sizeof(doctor_line), "Doctor: Dr. %s (ID: %d)",
00140                 doctors[idx].name, new_appt.doctor_id);

```

```

00141         break;
00142     }
00143
00144     // Step 3: Get Date
00145     while (1) {
00146         ui_clear_screen();
00147         ui_print_banner();
00148         const char* step3[] = {patient_line, doctor_line, "Date (DD-MM-YYYY):", ">> "};
00149         ui_print_menu("Create Appointment", step3, 4, UI_SIZE);
00150         utils_get_string(new_appt.date, DATE_SIZE);
00151
00152         if (strlen(new_appt.date) > 0) {
00153             snprintf(date_line, sizeof(date_line), "Date: %s", new_appt.date);
00154             break;
00155         }
00156         ui_print_error("Date cannot be empty!");
00157         ui_pause();
00158     }
00159
00160     // Step 4: Get Time Slot
00161     while (1) {
00162         ui_clear_screen();
00163         ui_print_banner();
00164         const char* step4[] = {patient_line, doctor_line, date_line, "Time (e.g. 10:00 AM):", ">> "};
00165         ui_print_menu("Create Appointment", step4, 5, UI_SIZE);
00166         utils_get_string(new_appt.time_slot, TIME_SIZE);
00167
00168         if (strlen(new_appt.time_slot) > 0) {
00169             snprintf(time_line, sizeof(time_line), "Time: %s", new_appt.time_slot);
00170             break;
00171         }
00172         ui_print_error("Time cannot be empty!");
00173         ui_pause();
00174     }
00175
00176     // Step 5: Get Reason
00177     while (1) {
00178         ui_clear_screen();
00179         ui_print_banner();
00180         const char* step5[] = {patient_line, doctor_line, date_line, time_line, "Reason for visit:",
00181                                ">> "};
00182         ui_print_menu("Create Appointment", step5, 6, UI_SIZE);
00183         utils_get_string(new_appt.reason, REASON_SIZE);
00184
00185         if (strlen(new_appt.reason) > 0) {
00186             break;
00187         }
00188         ui_print_error("Reason cannot be empty!");
00189         ui_pause();
00190     }
00191
00192     // Step 6: Confirm
00193     char reason_line[120];
00194     snprintf(reason_line, sizeof(reason_line), "Reason: %s", new_appt.reason);
00195     const char* step6[] = {
00196         patient_line, doctor_line, date_line, time_line, reason_line,
00197         "Confirm (Y/N):", ">> "
00198     };
00199     ui_clear_screen();
00200     ui_print_banner();
00201     ui_print_menu("Create Appointment", step6, 7, UI_SIZE);
00202     char confirm = utils_get_char();
00203
00204     if (confirm != 'Y' && confirm != 'y') {
00205         ui_print_info("Appointment creation cancelled.");
00206         ui_pause();
00207         return;
00208     }
00209
00210     // Add to array
00211     appointments[appointment_count] = new_appt;
00212     appointment_count++;
00213
00214     ui_clear_screen();
00215     ui_print_banner();
00216
00217     char id_line[70];
00218     snprintf(id_line, sizeof(id_line), "Appointment ID: %d", new_appt.id);
00219     const char* success_items[] = {
00220         id_line, patient_line, doctor_line, date_line, time_line,
00221         "Status: Pending",
00222         "Appointment created successfully!"
00223     };
00224     ui_print_menu("Appointment Created", success_items, 7, UI_SIZE);
00225     ui_pause();
00226     appointment_save_to_file();

```

```

00227 }
00228
00229 void ui_print_appointment(Appointment appt, int index) {
00230     int p_idx = patient_search_id(appt.patient_id);
00231     int d_idx = doctor_search_id(appt.doctor_id);
00232
00233     char id_line[70];
00234     snprintf(id_line, sizeof(id_line), "Appointment ID: %d", appt.id);
00235
00236     char patient_line[70];
00237     if (p_idx != -1) {
00238         snprintf(patient_line, sizeof(patient_line), "Patient: %s (ID: %d)",
00239                 patients[p_idx].name, appt.patient_id);
00240     } else {
00241         snprintf(patient_line, sizeof(patient_line), "Patient ID: %d", appt.patient_id);
00242     }
00243
00244     char doctor_line[70];
00245     if (d_idx != -1) {
00246         snprintf(doctor_line, sizeof(doctor_line), "Doctor: Dr. %s", doctors[d_idx].name);
00247     } else {
00248         snprintf(doctor_line, sizeof(doctor_line), "Doctor ID: %d", appt.doctor_id);
00249     }
00250
00251     char date_line[70];
00252     snprintf(date_line, sizeof(date_line), "Date: %s", appt.date);
00253
00254     char time_line[70];
00255     snprintf(time_line, sizeof(time_line), "Time: %s", appt.time_slot);
00256
00257     char reason_line[120];
00258     snprintf(reason_line, sizeof(reason_line), "Reason: %s", appt.reason);
00259
00260     char status_line[70];
00261     snprintf(status_line, sizeof(status_line), "Status: %s", appointment_status_str(appt.status));
00262
00263     const char* items[] = {
00264         id_line, patient_line, doctor_line, date_line, time_line, reason_line, status_line, ""
00265     };
00266
00267     char title[70];
00268     snprintf(title, sizeof(title), "Appointment %d", index + 1);
00269     ui_print_menu(title, items, 8, 72);
00270 }
00271
00272 void appointment_view_by_doctor(int doctor_id) {
00273     int count = 0;
00274     ui_clear_screen();
00275     ui_print_banner();
00276
00277     for (int i = 0; i < appointment_count; i++) {
00278         if (appointments[i].doctor_id == doctor_id &&
00279             appointments[i].status != APPT_CANCELLED) {
00280             ui_print_appointment(appointments[i], count++);
00281         }
00282     }
00283
00284     if (count == 0) {
00285         const char* menu_items[] = {"No appointments found!"};
00286         ui_print_menu("My Appointments", menu_items, 1, UI_SIZE);
00287     }
00288     ui_pause();
00289 }
00290
00291 void appointment_update_status(int appt_id, AppointmentStatus status) {
00292     int idx = appointment_search_id(appt_id);
00293     if (idx == -1) {
00294         ui_print_error("Appointment not found!");
00295         ui_pause();
00296         return;
00297     }
00298
00299     appointments[idx].status = status;
00300     appointment_save_to_file();
00301     ui_print_success("Appointment status updated!");
00302     ui_pause();
00303 }
00304
00305 void appointment_cancel(int appt_id) {
00306     appointment_update_status(appt_id, APPT_CANCELLED);
00307 }

```

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

<i>required_role</i>	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

<i>password</i>	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

<i>password</i>	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.](#)

```
00001
00005
00006 #include <stdio.h>
00007 #include <string.h>
00008 #include "../include/auth.h"
00009 #include "../include/utlis.h"
00010 #include "../include/ui.h"
00011 #include "../include/hospital.h"
00012 #include "../include/receptionist.h"
00013 #include "../include/admin.h"
00014 #include "../include/doctor_portal.h"
00015 #include "../include/doctor.h"
00016
00017 int auth_save_to_file(void) {
00018     FILE* file = fopen(USERS_FILE, "wb");
00019     if (file == NULL) {
00020         return -1;
00021     }
00022     if (fwrite(&user_count, sizeof(int), 1, file) != 1) {
00023         fclose(file);
00024         return -1;
00025     }
00026     if (fwrite(users, sizeof(User), user_count, file) != (size_t)user_count) {
00027         fclose(file);
00028         return -1;
00029     }
00030     fclose(file);
00031     return 0;
00032 }
00033
00034 int auth_load_from_file(void) {
00035     FILE* file = fopen(USERS_FILE, "rb");
00036     if (file == NULL) {
00037         return -1;
00038     }
00039     if (fread(&user_count, sizeof(int), 1, file) != 1) {
00040         fclose(file);
00041         return -1;
00042     }
00043     if (user_count < 0 || user_count > MAX_USERS) {
00044         fclose(file);
00045         user_count = 0;
00046         return -1;
00047     }
00048     if (fread(users, sizeof(User), user_count, file) != (size_t)user_count) {
00049         fclose(file);
00050         user_count = 0;
00051         return -1;
00052     }
00053     fclose(file);
00054     return 0;
00055 }
00056
00057 void auth_init_default_admin(void) {
00058     if (user_count == 0) {
00059         users[0].id = ADMIN_ID_START;
00060         strcpy(users[0].username, "admin");
00061         strcpy(users[0].password, "admin123");
00062         encrypt(users[0].password);
00063         users[0].role = ROLE_ADMIN;
00064         users[0].is_active = true;
00065         user_count = 1;
00066         auth_save_to_file();
00067     }
}
```

```

00068 }
00069
00070 void auth_register_user(void) {
00071     if (user_count >= MAX_USERS) {
00072         ui_print_error("Maximum user limit reached!");
00073         ui_pause();
00074         return;
00075     }
00076
00077     User new_user;
00078     Doctor new_doctor;
00079     char username_line[80], password_line[80], role_line[80];
00080     char name_line[NAME_LINE_SIZE], phone_line[PHONE_LINE_SIZE], email_line[EMAIL_LINE_SIZE];
00081     char spec_line[SPEC_LINE_SIZE], room_line[ROOM_LINE_SIZE];
00082     int selected_role = 0;
00083
00084     // Step 1: Select role first
00085     while (1) {
00086         ui_clear_screen();
00087         ui_print_banner();
00088         const char* step1[] = {
00089             "Admin",
00090             "Receptionist",
00091             "Doctor",
00092             "» ",
00093         };
00094         ui_print_menu("Register User", step1, 4, UI_SIZE);
00095         selected_role = utils_get_int();
00096
00097         switch (selected_role) {
00098             case 0:
00099                 ui_print_info("Cancelled.");
00100                 ui_pause();
00101                 return;
00102             case 1:
00103                 new_user.role = ROLE_ADMIN;
00104                 snprintf(role_line, sizeof(role_line), "Role: Admin");
00105                 new_user.id = ADMIN_ID_START + user_count;
00106                 break;
00107             case 2:
00108                 new_user.role = ROLE_RECEPTIONIST;
00109                 snprintf(role_line, sizeof(role_line), "Role: Receptionist");
00110                 new_user.id = RECEPTIONIST_ID_START + user_count;
00111                 break;
00112             case 3:
00113                 if (doctor_count >= MAX_DOCTORS) {
00114                     ui_print_error("Maximum doctor limit reached!");
00115                     ui_pause();
00116                     return;
00117                 }
00118                 new_user.role = ROLE_DOCTOR;
00119                 snprintf(role_line, sizeof(role_line), "Role: Doctor");
00120                 new_user.id = DOCTOR_ID_START + doctor_count;
00121                 new_doctor.id = new_user.id;
00122                 break;
00123             default:
00124                 ui_print_error("Invalid role! Try again.");
00125                 ui_pause();
00126                 continue;
00127         }
00128         break;
00129     }
00130
00131     // Step 2: Get username
00132     while (1) {
00133         ui_clear_screen();
00134         ui_print_banner();
00135         const char* step2[] = {role_line, "Username:", "» "};
00136         ui_print_menu("Register User", step2, 3, UI_SIZE);
00137         utils_get_string(new_user.username, USERNAME_SIZE);
00138
00139         if (strlen(new_user.username) < 3) {
00140             ui_print_error("Username must be at least 3 characters!");
00141             ui_pause();
00142             continue;
00143         }
00144
00145         // Check if username exists
00146         bool exists = false;
00147         for (int i = 0; i < user_count; i++) {
00148             if (strcmp(users[i].username, new_user.username) == 0) {
00149                 exists = true;
00150                 break;
00151             }
00152         }
00153         if (exists) {
00154             ui_print_error("Username already exists!");

```



```

00155         ui_pause();
00156         continue;
00157     }
00158
00159     snprintf(username_line, sizeof(username_line), "Username: %s", new_user.username);
00160     break;
00161 }
00162
00163 // Step 3: Get password
00164 while (1) {
00165     ui_clear_screen();
00166     ui_print_banner();
00167     const char* step3[] = {role_line, username_line, "Password:", ">> "};
00168     ui_print_menu("Register User", step3, 4, UI_SIZE);
00169     utils_get_string(new_user.password, PASSWORD_SIZE);
00170
00171     if (strlen(new_user.password) < 4) {
00172         ui_print_error("Password must be at least 4 characters!");
00173         ui_pause();
00174         continue;
00175     }
00176     snprintf(password_line, sizeof(password_line), "Password: ****");
00177     break;
00178 }
00179
00180 // If Doctor, collect additional profile data
00181 if (selected_role == 3) {
00182     // Step 4: Get full name
00183     while (1) {
00184         ui_clear_screen();
00185         ui_print_banner();
00186         const char* step4[] = {role_line, username_line, "Full Name:", ">> "};
00187         ui_print_menu("Register Doctor", step4, 4, UI_SIZE);
00188         utils_get_string(new_doctor.name, NAME_SIZE);
00189
00190         if (!utils_is_valid_name(new_doctor.name)) {
00191             ui_print_error("Invalid name!");
00192             ui_pause();
00193             continue;
00194         }
00195         utils_fix_name(new_doctor.name);
00196         snprintf(name_line, sizeof(name_line), "Name: %s", new_doctor.name);
00197         break;
00198     }
00199
00200     // Step 5: Get phone
00201     while (1) {
00202         ui_clear_screen();
00203         ui_print_banner();
00204         const char* step5[] = {role_line, username_line, name_line, "Phone (11 digits):", ">> "};
00205         ui_print_menu("Register Doctor", step5, 5, UI_SIZE);
00206         utils_get_string(new_doctor.phone, PHONE_SIZE);
00207
00208         if (!utils_is_valid_phone(new_doctor.phone)) {
00209             ui_print_error("Invalid phone!");
00210             ui_pause();
00211             continue;
00212         }
00213         snprintf(phone_line, sizeof(phone_line), "Phone: %s", new_doctor.phone);
00214         break;
00215     }
00216
00217     // Step 6: Get email
00218     while (1) {
00219         ui_clear_screen();
00220         ui_print_banner();
00221         const char* step6[] = {role_line, username_line, name_line, phone_line, "Email:", ">> "};
00222         ui_print_menu("Register Doctor", step6, 6, UI_SIZE);
00223         utils_get_string(new_doctor.email, EMAIL_SIZE);
00224
00225         if (strlen(new_doctor.email) < 5) {
00226             ui_print_error("Invalid email!");
00227             ui_pause();
00228             continue;
00229         }
00230         snprintf(email_line, sizeof(email_line), "Email: %s", new_doctor.email);
00231         break;
00232     }
00233
00234     // Step 7: Get specialization
00235     while (1) {
00236         ui_clear_screen();
00237         ui_print_banner();
00238         const char* step7[] = {role_line, username_line, name_line, phone_line, email_line,
00239 "Specialization:", ">> "};
00240         ui_print_menu("Register Doctor", step7, 7, UI_SIZE);
00241         utils_get_string(new_doctor.specialization, SPEC_SIZE);

```

