CSE471: System Analysis and Design
Project Report
Project Title: Music Studio and Instrument Store Website

Group No: 7, CSE471 Lab Section: 6, Spring 2025	
ID	**Name**
21201031	Maisha Fairooz
21201356	Yamin Adnan
22101486	Mustafiz Ahsan
22101229	Aparup Chowdhury
22101460	Arpita Saha

Submission Date: *10/05/2025*

Table of Contents

1. System Request

Business need:

The business needs a complete online platform that includes subscriptions for music lessons, artist communication, instrument sales and rentals, and bookings for music studios. This is necessary to address the increasing need for online availability in the music industry and offer a single service hub for artists and audiences.

Business requirements:

- Secure user authentication and profile management system
- Artist profile management with support for showcasing music releases and collaborations.
- Musician Collaboration Hub for artist interaction.
- Calendar-based booking and studio scheduling with real-time availability checks.
- Subscription-based access to music lessons.
- E-commerce capabilities include product listing, search/filter, cart, checkout, and payment gateway.
- Admin dashboard for managing users, inventory, bookings, and content (FAQ, blog, products).
- Admin dashboard for managing users, inventory, bookings, and content (FAQ, blog, products).

Business value:

- Increases income from the sale of musical instruments, lesson subscriptions, and rental fees.
- Enhances client satisfaction through self-service reservations, email, and in-app alerts, and customized promotions.
- Improves operational efficiency by managing inventory control, payments, and reservations.
- Improve awareness of the company through social media integration, blog hosting, and artist showcases.
- Increases customer satisfaction with a discount structure and point-based appreciation awards.

Special issues or constraints:

1. The system must abide by the relevant data privacy regulations.
2. A stable and secure connection is essential for third-party services like Spotify, YouTube.
3. Compatibility across platforms and devices (desktop, tablet, and mobile).
4. The infrastructure for in-app notifications and the secure email system (using Nodemailer) needs to be set up and maintained correctly.
5. Even when a lot of people are implementing the system at once, like at artist events or special promotions, it should still function properly and efficiently.

2. Functional Requirements

Module 1: User Authentication and Admin Profile Management

FR 1: The system must allow users to sign up, log in, and reset their passwords via email.

FR 2: Users (both Standard and Artist profile users) can edit personal details (name, phone, address, releases, etc.).

FR 3: Users (both Standard and Artist profile users) can see their calendar-based bookings, place order details, and earn points on their dashboard.

FR 4: Admin can manage and delete user profiles.

FR 5: Admins must be able to manage inventory, including adding and editing products for the store.

FR 6: Admin can cancel bookings and orders.

FR 7: The system must include an FAQ section where users can find information and raise issues.

FR 8: The website must include an About Us page with links to social media platforms like Instagram and YouTube.

FR 9: Admin can control and customize the point-based reward system (Configure point values for different activities, modify discounts and benefits for each tier, adjust point expiry rules and redemption rates).

Module 2: Artist Profile & Studio Bookings

FR 10: Artists can showcase their releases (several platforms might be integrated eg: YouTube, Spotify, etc) on their artist profile.

FR 11: The system must include a Musician Collaboration Hub, where artists can find and connect with other musicians for jam sessions.

FR 12: Users must be able to subscribe to music lessons through the platform.

FR 13: The system must provide a real-time availability check for studios and practice rooms.

FR 14: Users must be able to book studio/practice rooms based on location, date, time, and room size.

FR 15: Users must receive a booking confirmation and a digital receipt via email (sent through Nodemailer).

FR 16: Users must be able to cancel bookings, following a basic cancellation policy (e.g., "Cancel 24 hours before booking").

FR 17: Users must be able to contact an artist through the platform.

FR 18: Users will get email notifications for their respective tasks.

Module 3: E-Commerce & General Features

FR 19: The system must allow users to browse product listings categorized by type (guitars, drums, etc.).

FR 20: Users must be able to search for products and apply filters such as price range, instrument type, and brand.

FR 21: Each product must have a detailed page displaying images, descriptions, and other relevant information.

FR 22: Users must be able to add products to a shopping cart.

FR 23: The system must provide a checkout page with a payment gateway for purchasing items.

FR 24: Users must be able to comment (review) on products.

FR 25: The platform must support an instrument rental option, allowing users to rent musical instruments for practice pad sessions.

