ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Веб-застосунок для онлайн-запису клієнтів

Кваліфікаційна робота

Перший (бакалаврський рівень)

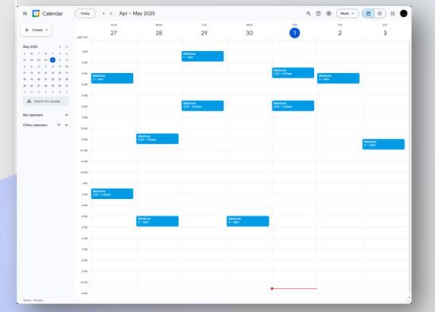
Автор:
Канцір Р.Б.,
ст. групи КІУКІ-21-6

Керівник:
Тимошенко Д.О.,
асист. каф. ЕОМ

АКТУАЛЬНІСТЬ РОБОТИ

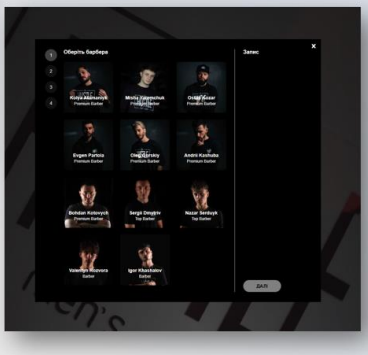
- зростання попиту на онлайн-сервіси у сфері послуг;
- потреба малого бізнесу у цифрових рішеннях;
- мінімізація часу взаємодії між клієнтом і сервісом;
- підвищення зручності для кінцевого користувача.

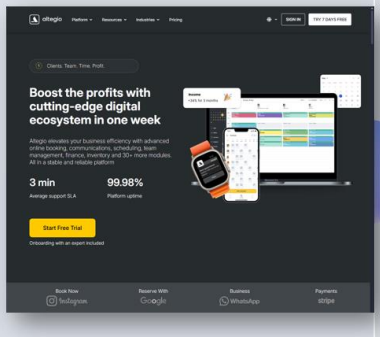
Недоліки існуючих рішень



👇Google Calendar

👇Кастомні рішення





👇Alteg.io

3

ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Рішення	Ручний запис	Google Calendar	Alteg.io	Кастомні рішення
Простота впровадження	+	+	-	-
Функціональність	-	-	+	+
Гнучкість налаштувань	-	-	+	+
Онлайн-запис без адміністратора	-	-	+	+
Підходить для малого бізнесу	+	+	-	-
Потребує технічних навичок	-	+	+	+
Функціональність	-	-	+	+
Велика вартість	-	-	+	+

4

ПОСТАНОВКА ЗАДАЧІ

- наявність чотирьох типів ролей користувачів: супер-адміністратор, адміністратор, майстер, клієнт;
- впровадження функціоналу модерації компаній для супер-адміністратора;
- забезпечення адміністратору можливості повного керування компанією, майстрами, послугами та перегляду аналітики;
- надання майстру доступу до персонального кабінету;
- реалізація процесу запису для клієнтів без потреби реєстрації;
- підтримка декількох способів запису: за майстром, за послугою або за часом;
- надсилання підтвердження запису на електронну пошту клієнта;
- впровадження можливості залишати відгуки після отримання послуги;