```

00241
00242         if (strlen(new_doctor.specialization) < 2) {
00243             ui_print_error("Invalid specialization!");
00244             ui_pause();
00245             continue;
00246         }
00247         snprintf(spec_line, sizeof(spec_line), "Specialization: %s", new_doctor.specialization);
00248         break;
00249     }
00250
00251     // Step 8: Get room number
00252     while (1) {
00253         ui_clear_screen();
00254         ui_print_banner();
00255         const char* step8[] = {role_line, name_line, phone_line, spec_line, "Room Number:", "» "};
00256         ui_print_menu("Register Doctor", step8, 6, UI_SIZE);
00257         new_doctor.room_number = utils_get_int();
00258
00259         if (new_doctor.room_number < 1 || new_doctor.room_number > 999) {
00260             ui_print_error("Invalid room number (1-999)!");
00261             ui_pause();
00262             continue;
00263         }
00264         snprintf(room_line, sizeof(room_line), "Room: %d", new_doctor.room_number);
00265         break;
00266     }
00267
00268     new_doctor.is_available = true;
00269     new_doctor.is_active = true;
00270
00271     // Confirm Doctor registration
00272     ui_clear_screen();
00273     ui_print_banner();
00274     const char* confirm_doc[] = {
00275         username_line,
00276         name_line,
00277         phone_line,
00278         email_line,
00279         spec_line,
00280         room_line,
00281         "Confirm registration? (Y/N):",
00282         "» "
00283     };
00284     ui_print_menu("Confirm Doctor Registration", confirm_doc, 8, UI_SIZE);
00285     char confirm = utils_get_char();
00286
00287     if (confirm != 'Y' && confirm != 'y') {
00288         ui_print_info("Registration cancelled.");
00289         ui_pause();
00290         return;
00291     }
00292
00293     // Save doctor
00294     doctors[doctor_count] = new_doctor;
00295     doctor_count++;
00296     doctor_available++;
00297     doctor_save_to_file();
00298
00299 } else {
00300     // Confirm Admin/Receptionist registration
00301     ui_clear_screen();
00302     ui_print_banner();
00303     const char* confirm_items[] = {
00304         username_line,
00305         role_line,
00306         "Confirm registration? (Y/N):",
00307         "» "
00308     };
00309     ui_print_menu("Register User", confirm_items, 4, UI_SIZE);
00310     char confirm = utils_get_char();
00311
00312     if (confirm != 'Y' && confirm != 'y') {
00313         ui_print_info("Registration cancelled.");
00314         ui_pause();
00315         return;
00316     }
00317 }
00318
00319 // Save user credentials
00320 new_user.is_active = true;
00321 encrypt(new_user.password); // Encryption
00322 users[user_count] = new_user;
00323 user_count++;
00324 auth_save_to_file();
00325
00326 ui_print_success("User registered successfully!");
00327 ui_pause();

```

```

00328 }
00329
00330 void auth_view_users(void) {
00331     ui_clear_screen();
00332     ui_print_banner();
00333
00334     if (user_count == 0) {
00335         const char* empty[] = {"No users found!"};
00336         ui_print_menu("All Users", empty, 1, UI_SIZE);
00337         ui_pause();
00338         return;
00339     }
00340
00341     for (int i = 0; i < user_count; i++) {
00342         char id_line[50], user_line[80], role_line[50], status_line[50];
00343
00344         snprintf(id_line, sizeof(id_line), "User ID: %d", users[i].id);
00345         snprintf(user_line, sizeof(user_line), "Username: %s", users[i].username);
00346
00347         const char* role_str;
00348         switch (users[i].role) {
00349             case ROLE_ADMIN: role_str = "Admin"; break;
00350             case ROLE_RECEPTIONIST: role_str = "Receptionist"; break;
00351             case ROLE_DOCTOR: role_str = "Doctor"; break;
00352             default: role_str = "Unknown";
00353         }
00354         snprintf(role_line, sizeof(role_line), "Role: %s", role_str);
00355         snprintf(status_line, sizeof(status_line), "Status: %s",
00356                 users[i].is_active ? "Active" : "Inactive");
00357
00358         const char* items[] = {id_line, user_line, role_line, status_line, ""};
00359
00360         char title[50];
00361         snprintf(title, sizeof(title), "User %d", i + 1);
00362         ui_print_menu(title, items, 5, UI_SIZE);
00363     }
00364     ui_pause();
00365 }
00366
00367 void auth_user_menu(void) {
00368     int choice;
00369
00370     do {
00371         ui_clear_screen();
00372         ui_print_banner();
00373
00374         const char* menu_items[] = {
00375             "Register New User",
00376             "View All Users",
00377             "Back",
00378             "» "
00379         };
00380
00381         ui_print_menu("User Management", menu_items, 4, UI_SIZE);
00382         choice = utils_get_int();
00383
00384         switch (choice) {
00385             case 1:
00386                 auth_register_user();
00387                 break;
00388             case 2:
00389                 auth_view_users();
00390                 break;
00391             case 3:
00392                 ui_print_info("Returning...");
00393                 ui_pause();
00394                 break;
00395             default:
00396                 ui_print_error("Invalid choice!");
00397                 ui_pause();
00398         }
00399     } while (choice != 3);
00400 }
00401
00402 void auth_role_login(UserRole required_role) {
00403     char username[USERNAME_SIZE];
00404     char password[PASSWORD_SIZE];
00405
00406     // Get role name for display
00407     const char* role_name;
00408     switch (required_role) {
00409         case ROLE_ADMIN:
00410             role_name = "Admin";
00411             break;
00412         case ROLE_RECEPTIONIST:
00413             if (receptionist_count == 0) {
00414                 ui_print_error("No receptionist account found!");

```

```

00415         ui_pause();
00416         return;
00417     }
00418     role_name = "Receptionist";
00419     break;
00420 case ROLE_DOCTOR:
00421     if (doctor_count == 0) {
00422         ui_print_error("No doctor account found!");
00423         ui_pause();
00424         return;
00425     }
00426     role_name = "Doctor";
00427     break;
00428 default:
00429     role_name = "User";
00430 }
00431
00432 char title[50];
00433 snprintf(title, sizeof(title), "%s Login", role_name);
00434
00435 // Get username
00436 ui_clear_screen();
00437 ui_print_banner();
00438
00439 const char* menu_items[] = {
00440     "Username:",
00441     "» "
00442 };
00443 ui_print_menu(title, menu_items, 2, UI_SIZE);
00444 utils_get_string(username, USERNAME_SIZE);
00445
00446 // Get password
00447 ui_clear_screen();
00448 ui_print_banner();
00449
00450 char user_line[80];
00451 snprintf(user_line, sizeof(user_line), "Username: %s", username);
00452 const char* pass_items[] = {
00453     user_line,
00454     "Password:",
00455     "» "
00456 };
00457 ui_print_menu(title, pass_items, 3, UI_SIZE);
00458 utils_get_string(password, PASSWORD_SIZE);
00459 encrypt(password); // Encryption for comparison
00460
00461 // Find user with matching role
00462 for (int i = 0; i < user_count; i++) {
00463     if (strcmp(users[i].username, username) == 0 &&
00464         strcmp(users[i].password, password) == 0 &&
00465         users[i].is_active &&
00466         users[i].role == required_role) {
00467
00468         current_user = &users[i];
00469
00470         // Route to appropriate portal
00471         switch (required_role) {
00472             case ROLE_ADMIN:
00473                 admin_main_menu();
00474                 break;
00475             case ROLE_RECEPTIONIST:
00476                 receptionist_menu();
00477                 break;
00478             case ROLE_DOCTOR:
00479                 doctor_portal_menu(users[i].id, username);
00480                 break;
00481             default:
00482                 break;
00483         }
00484
00485         current_user = NULL;
00486         return;
00487     }
00488 }
00489
00490 char error_msg[80];
00491 snprintf(error_msg, sizeof(error_msg), "Invalid credentials or not a %s!", role_name);
00492 ui_print_error(error_msg);
00493 ui_pause();
00494 }
00495
00496 void login_menu(void) {
00497     int choice;
00498
00499     do {
00500         ui_clear_screen();
00501         ui_print_banner();

```

```

00502
00503     const char* menu_items[] = {
00504         "Admin Login",
00505         "Receptionist Login",
00506         "Doctor Login",
00507         "Back",
00508         "» "
00509     };
00510
00511     ui_print_menu("Login", menu_items, 5, UI_SIZE);
00512     choice = utils_get_int();
00513
00514     switch (choice) {
00515         case 1:
00516             auth_role_login(ROLE_ADMIN);
00517             break;
00518         case 2:
00519             auth_role_login(ROLE_RECEPTIONIST);
00520             break;
00521         case 3:
00522             auth_role_login(ROLE_DOCTOR);
00523             break;
00524         case 4:
00525             return;
00526         default:
00527             ui_print_error("Invalid choice!");
00528             ui_pause();
00529     }
00530     } while (choice != 4);
00531 }
00532
00533 void encrypt(char* password) {
00534     for (int i = 0; password[i] != '\0'; i++) {
00535         password[i] = password[i] ^ 1;
00536     }
00537 }
00538
00539 void decrypt(char* password) {
00540     encrypt(password);
00541 }

```

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

<i>id</i>	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     int count = 0;
00060     ui_clear_screen();
00061     ui_print_banner();
00062
00063     if (doctor_available == 0 || doctor_count == 0) {
00064         const char* menu_items[] = {"No doctors found!"};
00065         ui_print_menu("View All Doctors", menu_items, 1, UI_SIZE);
00066         ui_pause();
00067         return;
00068     }
00069
00070     for (int i = 0; i < doctor_count; i++) {
00071         if (doctors[i].is_active) {
00072             ui_print_doctor(doctors[i], count++);
00073         }
00074     }
00075     ui_pause();
00076 }
00077
00078 void doctor_view_one() {
00079     int count = 0;
00080
00081
00082

```

```

00083     if (doctor_available == 0) {
00084         ui_clear_screen();
00085         ui_print_banner();
00086         const char* menu_items[] = {"No doctors found!"};
00087         ui_print_menu("View All Doctors", menu_items, 1, UI_SIZE);
00088         ui_pause();
00089         return;
00090     }
00091
00092     for (int i = 0; i < doctor_count; i++) {
00093         ui_clear_screen();
00094         ui_print_banner();
00095         if (doctors[i].is_active) {
00096             ui_print_doctor(doctors[i], count++);
00097             ui_pause();
00098         }
00099     }
00100 }
00101
00102 void doctor_view() {
00103
00104     const char* menu_items[] =
00105     {
00106         "All at once",
00107         "One after another",
00108         "Back to Doctor Menu",
00109         "» "
00110     };
00111
00112     int choice;
00113     do {
00114         ui_clear_screen();
00115         ui_print_banner();
00116         ui_print_menu("View Doctors", menu_items, 4, UI_SIZE);
00117         choice = utils_get_int();
00118         switch (choice) {
00119             case 1:
00120                 doctor_view_all();
00121                 break;
00122             case 2:
00123                 doctor_view_one();
00124                 break;
00125             case 3:
00126                 ui_print_info("Returning to doctor menu...");
00127                 ui_pause();
00128                 return;
00129             default:
00130                 ui_print_error("Invalid choice!");
00131                 ui_pause();
00132                 break;
00133         }
00134     } while (choice != 3);
00135 }
00136
00137 void doctor_search_by_id(void) {
00138
00139     int id;
00140
00141     do {
00142         ui_clear_screen();
00143         ui_print_banner();
00144
00145         const char* menu_items[] = {
00146             "Enter ID: ",
00147             "» "
00148         };
00149
00150         ui_print_menu("Search Doctor", menu_items, 2, UI_SIZE);
00151         id = utils_get_int();
00152
00153         if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
00154             ui_print_error("Invalid ID!");
00155             ui_pause();
00156             continue;
00157         }
00158         for (int i = 0; i < doctor_count; i++) {
00159             if (doctors[i].id == id) {
00160                 ui_clear_screen();
00161                 ui_print_banner();
00162
00163                 ui_print_doctor(doctors[i], (doctors[i].id - DOCTOR_ID_START));
00164                 ui_pause();
00165                 return;
00166             }
00167         }
00168         ui_print_error("Doctor not found!");
00169         ui_pause();

```

```

00170         return;
00171     } while (1);
00172 }
00173 }
00174
00175 void doctor_search_by_name(void) {
00176     char name[NAME_SIZE];
00177
00178     do {
00179         ui_clear_screen();
00180         ui_print_banner();
00181
00182         const char* menu_items[] = {
00183             "Enter name: ",
00184             "» "
00185         };
00186
00187         ui_print_menu("Search Doctor", menu_items, 2, UI_SIZE);
00188         utils_get_string(name, NAME_SIZE);
00189
00190         if (!utils_is_valid_name(name)) {
00191             ui_print_error("Invalid name!");
00192             ui_pause();
00193             continue;
00194         }
00195
00196         utils_fix_name(name);
00197         for (int i = 0; i < doctor_count; i++) {
00198             if (strcmp(doctors[i].name, name) == 0) {
00199                 ui_clear_screen();
00200                 ui_print_banner();
00201
00202                 ui_print_doctor(doctors[i], (doctors[i].id - DOCTOR_ID_START));
00203                 ui_pause();
00204                 return;
00205             }
00206         }
00207         ui_print_error("Doctor not found!");
00208         ui_pause();
00209         return;
00210     } while (1);
00211 }
00212
00213 void doctor_search_by_phone(void) {
00214     char phone[PHONE_SIZE];
00215
00216     do {
00217         ui_clear_screen();
00218         ui_print_banner();
00219
00220         const char* menu_items[] = {
00221             "Enter phone: ",
00222             "» "
00223         };
00224
00225         ui_print_menu("Search Doctor", menu_items, 2, UI_SIZE);
00226         utils_get_string(phone, PHONE_SIZE);
00227
00228         if (!utils_is_valid_phone(phone)) {
00229             ui_print_error("Invalid phone!");
00230             ui_pause();
00231             continue;
00232         }
00233         for (int i = 0; i < doctor_count; i++) {
00234             if (strcmp(doctors[i].phone, phone) == 0) {
00235                 ui_clear_screen();
00236                 ui_print_banner();
00237
00238                 ui_print_doctor(doctors[i], (doctors[i].id - DOCTOR_ID_START));
00239                 ui_pause();
00240                 return;
00241             }
00242         }
00243         ui_print_error("Doctor not found!");
00244         ui_pause();
00245         return;
00246     } while (1);
00247 }
00248
00249 void doctor_search_by(void) {
00250     int choice;
00251
00252     do {
00253         ui_clear_screen();
00254         ui_print_banner();
00255
00256         const char* menu_items[] = {