FR 26: Users will be able to accumulate points based on their activities (Booking practice or recording sessions, purchasing or renting musical instruments or accessories) and will be allocated to different categories (Bronze, Silver, Gold, Platinum)

FR 27: Users will get discounts based on their points categories (eg: 5% for Bronze, 10% for Silver, 15% for Gold, 20% for Platinum)

3. Technology (Framework, Languages)

JavaScript, MERN stack (MongoDB, ExpressJS, React.js, Node.js)

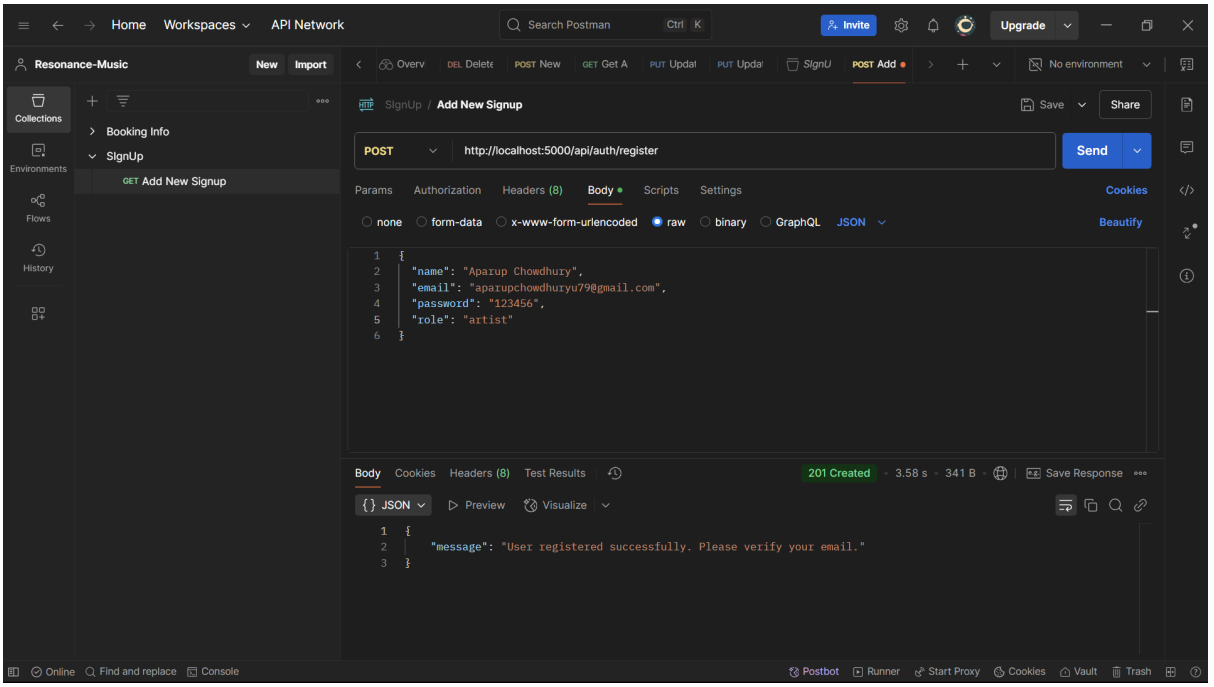
4. Backend Development

1. Sign Up:

```
// Register a new user
export const register = async (req, res) => {
  console.log('Received request to register a new user');
  try {
    const { name, email, password, role } = req.body;

    // Check if user already exists
    let user = await User.findOne({ email });
    if (user) {
      return res.status(400).json({ message: 'User already exists' });
    }

    // Validate role
    const validRoles = ['standard', 'artist'];
    const userRole = role === 'artist' ? 'artist' : 'standard';

    if (role && !validRoles.includes(userRole)) {
      return res.status(400).json({ message: 'Invalid role' });
    }

    // Hash password
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    // Create verification token
    const emailVerificationToken = crypto.randomBytes(32).toString('hex');

    // Create new user
    user = new User({
      name,
      email,
      password: hashedPassword,
      role: userRole,
      emailVerificationToken
    });

    await user.save();
```

```
    // Send verification email
    const verificationUrl = `${process.env.CLIENT_URL}/verify-email?token=${emailVerificationToken}`;
    await sendEmail(
      email,
      'Email Verification',
      `Please verify your email by clicking the link: ${verificationUrl}`
    );

    return res.status(201).json({ message: 'User registered successfully. Please verify your email.' });
  } catch (error) {
    console.error(error);
    return res.status(500).json({ message: 'Server error' });
  }
};
```

This API endpoint allows new users to register an account by submitting their name, email, password, and role (standard or artist). It checks for existing users, hashes the password securely, creates an email verification token, stores the new user in the database, and sends a verification email. If the registration is successful, it responds with a message prompting the user to verify their email.

