**VISOKA TEHNIČKA ŠKOLA**

**STRUKOVNIH STUDIJA**

**S** **U B O T I C A**

ZUBARSKA ORDINACIJA

**PROJEKAT**

**iz predmeta Web programiranje**

|  |  |
| --- | --- |
| **KANDIDAT(I)**  **Miloš Popović, 26123015**  **Sara Klipa, 26123014** | **MENTOR**  **prof. dr Zlatko Čović** |
| **SUBOTICA, 2025.** | |

Sadržaj

[1. Opis zadatka 4](#_Toc203401396)

[2. Realizacija zadatka 5](#_Toc203401397)

[Backend 5](#_Toc203401398)

[FRONTEND 15](#_Toc203401399)

[3. Struktura baze podataka (MySQL Workbench) 22](#_Toc203401400)

[4. Opis funkcionalnosti 23](#_Toc203401401)

[5. Korišćena literatura 26](#_Toc203401402)

# Opis zadatka

Cilj ovog projekta je razvoj web aplikacije za zakazivanje termina u zubarskoj ordinaciji, koja omogućava efikasnu komunikaciju između pacijenata, stomatologa i administratora. Sistem definiše četiri nivoa korisničkog pristupa: gost, registrovani korisnik, zubar i administrator, pri čemu svaki nivo ima svoje specifične funkcionalnosti i prava.

Gost može da pregleda osnovne informacije o ordinaciji, listu dostupnih zubara sa njihovim uslugama i da se registruje na sajt. Registrovani korisnik ima mogućnost da pregleda zubare, rezerviše termine, menja lične podatke, upravlja svojim rezervacijama, kao i da pristupa elektronskom kartonu tretmana. Takođe može otkazati rezervacije uz uslov da je otkaz najmanje četiri sata pre zakazanog termina, kao i da inicira zahtev za promenu lozinke.

Zubar ima pristup unosu i izmeni informacija o tretmanima korisnika, upravljanju sopstvenim radnim terminima, kao i pristupu elektronskom kartonu pacijenata. Administrator upravlja podacima o zubarima, uslugama i korisnicima, uključujući odobravanje ili zabranu pristupa sistemu.

Sistem obezbeđuje sigurnu registraciju korisnika sa verifikacijom putem aktivacionog linka poslatog na email, kao i sigurnu promenu lozinke. Svi termini su vremenski kontrolisani, uz proveru preklapanja i zauzetosti, a korisnicima se nakon rezervacije šalje potvrda sa jedinstvenim kodom za termin.

Posebna pažnja posvećena je sigurnosti podataka, gde su lozinke zaštićene bcrypt heširanjem, a administrativni deo sistema je zaštićen sesijama. Aplikacija koristi moderne web tehnologije (HTML, CSS, JavaScript, AJAX/Fetch API, PHP, MySQL), i pruža responzivan dizajn prilagođen različitim uređajima.

U sklopu projekta kreirana je i odgovarajuća baza podataka koja podržava sve navedene funkcionalnosti, dok su funkcionalnosti povezane sa slanjem emailova realizovane korišćenjem PHPMailer klase.

**Web adresa projekta**: <https://eclipse.stud.vts.su.ac.rs/index.html>

# Realizacija zadatka

## BACKEND

Backend se sastoji od malog MVC framework-a, koji funkcioniše slično kao osnovne komponente Symfony framework-a za PHP. Ulazna tačka za backend je **index.php** fajl, koji učitava sve environment promenljive pomoću **Dotenv** biblioteke, inicijalizuje Router klasu i poziva njenu resolve() metodu, koja obrađuje request i vraća odgovarajući response.

**Komponente backend-a:**

***1. Router klasa***

Router klasa skladišti sve rute na koje API odgovara unutar njenog $routes niza. $routes je 2D niz, čiji su ključevi sve request metode za koje postoje registrovane API rute (GET, POST, DELETE). Unutar svake metode su podnizovi koji sadrže API rutu i kontroler metodu koja se poziva za tu rutu. Na svaki request, index.php poziva **$router->resolve()** metodu, koja registruje sve kontroler funkcije za rute i middleware funkcije definisane za njih. Sve ove funkcije su u odgovarajućim klasama obeležene **PHP atributima**, i ruter ih očitava pomoću **Reflection API**-ja.

Ruter takođe obrađuje request URL i vrši obradu dinamičkih parametara - prepoznaje da /api/dentists/1 odgovara ruti /api/dentists/{id} i stavlja vrednost id parametra rute u $params svojstvo Request objekta, kako bi kasnije u kontrolerima bio omogućen pristup URL parametrima. Kada je URL obrađen, middleware funkcije pozvane i pronalazi se kontroler funkcija registrovana za zatraženu rutu i poziva se pomoću **call\_user\_func\_array()**.

