Appointment Booking Systems Component Document

*COCUS PT*

*July 2021*

Contents

[1. Introduction 3](#_Toc77192661)

[1.1 Audience 3](#_Toc77192662)

[1.2 What this guide covers 3](#_Toc77192663)

[1.3 Prerequisites 3](#_Toc77192664)

[1.4 Glossary 3](#_Toc77192665)

[2. Overview 4](#_Toc77192666)

[3. cocus-appointment-booking-api-server details 5](#_Toc77192667)

[3.1 config/default.json 5](#_Toc77192668)

[3.2 Mongo Database schema model for Appointment 5](#_Toc77192669)

[3.3 Exposed REST APIs 5](#_Toc77192670)

[3.3.1 Authentication API: 6](#_Toc77192671)

[3.3.2 Generate Password API: 6](#_Toc77192672)

[3.3.3 Appointment List API: 7](#_Toc77192673)

[3.3.4 Get Appointment by Id API: 7](#_Toc77192674)

[3.3.5 Create Appointment API: 7](#_Toc77192675)

[3.3.6 Patch/Update Appointment API: 8](#_Toc77192676)

[3.3.7 Cancel Appointment API: 9](#_Toc77192677)

[3.4 Email Service 10](#_Toc77192678)

[4. cocus-appointment-booking-client details 11](#_Toc77192679)

[4.1 Sign-in (home) Template 11](#_Toc77192680)

[4.2 List Appointment Template 12](#_Toc77192681)

[4.3 Create/Reschedule Appointment Template 14](#_Toc77192682)

[4.4 Calendar Template 15](#_Toc77192683)

[5. Logs 17](#_Toc77192684)

# Introduction

The purpose of this document is to document the “Appointment Management System” implementation to COCUS PT.

## Audience

This guide is intended to following audiences:

* Any end user, who can read and get familiar for system
* Any developer, who wish to consume data using APIs

## What this guide covers

This guide covers the following main topics:

* System’s UI design and implementation details
* System’s API related requirements and implementation details

## Prerequisites

Working knowledge of Typescript, NodeJS, Angular, HTML, CSS, Bootstrap and Postman

## Glossary

The following table defines the terms used in this document, including abbreviations.

|  |  |
| --- | --- |
| Term | Description |
|  |  |

# Overview

As part of system’s design, it provides ability for implementations in following two main areas:

* cocus-appointment-booking-api-server
* cocus-appointment-booking-client

# cocus-appointment-booking-api-server details

## config/default.json

It is a server level configuration file to configure following details:

* + - serverConfig

To change server specific settings as:

* port: to change application port
* host: to change application host
  + - userCredentials

To change authentication level settings as:

* username: to change username for server
* password: it is an encrypted password, use API ‘*/api/generatePassword*’ to create encrypted password
  + - databaseConnection

To change database connection settings as:

* dbUri: to provide remote/host database connection URL
  + - tokenConfiguration

To change JWT token related settings as:

* saltWorkFactor: to encrypt password
* accessTokenTtl: to provide time validity for token to expire
  + - emailConnections

To change email notification related settings as:

* domain: to change domain of the email server
* apiKey: to change API key from domain server
* from: to change from clause in email

## Mongo Database schema model for Appointment

Database Name: appointment-booking Collection Name: appointments

To modify schema please change below file:

*“cocus-appointment-booking\cocus-appointment-booking-api-server\src\model\* *Appointment.ts”*

|  |  |
| --- | --- |
| appointments | |
| \_id  subject  description participants  appointmentDate  appointmentPlace | ObjectId  String  String  String  Date (unique index)  String |

## Exposed REST APIs

Following are the exposed RESTful API from node server:

### Authentication API:

* Path: “/api/authenticate”
* Method: POST
* Request Body: {

"username": "\*\*\*",

"password": "\*\*\*"

}

* Desired Output: {

"ok": true,

"token": "\*\*\*"

}

* Error Response: {

"statusCode": 400,

"statusDescription": "Bad Request Exception",

"errorCode": 400,

"errorMessage": "Invalid password"

}

* Description:

This API is responsible for authentication user with the help of username and password and will provide token for the application use.