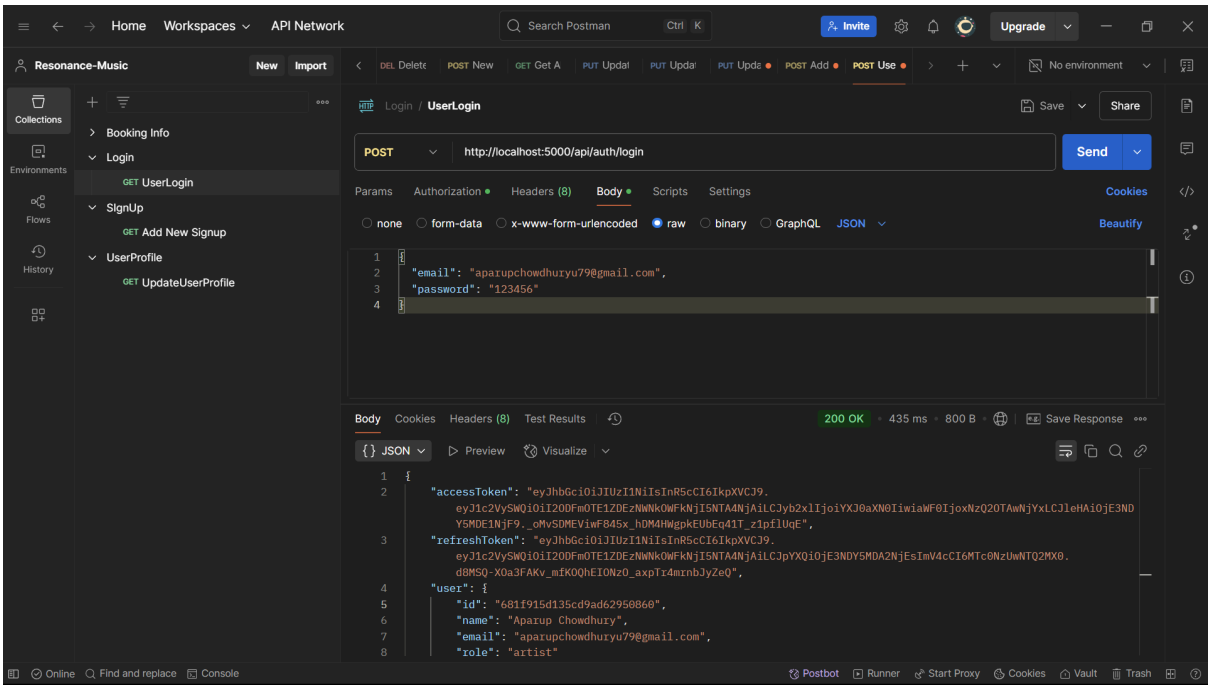


2. Login

```
// Login user
export const login = async (req, res) => {
  console.log('Login function called');
  try {
    const { email, password } = req.body;

    // Check if user exists
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: 'Invalid credentials' });
    }
    console.log("User found:", user);

    // Check if user is active
    if (!user.isActive) {
      return res.status(403).json({ message: 'Account has been restricted. Please contact admin.' });
    }
    console.log("User is active:", user.isActive);

    // Check if password is correct
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: 'Invalid credentials' });
    }
    console.log("Password matched:", isMatch);

    // Check if email is verified
    // if (!user.isEmailVerified) {
    //   console.log("Email not verified:", !user.isEmailVerified);
    //   return res.status(400).json({ message: 'Please verify your email before logging in' });
    // }
    // console.log("Email verified:", user.isEmailVerified);

    // Generate JWT tokens
    const accessToken = jwt.sign(
      { userId: user._id, role: user.role },
      process.env.JWT_SECRET,
```

```
      process.env.JWT_SECRET,