Najvažniji delovi Router klase:

*private* function registerRoute(string $route, string $method, array $action): void

{

$this->routes[$method][$route] = $action;

}

*public* function registerRoutes(array $controllers): void

{

foreach ($controllers as $controller) {

$reflectionController = new \ReflectionClass($controller);

foreach ($reflectionController->getMethods() as $method) {

$attributes = $method->getAttributes(Route::class);

foreach ($attributes as $attribute) {

$route = $attribute->newInstance();

$this->registerRoute($route->path, $route->method, [$controller, $method->getName()]);

}

}

}

}

*public* function resolve(): void

{

try {

$method = $this->req->getMethod();

$path = $this->req->getPath();

$params = $this->resolveParams($method, $path);

$this->req->params = $params[1];

*// if the route has a parameter, like {id}, $path doesn't contain the placeholder, but the value*

*// in order for the controller to be fetched from the $routes array, $params[0] has to be*

*// used, because it contains the parameter placeholder*

[$class, $controller] = $this->routes[$method][$params[0]];

$res = null;

if (class\_exists($class) && method\_exists($class, $controller)) {

$class = new $class();

*// if there's middleware registered to a function with this name, call the mw stack*

if (isset($this->middleware[$controller])) {

foreach ($this->middleware[$controller] as $middleware) {

[$mwClass, $mwMethod] = $middleware['function'];

if (class\_exists($mwClass) && method\_exists($mwClass, $mwMethod)) {

$mwClass = new $mwClass();

call\_user\_func\_array([$mwClass, $mwMethod], ['req' => $this->req, 'args' => $middleware['args']]);

}

}

}

$res = call\_user\_func\_array([$class, $controller], ['req' => $this>req]);

}

}

catch (\PDOException $e) {

$res = new JSONResponse([

'status' => 'fail',

'message' => 'something went wrong when connecting to the database!',

'details' => $e->getMessage()

], 500);

}

catch (APIException $e) {

if ($e->extraInfo) {

$res = new JSONResponse([

'status' => 'fail',

'message' => $e->getMessage(),

'errors' => $e->extraInfo

], $e->statusCode);

}

else {

$res = new JSONResponse([

'status' => 'fail',

'message' => $e->getMessage(),

], $e->statusCode);

}

}

catch (\Throwable $t) {

$res = new JSONResponse([

'status' => 'fail',

'message' => $t->getMessage(),

], 500);

}

finally {

if (null === $res) {

$res = new JSONResponse([

'status' => 'fail',

'message' => 'route not defined!',

], 404);

}

echo $res->send();

}

***2. Controller klase***

Svaki resurs aplikacije (Dentist, Appointment, Service, User) ima svoju Controller i Model klasu. Kontroler klase su odgovorne za obrađivanje request-ova i vraćanje odgovarajućeg response-a. One primaju request i vraćaju response u JSON formatu. Program zna da je neka funkcija u klasi kontroler funkcija zbog custom PHP atributa - Route, koji ruter čita i na osnovu toga dodaje tu funkciju u svoj $routes niz. Sve kontroler klase nasleđuju **BaseController** apstraktnu klasu, koja ima definisan json() metod, koji se koristi za vraćanje JSON response-a. Svaka kontroler klasa prima jedan argument tipa klase **Request**, koja opisuje dolazeći HTTP request i definiše utility metode koje kontrolerima omogućavaju da pročitaju dolazeće podatke u slučaju POST requestova, ili da pristupe URL parametrima. Za komunikaciju sa bazom, kontroleri koriste model klase.

Primer jedne od kontroler funkcija:

#[Route(path: '/api/users', method: 'get')]

#[Middleware(function: [AuthMiddleware::class, 'authenticate'])]

#[Middleware(function: [AuthMiddleware::class, 'authorize'], args: ['dentist', 'admin'])]

*public* function getAllUsers(Request $req): JSONResponse

{

$model = new UserModel();

$users = $model->getAllUsers();

if (!$users) {

throw new APIException('No users found!', 404);

}

return $this->json([

'status' => 'success',

'data' => [

'users' => $users

]

]);

}

***3. Middleware klase***

Middleware klase definišu metode koje se izvršavaju pre glavne kontroler funkcije. Postavljaju se kao atributi na one kontroler funkcije pre kojih želimo da izvršimo middleware. Unutar projekta su prisutne samo dve middleware funkcije koje se koriste za autentifikaciju i autorizaciju korisnika. Funkcija za autentifikaciju proverava da li je korisnik koji pravi re quest ulogovan, a funkcija za autorizaciju proverava da li korisnik ima dozvole da pristupi datoj ruti. Ako korisnik ne ispunjava ove uslove, baca se custom Exception podklasa - **APIException** i kontroler funkcija se ne izvršava.

Primer middleware klase:

class AuthMiddleware

{

*public* function authenticate(Request $req, array $args): void

{

$user = SessionManager::get('user');

if (!$user) {

throw new APIException('You need to be logged in to access this route!', 403);

}

$req->user = $user;

}

*public* function authorize(Request $req, array $args): void

{

if (!in\_array($req->user['role'], $args)) {

throw new APIException('You do not have permission to access this route!', 403);

}

}

}

***4. Model klase***

Svaki resurs u aplikaciji takođe ima i sopstvenu model klasu koja je odgovorna za komunikaciju sa bazom. Model klase sadrže metode za sve operacije sa podacima. Pojedine metode za čitanje podataka spajaju podatke iz više tabela i takođe ih formatiraju tako da response API-ja bude u formatu pogodnom za kasniji rad. Ako se desi da u response-u ima podataka iz više tabela, biće formatirani tako da su podaci iz sporednih tabela poseban niz u rezultatu funkcije. Na taj način, kada se iskoristi json\_encode(), ti nizovi će postati JSON, i kasnije JavaScript objekti za lakši rad na frontendu. Model klase konekciju sa bazom dobijaju korišćenjem **Db klase**, koja je zadužena za uspostavljanje te konekcije. U toku rada backenda, otvara se samo jedna konekcija sa bazom, zato što Db klasa koristi **singleton pattern** - ima jedan statički metod, **Db::getConnection()**, koji vraća PDO database handler objekat koji je kreiran prvi put kada se instancirala Db klasa.