It will make use of ‘jsonwebtoken’ package for creating token.

### Generate Password API:

* Path: “/api/generatePassword”
* Method: POST
* Request Body: {

"password": "123"

}

* Desired Output: {

"ok": true,

"originalPassword": "\*\*\*",

"encryptedPassword": "\*\*\*"

}

* Error Response: {

"statusCode": 400,

"statusDescription": "Bad Request Exception",

"errorCode": 400,

"errorMessage": "Missing property 'password' in request body"

}

* Description:

This API is responsible for creating encrypted password using bcrypt package.

### Appointment List API:

* Path: “/api/appointment”
* Method: GET
* Desired Output: [

{

"\_id": "60ec3a4db99eea460cb1b3b8",

"subject": "Some subject",

"description": "Some description",

"participants": "list of email ids separated with ‘;’",

"appointmentDate": "some date",

"appointmentPlace": "some place",

"\_\_v": 0

}

]

* Description:

This API is responsible for getting all appointments created in backdate from Mongo DB collection.

It also supports query parameter ‘*date’* in case requesting date specific appointment.

### Get Appointment by Id API:

* Path: “/api/appointment/:id”
* Method: GET
* Desired Output: {

"\_id": "60ec3a4db99eea460cb1b3b8",

"subject": "Some subject",

"description": "Some description",

"participants": "list of email ids separated with ‘;’",

"appointmentDate": "some date",

"appointmentPlace": "some place",

"\_\_v": 0

}

* Description:

This API is responsible for getting specific appointment by id created in backdate from Mongo DB collection.

### Create Appointment API:

* Path: “/api/appointment/”
* Method: POST
* Request Body: {

"subject": "Some subject",

"description": " Some description",

"participants": "Some email ids",

"appointmentDate": "Some date",

"appointmentPlace": "Some place"

}

* Desired Output: {

"\_id": "60ec3a4db99eea460cb1b3b8",

"subject": "Some subject",

"description": "Some description",

"participants": "list of email ids separated with ‘;’",

"appointmentDate": "some date",

"appointmentPlace": "some place",

"\_\_v": 0

}

* Error Response: {

"statusCode": 500,

"statusDescription": "Internal Server Error",

"errorCode": 500,

"errorMessage": "Error creating appointment. E11000 duplicate key error collection: appointment-booking.appointments index: appointmentDate\_1 dup key: { appointmentDate: new Date(1626366900000) }"

}

* Description:

This API is responsible for creating appointment.

### Patch/Update Appointment API:

* Path: “/api/appointment/:id”
* Method: PATCH
* Request Body: {

"subject": "Some subject",

"description": " Some description",

"participants": "Some email ids",

"appointmentDate": "Some date",

"appointmentPlace": "Some place"

}

* Desired Output: {

"\_id": "60ec3a4db99eea460cb1b3b8",

"subject": "Some subject",

"description": "Some description",

"participants": "list of email ids separated with ‘;’",

"appointmentDate": "some date",

"appointmentPlace": "some place",

"\_\_v": 0

}

* Error Response: {

"statusCode": 500,

"statusDescription": "Internal Server Error",

"errorCode": 500,

"errorMessage": "Error creating appointment. E11000 duplicate key error collection: appointment-booking.appointments index: appointmentDate\_1 dup key: { appointmentDate: new Date(1626366900000) }"

}

* Description:

This API is responsible for patch/update appointment.

### Cancel Appointment API:

* Path: “/api/appointment/:id”
* Method: DELETE
* Desired Output: {

"\_id": "60ec3a4db99eea460cb1b3b8",

"subject": "Some subject",

"description": "Some description",

"participants": "list of email ids separated with ‘;’",

"appointmentDate": "some date",

"appointmentPlace": "some place",

"\_\_v": 0

}

* Error Response: {

"statusCode": 500,

"statusDescription": "Internal Server Error",

"errorCode": 500,

"errorMessage": "Error creating appointment. E11000 duplicate key error collection: appointment-booking.appointments index: appointmentDate\_1 dup key: { appointmentDate: new Date(1626366900000) }"

}

* Description:

This API is responsible for cancel/delete appointment.