      { expiresIn: '15m' }
    );

    const refreshToken = jwt.sign(
      { userId: user._id },
      process.env.JWT_REFRESH_SECRET,
      { expiresIn: '7d' }
    );

    // Save refresh token to database
    const tokenDoc = new Token({
      userId: user._id,
      token: refreshToken,
      type: 'refresh',
      expiresAt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000) // 7 days
    });

    await tokenDoc.save();

    return res.status(200).json({
      accessToken,
      refreshToken,
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
        role: user.role
      }
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

This login API authenticates a user by checking their email and password. If the credentials are valid and the account is active, it generates and returns a short-lived access token and a long-lived refresh token, along with basic user details. It also stores the refresh token in the database for session management.



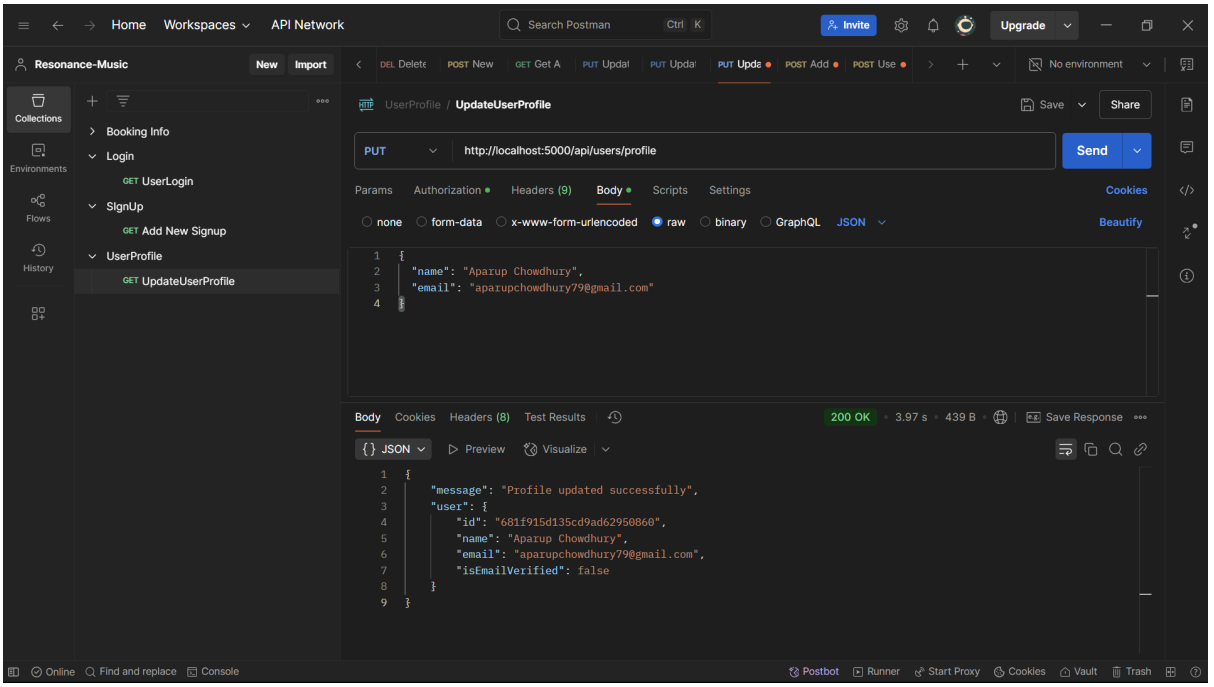
3. Update User Profile :

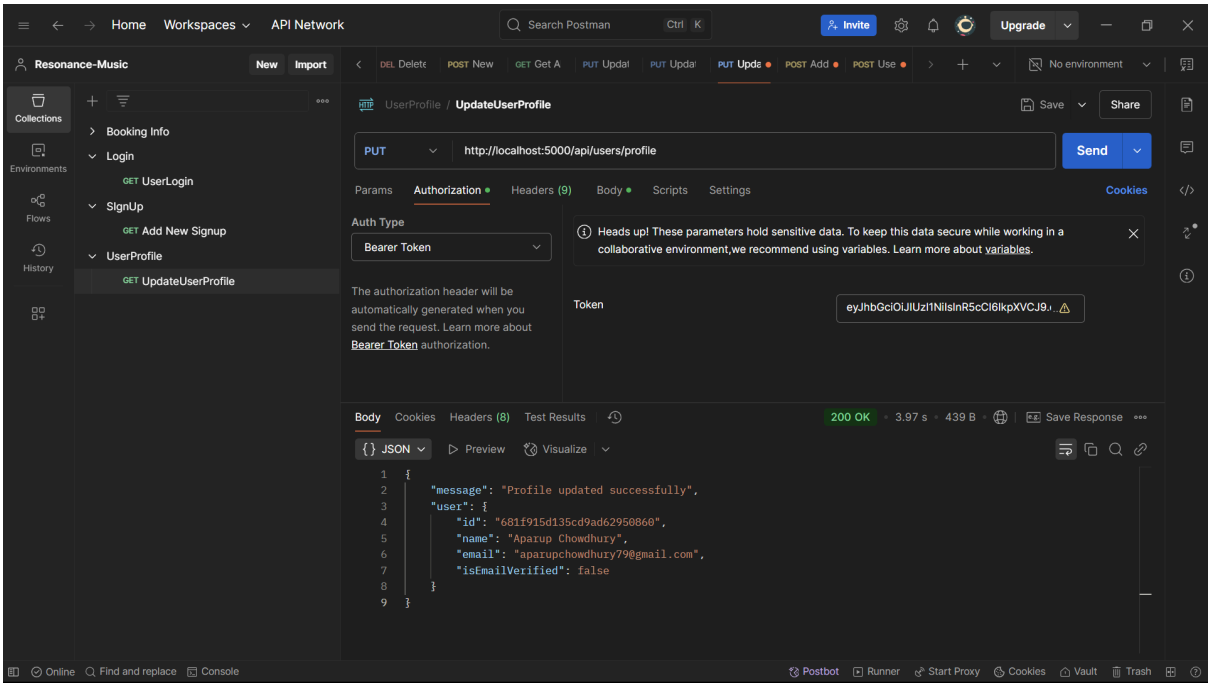
```
// Update user profile
export const updateProfile = async (req, res) => {
  try {
    const { name, email } = req.body;