Primer metode iz jedne od model klasa:

*public* function getActiveAppointmentsForUser(int $userId): array

{

$dbh = Db::getConnection();

$query = "SELECT a.id AS appointment\_id,a.user\_id, a.dentist\_id, a.scheduled\_at, a.price, a.duration, ac.code, d.id AS dentist\_id, d.first\_name, d.last\_name, d.email, d.photo

FROM appointment a

INNER JOIN appointment\_codes ac ON a.id = ac.appointment\_id

INNER JOIN dentist d ON a.dentist\_id = d.id

WHERE a.user\_id = :userId AND a.scheduled\_at > NOW()";

$stmt = $dbh->prepare($query);

$stmt->bindValue(':userId', $userId);

$stmt->execute();

$res = $stmt->fetchAll(PDO::FETCH\_ASSOC);

$activeAppointments = [];

foreach ($res as $appointment) {

$servicesQuery = "SELECT \* FROM appointment\_service app\_sr INNER JOIN service s ON app\_sr.service\_id = s.id WHERE app\_sr.appointment\_id = :appointmentId";

$servicesStmt = $dbh->prepare($servicesQuery);

$servicesStmt->bindValue(':appointmentId', $appointment['appointment\_id']);

$servicesStmt->execute();

$services = $servicesStmt->fetchAll(PDO::FETCH\_ASSOC);

$activeAppointments[] = [

'id' => $appointment['appointment\_id'],

'user\_id' => $appointment['user\_id'],

'scheduled\_at' => $appointment['scheduled\_at'],

'price' => $appointment['price'],

'duration' => $appointment['duration'],

'code' => $appointment['code'],

'dentist' => [

'id' => $appointment['dentist\_id'],

'first\_name' => $appointment['first\_name'],

'last\_name' => $appointment['last\_name'],

'email' => $appointment['email'],

'photo' => $appointment['photo']

],

'services' => $services

];

}

return $activeAppointments;

}

**5. Servis klase**

Servise klase služe da obavljaju određeni posao koji je potreban na više mesta u više kontrolera, ili samo da doprinesu tome da svaki kontroler radi isključivo upravljanje resursom za koji je određen, umesto da radi i npr. slanje imejlova pored toga. Na ovaj način postiže se **separation of concerns**. Servis klase koje se nalaze u projektu su:

* **Mailer** - zadužen za slanje mejlova i konfigurisanje transporta
* **SessionManager -** zadužen za upisivanje, čitanje i brisanje podataka iz $\_SESSION promenljive
* **TokenGenerator -** zadužen za generisanje tokena pomoću bin2hex(random\_bytes()) funkcija. Ova funkcionalnost je izvučena u posebnu klasu zato što je generisanje tokena potrebno na više mesta - password reset token, account activation token i token koji korisnik dobija kada zakaže termin kod zubara. Pomoću parametra funkciji je omogućeno generisanje tokena drugačije dužine kako bi oni tokeni koji se prikaziji korisniku, kao token za pregled, izgledali donekle čitljivije.
* **Validator -** zadužen za validaciju podataka koji se šalju kada korisnik pošalje request za kreiranje naloga. Ova funkcionalnost je izvučena u posebnu klasu takođe kako bi svaki deo sistema radio samo svoju glavnu ulogu i kako bi posao kontroler metoda ostao da obrađuju request i vrate response, a ne da i validiraju podatke.

Primer jedne od servis klasa:

class SessionManager

{

*public* *static* function set(string $key, mixed $value): void

{

if (session\_status() !== PHP\_SESSION\_ACTIVE) {

session\_start();

}

$\_SESSION[$key] = $value;

}

*public* *static* function get(string $key): mixed

{

if (session\_status() !== PHP\_SESSION\_ACTIVE) {

session\_start();

}

return $\_SESSION[$key] ?? null;

}

*public* *static* function remove(string $key): void

{

if (session\_status() !== PHP\_SESSION\_ACTIVE) {

session\_start();

}

unset($\_SESSION[$key]);

}

}

**Povezivanje frontenda i backenda**

Frontend i backend su povezani pomoću **.htaccess** fajla koji svaki request koji počinje sa /api prosleđuje na index.php, koji dalje pomoću Router klase određuje šta treba da radi sa requestom i vraća response, a sve ostale requestove prosleđuje na destinaciju koja je i zatražena.

.haccess fajl:

RewriteEngine On

RewriteBase /

RewriteRule ^api/(.\*)$ src/index.php [L]

RewriteCond %{REQUEST\_FILENAME} -f [OR]

RewriteCond %{REQUEST\_FILENAME} -d

RewriteRule ^ - [L]

## FRONTEND

Frontend deo aplikacije razvijen je korišćenjem HTML, CSS (Bootstrap 5), JavaScript (ES6), Fetch API-ja i FullCalendar biblioteke. Aplikacija je potpuno responzivna i prilagođena radu na desktop i mobilnim uređajima. Vizuelni izgled stranica je zasnovan na jednostavnom Bootstrap dizajnu koji je dodatno stilizovan prema potrebama projekta. Aplikacija se oslanja na asinhronu komunikaciju između korisničkog interfejsa i backend API-ja, koristeći Fetch za učitavanje i ažuriranje podataka kao što su zakazani termini, elektronski kartoni i korisničke napomene. Sav JavaScript kod organizovan je u posebne fajlove unutar direktorijuma assets/js/, po logičkim celinama i korisničkim ulogama. U nastavku su prikazani najvažniji segmenti JavaScript koda sa objašnjenjima.