```

```

00257         "Search by doctor ID",
00258         "Search by doctor name",
00259         "Search by doctor phone",
00260         "Back to Doctor Menu",
00261         "» "
00262     };
00263
00264     ui_print_menu("Search Doctor", menu_items, 5, UI_SIZE);
00265     choice = utils_get_int();
00266
00267     switch (choice) {
00268     case 1:
00269         doctor_search_by_id();
00270         break;
00271     case 2:
00272         doctor_search_by_name();
00273         break;
00274     case 3:
00275         doctor_search_by_phone();
00276         break;
00277     case 4:
00278         ui_print_info("Returning to doctor menu...");
00279         ui_pause();
00280         break;
00281     default:
00282         ui_print_error("Invalid choice! Please try again.");
00283         ui_pause();
00284     }
00285     } while (choice != 4);
00286 }
00287
00288 int doctor_search_id (int id) {
00289     for (int i = 0; i < doctor_count; i++) {
00290         if (doctors[i].id == id) {
00291             return i;
00292         }
00293     }
00294     return -1;
00295 }
00296
00297 void doctor_update_name(const char* menu_items[], int index) {
00298     ui_clear_screen();
00299     ui_print_banner();
00300
00301     const char* menu[] = {
00302         menu_items[0],
00303         "Enter new name: ",
00304         "» "
00305     };
00306
00307     char name[NAME_SIZE];
00308     ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
00309     utils_get_string(name, NAME_SIZE);
00310     utils_fix_name(name);
00311
00312     if (!utils_is_valid_name(name)) {
00313         ui_print_error("Invalid name! Could not update.");
00314         ui_pause();
00315         return;
00316     }
00317     doctors[index].name[0] = '\0';
00318     strncpy(doctors[index].name, name, NAME_SIZE);
00319 }
00320
00321 void doctor_update_phone(const char* menu_items[], int index) {
00322     ui_clear_screen();
00323     ui_print_banner();
00324
00325     const char* menu[] = {
00326         menu_items[1],
00327         "Enter new phone: ",
00328         "» "
00329     };
00330
00331     char phone[PHONE_SIZE];
00332     ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
00333     utils_get_string(phone, PHONE_SIZE);
00334
00335     if (!utils_is_valid_phone(phone)) {
00336         ui_print_error("Invalid phone! Could not update.");
00337         ui_pause();
00338         return;
00339     }
00340     doctors[index].phone[0] = '\0';
00341     strncpy(doctors[index].phone, phone, PHONE_SIZE);
00342 }
00343

```

```

00344 void doctor_update_email(const char* menu_items[], int index) {
00345     ui_clear_screen();
00346     ui_print_banner();
00347
00348     const char* menu[] = {
00349         menu_items[2],
00350         "Enter new email: ",
00351         "» "
00352     };
00353
00354     char email[EMAIL_SIZE];
00355     ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
00356     utils_get_string(email, EMAIL_SIZE);
00357
00358     if (!utils_is_valid_email(email)) {
00359         ui_print_error("Invalid email! Could not update.");
00360         ui_pause();
00361         return;
00362     }
00363     doctors[index].email[0] = '\0';
00364     strncpy(doctors[index].email, email, EMAIL_SIZE);
00365 }
00366
00367 void doctor_update_specialization(const char* menu_items[], int index) {
00368     ui_clear_screen();
00369     ui_print_banner();
00370
00371     const char* menu[] = {
00372         menu_items[3],
00373         "Enter new specialization: ",
00374         "» "
00375     };
00376
00377     char spec[SPEC_SIZE];
00378     ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
00379     utils_get_string(spec, SPEC_SIZE);
00380
00381     if (strlen(spec) == 0) {
00382         ui_print_error("Specialization cannot be empty! Could not update.");
00383         ui_pause();
00384         return;
00385     }
00386     utils_fix_name(spec);
00387     doctors[index].specialization[0] = '\0';
00388     strncpy(doctors[index].specialization, spec, SPEC_SIZE);
00389 }
00390
00391 void doctor_update_room(const char* menu_items[], int index) {
00392     ui_clear_screen();
00393     ui_print_banner();
00394
00395     const char* menu[] = {
00396         menu_items[4],
00397         "Enter new room number: ",
00398         "» "
00399     };
00400
00401     ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
00402     int room = utils_get_int();
00403
00404     if (room <= 0 || room >= 1000) {
00405         ui_print_error("Invalid room number! Could not update.");
00406         ui_pause();
00407         return;
00408     }
00409     doctors[index].room_number = room;
00410 }
00411
00412 void doctor_update_availability(const char* menu_items[], int index) {
00413     ui_clear_screen();
00414     ui_print_banner();
00415
00416     const char* menu[] = {
00417         "Set availability (1 for available, 2 for unavailable): ",
00418         menu_items[5],
00419         "» "
00420     };
00421
00422     int input;
00423     ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
00424     input = utils_get_int();
00425
00426     if (input == 1) {
00427         doctors[index].is_available = true;
00428     } else if (input == 2) {
00429         doctors[index].is_available = false;
00430     } else {

```

```

00431         ui_print_error("Invalid input! Could not update.");
00432         ui_pause();
00433     }
00434 }
00435
00436 void doctor_update_status(const char* menu_items[], int index) {
00437     ui_clear_screen();
00438     ui_print_banner();
00439
00440     const char* menu[] = {
00441         menu_items[6],
00442         "Enter new status: (1 for active, 2 for inactive) ",
00443         "» "
00444     };
00445
00446     bool status;
00447     int input;
00448     ui_print_menu("Update Doctor", menu, 3, UI_SIZE);
00449     input = utils_get_int();
00450
00451     if (input == 1) {
00452         status = true;
00453     } else if (input == 2) {
00454         status = false;
00455     }
00456     doctors[index].is_active = status;
00457 }
00458
00459 void doctor_update_using_id() {
00460     ui_clear_screen();
00461     ui_print_banner();
00462
00463     int id;
00464
00465     do {
00466         ui_clear_screen();
00467         ui_print_banner();
00468
00469         const char* enter_id_items[] = {
00470             "Enter doctor ID: (0 to go back) ",
00471             "» "
00472         };
00473
00474         ui_print_menu("Update Doctor", enter_id_items, 2, UI_SIZE);
00475         id = utils_get_int();
00476
00477         if (id == 0) return;
00478
00479         if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
00480             ui_print_error("Invalid ID!");
00481             ui_pause();
00482         } else {
00483             ui_print_info("Doctor found!");
00484             ui_pause();
00485             break;
00486         }
00487     } while (1);
00488
00489     int index = doctor_search_id(id);
00490     if (index == -1) return;
00491
00492     char name_line[NAME_LINE_SIZE], phone_line[PHONE_LINE_SIZE], email_line[70], spec_line[70],
00493     room_line[70], avail_line[70], status_line[STATUS_LINE_SIZE];
00494
00495     snprintf(name_line, NAME_LINE_SIZE, "Name: %s", doctors[index].name);
00496     snprintf(phone_line, PHONE_LINE_SIZE, "Phone: %s", doctors[index].phone);
00497     snprintf(email_line, sizeof(email_line), "Email: %s", doctors[index].email);
00498     snprintf(spec_line, sizeof(spec_line), "Specialization: %s", doctors[index].specialization);
00499     snprintf(room_line, sizeof(room_line), "Room Number: %d", doctors[index].room_number);
00500
00501     int doctor_id = doctors[index].id;
00502
00503     if (doctors[index].is_available) {
00504         snprintf(avail_line, sizeof(avail_line), "Availability: Available");
00505     } else {
00506         snprintf(avail_line, sizeof(avail_line), "Availability: Unavailable");
00507     }
00508
00509     if (doctors[index].is_active) {
00510         snprintf(status_line, STATUS_LINE_SIZE, "Status: Active");
00511     } else {
00512         snprintf(status_line, STATUS_LINE_SIZE, "Status: Inactive");
00513     }
00514
00515     const char* menu_items[] = {
00516         name_line,
00517         phone_line,
00518         email_line,

```

```
00517         spec_line,
00518         room_line,
00519         avail_line,
00520         "Go back",
00521         "» "
00522     };
00523
00524     char title[50];
00525     snprintf(title, 50, "Update Doctor | ID: %d", doctor_id);
00526
00527     ui_clear_screen();
00528     ui_print_banner();
00529     ui_print_menu(title, menu_items, 8, UI_SIZE);
00530
00531     int choice = utils_get_int();
00532
00533     switch (choice) {
00534     case 1:
00535         doctor_update_name(menu_items, index);
00536         break;
00537     case 2:
00538         doctor_update_phone(menu_items, index);
00539         break;
00540     case 3:
00541         doctor_update_email(menu_items, index);
00542         break;
00543     case 4:
00544         doctor_update_specialization(menu_items, index);
00545         break;
00546     case 5:
00547         doctor_update_room(menu_items, index);
00548         break;
00549     case 6:
00550         doctor_update_availability(menu_items, index);
00551         break;
00552     case 7:
00553         ui_print_info("Returning to doctor menu...");
00554         ui_pause();
00555         break;
00556     default:
00557         ui_print_error("Invalid choice! Please try again.");
00558         ui_pause();
00559         break;
00560     }
00561 }
00562
00563 void doctor_deactivate_account() {
00564     int id;
00565     while (1) {
00566         ui_clear_screen();
00567         ui_print_banner();
00568
00569         const char* enter_id_items[] = {
00570             "Enter doctor ID (0 to cancel): ",
00571             "» "
00572         };
00573
00574         ui_print_menu("Deactivate Doctor", enter_id_items, 2, UI_SIZE);
00575         id = utils_get_int();
00576
00577         if (id == 0) {
00578             ui_print_info("User Pressed 0. \nCanceling deactivation...");
00579             ui_pause();
00580             return;
00581         }
00582
00583         if (!utils_is_valid_id(id, ROLE_DOCTOR)) {
00584             ui_print_error("Invalid ID!");
00585             ui_pause();
00586             continue;
00587         }
00588         break;
00589     }
00590
00591     int index = doctor_search_id(id);
00592     if (index == -1) {
00593         ui_print_error("Doctor not found!");
00594         ui_pause();
00595         return;
00596     }
00597
00598     ui_clear_screen();
00599     ui_print_banner();
00600     ui_print_doctor(doctors[index], index);
00601
00602     const char* menu[] = {
00603         "Confirm Deactivation",
```

```

00604         "Cancel",
00605         "» "
00606     };
00607
00608     ui_print_menu("Deactivate Doctor", menu, 3, UI_SIZE);
00609     int input = utils_get_int();
00610
00611     if (input == 1) {
00612         doctors[index].is_active = false;
00613         ui_print_success("Doctor deactivated successfully!");
00614         doctor_available--;
00615         doctor_unavailable++;
00616         ui_pause();
00617     } else {
00618         ui_print_info("Deactivation cancelled.");
00619         ui_pause();
00620     }
00621 }
00622
00623 void doctor_view_discharged(void) {
00624     int count = 0;
00625     ui_clear_screen();
00626     ui_print_banner();
00627
00628     if (doctor_unavailable == 0) {
00629         const char* menu_items[] = {"No inactive doctors found!"};
00630         ui_print_menu("Inactive Doctors", menu_items, 1, UI_SIZE);
00631         ui_pause();
00632         return;
00633     }
00634
00635     for (int i = 0; i < doctor_count; i++) {
00636         if (!doctors[i].is_active) {
00637             ui_print_doctor(doctors[i], count++);
00638         }
00639     }
00640     ui_pause();
00641 }

```

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

<i>doctor↔ _id</i>	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

<i>doctor↔ _id</i>	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

<i>doctor_id</i>	The doctor's ID.
<i>doctor_name</i>	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

<i>doctor↔ _id</i>	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

<i>doctor↔ _id</i>	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

<i>doctor↔ _id</i>	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

<i>doctor↔ _id</i>	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

<i>doctor_id</i>	The doctor's ID.
<i>today_date</i>	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.](#)

```

00001
00008
00009 #include <stdio.h>
00010 #include <string.h>
00011 #include "../include/doctor_portal.h"
00012 #include "../include/appointment.h"
00013 #include "../include/patient.h"
00014 #include "../include/doctor.h"
00015 #include "../include/utils.h"
00016 #include "../include/ui.h"
00017 #include "../include/hospital.h"
00018
00019 void doctor_portal_view_appointments(int doctor_id) {
00020     appointment_view_by_doctor(doctor_id);
00021 }
00022
00023 void doctor_portal_view_pending(int doctor_id) {
00024     int count = 0;
00025     ui_clear_screen();
00026     ui_print_banner();
00027
00028     for (int i = 0; i < appointment_count; i++) {
00029         if (appointments[i].doctor_id == doctor_id &&
00030             appointments[i].status == APPT_PENDING) {
00031             ui_print_appointment(appointments[i], count++);
00032         }
00033     }
00034
00035     if (count == 0) {
00036         const char* menu_items[] = {"No pending appointments!"};
00037         ui_print_menu("Pending Appointments", menu_items, 1, UI_SIZE);
00038     }
00039     ui_pause();
00040 }
00041
00042 void doctor_portal_view_today(int doctor_id, const char* today_date) {
00043     int count = 0;
00044     ui_clear_screen();
00045     ui_print_banner();
00046
00047     for (int i = 0; i < appointment_count; i++) {
00048         if (appointments[i].doctor_id == doctor_id &&
00049             strcmp(appointments[i].date, today_date) == 0 &&
00050             appointments[i].status != APPT_CANCELLED) {
00051             ui_print_appointment(appointments[i], count++);
00052         }
00053     }
00054
00055     if (count == 0) {
00056         const char* menu_items[] = {"No appointments for today!"};
00057         ui_print_menu("Today's Appointments", menu_items, 1, UI_SIZE);
00058     }
00059     ui_pause();
00060 }
00061
00062 void doctor_portal_view_patient(void) {
00063     int patient_id;
00064
00065     ui_clear_screen();
00066     ui_print_banner();
00067
00068     const char* menu_items[] = {
00069         "Enter Patient ID:",
00070         "» "
00071     };
00072
00073     ui_print_menu("View Patient Details", menu_items, 2, UI_SIZE);
00074     patient_id = utils_get_int();
00075
00076     if (!utils_is_valid_id(patient_id, ROLE_PATIENT)) {
00077         ui_print_error("Invalid patient ID!");