    // Find user
    let user = await User.findById(req.user.userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Check if email is being changed
    if (email && email !== user.email) {
      // Check if new email already exists
      const emailExists = await User.findOne({ email });
      if (emailExists) {
        return res.status(400).json({ message: 'Email already in use' });
      }

      // Generate new verification token
      const emailVerificationToken = crypto.randomBytes(32).toString('hex');

      // Update user with new email and verification token
      user.email = email;
      user.isEmailVerified = false;
      user.emailVerificationToken = emailVerificationToken;

      // Send verification email
      const verificationUrl = `${process.env.CLIENT_URL}/verify-email?token=${emailVerificationToken}`;
      await sendEmail(
        email,
        'Email Verification',
        `Please verify your new email by clicking the link: ${verificationUrl}`
      );
    }
```

```
        email,
        'Email Verification',
        `Please verify your new email by clicking the link: ${verificationUrl}`
      );
    }

    // Update name if provided
    if (name) {
      user.name = name;
    }

    user.updatedAt = Date.now();
    await user.save();

    res.status(200).json({
      message: 'Profile updated successfully',
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
        isEmailVerified: user.isEmailVerified
      }
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

This update user profile API allows authenticated users to change their name and/or email. If the email is updated, it checks for duplicates, resets the verification status, generates a new

token, and sends a verification email. The updated user data is saved and returned, excluding sensitive fields like the password.