*Home stranica*

Na pocetnoj stranici se izlistavaju svi zubari sa svojom fotografijom i informacijama. Taj deo odnosno kartice u kojima se oni prikazuju su realizane putem sledeceg JavaScript koda

fetch('/dentists/api/dentists')  
 .then(res => res.json())  
 .then(data => {  
 const container = *document*.getElementById('dentist-card');  
 const dentists = data.data.dentists;  
 dentists.forEach(d => {  
 const name = d.first\_name + ' ' + d.last\_name;  
 const col = *document*.createElement('div');  
 col.className = 'col-md-4 mb-4';  
 col.innerHTML = `  
 <div class="card h-100">  
 <img src="/dentists${d.photo}" class="card-img-top" alt="${name}">  
 <div class="card-body">  
 <h5 class="card-title">${name}</h5>  
 <p class="card-text"><strong>Specialization:</strong> ${d.specialization}</p>  
 <div class="mb-2">  
 ${(d.services || []).map(service => `<span class="badge rounded-pill bg-secondary">${service.name}</span>`).join('')}  
 </div>  
 <button class="btn"><a href="/dentists/public/makeAppointment.html?dentist=${d.id}" class="btn-link">Book an appointment</a></button>  
 </div>  
 </div>  
 `;  
 container.appendChild(col);  
 });  
 })  
 .catch(err => {  
 *console*.error('Error fetch:', err);  
 });

*Sign in and Sign up stranica*

// On sign-in button click: validate, send login POST request, handle response

signInBtn.addEventListener('click', async (e) => {

e.preventDefault()

const email = emailInput.value.trim()

const password = passwordInput.value.trim()

if (!validate(email, password)) return

try {

const res = await fetch('/dentists/api/login', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

credentials: 'include',

body: JSON.stringify({ email, password })

})

const json = await res.json()

if (json.status === 'success') {

const user = json.data.user

localStorage.setItem('userId', user.id)

localStorage.setItem('role', user.role)

if (user.role === 'dentist') {

localStorage.setItem('dentistId', user.id)

showModalSuccess('Login successful!')

setTimeout(() => window.location.href = '/dentists/public/dentistDashboard.html', 1000)

} else if (user.role === 'admin') {

localStorage.setItem('adminId', user.id)

showModalSuccess('Login successful!')

setTimeout(() => window.location.href = '/dentists/public/adminDashboard.html', 1000)

} else {

showModalSuccess('Login successful!')

setTimeout(() => window.location.href = '/dentists/index.html', 1000)

}

} else {

showModalFail(json.message || 'Invalid login.')

}

} catch (error) {

console.error('Login error:', error)

showModalFail('Something went wrong. Please try again.')

}

})

Na klik dugmeta validira unos, šalje login zahtev, čuva podatke o korisniku u localStorage i preusmerava korisnika na odgovarajuću stranicu prema ulozi. Prikazuje poruke uspeha ili greške.

*Dentist Panel – Calendar View and Notes + Patient Health Record*

//get all appointments for dentist

const getAppointments = async () => {

const res = await fetch(`/dentists/api/appointments/dentist/${dentistId}`);

const json = await res.json();

return json.data.appointments;

}

Ova funkcija šalje asinhroni GET zahtev backendu i dobavlja sve termine za prijavljenog zubara. Rezultat se parsira kao JSON i vraća se niz termina koje će prikazati FullCalendar.

//calender initialization

document.addEventListener('DOMContentLoaded', async () => {

const calendarEl = document.getElementById('calendar');

const calendar = new FullCalendar.Calendar(calendarEl, {

initialView: 'timeGridWeek',

editable: false,

locale: 'en',

events: await getAppointments(),

eventClick: handleAppointmentClick

});

calendar.render();

});

Nakon što se DOM učita, inicijalizuje se FullCalendar koji prikazuje termine u nedeljnom prikazu. Funkcija eventClick se poziva kada se klikne na zakazani termin i tada se otvara modal za unos napomene.

//insert treatment note

document.getElementById("save").addEventListener("click", async () => {

const code = document.getElementById("appointmentCode").value;

const note = document.getElementById("noteText").value;

const res = await fetch(`/dentists/api/appointments/${id}/note`,

{

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ note })

});

const data = await res.json();

alert(data.message);

location.reload();

});

Klikom na dugme "save", šalje se POST zahtev sa tekstom napomene koji se vezuje za određeni termin (po ID tretmana). Nakon unosa, korisniku se prikazuje poruka, a stranica se osvežava da bi se ažurirao prikaz termina.

//delete function

deleteBtn.addEventListener('click', async () => {  
 const id = appointmentIdInput.value;  
 if (!id || isNaN(id)) {  
 alert('Select a valid appointment first.');  
 return;  
 }  
  
 if (!confirm('Are you sure you want to delete this appointment?')) return;  
  
 try {  
 const res = await fetch(`/dentists/api/appointments/${id}/delete`, {  
 method: 'DELETE',  
 credentials: 'include'  
 });  
  
 if (!res.ok) throw new Error('Failed to delete appointment');  
  
 alert('Appointment deleted successfully!');  
 appointmentForm.reset();  
 calendar.refetchEvents();  
 } catch (err) {  
 alert(err.message);  
 }  
});

Ovaj ima event listener klik na dugme za brisanje termina i proverava da li je selektovan validan ID termina. Ako korisnik potvrdi brisanje, šalje se DELETE zahtev backendu; u slučaju uspeha prikazuje se poruka, resetuje forma i osvežava kalendar.

document.addEventListener('DOMContentLoaded', async () => {

const dentistId = localStorage.getItem('dentistId');

const container = document.getElementById('records-container');

try {

const res = await fetch(`/dentists/api/medicalRecords/dentist/${dentistId}/allPatients`);

if (!res.ok) throw new Error('Failed to fetch records from API');

const json = await res.json();

const appointments = json.data.appointments;

if (!appointments.length) {

container.innerHTML = '<p>No medical records found.</p>';

return;