## Email Service

We have used “mailgun-js” package for sending email notification to participants.

After every create appointment participant will receive email notification.

Note: Currently we’re using sandbox environment. So before receiving email, we need to register it on mailgun portal.

# cocus-appointment-booking-client details

This section will provide all UI related details.

## Sign-in (home) Template

This template is used to make sign-in or login to appointment booking system.

User needs to provide username and password, which is mandatory.

Graphical user interface, application

Description automatically generated

A screenshot of a computer

Description automatically generated

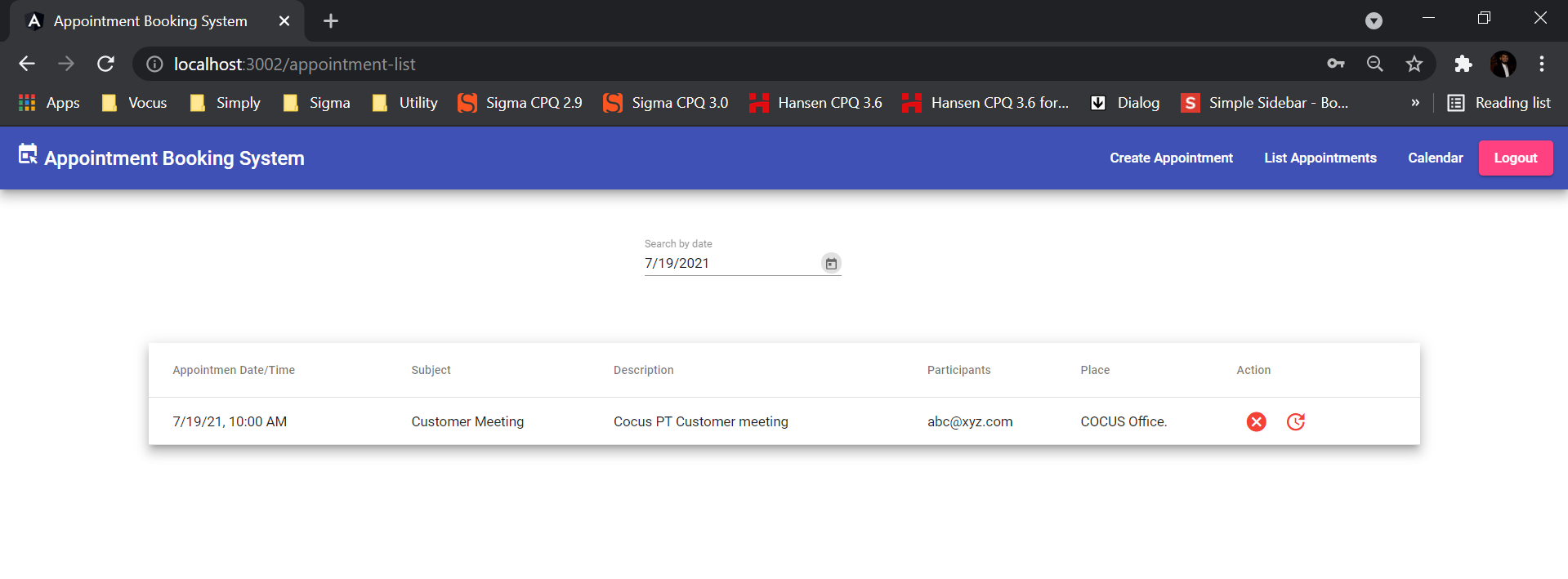
## List Appointment Template

This template is used to list all appointment booked in past date.