```

```

00078         ui_pause();
00079         return;
00080     }
00081
00082     int idx = patient_search_id(patient_id);
00083     if (idx == -1) {
00084         ui_print_error("Patient not found!");
00085         ui_pause();
00086         return;
00087     }
00088
00089     ui_clear_screen();
00090     ui_print_banner();
00091     ui_print_patient(patients[idx], idx);
00092     ui_pause();
00093 }
00094
00095 void doctor_portal_complete_appointment(int doctor_id) {
00096     int appt_id;
00097
00098     // First show pending appointments
00099     ui_clear_screen();
00100     ui_print_banner();
00101
00102     int count = 0;
00103     for (int i = 0; i < appointment_count; i++) {
00104         if (appointments[i].doctor_id == doctor_id &&
00105             (appointments[i].status == APPT_PENDING ||
00106              appointments[i].status == APPT_CONFIRMED)) {
00107             ui_print_appointment(appointments[i], count++);
00108         }
00109     }
00110
00111     if (count == 0) {
00112         const char* menu_items[] = {"No appointments to complete!"};
00113         ui_print_menu("Complete Appointment", menu_items, 1, UI_SIZE);
00114         ui_pause();
00115         return;
00116     }
00117
00118     const char* input_items[] = {
00119         "Enter Appointment ID to mark as completed (0 to cancel):",
00120         ">> "
00121     };
00122     ui_print_menu("Complete Appointment", input_items, 2, UI_SIZE);
00123     appt_id = utils_get_int();
00124
00125     if (appt_id == 0) return;
00126
00127     int idx = appointment_search_id(appt_id);
00128     if (idx == -1) {
00129         ui_print_error("Appointment not found!");
00130         ui_pause();
00131         return;
00132     }
00133
00134     if (appointments[idx].doctor_id != doctor_id) {
00135         ui_print_error("This appointment is not assigned to you!");
00136         ui_pause();
00137         return;
00138     }
00139
00140     appointments[idx].status = APPT_COMPLETED;
00141     appointment_save_to_file();
00142     ui_print_success("Appointment marked as completed!");
00143     ui_pause();
00144 }
00145
00146 void doctor_portal_cancel_appointment(int doctor_id) {
00147     int appt_id;
00148
00149     // First show pending appointments
00150     ui_clear_screen();
00151     ui_print_banner();
00152
00153     int count = 0;
00154     for (int i = 0; i < appointment_count; i++) {
00155         if (appointments[i].doctor_id == doctor_id &&
00156             appointments[i].status == APPT_PENDING) {
00157             ui_print_appointment(appointments[i], count++);
00158         }
00159     }
00160
00161     if (count == 0) {
00162         const char* menu_items[] = {"No appointments to cancel!"};
00163         ui_print_menu("Cancel Appointment", menu_items, 1, UI_SIZE);
00164         ui_pause();

```

```

00165         return;
00166     }
00167
00168     const char* input_items[] = {
00169         "Enter Appointment ID to cancel (0 to go back):",
00170         "» "
00171     };
00172     ui_print_menu("Cancel Appointment", input_items, 2, UI_SIZE);
00173     appt_id = utils_get_int();
00174
00175     if (appt_id == 0) return;
00176
00177     int idx = appointment_search_id(appt_id);
00178     if (idx == -1) {
00179         ui_print_error("Appointment not found!");
00180         ui_pause();
00181         return;
00182     }
00183
00184     if (appointments[idx].doctor_id != doctor_id) {
00185         ui_print_error("This appointment is not assigned to you!");
00186         ui_pause();
00187         return;
00188     }
00189
00190     appointments[idx].status = APPT_CANCELLED;
00191     appointment_save_to_file();
00192     ui_print_success("Appointment cancelled!");
00193     ui_pause();
00194 }
00195
00196 void doctor_portal_update_availability(int doctor_id) {
00197     int idx = doctor_search_id(doctor_id);
00198     if (idx == -1) {
00199         ui_print_error("Doctor not found!");
00200         ui_pause();
00201         return;
00202     }
00203
00204     ui_clear_screen();
00205     ui_print_banner();
00206
00207     char current_status[50];
00208     snprintf(current_status, sizeof(current_status), "Current Status: %s",
00209              doctors[idx].is_available ? "Available" : "Unavailable");
00210
00211     const char* menu_items[] = {
00212         "Set availability (1 for available, 2 for unavailable): ",
00213         current_status,
00214         "Back",
00215         "» "
00216     };
00217
00218     ui_print_menu("Update Availability", menu_items, 4, UI_SIZE);
00219     int choice = utils_get_int();
00220
00221     switch (choice) {
00222     case 1:
00223         doctors[idx].is_available = true;
00224         doctor_save_to_file();
00225         ui_print_success("Status set to Available!");
00226         ui_pause();
00227         break;
00228     case 2:
00229         doctors[idx].is_available = false;
00230         doctor_save_to_file();
00231         ui_print_success("Status set to Unavailable!");
00232         ui_pause();
00233         break;
00234     case 3:
00235         return;
00236     default:
00237         ui_print_error("Invalid choice!");
00238         ui_pause();
00239     }
00240 }
00241
00242 void doctor_portal_view_profile(int doctor_id) {
00243     int idx = doctor_search_id(doctor_id);
00244     if (idx == -1) {
00245         ui_print_error("Doctor not found!");
00246         ui_pause();
00247         return;
00248     }
00249
00250     ui_clear_screen();
00251     ui_print_banner();

```

```

00252     ui_print_doctor(doctors[idx], idx);
00253     ui_pause();
00254 }
00255
00256 void doctor_portal_menu(int doctor_id, const char* doctor_name) {
00257     int choice;
00258
00259     // Build title with doctor's name
00260     char title[80];
00261     snprintf(title, sizeof(title), "Doctor Portal | Dr. %s", doctor_name);
00262
00263     do {
00264         ui_clear_screen();
00265         ui_print_banner();
00266
00267         const char* menu_items[] = {
00268             "View All My Appointments",
00269             "View Pending Appointments",
00270             "View Patient Details",
00271             "Complete Appointment",
00272             "Cancel Appointment",
00273             "Update My Availability",
00274             "View My Profile",
00275             "Logout",
00276             "» "
00277         };
00278
00279         ui_print_menu(title, menu_items, 9, UI_SIZE);
00280         choice = utils_get_int();
00281
00282         switch (choice) {
00283             case 1:
00284                 doctor_portal_view_appointments(doctor_id);
00285                 break;
00286             case 2:
00287                 doctor_portal_view_pending(doctor_id);
00288                 break;
00289             case 3:
00290                 doctor_portal_view_patient();
00291                 break;
00292             case 4:
00293                 doctor_portal_complete_appointment(doctor_id);
00294                 break;
00295             case 5:
00296                 doctor_portal_cancel_appointment(doctor_id);
00297                 break;
00298             case 6:
00299                 doctor_portal_update_availability(doctor_id);
00300                 break;
00301             case 7:
00302                 doctor_portal_view_profile(doctor_id);
00303                 break;
00304             case 8:
00305                 ui_print_info("Logging out...");
00306                 ui_pause();
00307                 break;
00308             default:
00309                 ui_print_error("Invalid choice!");
00310                 ui_pause();
00311         }
00312     } while (choice != 8);
00313 }

```

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];
```

```

00031 Appointment appointments[MAX_APPOINTMENTS];
00032
00033 int patient_count = 0;
00034 int patient_available = 0;
00035 int patient_unavailable = 0;
00036 int doctor_count = 0;
00037 int doctor_available = 0;
00038 int doctor_unavailable = 0;
00039 int receptionist_count = 0;
00040 int receptionist_available = 0;
00041 int receptionist_unavailable = 0;
00042 int user_count = 0;
00043 int appointment_count = 0;
00044
00045 User* current_user = NULL;
00046
00047 void hospital_init(void) {
00048     patient_load_from_file();
00049     doctor_load_from_file();
00050     appointment_load_from_file();
00051     auth_load_from_file();
00052     auth_init_default_admin();
00053 }
00054
00055 void show_about(void) {
00056     ui_clear_screen();
00057     ui_print_banner();
00058
00059     const char* about_items[] = {
00060         "Healthcare Management System (HMS)",
00061         "Version: 1.0",
00062         "Developed for efficient hospital management",
00063         "including patient, doctor, and appointment",
00064         "tracking with role-based access control.",
00065         "Roles: Admin, Receptionist, Doctor",
00066         ""
00067     };
00068
00069     ui_print_menu("About HMS", about_items, 7, UI_SIZE);
00070     ui_pause();
00071 }
00072
00073 void print_help(const char* program_name) {
00074     printf("\n" BOLD BRIGHT_RED "Healthcare Management System" RESET BOLD SOFT_GREEN " v%s" RESET
00075         "\n\n", VERSION);
00076     printf(SOFT_YELLOW "Usage:" RESET " %s [option] or .\\hms.exe [option] for windows\n",
00077         program_name);
00078     printf(SOFT_YELLOW "Usage:" RESET " ./hms.out [option] for linux\n\n");
00079     printf(SOFT_YELLOW "Options:" RESET "\n");
00080     printf("  -h, --help      Show this help message\n");
00081     printf("  -v, --version   Show version information\n");
00082     printf("  -a, --about     Show about information\n");
00083     printf("  -l, --login     Go directly to login menu\n");
00084     printf("\n");
00085     printf("If no options are provided, the interactive menu will start.\n\n");
00086 }
00087
00088 void print_version(void) {
00089     printf("\n" BOLD "Healthcare Management System" RESET "\n");
00090     printf("Version: " SOFT_GREEN "%s" RESET "\n", VERSION);
00091     printf("Built with C\n\n");
00092 }
00093
00094 void ensure_data_dir(void) {
00095     #ifdef _WIN32
00096         _mkdir("data");
00097     #else
00098         mkdir("data", 0755);
00099     #endif
00100 }

```

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/utlis.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

<i>id</i>	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.](#)

```
00001
00007
00008 #include <stdio.h>
00009 #include <string.h>
00010 #include "../include/patient.h"
00011 #include "../include/utils.h"
00012 #include "../include/ui.h"
00013 #include "../include/hospital.h"
00014
00015 int patient_save_to_file(void) {
00016     FILE* file = fopen(PATIENTS_FILE, "wb");
00017     if (file == NULL) {
00018         return -1;
00019     }
00020     if (fwrite(&patient_count, sizeof(int), 1, file) != 1 ||
00021         fwrite(&patient_available, sizeof(int), 1, file) != 1 ||
00022         fwrite(patients, sizeof(Patient), patient_count, file) != (size_t)patient_count) {
00023         fclose(file);
00024         return -1;
00025     }
00026     fclose(file);
00027     return 0;
00028 }
00029
00030 int patient_load_from_file(void) {
00031     FILE* file = fopen(PATIENTS_FILE, "rb");
00032     if (file == NULL) {
00033         return -1;
00034     }
00035     if (fread(&patient_count, sizeof(int), 1, file) != 1 ||
00036         fread(&patient_available, sizeof(int), 1, file) != 1) {
00037         fclose(file);
00038         return -1;
00039     }
00040     if (patient_count < 0 || patient_count > MAX_PATIENTS) {
00041         fclose(file);
00042         patient_count = 0;
00043         return -1;
00044     }
00045     if (fread(patients, sizeof(Patient), patient_count, file) != (size_t)patient_count) {
00046         fclose(file);
00047         patient_count = 0;
00048         return -1;
00049     }
00050     fclose(file);
00051     return 0;
00052 }
00053
00054 int patient_generate_id(void) {
00055     return PATIENT_ID_START + patient_count;
00056 }
00057
00058 void patient_add(void) {
00059     if (patient_count >= MAX_PATIENTS) {
```