}

const table = document.createElement('table');

table.className = 'table table-striped';

table.innerHTML = `

<thead>

<tr>

<th>Patient</th>

<th>Dentist</th>

<th>Services</th>

<th>Date & Time</th>

<th>Duration (min)</th>

<th>Note</th>

</tr>

</thead>

<tbody>

${appointments.map(app => `

<tr>

<td>${app.user\_first\_name} ${app.user\_last\_name}</td>

<td>${app.dentist\_first\_name} ${app.dentist\_last\_name}</td>

<td>${app.services?.length ? app.services.join(', ') : 'No services'}</td>

<td>${app.scheduled\_at}</td>

<td>${app.duration ?? ''}</td>

<td>${app.note?.trim() ? app.note : 'No note'}</td>

</tr>

`).join('')}

</tbody>

`;

container.appendChild(table);

} catch (err) {

console.error('Error loading records:', err);

container.innerHTML = '<p>Error loading medical records.</p>';

}

});

Ovaj deo koda se odnosi na allRecord.js u kome se nalazi funkcija za tabelarno prikazivanje svih elektronskih kartona pacijenata odredjenog doktora. JSON podaci iy baze se dobijaju putem dentists/api/medicalRecords/dentist/${dentistId}/allPatients API.

*Make Appointment page (Java Script only)*

// Redirect if user role is not 'user'

const role = localStorage.getItem('role')

if (!role || role !== 'user') {

window.location.href = '/dentists/public/signin.html'

}

Proverava da li je u localStorage sačuvana uloga korisnika i preusmerava na stranicu za prijavu ako nije validan korisnik.

const fetchAPI = async (route, method = 'GET', data = {}) => {

let res

if (method !== 'GET') {

res = await fetch(route, {

method,

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(data)

})

} else {

res = await fetch(route)

}

const json = await res.json()

return json

}

Funkcija za komunikaciju sa backend API-jem koja podržava GET i POST/DELETE pozive sa JSON podacima.

// Format duration from minutes to string like "1h 30min"

const formatDuration = (durationMins) => {

let res = ``

const hours = Math.floor(durationMins / 60)

if (hours !== 0) res += `${hours}h`

const minutes = durationMins % 60

if (minutes !== 0) res += ` ${minutes}min`

return res

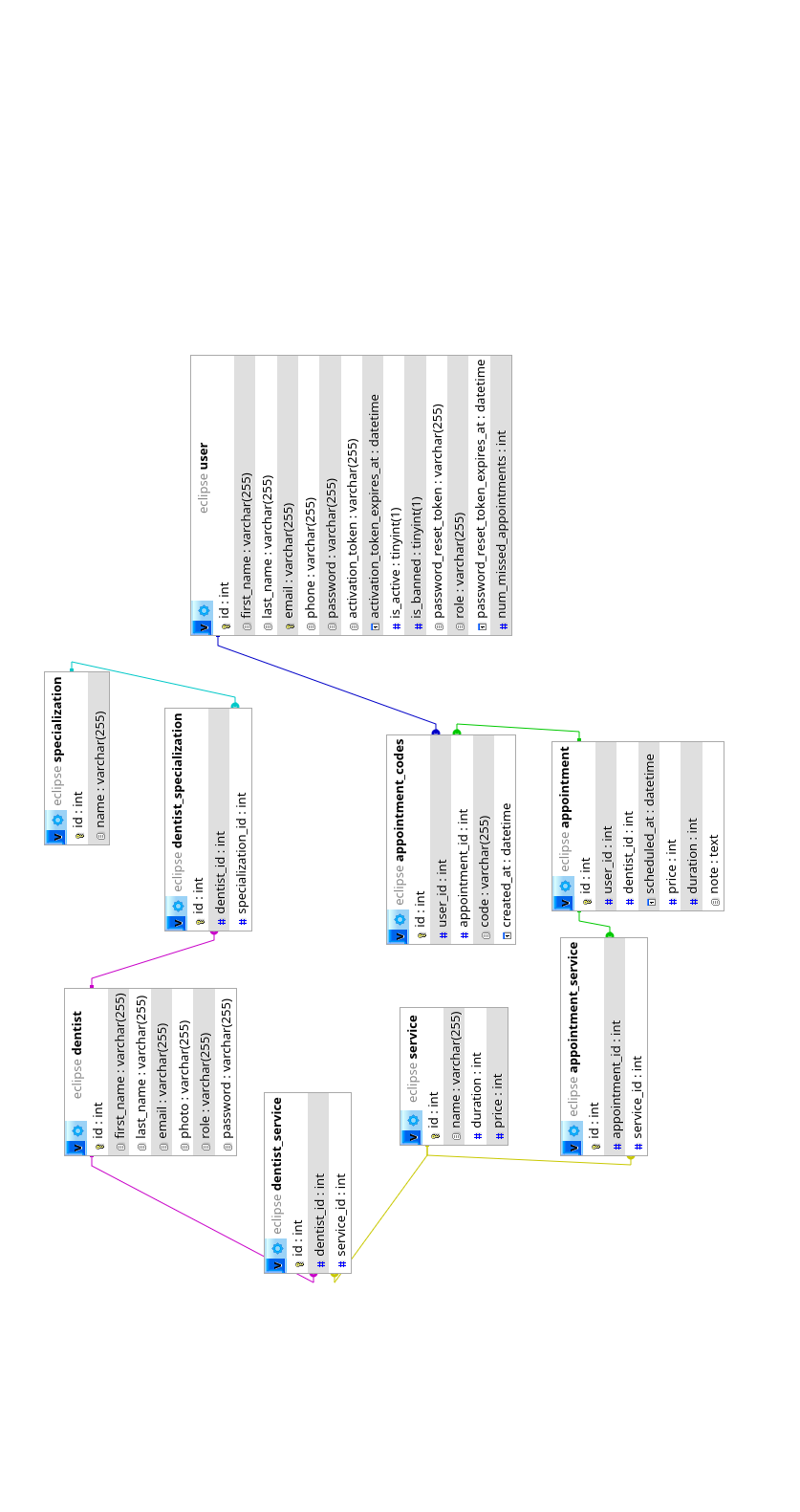
Konvertuje trajanje u minutima u čitljiv format sa satima i minutima za prikaz korisniku.