5

ВИКОРИСТАНІ ТЕХНОЛОГІЇ

NextJS

TypeScript

Tailwind CSS

Cursor

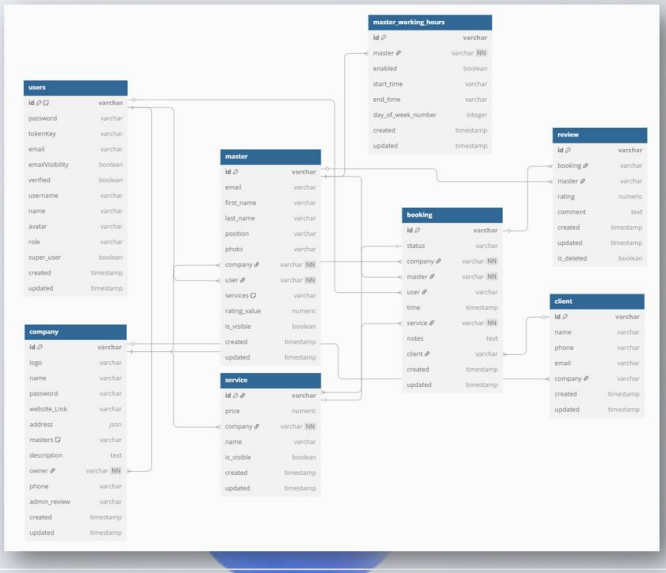
PocketBase

Vercel

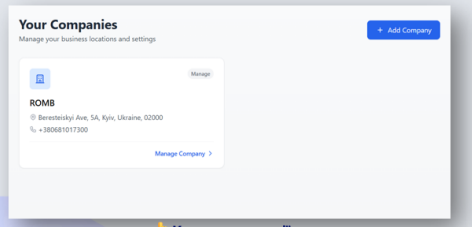
Pocketi

6

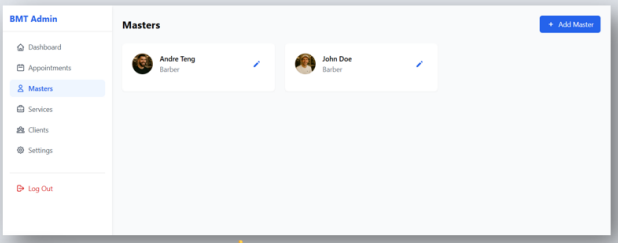
СТРУКТУРА БАЗИ ДАНИХ



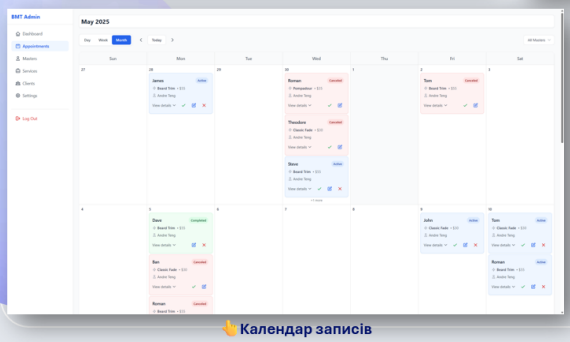
АДМІНІСТРАТОР



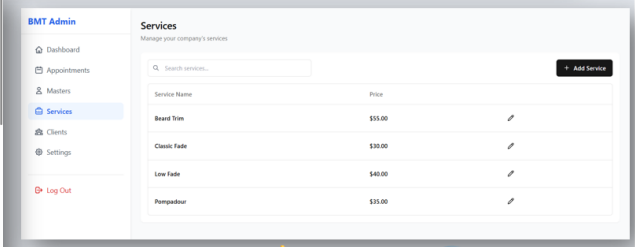
Картка компанії



Список майстрів

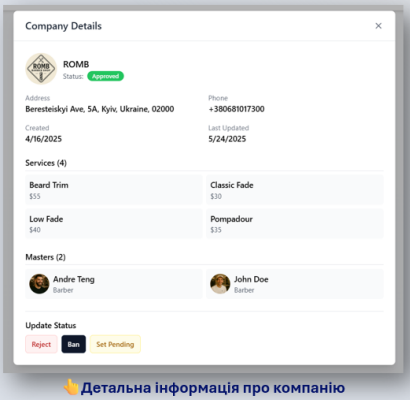


Календар записів

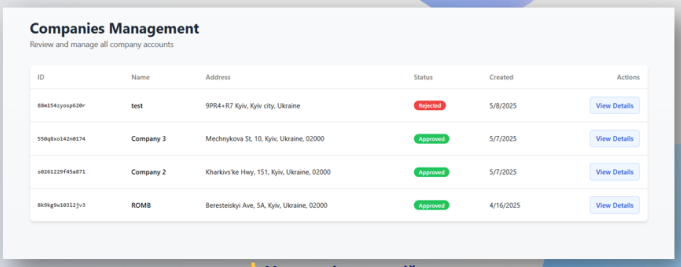


Список послуг

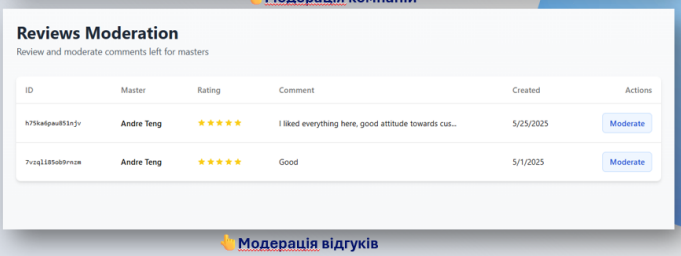
СУПЕР-АДМІНІСТРАТОР



👆Детальна інформація про компанію

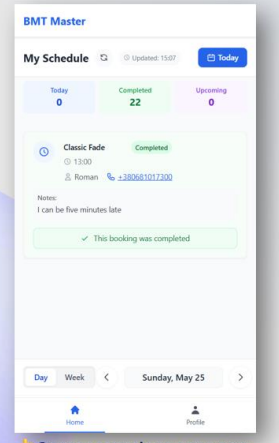


👆Модерація компаній

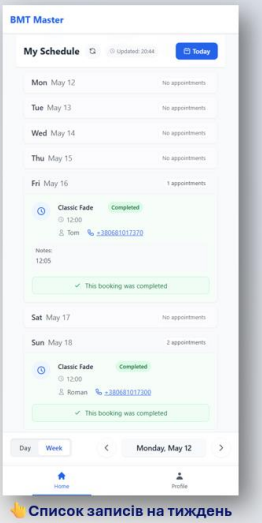


👆Модерація відгуків

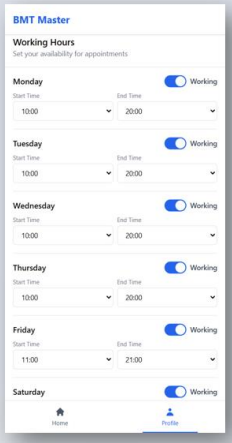
МАЙСТЕР



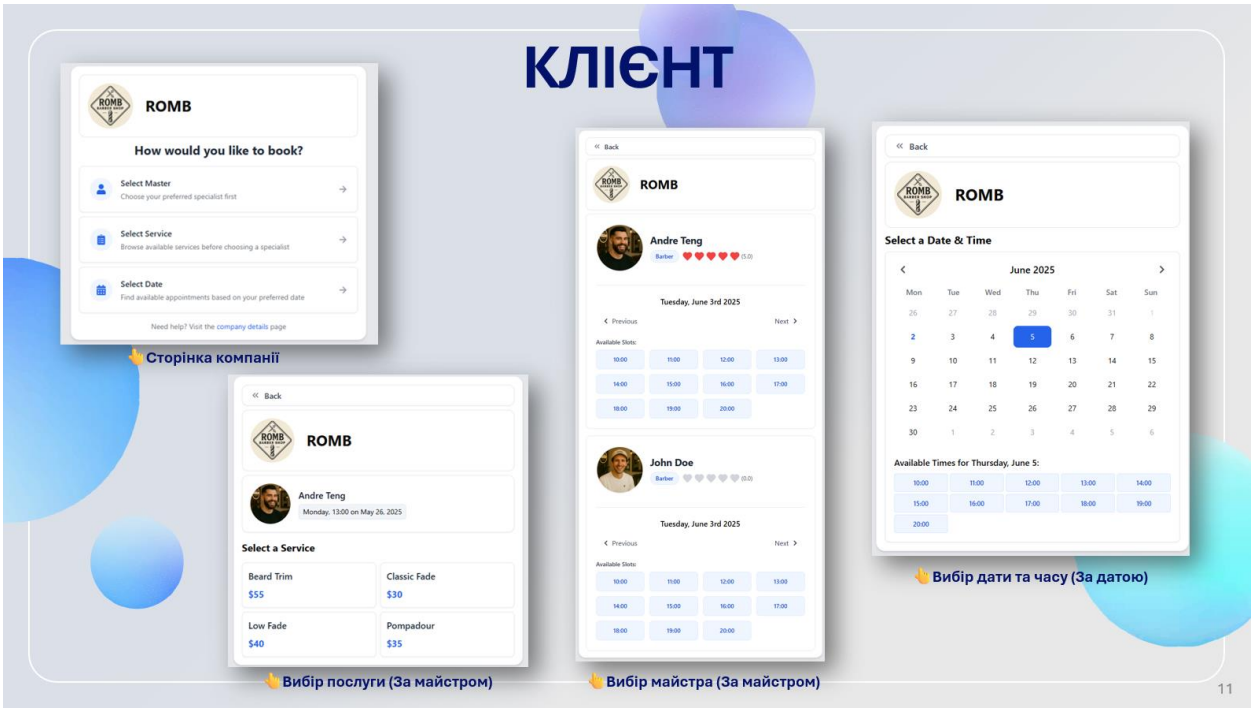
👆Список записів на один день

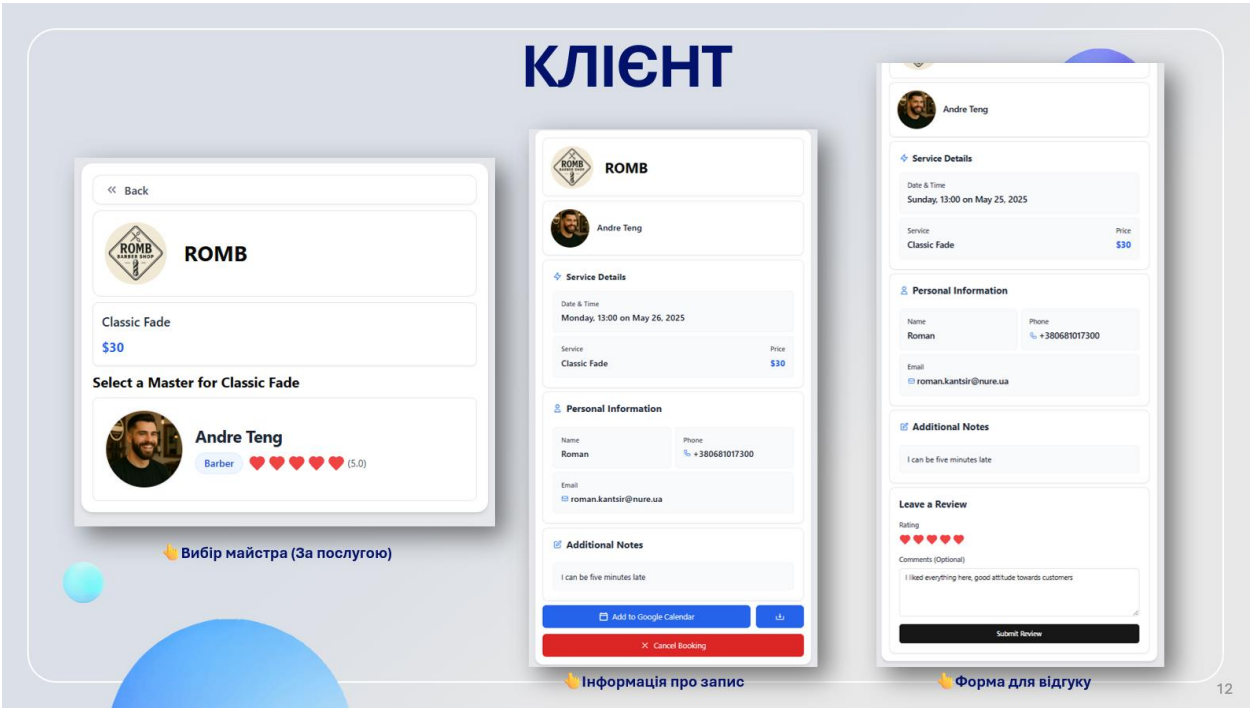


👆Список записів на тиждень



👆Налаштування робочого часу


КЛІЄНТ
ROMB
How would you like to book?
Select Master
Choose your preferred specialist first
Select Service
Browse available services before choosing a specialist
Select Date
Find available appointments based on your preferred date
Need help? Visit the company details page
Сторінка компанії
Back
ROMB
Andre Teng
Monday, 13:00 on May 26, 2025
Select a Service
Beard Trim
$55
Classic Fade
$30
Low Fade
$40
Pompadour
$35
Вибір послуги (За майстром)
Back
ROMB
Andre Teng
Barber
Tuesday, June 3rd 2025
Previous
Next
Available Slots:
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
John Doe
Barber
Tuesday, June 3rd 2025
Previous
Next
Available Slots:
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