```

00060         ui_print_error("Error: Maximum patient limit reached!");
00061         ui_pause();
00062         return;
00063     }
00064
00065     // Create new patient
00066     Patient new_patient;
00067     new_patient.id = patient_generate_id();
00068     new_patient.is_active = true;
00069
00070     // lines for menu
00071     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];
00072
00073     // Step 1: Get Name
00074     while (1) {
00075         ui_clear_screen();
00076         ui_print_banner();
00077         const char* step1[] = {"Name:", "» "};
00078         ui_print_menu("Add Patient", step1, 2, UI_SIZE);
00079         utils_get_string(new_patient.name, NAME_SIZE);
00080         if (utils_is_valid_name(new_patient.name)) {
00081             utils_fix_name(new_patient.name);
00082             break;
00083         }
00084         ui_print_error("Invalid name! Please try again.");
00085         ui_pause();
00086     }
00087
00088     // Step 2: Get Age
00089     while (1) {
00090         ui_clear_screen();
00091         ui_print_banner();
00092         snprintf(name_line, sizeof(name_line), "Name: %s", new_patient.name);
00093         const char* step2[] = {name_line, "Age:", "» "};
00094         ui_print_menu("Add Patient", step2, 3, UI_SIZE);
00095         new_patient.age = utils_get_int();
00096         if (new_patient.age > 0 && new_patient.age < 120) {
00097             break;
00098         }
00099         ui_print_error("Invalid age! Please try again.");
00100         ui_pause();
00101     }
00102
00103     // Step 3: Get Gender
00104     while (1) {
00105         ui_clear_screen();
00106         ui_print_banner();
00107         snprintf(age_line, sizeof(age_line), "Age: %d", new_patient.age);
00108         const char* step3[] = {name_line, age_line, "Gender (M/F):", "» "};
00109         ui_print_menu("Add Patient", step3, 4, UI_SIZE);
00110         char gender_input = utils_get_char();
00111         if (gender_input == 'M' || gender_input == 'm') {
00112             new_patient.gender = MALE;
00113             break;
00114         } else if (gender_input == 'F' || gender_input == 'f') {
00115             new_patient.gender = FEMALE;
00116             break;
00117         }
00118         ui_print_error("Invalid gender! Please try again.");
00119         ui_pause();
00120     }
00121
00122     // Step 4: Get Phone
00123     while (1) {
00124         ui_clear_screen();
00125         ui_print_banner();
00126         snprintf(gender_line, sizeof(gender_line), "Gender: %s", (new_patient.gender == MALE) ? "Male"
: "Female");
00127         const char* step4[] = {name_line, age_line, gender_line, "Phone (11 digits):", "» "};
00128         ui_print_menu("Add Patient", step4, 5, UI_SIZE);
00129         utils_get_string(new_patient.phone, PHONE_SIZE);
00130         if (utils_is_valid_phone(new_patient.phone)) {
00131             break;
00132         }
00133         ui_print_error("Invalid phone! Please try again.");
00134         ui_pause();
00135     }
00136
00137     // Step 5: Get Address
00138     while (1) {
00139         ui_clear_screen();
00140         ui_print_banner();
00141         snprintf(phone_line, sizeof(phone_line), "Phone: %s", new_patient.phone);
00142         const char* step5[] = {name_line, age_line, gender_line, phone_line, "Address:", "» "};
00143         ui_print_menu("Add Patient", step5, 6, UI_SIZE);
00144         utils_get_string(new_patient.address, ADDRESS_SIZE);

```

```

00145         if (utils_is_valid_address(new_patient.address)) {
00146             break;
00147         }
00148         ui_print_error("Invalid address! Please try again.");
00149         ui_pause();
00150     }
00151
00152     // Step 6: Get Blood Group
00153     while (1) {
00154         ui_clear_screen();
00155         ui_print_banner();
00156         snprintf(address_line, sizeof(address_line), "Address: %s", new_patient.address);
00157         const char* step6[] =
00158         {
00159             name_line, age_line, gender_line, phone_line, address_line,
00160             "Blood Group (e.g. A+, O-, or U for Unknown):", "» "
00161         };
00162         ui_print_menu("Add Patient", step6, 7, UI_SIZE);
00163         utils_get_string(new_patient.blood_group, BLOOD_SIZE);
00164         if (utils_is_valid_blood_group(new_patient.blood_group)) {
00165             utils_str_to_upper(new_patient.blood_group);
00166             break;
00167         }
00168         ui_print_error("Invalid blood group! Please try again.");
00169         ui_pause();
00170     }
00171
00172     // Step 7: Confirm
00173     snprintf(blood_group_line, sizeof(blood_group_line), "Blood Group: %s", new_patient.blood_group);
00174     const char* step7[] =
00175     {
00176         name_line, age_line, gender_line, phone_line, address_line, blood_group_line,
00177         "Confirm (Y/N):", "» "
00178     };
00179     ui_clear_screen();
00180     ui_print_banner();
00181     ui_print_menu("Add Patient", step7, 8, UI_SIZE);
00182     char confirm = utils_get_char();
00183     if (confirm != 'Y' && confirm != 'y') {
00184         ui_print_info("Patient addition cancelled.");
00185         ui_pause();
00186         return;
00187     }
00188
00189     // Add to array
00190     patients[patient_count] = new_patient;
00191     patient_count++;
00192     patient_available++;
00193
00194     ui_clear_screen();
00195     ui_print_banner();
00196
00197     char id_line[70];
00198     snprintf(id_line, sizeof(id_line), "Patient ID: %d", new_patient.id);
00199     const char* success_items[] =
00200     {
00201         id_line, name_line, age_line,
00202         gender_line, phone_line,
00203         address_line, blood_group_line,
00204         "Patient added successfully!"
00205     };
00206     ui_print_menu("Patient Added", success_items, 8, UI_SIZE);
00207     ui_pause();
00208 }
00209 }
00210
00211 void patient_view_all() {
00212
00213     int count = 0;
00214     ui_clear_screen();
00215     ui_print_banner();
00216
00217     if (patient_available == 0 || patient_count == 0) {
00218         const char* menu_items[] = {"No patients found!"};
00219         ui_print_menu("View All Patients", menu_items, 1, UI_SIZE);
00220         ui_pause();
00221         return;
00222     }
00223
00224     for (int i = 0; i < patient_count; i++) {
00225         if (patients[i].is_active) {
00226             ui_print_patient(patients[i], count++);
00227         }
00228     }
00229     ui_pause();
00230 }
00231

```



```

00232 void patient_view_one() {
00233     int count = 0;
00234
00235     if (patient_available == 0) {
00236         ui_clear_screen();
00237         ui_print_banner();
00238         const char* menu_items[] = {"No patients found!"};
00239         ui_print_menu("View All Patients", menu_items, 1, UI_SIZE);
00240         ui_pause();
00241         return;
00242     }
00243
00244     for (int i = 0; i < patient_count; i++) {
00245         ui_clear_screen();
00246         ui_print_banner();
00247         if (patients[i].is_active) {
00248             ui_print_patient(patients[i], count++);
00249             ui_pause();
00250         }
00251     }
00252 }
00253
00254 void patient_view() {
00255     const char* menu_items[] =
00256     {
00257         "All at once",
00258         "One after another",
00259         "Back to Patient Menu",
00260         "» "
00261     };
00262
00263     int choice;
00264     do {
00265         ui_clear_screen();
00266         ui_print_banner();
00267         ui_print_menu("View Patients", menu_items, 4, UI_SIZE);
00268         choice = utils_get_int();
00269         switch (choice) {
00270             case 1:
00271                 patient_view_all();
00272                 break;
00273             case 2:
00274                 patient_view_one();
00275                 break;
00276             case 3:
00277                 ui_print_info("Returning to receptionist menu...");
00278                 ui_pause();
00279                 return;
00280             default:
00281                 ui_print_error("Invalid choice!");
00282                 ui_pause();
00283                 break;
00284         }
00285     } while (choice != 3);
00286 }
00287
00288 void patient_search_by_id(void) {
00289     int id;
00290
00291     do {
00292         ui_clear_screen();
00293         ui_print_banner();
00294
00295         const char* menu_items[] = {
00296             "Enter ID: ",
00297             "» "
00298         };
00299
00300         ui_print_menu("Search Patient", menu_items, 2, UI_SIZE);
00301         id = utils_get_int();
00302
00303         if (!utils_is_valid_id(id, ROLE_PATIENT)) {
00304             ui_print_error("Invalid ID!");
00305             ui_pause();
00306             continue;
00307         }
00308
00309         for (int i = 0; i < patient_count; i++) {
00310             if (patients[i].id == id) {
00311                 ui_clear_screen();
00312                 ui_print_banner();
00313
00314                 ui_print_patient(patients[i], (patients[i].id - 1001));
00315                 ui_pause();
00316                 return;
00317             }
00318         }

```

```
00319         }
00320     }
00321     ui_print_error("Patient not found!");
00322     ui_pause();
00323     return;
00324 } while (1);
00325
00326 }
00327
00328 void patient_search_by_name(void) {
00329     char name[NAME_SIZE];
00330
00331     do {
00332         ui_clear_screen();
00333         ui_print_banner();
00334
00335         const char* menu_items[] = {
00336             "Enter name: ",
00337             "» "
00338         };
00339
00340         ui_print_menu("Search Patient", menu_items, 2, UI_SIZE);
00341         utils_get_string(name, NAME_SIZE);
00342
00343         if (!utils_is_valid_name(name)) {
00344             ui_print_error("Invalid name!");
00345             ui_pause();
00346             continue;
00347         }
00348
00349         utils_fix_name(name);
00350         for (int i = 0; i < patient_count; i++) {
00351             if (strcmp(patients[i].name, name) == 0) {
00352                 ui_clear_screen();
00353                 ui_print_banner();
00354
00355                 ui_print_patient(patients[i], (patients[i].id - 1001));
00356                 ui_pause();
00357                 return;
00358             }
00359         }
00360         ui_print_error("Patient not found!");
00361         ui_pause();
00362         return;
00363     } while (1);
00364 }
00365
00366 void patient_search_by_phone(void) {
00367     char phone[PHONE_SIZE];
00368
00369     do {
00370         ui_clear_screen();
00371         ui_print_banner();
00372
00373         const char* menu_items[] = {
00374             "Enter phone: ",
00375             "» "
00376         };
00377
00378         ui_print_menu("Search Patient", menu_items, 2, UI_SIZE);
00379         utils_get_string(phone, PHONE_SIZE);
00380
00381         if (!utils_is_valid_phone(phone)) {
00382             ui_print_error("Invalid phone!");
00383             ui_pause();
00384             continue;
00385         }
00386         for (int i = 0; i < patient_count; i++) {
00387             if (strcmp(patients[i].phone, phone) == 0) {
00388                 ui_clear_screen();
00389                 ui_print_banner();
00390
00391                 ui_print_patient(patients[i], (patients[i].id - 1001));
00392                 ui_pause();
00393                 return;
00394             }
00395         }
00396         ui_print_error("Patient not found!");
00397         ui_pause();
00398         return;
00399     } while (1);
00400 }
00401
00402 void patient_search_by(void) {
00403     int choice;
00404
00405     do {
```

```

00406     ui_clear_screen();
00407     ui_print_banner();
00408
00409     const char* menu_items[] = {
00410         "Search by patient ID",
00411         "Search by patient name",
00412         "Search by patient phone",
00413         "Back to Patient Menu",
00414         "» "
00415     };
00416
00417     ui_print_menu("Search Patient", menu_items, 5, UI_SIZE);
00418     choice = utils_get_int();
00419
00420     switch (choice) {
00421         case 1:
00422             patient_search_by_id();
00423             break;
00424         case 2:
00425             patient_search_by_name();
00426             break;
00427         case 3:
00428             patient_search_by_phone();
00429             break;
00430         case 4:
00431             ui_print_info("Returning to receptionist menu...");
00432             ui_pause();
00433             break;
00434         default:
00435             ui_print_error("Invalid choice! Please try again.");
00436             ui_pause();
00437     }
00438     } while (choice != 4);
00439 }
00440
00441 int patient_search_id (int id) {
00442     for (int i = 0; i < patient_count; i++) {
00443         if (patients[i].id == id) {
00444             return i;
00445         }
00446     }
00447     return -1;
00448 }
00449
00450 void patient_update_name(const char* menu_items[], int index) {
00451     ui_clear_screen();
00452     ui_print_banner();
00453
00454     const char* menu[] = {
00455         menu_items[0],
00456         "Enter new name: ",
00457         "» "
00458     };
00459
00460     char name[NAME_SIZE];
00461     ui_print_menu("Update Patient", menu, 3, UI_SIZE);
00462     utils_get_string(name, NAME_SIZE);
00463     utils_fix_name(name);
00464
00465     if (!utils_is_valid_name(name)) {
00466         ui_print_error("Invalid name! Could not update.");
00467         ui_pause();
00468         return;
00469     }
00470     patients[index].name[0] = '\0';
00471     strncpy(patients[index].name, name, NAME_SIZE);
00472 }
00473
00474 void patient_update_phone(const char* menu_items[], int index) {
00475     ui_clear_screen();
00476     ui_print_banner();
00477
00478     const char* menu[] = {
00479         menu_items[1],
00480         "Enter new phone: ",
00481         "» "
00482     };
00483
00484     char phone[PHONE_SIZE];
00485     ui_print_menu("Update Patient", menu, 3, UI_SIZE);
00486     utils_get_string(phone, PHONE_SIZE);
00487
00488     if (!utils_is_valid_phone(phone)) {
00489         ui_print_error("Invalid phone! Could not update.");
00490         ui_pause();
00491         return;
00492     }

```

```

00493     patients[index].phone[0] = '\0';
00494     strncpy(patients[index].phone, phone, PHONE_SIZE);
00495 }
00496
00497 void patient_update_address(const char* menu_items[], int index) {
00498     ui_clear_screen();
00499     ui_print_banner();
00500
00501     const char* menu[] = {
00502         menu_items[2],
00503         "Enter new address: ",
00504         "» "
00505     };
00506
00507     char address[ADDRESS_SIZE];
00508     ui_print_menu("Update Patient", menu, 3, UI_SIZE);
00509     utils_get_string(address, ADDRESS_SIZE);
00510
00511     if (!utils_is_valid_address(address)) {
00512         ui_print_error("Invalid address! Could not update.");
00513         ui_pause();
00514         return;
00515     }
00516     patients[index].address[0] = '\0';
00517     strncpy(patients[index].address, address, ADDRESS_SIZE);
00518 }
00519
00520 void patient_update_blood_group(const char* menu_items[], int index) {
00521     ui_clear_screen();
00522     ui_print_banner();
00523
00524     const char* menu[] = {
00525         menu_items[3],
00526         "Enter new blood group: ",
00527         "» "
00528     };
00529
00530     char blood_group[BLOOD_SIZE];
00531     ui_print_menu("Update Patient", menu, 3, UI_SIZE);
00532     utils_get_string(blood_group, BLOOD_SIZE);
00533
00534     if (!utils_is_valid_blood_group(blood_group)) {
00535         ui_print_error("Invalid blood group! Could not update.");
00536         ui_pause();
00537         return;
00538     }
00539     utils_str_to_upper(blood_group);
00540     patients[index].blood_group[0] = '\0';
00541     strncpy(patients[index].blood_group, blood_group, BLOOD_SIZE);
00542 }
00543
00544 void patient_update_gender(const char* menu_items[], int index) {
00545     ui_clear_screen();
00546     ui_print_banner();
00547
00548     const char* menu[] = {
00549         menu_items[4],
00550         "Enter new gender: ",
00551         "» "
00552     };
00553
00554     char gender_input;
00555     ui_print_menu("Update Patient", menu, 3, UI_SIZE);
00556     while (1) {
00557         gender_input = utils_get_char();
00558         if (gender_input == 'M' || gender_input == 'm' || gender_input == 'F' || gender_input == 'f')
00559         {
00560             break;
00561         }
00562         ui_print_error("Invalid gender!");
00563         ui_pause();
00564     }
00565     if (gender_input == 'M' || gender_input == 'm') {
00566         patients[index].gender = MALE;
00567     } else if (gender_input == 'F' || gender_input == 'f') {
00568         patients[index].gender = FEMALE;
00569     }
00570 }
00571
00572 void patient_update_status(const char* menu_items[], int index) {
00573     ui_clear_screen();
00574     ui_print_banner();
00575
00576     const char* menu[] = {
00577         menu_items[5],
00578         "Enter new status: (1 for active, 2 for inactive) ",
00579         "» "

```