Ono što je bitno napomenuti jeste da se ovi JavaScript fajlovi “povezuju” sa html stranicama unutar id elemenata na html stranici. Na primer:

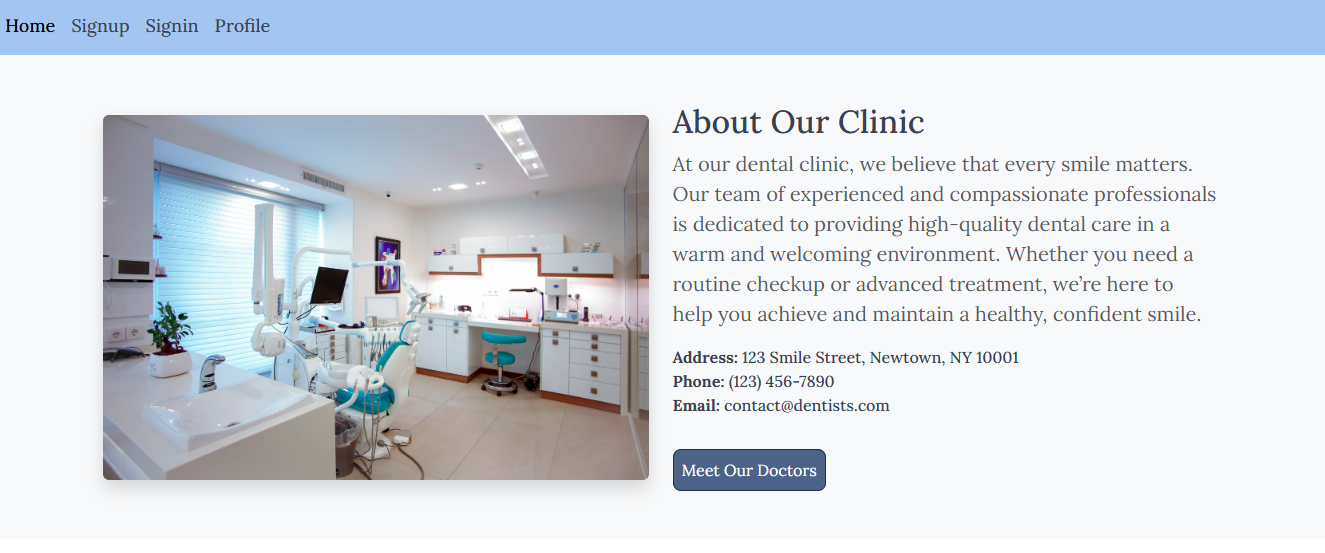
<section class="doctor-section" id="section2">  
 <header class="browse-header">  
 <h2>Browse dentists:</h2>  
 </header>  
 <div class="container mt-4">  
 <div class="row" id="dentist-card">  
  
 </div>  
 </div>  
</section>

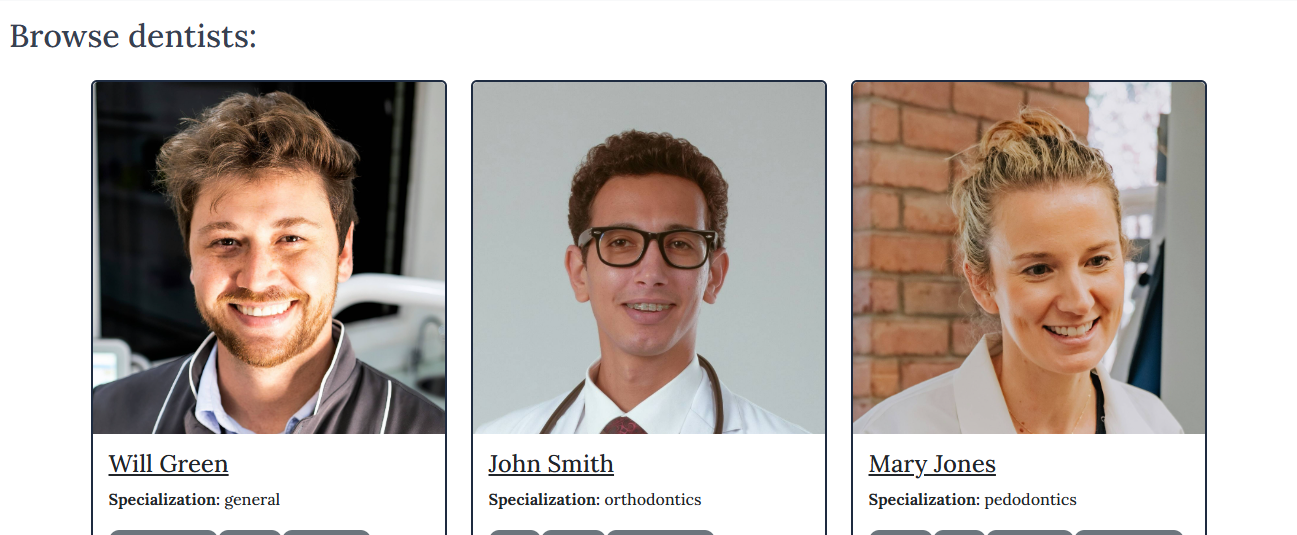
Preko id se pristupa dentist-card odosno tu se ispisuju dentist-card sa js