Вибір майстра (За майстром)
Back
ROMB
Select a Date & Time
June 2025
Mon
Tue
Wed
Thu
Fri
Sat
Sun
Available Times for Thursday, June 5:
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
Вибір дати та часу (За датою)
11


КЛІЄНТ
Back
ROMB
Classic Fade
$30
Select a Master for Classic Fade
Andre Teng
Barber
(5.0)
Вибір майстра (За послугою)
ROMB
Andre Teng
Service Details
Date & Time
Monday, 13:00 on May 26, 2025
Service
Classic Fade
Price
$30
Personal Information
Name
Roman
Phone
+380681017300
Email
roman.kantsir@nure.ua
Additional Notes
I can be five minutes late
Add to Google Calendar
Cancel Booking
Інформація про запис
Andre Teng
Service Details
Date & Time
Sunday, 13:00 on May 25, 2025
Service
Classic Fade
Price
$30
Personal Information
Name
Roman
Phone
+380681017300
Email
roman.kantsir@nure.ua
Additional Notes
I can be five minutes late
Leave a Review
Rating
Comments (Optional)
I liked everything here, good attitude towards customers
Submit Review
Форма для відгуку
12

ВИСНОВОК

У процесі виконання кваліфікаційної роботи було проаналізовано існуючі рішення для онлайн-запису, виявлено їхній функціонал та недоліки. Отримані результати стали основою для створення власного вебзастосунку, адаптованого під актуальні потреби користувачів.

Розроблений застосунок реалізує:

- підтримку чотирьох ролей (супер-адміністратор, адміністратор, майстер, клієнт);
- онлайн-запис з різними сценаріями (за майстром, послугою або датою);
- керування компаніями, послугами, майстрами, графіками;
- сповіщення, відгуки, календар та аналітику.

Результати роботи представлені у рамках п’ятнадцятої міжнародної науково-технічної конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління»