```

00579     };
00580
00581     bool status;
00582     int input;
00583     ui_print_menu("Update Patient", menu, 3, UI_SIZE);
00584     input = utils_get_int();
00585
00586     if (input == 1) {
00587         status = true;
00588     } else if (input == 2) {
00589         status = false;
00590     }
00591     patients[index].is_active = status;
00592 }
00593
00594 void patient_update_using_id() {
00595     ui_clear_screen();
00596     ui_print_banner();
00597
00598     int id;
00599
00600     do {
00601         ui_clear_screen();
00602         ui_print_banner();
00603
00604         const char* enter_id_items[] = {
00605             "Enter patient ID: (0 to go back) ",
00606             "» "
00607         };
00608
00609         ui_print_menu("Update Patient", enter_id_items, 2, UI_SIZE);
00610         id = utils_get_int();
00611
00612         if (id == 0) return;
00613
00614         if (!utils_is_valid_id(id, ROLE_PATIENT)) {
00615             ui_print_error("Invalid ID!");
00616             ui_pause();
00617         } else {
00618             ui_print_info("Patient found!");
00619             ui_pause();
00620             break;
00621         }
00622     } while (1);
00623
00624     int index = patient_search_id(id);
00625     if (index == -1) return;
00626
00627     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];
00628
00629     snprintf(name_line, NAME_LINE_SIZE, "Name: %s", patients[index].name);
00630     snprintf(phone_line, PHONE_LINE_SIZE, "Phone: %s", patients[index].phone);
00631     snprintf(address_line, ADDRESS_LINE_SIZE, "Address: %s", patients[index].address);
00632     snprintf(blood_group_line, BLOOD_LINE_SIZE, "Blood Group: %s", patients[index].blood_group);
00633
00634     int patient_id = patients[index].id;
00635
00636     if (patients[index].gender == MALE) {
00637         snprintf(gender_line, GENDER_LINE_SIZE, "Gender: Male");
00638     } else {
00639         snprintf(gender_line, GENDER_LINE_SIZE, "Gender: Female");
00640     }
00641     if (patients[index].is_active) {
00642         snprintf(status_line, STATUS_LINE_SIZE, "Status: Active");
00643     } else {
00644         snprintf(status_line, STATUS_LINE_SIZE, "Status: Discharged");
00645     }
00646
00647     const char* menu_items[] = {
00648         name_line,
00649         phone_line,
00650         address_line,
00651         blood_group_line,
00652         gender_line,
00653         "Go back",
00654         "» "
00655     };
00656
00657     char title[50];
00658     snprintf(title, 50, "Update Patient | ID: %d", patient_id);
00659
00660     ui_clear_screen();
00661     ui_print_banner();
00662     ui_print_menu(title, menu_items, 7, UI_SIZE);
00663
00664     int choice = utils_get_int();

```

```

00665
00666     switch (choice) {
00667     case 1:
00668         patient_update_name(menu_items, index);
00669         break;
00670     case 2:
00671         patient_update_phone(menu_items, index);
00672         break;
00673     case 3:
00674         patient_update_address(menu_items, index);
00675         break;
00676     case 4:
00677         patient_update_blood_group(menu_items, index);
00678         break;
00679     case 5:
00680         patient_update_gender(menu_items, index);
00681         break;
00682     case 6:
00683         ui_print_info("Returning to receptionist menu...");
00684         ui_pause();
00685         break;
00686     default:
00687         ui_print_error("Invalid choice! Please try again.");
00688         ui_pause();
00689         break;
00690     }
00691 }
00692
00693 void patient_discharge() {
00694     int id;
00695     while (1) {
00696         ui_clear_screen();
00697         ui_print_banner();
00698
00699         const char* enter_id_items[] = {
00700             "Enter patient ID (0 to cancel): ",
00701             "» "
00702         };
00703
00704         ui_print_menu("Discharge Patient", enter_id_items, 2, UI_SIZE);
00705         id = utils_get_int();
00706
00707         if (id == 0) {
00708             ui_print_info("User Pressed 0. \nCanceling deletion...");
00709             ui_pause();
00710             return;
00711         }
00712
00713         if (!utils_is_valid_id(id, ROLE_PATIENT)) {
00714             ui_print_error("Invalid ID!");
00715             ui_pause();
00716             continue;
00717         }
00718         break;
00719     }
00720
00721     int index = patient_search_id(id);
00722     if (index == -1) {
00723         ui_print_error("Patient not found!");
00724         ui_pause();
00725         return;
00726     }
00727
00728     ui_clear_screen();
00729     ui_print_banner();
00730     ui_print_patient(patients[index], index);
00731
00732     const char* menu[] = {
00733         "Confirm Discharge",
00734         "Cancel",
00735         "» "
00736     };
00737
00738     ui_print_menu("Discharge Patient", menu, 3, UI_SIZE);
00739     int input = utils_get_int();
00740
00741     if (input == 1) {
00742         patients[index].is_active = false;
00743         ui_print_success("Patient discharged successfully!");
00744         patient_available--;
00745         patient_unavailable++;
00746         ui_pause();
00747     } else {
00748         ui_print_info("Discharge cancelled.");
00749         ui_pause();
00750     }
00751 }

```

```

00752
00753 void patient_view_discharged(void) {
00754     int count = 0;
00755     ui_clear_screen();
00756     ui_print_banner();
00757
00758     if (patient_unavailable == 0) {
00759         const char* menu_items[] = {"No discharged patients found!"};
00760         ui_print_menu("Discharged Patients", menu_items, 1, UI_SIZE);
00761         ui_pause();
00762         return;
00763     }
00764
00765     for (int i = 0; i < patient_count; i++) {
00766         if (!patients[i].is_active) {
00767             ui_print_patient(patients[i], count++);
00768         }
00769     }
00770     ui_pause();
00771 }

```

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/utlis.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

<i>id</i>	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.](#)

```
00001
00005
00006 #include <stdio.h>
00007 #include <string.h>
00008 #include "../include/receptionist.h"
00009 #include "../include/patient.h"
00010 #include "../include/doctor.h"
00011 #include "../include/appointment.h"
00012 #include "../include/utils.h"
00013 #include "../include/ui.h"
00014 #include "../include/hospital.h"
00015
00016 void receptionist_patient_menu(void) {
00017     int choice;
00018
00019     do {
00020         ui_clear_screen();
00021         ui_print_banner();
00022
00023         const char* menu_items[] = {
00024             "Add Patient",
00025             "View Active Patients",
00026             "View Discharged Patients",
00027             "Search Patient",
00028             "Update Patient",
00029             "Discharge Patient",
00030             "Back",
00031             "» "
00032         };
00033
00034         ui_print_menu("Patient Management", menu_items, 8, UI_SIZE);
00035         choice = utils_get_int();
00036
00037         switch (choice) {
00038             case 1:
00039                 patient_add();
00040                 patient_save_to_file();
00041                 break;
00042             case 2:
00043                 patient_view_all();
00044                 break;
00045             case 3:
00046                 patient_view_discharged();
00047                 break;
00048             case 4:
00049                 patient_search_by();
00050                 break;
00051             case 5:
00052                 patient_update_using_id();
00053                 patient_save_to_file();
00054                 break;
00055             case 6:
00056                 patient_discharge();
00057                 patient_save_to_file();
00058                 break;
00059             case 7:
00060                 ui_print_info("Returning to receptionist menu...");
00061                 ui_pause();
00062                 break;
00063             default:
00064                 ui_print_error("Invalid choice!");
```

```

00065         ui_pause();
00066     }
00067     } while (choice != 7);
00068 }
00069
00070 void receptionist_appointment_menu(void) {
00071     int choice;
00072
00073     do {
00074         ui_clear_screen();
00075         ui_print_banner();
00076
00077         const char* menu_items[] = {
00078             "Create Appointment",
00079             "View All Appointments",
00080             "Back",
00081             "» "
00082         };
00083
00084         ui_print_menu("Appointment Management", menu_items, 4, UI_SIZE);
00085         choice = utils_get_int();
00086
00087         switch (choice) {
00088             case 1:
00089                 appointment_create();
00090                 break;
00091             case 2:
00092                 ui_clear_screen();
00093                 ui_print_banner();
00094                 for (int i = 0; i < appointment_count; i++) {
00095                     ui_print_appointment(appointments[i], i);
00096                 }
00097                 if (appointment_count == 0) {
00098                     const char* no_appt[] = {"No appointments found!"};
00099                     ui_print_menu("All Appointments", no_appt, 1, UI_SIZE);
00100                 }
00101                 ui_pause();
00102                 break;
00103             case 3:
00104                 ui_print_info("Returning to receptionist menu...");
00105                 ui_pause();
00106                 break;
00107             default:
00108                 ui_print_error("Invalid choice!");
00109                 ui_pause();
00110         }
00111     } while (choice != 3);
00112 }
00113
00114 void receptionist_menu(void) {
00115     int choice;
00116
00117     do {
00118         ui_clear_screen();
00119         ui_print_banner();
00120
00121         const char* menu_items[] = {
00122             "Patient Management",
00123             "Appointment Management",
00124             "Logout",
00125             "» "
00126         };
00127
00128         ui_print_menu("Receptionist Portal", menu_items, 4, UI_SIZE);
00129         choice = utils_get_int();
00130
00131         switch (choice) {
00132             case 1:
00133                 receptionist_patient_menu();
00134                 break;
00135             case 2:
00136                 receptionist_appointment_menu();
00137                 break;
00138             case 3:
00139                 ui_print_info("Logging out...");
00140                 ui_pause();
00141                 break;
00142             default:
00143                 ui_print_error("Invalid choice!");
00144                 ui_pause();
00145         }
00146     } while (choice != 3);
00147 }
00148
00149 /*
00150 *=====
00151 *
RECEPTIONIST DATA FUNCTIONS

```

```

00152  *=====
00153  */
00154
00155 int receptionist_save_to_file(void) {
00156     FILE* file = fopen(RECEPTIONISTS_FILE, "wb");
00157     if (file == NULL) {
00158         return -1;
00159     }
00160     if (fwrite(&receptionist_count, sizeof(int), 1, file) != 1 ||
00161         fwrite(&receptionist_available, sizeof(int), 1, file) != 1 ||
00162         fwrite(receptionists, sizeof(Receptionist), receptionist_count, file) !=
00163         (size_t)receptionist_count) {
00164         fclose(file);
00165         return -1;
00166     }
00167     fclose(file);
00168     return 0;
00169 }
00170 int receptionist_load_from_file(void) {
00171     FILE* file = fopen(RECEPTIONISTS_FILE, "rb");
00172     if (file == NULL) {
00173         return -1;
00174     }
00175     if (fread(&receptionist_count, sizeof(int), 1, file) != 1 ||
00176         fread(&receptionist_available, sizeof(int), 1, file) != 1) {
00177         fclose(file);
00178         return -1;
00179     }
00180     if (receptionist_count < 0 || receptionist_count > MAX_RECEPTIONISTS) {
00181         fclose(file);
00182         receptionist_count = 0;
00183         return -1;
00184     }
00185     if (fread(receptionists, sizeof(Receptionist), receptionist_count, file) !=
00186     (size_t)receptionist_count) {
00187         fclose(file);
00188         receptionist_count = 0;
00189         return -1;
00190     }
00191     fclose(file);
00192     return 0;
00193 }
00194 int receptionist_search_id(int id) {
00195     for (int i = 0; i < receptionist_count; i++) {
00196         if (receptionists[i].id == id) {
00197             return i;
00198         }
00199     }
00200     return -1;
00201 }
00202
00203 void receptionist_view_all(void) {
00204     int count = 0;
00205     ui_clear_screen();
00206     ui_print_banner();
00207
00208     if (receptionist_available == 0 || receptionist_count == 0) {
00209         const char* menu_items[] = {"No receptionists found!"};
00210         ui_print_menu("View All Receptionists", menu_items, 1, UI_SIZE);
00211         ui_pause();
00212         return;
00213     }
00214
00215     for (int i = 0; i < receptionist_count; i++) {
00216         if (receptionists[i].is_active) {
00217             ui_print_receptionist(receptionists[i], count++);
00218         }
00219     }
00220     ui_pause();
00221 }
00222
00223 void receptionist_view_discharged(void) {
00224     int count = 0;
00225     ui_clear_screen();
00226     ui_print_banner();
00227
00228     if (receptionist_unavailable == 0) {
00229         const char* menu_items[] = {"No inactive receptionists found!"};
00230         ui_print_menu("Inactive Receptionists", menu_items, 1, UI_SIZE);
00231         ui_pause();
00232         return;
00233     }
00234
00235     for (int i = 0; i < receptionist_count; i++) {
00236         if (!receptionists[i].is_active) {

```