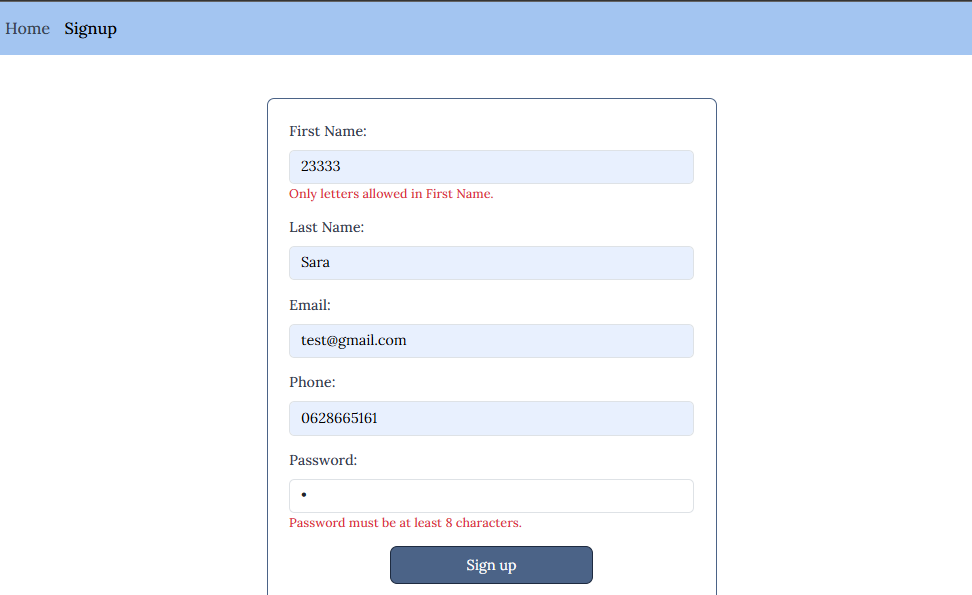
# Struktura baze podataka (MySQL Workbench)