ДОДАТОК Б

Вихідний код

Б.1 Auth методи

```
import { UserRole } from "@/constants/enums";

const TOKEN_KEY = 'auth_token';

export const auth = {
  login: async (email: string, password: string) => {
    const response = await fetch('/api/auth/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email, password }),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || 'Login failed');
    }

    const data = await response.json();

    if (data.token) {
      localStorage.setItem(TOKEN_KEY, data.token);

      const authInterceptor = (url: string, options:
RequestInit) => {
        if (!options.headers) {
          options.headers = {};
        }

        options.headers = {
          ...options.headers,
          'Authorization': `Bearer ${data.token}`
        };

        return { url, options };
      };
    }

    return data.user;
  },
```

```
  logout: async () => {
    localStorage.removeItem(TOKEN_KEY);

    await fetch('/api/auth/logout', {
      method: 'POST',
    });
  },

  getToken: () => {
    if (typeof window !== 'undefined') {
      return localStorage.getItem(TOKEN_KEY);
    }
    return null;
  },

  register: async (userData: { name: string, email: string,
password: string, passwordConfirm: string }) => {
    const response = await fetch('/api/auth/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(userData),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || 'Registration failed');
    }

    const data = await response.json();

    if (data.token) {
      localStorage.setItem(TOKEN_KEY, data.token);
    } else {
      console.error('No token received from registration');
      throw new Error('Authentication failed after
registration');
    }

    return data.user;
  },

  getCurrentUser: async () => {
    try {
      const token = auth.getToken();

      if (!token) {
        return null;
      }

      const response = await fetch('/api/auth/user', {
        headers: {
```

```
          'Authorization': `Bearer ${token}`
        }
      });

      if (!response.ok) {
        if (response.status === 401) {
          localStorage.removeItem(TOKEN_KEY);
          return null;
        }

        const errorData = await response.json();
        return null;
      }

      const data = await response.json();
      if (!data.user) {
        return null;
      }

      return data.user;
    } catch (error) {
      return null;
    }
  },

  isAuthenticated: () => {
    const token = auth.getToken();
    return !!token;
  },

  checkRole: async (role: UserRole) => {
    const user = await auth.getCurrentUser();
    return user?.role === role;
  },

  requestPasswordReset: async (email: string) => {
    const response = await fetch('/api/auth/reset-password', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email }),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || 'Failed to send
password reset email');
    }
  },

  resetPassword: async (token: string, password: string) => {
    const response = await fetch('/api/auth/reset-
```

```
password/confirm', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ token, password }),
    });

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || 'Failed to reset
password');
    }
  }
};
```