```

00237         ui_print_receptionist(receptionists[i], count++);
00238     }
00239 }
00240 ui_pause();
00241 }
00242
00243 void receptionist_discharge(void) {
00244     int id;
00245     while (1) {
00246         ui_clear_screen();
00247         ui_print_banner();
00248
00249         const char* enter_id_items[] = {
00250             "Enter receptionist ID (0 to cancel): ",
00251             "» "
00252         };
00253
00254         ui_print_menu("Deactivate Receptionist", enter_id_items, 2, UI_SIZE);
00255         id = utils_get_int();
00256
00257         if (id == 0) {
00258             ui_print_info("Cancelled.");
00259             ui_pause();
00260             return;
00261         }
00262
00263         if (!utils_is_valid_id(id, ROLE_RECEPTIONIST)) {
00264             ui_print_error("Invalid ID!");
00265             ui_pause();
00266             continue;
00267         }
00268         break;
00269     }
00270
00271     int index = receptionist_search_id(id);
00272     if (index == -1) {
00273         ui_print_error("Receptionist not found!");
00274         ui_pause();
00275         return;
00276     }
00277
00278     ui_clear_screen();
00279     ui_print_banner();
00280     ui_print_receptionist(receptionists[index], index);
00281
00282     const char* menu[] = {
00283         "Confirm Deactivate",
00284         "Cancel",
00285         "» "
00286     };
00287
00288     ui_print_menu("Deactivate Receptionist", menu, 3, UI_SIZE);
00289     int input = utils_get_int();
00290
00291     if (input == 1) {
00292         receptionists[index].is_active = false;
00293         ui_print_success("Receptionist deactivated successfully!");
00294         receptionist_available--;
00295         receptionist_unavailable++;
00296         receptionist_save_to_file();
00297         ui_pause();
00298     } else {
00299         ui_print_info("Deactivation cancelled.");
00300         ui_pause();
00301     }
00302 }

```

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

<i>doctor</i>	The doctor to print.
<i>index</i>	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

<i>message</i>	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

<i>title</i>	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

<i>message</i>	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

<i>title</i>	The title of the menu.
--------------	------------------------

<i>items</i>	The array of menu items.
<i>item_count</i>	The number of menu items.
<i>box_width</i>	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

<i>patient</i>	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

<i>receptionist</i>	The receptionist to print.
<i>index</i>	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

<i>message</i>	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

<i>message</i>	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/utlis.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 void ui_print_banner(void){
00056     unsigned char b = 219; // Full block character
00057
00058     char* hms =
00059     "#####\n"
00060     "#                                           #\n"
00061     "#           X X X X XXXX                    #\n"
00062     "#           X X XX XX X                      #\n"
00063     "#           XXXX X X X XXXX                  #\n"
00064     "#           X X X X X                        #\n"
00065     "#           X X X X XXXX                    #\n"
00066     "#           HEALTHCARE MANAGEMENT SYSTEM    #\n"
00067     "#                                           #\n"
00068     "#####\n";
00069
00070     printf("\n");
00071
00072     while (*hms) {
00073         if (*hms == 'X') {
00074             printf(BRIGHT_RED "%c%c" RESET, b, b);
00075         }
00076         else if (*hms == '#') {
00077             printf(SOFT_GRAY "%c%c" RESET, b, b);
00078         }
00079         else if (*hms == '\n') {
00080             printf("\n");
00081         }
00082         else if (isalpha(*hms)){
00083             printf(BOLD CYAN "%c" RESET " ", *hms);
00084         }
00085     }

```



```

00086     }
00087     else {
00088         printf(" ");
00089     }
00090     hms++;
00091 }
00092 printf("\n");
00093 }
00094
00095 void ui_print_menu
00096 (
00097     const char *title,
00098     const char *items[],
00099     int item_count,
00100     int box_width
00101 ) {
00102     unsigned char h = 205; //
00103     unsigned char v = 186; //
00104     unsigned char tl = 201; //
00105     unsigned char tr = 187; //
00106     unsigned char bl = 200; //
00107     unsigned char br = 188; //
00108
00109     printf(BRIGHT_BLACK "%c", tl);
00110     for (int i = 0; i < box_width; i++) printf("%c", h);
00111     printf("%c" RESET "\n", tr);
00112
00113     char title_upper[100];
00114     strncpy(title_upper, title, sizeof(title_upper) - 1);
00115     title_upper[sizeof(title_upper) - 1] = '\0';
00116     utils_str_to_upper(title_upper);
00117
00118     int title_len = strlen(title_upper) + 4;
00119     int title_padding = (box_width - title_len) / 2;
00120     printf(BRIGHT_BLACK "%c" RESET, v);
00121     for (int i = 0; i < title_padding; i++) printf(" ");
00122     ui_print_header(title_upper);
00123     for (int i = 0; i < box_width - title_padding - title_len; i++) printf(" ");
00124     printf(BRIGHT_BLACK "%c" RESET "\n", v);
00125
00126     printf(BRIGHT_BLACK "%c" RESET, v);
00127     for (int i = 0; i < box_width; i++) printf(" ");
00128     printf(BRIGHT_BLACK "%c" RESET "\n", v);
00129
00130     for (int i = 0; i < item_count - 1; i++) {
00131         int item_len = strlen(items[i]);
00132         printf(BRIGHT_BLACK "%c" RESET " " SOFT_YELLOW_BOLD "%d. %s" RESET, v, i + 1, items[i]);
00133         for (int j = 0; j < box_width - item_len - 5; j++) printf(" ");
00134         printf(BRIGHT_BLACK "%c" RESET "\n", v);
00135     }
00136
00137     printf(BRIGHT_BLACK "%c" RESET, v);
00138     for (int i = 0; i < box_width; i++) printf(" ");
00139     printf(BRIGHT_BLACK "%c" RESET "\n", v);
00140     printf(BRIGHT_BLACK "%c", bl);
00141     for (int i = 0; i < box_width; i++) printf("%c", h);
00142     printf("%c" RESET "\n\n", br);
00143
00144     printf(BOLD SOFT_GREEN "%s" RESET, items[item_count - 1]);
00145 }
00146
00147 void ui_print_patient(Patient patient, int index) {
00148
00149     char id_line[ID_LINE_SIZE];
00150     snprintf(id_line, sizeof(id_line), "Patient ID: %d", patient.id);
00151
00152     char name_line[NAME_LINE_SIZE];
00153     snprintf(name_line, sizeof(name_line), "Name: %s", patient.name);
00154
00155     char age_line[AGE_LINE_SIZE];
00156     snprintf(age_line, sizeof(age_line), "Age: %d", patient.age);
00157
00158     char gender_line[GENDER_LINE_SIZE];
00159     snprintf(gender_line, sizeof(gender_line), "Gender: %s", patient.gender == MALE ? "Male" :
00160 "Female");
00161
00162     char phone_line[PHONE_LINE_SIZE];
00163     snprintf(phone_line, sizeof(phone_line), "Phone: %s", patient.phone);
00164
00165     char address_line[ADDRESS_LINE_SIZE];
00166     snprintf(address_line, sizeof(address_line), "Address: %s", patient.address);
00167
00168     char blood_group_line[BLOOD_LINE_SIZE];
00169     snprintf(blood_group_line, sizeof(blood_group_line), "Blood Group: %s", patient.blood_group);
00170
00171     char status_line[STATUS_LINE_SIZE];
00172     snprintf(status_line, sizeof(status_line), "Status: %s", patient.is_active ? "Active" :

```

```

    "Inactive");
00172
00173     const char* items[] = {
00174         id_line,
00175         name_line,
00176         age_line,
00177         gender_line,
00178         phone_line,
00179         address_line,
00180         blood_group_line,
00181         status_line,
00182         ""
00183     };
00184
00185     char title[70];
00186     snprintf(title, sizeof(title), "Patient %d", index + 1);
00187     ui_print_menu(title, items, 9, 72);
00188 }
00189
00190 void ui_print_doctor(Doctor doctor, int index) {
00191
00192     char id_line[70];
00193     snprintf(id_line, sizeof(id_line), "Doctor ID: %d", doctor.id);
00194
00195     char name_line[70];
00196     snprintf(name_line, sizeof(name_line), "Name: %s", doctor.name);
00197
00198     char phone_line[70];
00199     snprintf(phone_line, sizeof(phone_line), "Phone: %s", doctor.phone);
00200
00201     char email_line[70];
00202     snprintf(email_line, sizeof(email_line), "Email: %s", doctor.email);
00203
00204     char spec_line[70];
00205     snprintf(spec_line, sizeof(spec_line), "Specialization: %s", doctor.specialization);
00206
00207     char room_line[70];
00208     snprintf(room_line, sizeof(room_line), "Room Number: %d", doctor.room_number);
00209
00210     char avail_line[70];
00211     snprintf(avail_line, sizeof(avail_line), "Available: %s", doctor.is_available ? "Yes" : "No");
00212
00213     char status_line[70];
00214     snprintf(status_line, sizeof(status_line), "Status: %s", doctor.is_active ? "Active" :
    "Inactive");
00215
00216     const char* items[] = {
00217         id_line,
00218         name_line,
00219         phone_line,
00220         email_line,
00221         spec_line,
00222         room_line,
00223         avail_line,
00224         status_line,
00225         ""
00226     };
00227
00228     char title[70];
00229     snprintf(title, sizeof(title), "Doctor %d", index + 1);
00230     ui_print_menu(title, items, 9, 72);
00231 }
00232
00233 void ui_print_receptionist(Receptionist receptionist, int index) {
00234
00235     char id_line[70];
00236     snprintf(id_line, sizeof(id_line), "Receptionist ID: %d", receptionist.id);
00237
00238     char name_line[70];
00239     snprintf(name_line, sizeof(name_line), "Name: %s", receptionist.name);
00240
00241     char phone_line[70];
00242     snprintf(phone_line, sizeof(phone_line), "Phone: %s", receptionist.phone);
00243
00244     char email_line[70];
00245     snprintf(email_line, sizeof(email_line), "Email: %s", receptionist.email);
00246
00247     char avail_line[70];
00248     snprintf(avail_line, sizeof(avail_line), "Available: %s", receptionist.is_available ? "Yes" :
    "No");
00249
00250     char status_line[70];
00251     snprintf(status_line, sizeof(status_line), "Status: %s", receptionist.is_active ? "Active" :
    "Inactive");
00252
00253     const char* items[] = {
00254         id_line,

```

```

00255         name_line,
00256         phone_line,
00257         email_line,
00258         avail_line,
00259         status_line,
00260         ""
00261     };
00262
00263     char title[70];
00264     snprintf(title, sizeof(title), "Receptionist %d", index + 1);
00265     ui_print_menu(title, items, 7, 72);
00266 }
00267
00268 void ui_dummy_loading(int time) {
00269     ui_clear_screen();
00270     ui_print_banner();
00271
00272     const char arr[] = {'|', '/', '-', '\\'};
00273     int i = 0;
00274     int loading_time = 0;
00275     printf("\n\n");
00276     while (1) {
00277         printf(BOLD BRIGHT_RED "\rLoading data from files... " RESET "%c", arr[i]);
00278         fflush(stdout);
00279         i = (i + 1) % 4;
00280
00281         #ifdef _WIN32
00282             Sleep(100);
00283         #else
00284             usleep(100000);
00285         #endif
00286
00287         loading_time++;
00288         if (loading_time == time) break;
00289     }
00290
00291     ui_print_success("Successfully loaded data from files!\n");
00292
00293     ui_pause();
00294
00295 }

```

6.39 src/utls.c File Reference

Utility functions for Healthcare Management System.

```

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "../include/utls.h"

```

Functions

- void [utls_clear_input_buffer](#) (void)
- void [utls_pause](#) (void)
- int [utls_get_int](#) (void)
- float [utls_get_float](#) (void)
- double [utls_get_double](#) (void)
- char [utls_get_char](#) (void)
- char * [utls_get_string](#) (char *str, size_t size)
- bool [utls_is_valid_phone](#) (const char *phone)
- bool [utls_is_valid_email](#) (const char *email)
- bool [utls_is_valid_name](#) (const char *name)
- bool [utls_is_valid_id](#) (int id, [UserRole](#) role)
- char * [utls_str_to_upper](#) (char *str)
- char * [utls_fix_name](#) (char *name)
- bool [utls_is_valid_blood_group](#) (const char *blood_group)
- bool [utls_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

<i>name</i>	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 [utls.c](#).

6.39.2.6 utls_get_int()

```
int utls_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 [utls.c](#).

6.39.2.7 utls_get_string()

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

Scans a string value from the user.

Parameters

<i>str</i>	The buffer to store the string.
<i>size</i>	The maximum number of characters to read.

Returns

Pointer to string on success, NULL on failure.

Definition at line 61 of file [utls.c](#).

6.39.2.8 utls_is_valid_address()

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

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

Parameters

<i>address</i>	The address string to validate.
----------------	---------------------------------

Returns

true if valid, false otherwise.

Definition at line 204 of file [utls.c](#).

6.39.2.9 utls_is_valid_blood_group()

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

Validates if a blood group is in correct format.

Parameters

<i>blood_group</i>	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

<i>email</i>	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

<i>id</i>	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

<i>name</i>	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

<i>str</i>	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;
```

```
00042 }
00043
00044 double utils_get_double(void) {
00045     double num;
00046     while (scanf("%lf", &num) != 1) {
00047         printf("Invalid input. Please enter a double: ");
00048         utils_clear_input_buffer();
00049     }
00050     utils_clear_input_buffer();
00051     return num;
00052 }
00053
00054 char utils_get_char(void) {
00055     char ch;
00056     scanf("%c", &ch);
00057     utils_clear_input_buffer();
00058     return ch;
00059 }
00060
00061 char* utils_get_string(char *str, size_t size) {
00062     while (1) {
00063         if (fgets(str, size, stdin) != NULL) {
00064             str[strcspn(str, "\n")] = '\0';
00065             if (str[0] != '\0') {
00066                 return str;
00067             }
00068             printf("Input cannot be empty. Try again: ");
00069         } else {
00070             return NULL;
00071         }
00072     }
00073 }
00074
00075
00076
00077
00078
00079
00080
00081 bool utils_is_valid_phone(const char *phone) {
00082     if (phone == NULL || phone[0] == '\0') return false;
00083
00084     int digit_count = 0;
00085     for (size_t i = 0; phone[i] != '\0'; i++) {
00086         if (!isdigit(phone[i])) return false;
00087         digit_count++;
00088     }
00089
00090     if (digit_count != 11) return false;
00091     if (phone[0] != '0' || phone[1] != '1') return false;
00092     if (phone[2] < '3' || phone[2] > '9') return false;
00093
00094     return true;
00095 }
00096
00097 bool utils_is_valid_email(const char *email) {
00098     if (email == NULL || email[0] == '\0') return false;
00099
00100     int at_count = 0;
00101     int at_position = -1;
00102     int last_dot_position = -1;
00103     int length = 0;
00104
00105     for (size_t i = 0; email[i] != '\0'; i++) {
00106         char c = email[i];
00107
00108         if (c != '@' && !(isdigit(c)) && !(isalpha(c)) && c != '.' && c != '_') return false;
00109
00110         if (c == '@') {
00111             at_count++;
00112             at_position = i;
00113         }
00114         if (c == '.') {
00115             last_dot_position = i;
00116         }
00117         length++;
00118     }
00119
00120     if (at_count != 1) return false;
00121     if (at_position == 0) return false;
00122     if (last_dot_position <= at_position) return false;
00123     if (last_dot_position == length - 1) return false;
00124
00125     return true;
00126 }
00127
00128 bool utils_is_valid_name(const char *name) {
00129     if (name == NULL || name[0] == '\0') return false;
00130
00131     for (size_t i = 0; name[i] != '\0'; i++) {
00132         if (!isalpha(name[i]) && name[i] != ' ') {
00133             return false;
00134         }
00135     }
00136 }
```



```
00134     }
00135 }
00136
00137     return true;
00138 }
00139
00140 bool utils_is_valid_id(int id, UserRole role) {
00141     if (id < 0) return false;
00142
00143     if (role == ROLE_PATIENT) {
00144         if (id < 1001) return false;
00145         if (id >= 2001) return false;
00146     } else if (role == ROLE_DOCTOR) {
00147         if (id < 2001) return false;
00148         if (id >= 3001) return false;
00149     } else if (role == ROLE_ADMIN) {
00150         if (id < 3001) return false;
00151         if (id >= 4001) return false;
00152     } else if (role == ROLE_RECEPTIONIST) {
00153         if (id < 4001) return false;
00154         if (id >= 5001) return false;
00155     }
00156
00157     return true;
00158 }
00159
00160
00161 char* utils_str_to_upper(char *str) {
00162     if (str == NULL) return NULL;
00163     for (int i = 0; str[i] != '\0'; i++) {
00164         str[i] = toupper(str[i]);
00165     }
00166     return str;
00167 }
00168
00169 char* utils_fix_name(char *name) {
00170     if (name == NULL || name[0] == '\0') return name;
00171
00172     bool new_word = true;
00173     for (int i = 0; name[i] != '\0'; i++) {
00174         if (name[i] == ' ') {
00175             new_word = true;
00176         } else if (new_word) {
00177             name[i] = toupper(name[i]);
00178             new_word = false;
00179         } else {
00180             name[i] = tolower(name[i]);
00181         }
00182     }
00183     return name;
00184 }
00185
00186 bool utils_is_valid_blood_group(const char *blood_group) {
00187     if (blood_group == NULL || blood_group[0] == '\0') return false;
00188
00189     const char* valid_blood_groups[] = {
00190         "A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-", "U",
00191         "a+", "a-", "b+", "b-", "ab+", "ab-", "o+", "o-", "u"
00192     };
00193     int count = sizeof(valid_blood_groups) / sizeof(valid_blood_groups[0]);
00194
00195     for (int i = 0; i < count; i++) {
00196         if (strcmp(blood_group, valid_blood_groups[i]) == 0) {
00197             return true;
00198         }
00199     }
00200
00201     return false;
00202 }
00203
00204 bool utils_is_valid_address(const char *address) {
00205     if (address == NULL || address[0] == '\0') return false;
00206
00207     for (int i = 0; address[i] != '\0'; i++) {
00208         char c = address[i];
00209         if (!isalnum(c) && c != ' ' && c != ',' && c != '.') {
00210             return false;
00211         }
00212     }
00213     return true;
00214 }
```