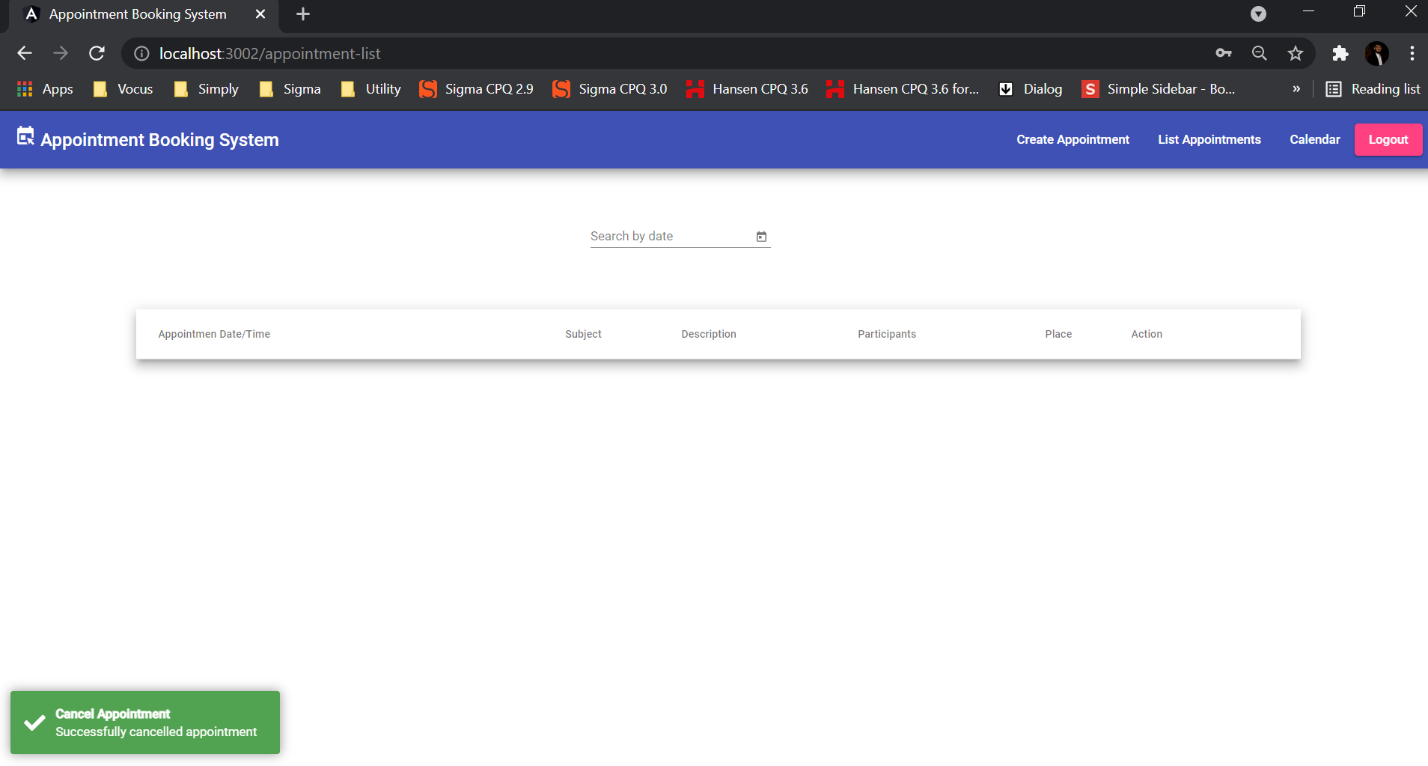
A screenshot of a computer

Description automatically generated

User can also view date specific appointments by selecting date.



User can also cancel/delete appointment by clicking on cancel button provided on each appointment row.



User can also reschedule/update appointment by clicking on reschedule button provided on each appointment row.

## Create/Reschedule Appointment Template

This template is used to create or reschedule appointment booked in past date.

User can create appointment using below template.

Graphical user interface, application

Description automatically generated

User can click on Reschedule appointment on list appointment template and it will redirect to this template.

Graphical user interface, application

Description automatically generated

## Calendar Template

This template is useful to render all appointments booked in past using a calendar.

We can watch appointment on daily, monthly, weekly basis.