Б.2 Модель Booking

```
import { Service } from "./service";
import { Master } from "./master";
import { Company } from "./company";
import { Client } from "./client";
import { RecordModel } from "pocketbase";
import { bookingService } from "@/services/booking";

export interface CreateBookingData {
  companyId: string;
  masterId: string;
  serviceId: string;
  time: Date;
  name: string;
  phone: string;
  email: string;
  notes?: string;
  captchaToken?: string | null;
}

export interface Booking {
  id: string;
  created: Date;
  updated: Date;
  status: string;
  company: string;
  master: Master;
  service: Service;
  client: Client;
  time: Date;
  notes?: string;
}

export interface BookingRecord extends RecordModel {
  created: string;
```

```
  updated: string;
  status: string;
  company: string;
  master: string;
  service: string;
  client: string;
  time: string;
  notes?: string;
  expand?: {
    master?: Master;
    service?: Service;
    company?: Company;
    client?: Client;
  };
}

export interface BookingValidationError extends Error {
  field?: string;
  code: 'SLOT_UNAVAILABLE' | 'INVALID_DATE' | 'INVALID_CLIENT' |
'CREATION_FAILED';
}

export const mapToBooking = bookingService.mapToBooking;
```

Б.3 Методи моделі Booking

```
import { BookingStatus } from "@/constants/enums";
import { Booking, BookingRecord, BookingValidationError,
CreateBookingData } from "@/types/booking";
import { Client } from "@/types/client";
import { Master } from "@/types/master";
import { Service } from "@/types/service";
import { User } from "@/types/user";
import { formatToPocketBaseTime } from "@/utils/date";
import { TIME_SLOTS } from "@/constants/TimeSlots";

export const bookingService = {
  mapToBooking: (record: BookingRecord): Booking => {
    let masterData: Master;
    if (record.expand?.master) {
      masterData = record.expand.master;
    } else if (typeof record.master === 'string') {
      masterData = {
        id: record.master,
        first_name: 'Unknown',
        last_name: 'Master',
        company: record.company,
        services: [],
        user: { id: '', email: '', created: '', updated: '' } as
User,
        created: '',
        updated: '',
```

```
      };
    } else {
      masterData = record.master as unknown as Master;
    }

    let serviceData: Service;
    if (record.expand?.service) {
      serviceData = record.expand.service;
    } else if (typeof record.service === 'string') {
      serviceData = {
        id: record.service,
        name: 'Unknown Service',
        price: 0,
        company: record.company,
      };
    } else {
      serviceData = record.service as unknown as Service;
    }

    let clientData: Client;
    if (record.expand?.client) {
      clientData = record.expand.client;
    } else if (typeof record.client === 'string') {
      clientData = {
        id: record.client,
        name: 'Unknown Client',
        phone: '',
        email: '',
        company: record.company,
        created: '',
        updated: '',
      };
    } else {
      clientData = record.client as unknown as Client;
    }

    return {
      id: record.id,
      created: new Date(record.created),
      updated: new Date(record.updated),
      status: record.status || 'Active',
      company: record.company,
      master: masterData,
      service: serviceData,
      client: clientData,
      time: new Date(record.time),
      notes: record.notes,
    }
  },

  getBookingByMasterId: async (masterId: string):
Promise<Booking[]> => {
    try {
```

```
      const response = await
fetch(`/api/booking?masterId=${masterId}`);
      if (!response.ok) {
        console.warn(`Failed to fetch bookings for master ID
${masterId}: ${response.status} ${response.statusText}`);
        return [];
      }

      const data = await response.json();

      if (!data) {
        console.warn(`Empty response from booking API for master
ID ${masterId}`);