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.](#)

```

00001 #include <stdio.h>
00002 #include "../include/utils.h"
00003 #include <stdbool.h>
00004
00005 int main() {
00006     char phone[12];
00007     utils_get_string(phone, 12);
00008     if(utils_is_valid_phone(phone)) {
00009         printf("Valid phone number\n");
00010     } else {
00011         printf("Invalid phone number\n");
00012     }
00013     if (utils_is_valid_email("noice.nice@gmail.com")) {
00014         printf("Valid email\n");
00015     } else {
00016         printf("Invalid email\n");
00017     }
00018     utils_pause();
00019     return 0;
00020 }
```

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.](#)

```

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.](#)

```
00001 #include <stdio.h>
00002 #include "../include/ui.h"
00003 #include "../include/utils.h"
00004 #include "../include/patient.h"
00005 #include "../include/receptionist.h"
00006
00007 int main() {
00008     // ui_clear_screen();
00009     // ui_print_banner();
00010     // printf("\n\n\n");
00011     // // ui_print_start_menu();
00012     // const char* menu[] = {"Login", "Register", "Exit"};
00013     // ui_print_menu("Main Menu", menu, 3, 72);
00014     // ui_pause();
00015
00016     receptionist_menu();
00017     return 0;
00018 }
```

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 [utills_test.c](#).

6.49.1.2 test_utils_get_char()

```
void test_utils_get_char ()
```

Definition at line 15 of file [utills_test.c](#).

6.49.1.3 test_utils_get_double()

```
void test_utils_get_double ()
```

Definition at line 35 of file [utills_test.c](#).

6.49.1.4 test_utils_get_float()

```
void test_utils_get_float ()
```

Definition at line 25 of file [utills_test.c](#).

6.49.1.5 test_utils_get_int()

```
void test_utils_get_int ()
```

Definition at line 5 of file [utills_test.c](#).

6.49.1.6 test_utils_get_string()

```
void test_utils_get_string ()
```

Definition at line 45 of file [utills_test.c](#).

6.49.1.7 test_utils_is_valid_address()

```
void test_utils_is_valid_address ()
```

Definition at line 139 of file [utills_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 [utills_test.c](#).

6.49.1.9 test_utils_is_valid_email()

```
void test_utils_is_valid_email ()
```

Definition at line 74 of file [utills_test.c](#).

6.49.1.10 test_utils_is_valid_phone()

```
void test_utils_is_valid_phone ()
```

Definition at line 56 of file [utills_test.c](#).

6.49.1.11 test_utils_str_to_upper()

```
void test_utils_str_to_upper ()
```

Definition at line 92 of file [utills_test.c](#).

6.50 utils_test.c

[Go to the documentation of this file.](#)

```

00001 #include <stdio.h>
00002 #include <string.h>
00003 #include "../include/utils.h"
00004
00005 void test_utils_get_int() {
00006     printf("Enter an integer: ");
00007     int expected = 5;
00008     int actual = utils_get_int();
00009     int success = (actual == expected);
00010     printf(" Expected: %d\n", expected);
00011     printf(" Actual:   %d\n", actual);
00012     printf(" Success:  %s\n\n", success ? "Yes" : "No");
00013 }
00014
00015 void test_utils_get_char() {
00016     printf("Enter a character: ");
00017     char expected = 'c';
00018     char actual = utils_get_char();
00019     int success = (actual == expected);
00020     printf(" Expected: %c\n", expected);
00021     printf(" Actual:   %c\n", actual);
00022     printf(" Success:  %s\n\n", success ? "Yes" : "No");
00023 }
00024
00025 void test_utils_get_float() {
00026     printf("Enter a float: ");
00027     float expected = 3.14;
00028     float actual = utils_get_float();
00029     int success = (actual >= 3.13 && actual <= 3.15);
00030     printf(" Expected: %.2f\n", expected);
00031     printf(" Actual:   %.2f\n", actual);
00032     printf(" Success:  %s\n\n", success ? "Yes" : "No");
00033 }
00034
00035 void test_utils_get_double() {
00036     printf("Enter a double: ");
00037     double expected = 3.141593;
00038     double actual = utils_get_double();
00039     int success = (actual >= 3.141592 && actual <= 3.141594);
00040     printf(" Expected: %.6f\n", expected);
00041     printf(" Actual:   %.6f\n", actual);
00042     printf(" Success:  %s\n\n", success ? "Yes" : "No");
00043 }
00044
00045 void test_utils_get_string() {
00046     printf("Enter a string: ");
00047     char expected[] = "happy birthday";
00048     char actual[50];
00049     utils_get_string(actual, 50);
00050     int success = (strcmp(actual, expected) == 0);
00051     printf(" Expected: %s\n", expected);
00052     printf(" Actual:   %s\n", actual);
00053     printf(" Success:  %s\n\n", success ? "Yes" : "No");
00054 }
00055
00056 void test_utils_is_valid_phone() {
00057     printf("Testing utils_is_valid_phone():\n\n");
00058
00059     char *phone1 = "01315648613";
00060     int expected1 = 1, actual1 = utils_is_valid_phone(phone1);
00061     printf(" Input:   %s\n", phone1);
00062     printf(" Expected: %s\n", expected1 ? "Valid" : "Invalid");
00063     printf(" Actual:   %s\n", actual1 ? "Valid" : "Invalid");
00064     printf(" Success:  %s\n\n", (expected1 == actual1) ? "Yes" : "No");
00065
00066     char *phone2 = "1234567890";
00067     int expected2 = 0, actual2 = utils_is_valid_phone(phone2);
00068     printf(" Input:   %s\n", phone2);
00069     printf(" Expected: %s\n", expected2 ? "Valid" : "Invalid");
00070     printf(" Actual:   %s\n", actual2 ? "Valid" : "Invalid");
00071     printf(" Success:  %s\n\n", (expected2 == actual2) ? "Yes" : "No");
00072 }
00073
00074 void test_utils_is_valid_email() {
00075     printf("Testing utils_is_valid_email():\n\n");
00076
00077     char *email1 = "this.is.an.email@gmail.com";
00078     int expected1 = 1, actual1 = utils_is_valid_email(email1);
00079     printf(" Input:   %s\n", email1);
00080     printf(" Expected: %s\n", expected1 ? "Valid" : "Invalid");
00081     printf(" Actual:   %s\n", actual1 ? "Valid" : "Invalid");
00082     printf(" Success:  %s\n\n", (expected1 == actual1) ? "Yes" : "No");
00083 }

```

```

00084     char *email2 = "noatsign.com";
00085     int expected2 = 0, actual2 = utils_is_valid_email(email2);
00086     printf(" Input: %s\n", email2);
00087     printf(" Expected: %s\n", expected2 ? "Valid" : "Invalid");
00088     printf(" Actual: %s\n", actual2 ? "Valid" : "Invalid");
00089     printf(" Success: %s\n", (expected2 == actual2) ? "Yes" : "No");
00090 }
00091
00092 void test_utils_str_to_upper() {
00093     printf("Testing utils_str_to_upper():\n\n");
00094
00095     char input1[] = "hello world";
00096     char expected1[] = "HELLO WORLD";
00097     char *actual1 = utils_str_to_upper(input1);
00098     int success1 = (strcmp(actual1, expected1) == 0);
00099     printf(" Input: hello world\n");
00100     printf(" Expected: %s\n", expected1);
00101     printf(" Actual: %s\n", actual1);
00102     printf(" Success: %s\n", success1 ? "Yes" : "No");
00103
00104     char input2[] = "Test123";
00105     char expected2[] = "TEST123";
00106     char *actual2 = utils_str_to_upper(input2);
00107     int success2 = (strcmp(actual2, expected2) == 0);
00108     printf(" Input: Test123\n");
00109     printf(" Expected: %s\n", expected2);
00110     printf(" Actual: %s\n", actual2);
00111     printf(" Success: %s\n", success2 ? "Yes" : "No");
00112 }
00113
00114 void test_utils_is_valid_blood_group() {
00115     printf("Testing utils_is_valid_blood_group():\n\n");
00116
00117     char *blood1 = "A+";
00118     int expected1 = 1, actual1 = utils_is_valid_blood_group(blood1);
00119     printf(" Input: %s\n", blood1);
00120     printf(" Expected: %s\n", expected1 ? "Valid" : "Invalid");
00121     printf(" Actual: %s\n", actual1 ? "Valid" : "Invalid");
00122     printf(" Success: %s\n", (expected1 == actual1) ? "Yes" : "No");
00123
00124     char *blood2 = "AB-";
00125     int expected2 = 1, actual2 = utils_is_valid_blood_group(blood2);
00126     printf(" Input: %s\n", blood2);
00127     printf(" Expected: %s\n", expected2 ? "Valid" : "Invalid");
00128     printf(" Actual: %s\n", actual2 ? "Valid" : "Invalid");
00129     printf(" Success: %s\n", (expected2 == actual2) ? "Yes" : "No");
00130
00131     char *blood3 = "X+";
00132     int expected3 = 0, actual3 = utils_is_valid_blood_group(blood3);
00133     printf(" Input: %s\n", blood3);
00134     printf(" Expected: %s\n", expected3 ? "Valid" : "Invalid");
00135     printf(" Actual: %s\n", actual3 ? "Valid" : "Invalid");
00136     printf(" Success: %s\n", (expected3 == actual3) ? "Yes" : "No");
00137 }
00138
00139 void test_utils_is_valid_address() {
00140     printf("Testing utils_is_valid_address():\n\n");
00141
00142     char *addr1 = "123 Main Street, Dhaka";
00143     int expected1 = 1, actual1 = utils_is_valid_address(addr1);
00144     printf(" Input: %s\n", addr1);
00145     printf(" Expected: %s\n", expected1 ? "Valid" : "Invalid");
00146     printf(" Actual: %s\n", actual1 ? "Valid" : "Invalid");
00147     printf(" Success: %s\n", (expected1 == actual1) ? "Yes" : "No");
00148
00149     char *addr2 = "Apt. 5, Block B";
00150     int expected2 = 1, actual2 = utils_is_valid_address(addr2);
00151     printf(" Input: %s\n", addr2);
00152     printf(" Expected: %s\n", expected2 ? "Valid" : "Invalid");
00153     printf(" Actual: %s\n", actual2 ? "Valid" : "Invalid");
00154     printf(" Success: %s\n", (expected2 == actual2) ? "Yes" : "No");
00155
00156     char *addr3 = "Invalid@Address#123";
00157     int expected3 = 0, actual3 = utils_is_valid_address(addr3);
00158     printf(" Input: %s\n", addr3);
00159     printf(" Expected: %s\n", expected3 ? "Valid" : "Invalid");
00160     printf(" Actual: %s\n", actual3 ? "Valid" : "Invalid");
00161     printf(" Success: %s\n", (expected3 == actual3) ? "Yes" : "No");
00162 }
00163
00164 int main() {
00165     printf("=== UTILITY FUNCTIONS TEST ===\n\n");
00166
00167     // test_utils_get_int();
00168     // test_utils_get_char();
00169     // test_utils_get_float();
00170     // test_utils_get_double();

```

```
00171     // test_utils_get_string();
00172     test_utils_is_valid_phone();
00173     test_utils_is_valid_email();
00174     test_utils_str_to_upper();
00175     test_utils_is_valid_blood_group();
00176     test_utils_is_valid_address();
00177
00178     return 0;
00179 }
```