Monthly

A screenshot of a computer

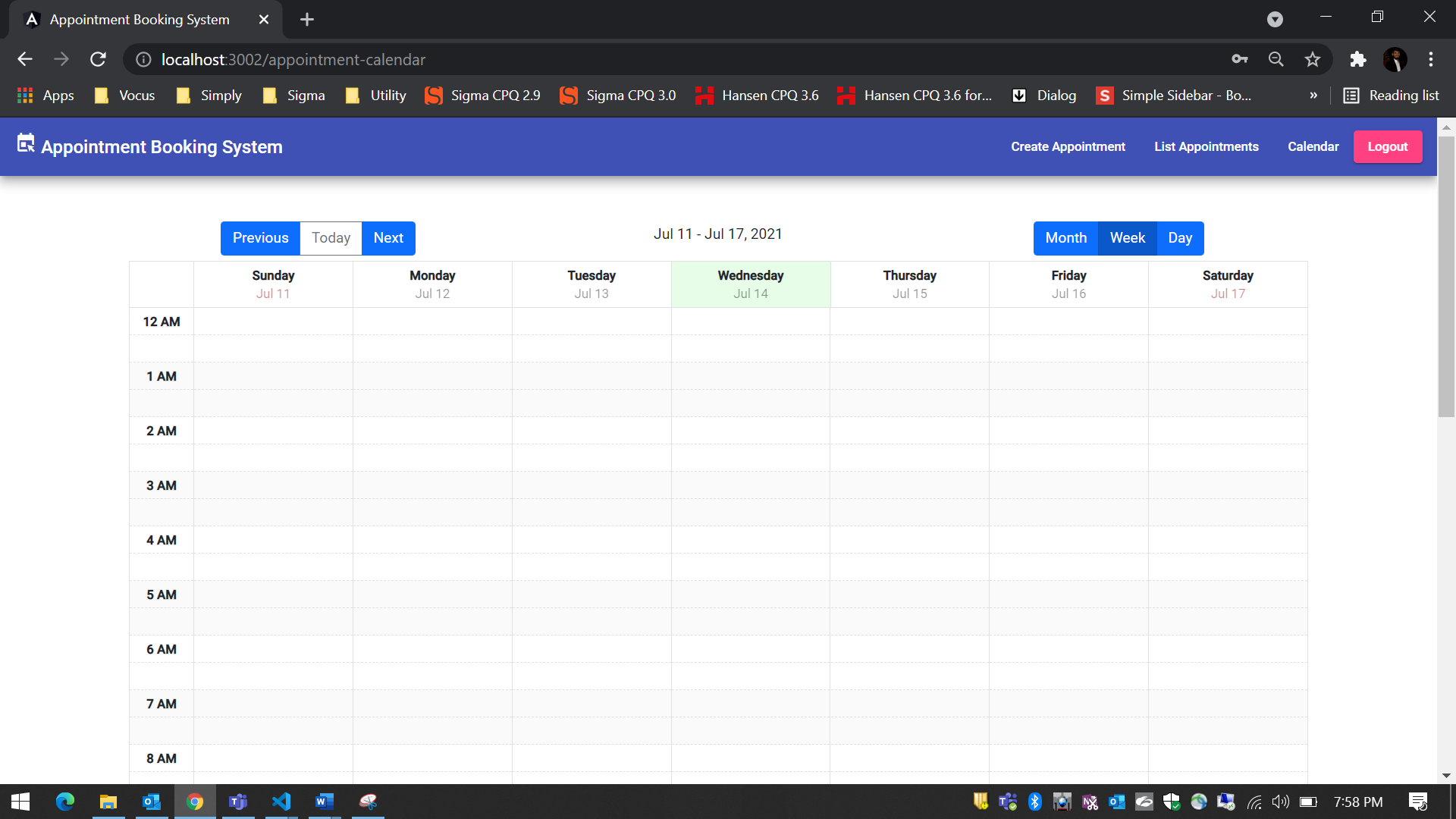
Description automatically generated

Daily

A picture containing text, screenshot, monitor

Description automatically generated

Weekly



# Logs

Server application has logging mechanism created using “dayjs” and “pino”.

Logs are created in server root directory.