        return [];
      }

      const records = Array.isArray(data) ? data : (data.items
|| []);

      if (!Array.isArray(records)) {
        console.warn(`Invalid response format from booking API
for master ID ${masterId}`);
        return [];
      }


      return records.map((record: BookingRecord) =>
bookingService.mapToBooking(record));
    } catch (error) {
      console.error("Error fetching bookings by master ID:",
error);
      return [];
    }
  },

  updateBookingStatus: async (bookingId: string, newStatus:
BookingStatus): Promise<boolean> => {
    try {
      const response = await fetch(`/api/booking/${bookingId}`,
{
        method: 'PATCH',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ status: newStatus }),
      });
      return response.ok;
    } catch (error) {
      return false;
    }
  },

  fetchMasterWorkingHours: async (masterId: string, day:
```

```
number): Promise<any> => {
    const response = await
fetch(`/api/master/${masterId}/working-hours?day=${day}`);
    if (!response.ok) return null;

    const data = await response.json();
    return data?.items?.[0];
  },

  fetchMasterBookings: async (masterId: string, formattedDate:
string): Promise<any[]> => {
    const response = await
fetch(`/api/booking?masterId=${masterId}&date=${formattedDate}`)
;
    if (!response.ok) return [];

    return await response.json();
  },

  getBookedTimeSlots: (bookings: any[]): Set<string> => {
    return new Set(
      bookings
        .filter(booking => booking.time)
        .map(booking => new
Date(booking.time).toLocaleTimeString("en-US", {
          hour: "2-digit", minute: "2-digit", hour12: false
        }))
    );
  },

  getTimeInMinutes: (timeString: string): number => {
    const [hours, minutes] = timeString.split(":").map(Number);
    return hours * 60 + minutes;
  },

  isSlotAvailable: (
    slot: string,
    startMinutes: number,
    endMinutes: number,
    bookedSlots: Set<string>
  ): boolean => {
    const [hours, minutes] = slot.split(":").map(Number);
    const slotMinutes = hours * 60 + minutes;

    if (slotMinutes < startMinutes || slotMinutes > endMinutes)
{
      return false;
    }

    return !bookedSlots.has(slot);
  },

  getAvailableSlots: async (masterId: string, date: Date):
```

```
Promise<string[]> => {
    try {
      const dayOfWeek = date.getDay();
      const formattedDate = formatToPocketBaseTime(date).split('
')[0];

      const workingHours = await
bookingService.fetchMasterWorkingHours(masterId, dayOfWeek);
      const bookings = await
bookingService.fetchMasterBookings(masterId, formattedDate);

      if (!workingHours || !workingHours.start_time ||
!workingHours.end_time) {
        return [];
      }

      const bookedSlots =
bookingService.getBookedTimeSlots(bookings);
      const startMinutes =
bookingService.getTimeInMinutes(workingHours.start_time);
      const endMinutes =
bookingService.getTimeInMinutes(workingHours.end_time);

      return TIME_SLOTS.filter(slot =>
        bookingService.isSlotAvailable(slot, startMinutes,
endMinutes, bookedSlots)
      );
    } catch (error) {
      console.error("Error getting available slots:", error);
      return [];
    }
  },

  verifySlotAvailability: async (masterId: string, date: Date,
time: string): Promise<boolean> => {
    try {
      const availableSlots = await
bookingService.getAvailableSlots(masterId, date);
      return availableSlots.includes(time);
    } catch (error) {
      console.error("Error verifying slot availability:",
error);
      return false;
    }
  },

  getBookingsForDay: async (masterId: string, date: Date):
Promise<any[]> => {
    try {
      const formattedDate = formatToPocketBaseTime(date).split('
')[0];
      const response = await
fetch(`/api/booking?masterId=${masterId}&date=${formattedDate}`)
```

```
;

      if (!response.ok) {
        console.error(`Error fetching bookings:
${response.status} ${response.statusText}`);