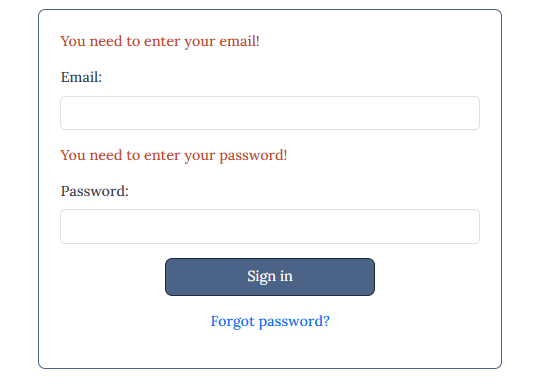
****

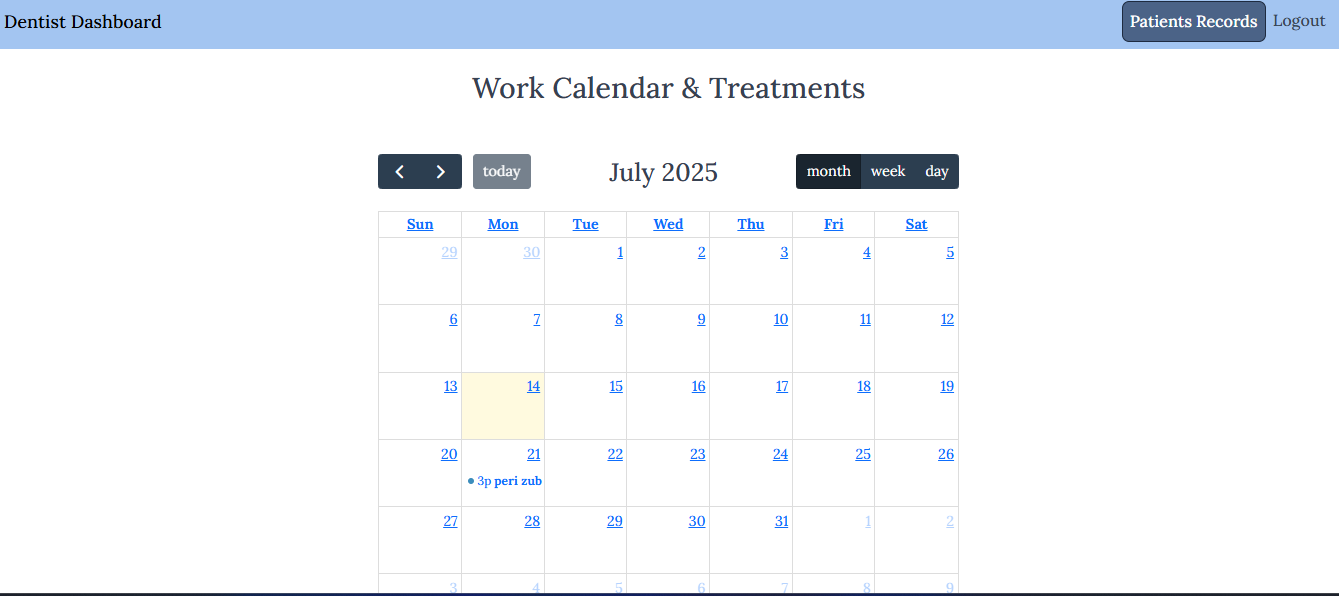
# Opis funkcionalnosti

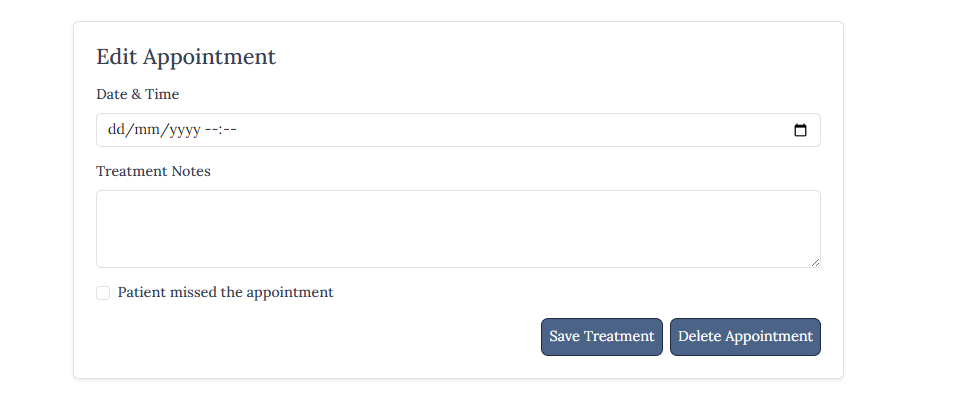




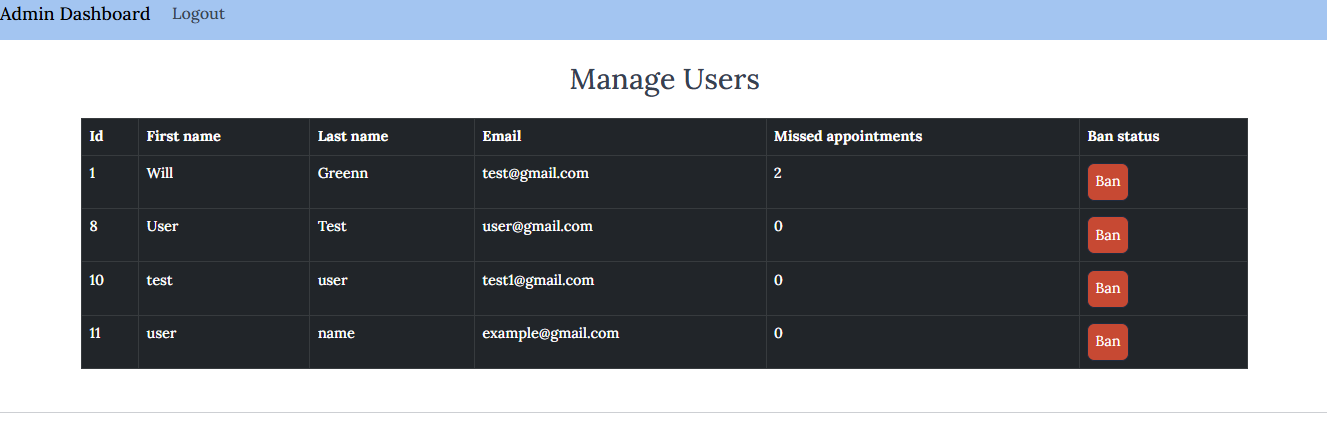
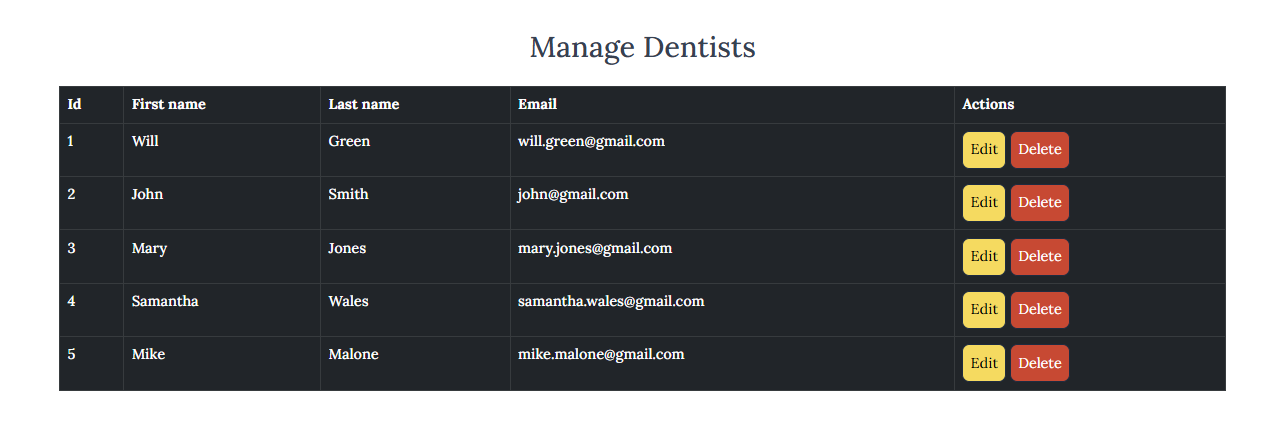
****

****

****

****

****

# Korišćena literatura

W1. <http://www.w3schools.com/>

W2. <http://www.jquery.com>

W3. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>

W4. <https://unsplash.com/>

W5. <https://www.php.net/>