        return [];
      }

      const data = await response.json();
      return Array.isArray(data) ? data : (data.items || []);
    } catch (error) {
      console.error("Error getting bookings for day:", error);
      return [];
    }
  },

  validateBookingTime: async (masterId: string, time: Date):
Promise<void> => {
    const timeString = time.toLocaleTimeString("en-US", { hour:
"2-digit", minute: "2-digit", hour12: false });
    const isAvailable = await
bookingService.verifySlotAvailability(masterId, time,
timeString);

    if (!isAvailable) {
      const error = new Error("Selected time slot is not
available") as BookingValidationError;
      error.field = "time";
      error.code = "SLOT_UNAVAILABLE";
      throw error;
    }
  },

  prepareBookingData: async (data: CreateBookingData):
Promise<Record<string, any>> => {
    await bookingService.validateBookingTime(data.masterId,
data.time);

    let clientId;
    try {
      const response = await fetch('/api/client/upsert', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          name: data.name,
          phone: data.phone,
          email: data.email,
          company: data.companyId,
        }),
      });
```

```
      if (!response.ok) {
        throw new Error(`Error upserting client:
${response.statusText}`);
      }

      const result = await response.json();
      clientId = result.id;
    } catch (error) {
      const clientError = new Error("Failed to create client")
as BookingValidationError;
      clientError.field = "client";
      clientError.code = "INVALID_CLIENT";
      throw clientError;
    }

    return {
      company: data.companyId,
      master: data.masterId,
      service: data.serviceId,
      client: clientId,
      time: formatToPocketBaseTime(data.time),
      notes: data.notes || "",
      status: "Active",
    };
  },

  createBooking: async (data: CreateBookingData) => {
    try {
      const bookingData = await
bookingService.prepareBookingData(data);

      const response = await fetch('/api/booking/create', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          ...bookingData,
          captchaToken: data.captchaToken,
        }),
      });

      if (!response.ok) {
        const errorData = await response.json();
        const error = new Error(errorData.error || "Failed to
create booking") as BookingValidationError;
        error.code = "CREATION_FAILED";
        throw error;
      }

      const newBooking = await response.json();
      return newBooking;
    } catch (error) {
```

```
      if ((error as any).code) {
        throw error;
      }

      const bookingError = new Error(error instanceof Error ?
error.message : "Unknown error") as BookingValidationError;
      bookingError.code = "CREATION_FAILED";
      throw bookingError;
    }
  },

  getAvailableSlotsForAllMasters: async (masterIds: string[],
date: Date): Promise<string[]> => {
    try {
      const allSlotsPromises = masterIds.map(id =>
bookingService.getAvailableSlots(id, date));
      const allSlots = await Promise.all(allSlotsPromises);

      if (allSlots.length === 0) return [];

      return allSlots.reduce((commonSlots, masterSlots) =>
        commonSlots.filter(slot => masterSlots.includes(slot))
      );
    } catch (error) {
      console.error("Error getting common available slots:",
error);
      return [];
    }
  },

  getAllBookingsForCompany: async (companyId: string):
Promise<Booking[]> => {
    try {
      const response = await
fetch(`/api/booking?companyId=${companyId}`);
      if (!response.ok) {
        return [];
      }

      const data = await response.json();
      const records = Array.isArray(data) ? data : (data.items
|| []);

      return records.map((record: BookingRecord) =>
bookingService.mapToBooking(record));
    } catch (error) {
      console.error("Error fetching bookings for company:",
error);
      return [];
    }
  }
};
```