4. Create New Product :

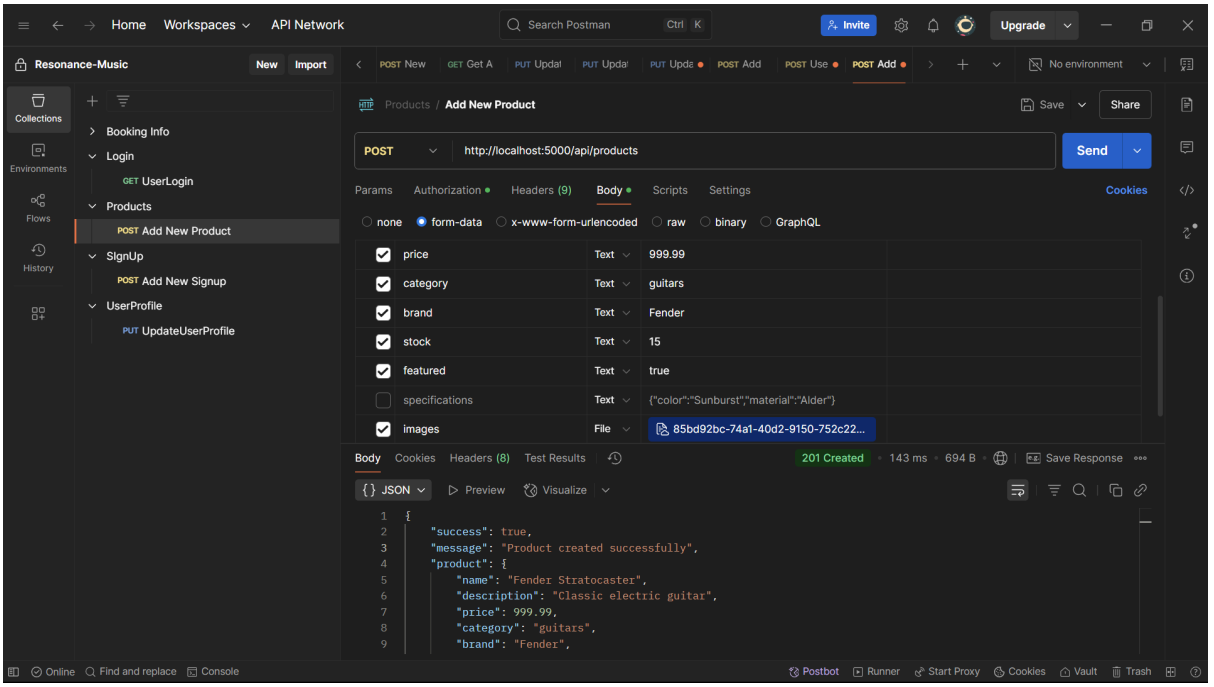
```
// Admin: Create new product
export const createProduct = async (req, res) => {
  try {
    const {
      name,
      description,
      price,
      category,
      brand,
      stock,
      specifications,
      featured
    } = req.body;

    // Get image paths from middleware
    const images = req.filePaths || [];

    if (images.length === 0) {
      return res.status(400).json({
        success: false,
        message: 'At least one product image is required'
      });
    }

    const product = new Product({
      name,
      description,
      price,
      category,
      brand,
      stock,
      images,
      specifications: specifications || {},
      featured: featured || false
    });

    await product.save();
```

```
    await product.save();

    res.status(201).json({
      success: true,
      message: 'Product created successfully',
      product
    });
  } catch (error) {
    console.error('Error creating product:', error);
    res.status(500).json({
      success: false,
      message: 'Server error',
      error: error.message
    });
  }
};

// Admin: Update product
export const updateProduct = async (req, res) => {
  try {
    const { id } = req.params;
    const {
      name,
      description,
      price,
      category,
      brand,
      stock,
      specifications,
      featured
    } = req.body;

    const product = await Product.findById(id);

    if (!product) {
      return res.status(404).json({
```

```
    if (!product) {
      return res.status(404).json({
        success: false,
        message: 'Product not found'
      });
    }

    // Update fields if provided
    if (name) product.name = name;
    if (description) product.description = description;
    if (price !== undefined) product.price = price;
    if (category) product.category = category;
    if (brand) product.brand = brand;
    if (stock !== undefined) product.stock = stock;
    if (specifications) product.specifications = specifications;
    if (featured !== undefined) product.featured = featured;

    // Add new images if provided
    if (req.filePaths && req.filePaths.length > 0) {
      product.images = [...product.images, ...req.filePaths];
    }

    await product.save();

    res.status(200).json({
      success: true,
      message: 'Product updated successfully',
      product
    });
  } catch (error) {
    console.error('Error updating product:', error);
    res.status(500).json({
      success: false,
      message: 'Server error',
      error: error.message
    });
```

```
    res.status(200).json({
      success: true,
      message: 'Product updated successfully',
      product
    });
  } catch (error) {
    console.error('Error updating product:', error);
    res.status(500).json({
      success: false,
      message: 'Server error',
      error: error.message
    });
  }
};
```

This Admin Create Product API enables administrators to add new products to the system by submitting essential details such as name, description, price, category, brand, stock quantity, featured status, specifications, and product images. It ensures that at least one image is provided and that the category matches predefined valid values. The specifications field, typically sent as a JSON string in form-data, is parsed into an object on the server before being saved. Once validated and processed, the product is stored in the database, and a success response with the newly created product details is returned. This API is critical for managing the product catalog in an e-commerce or inventory system.

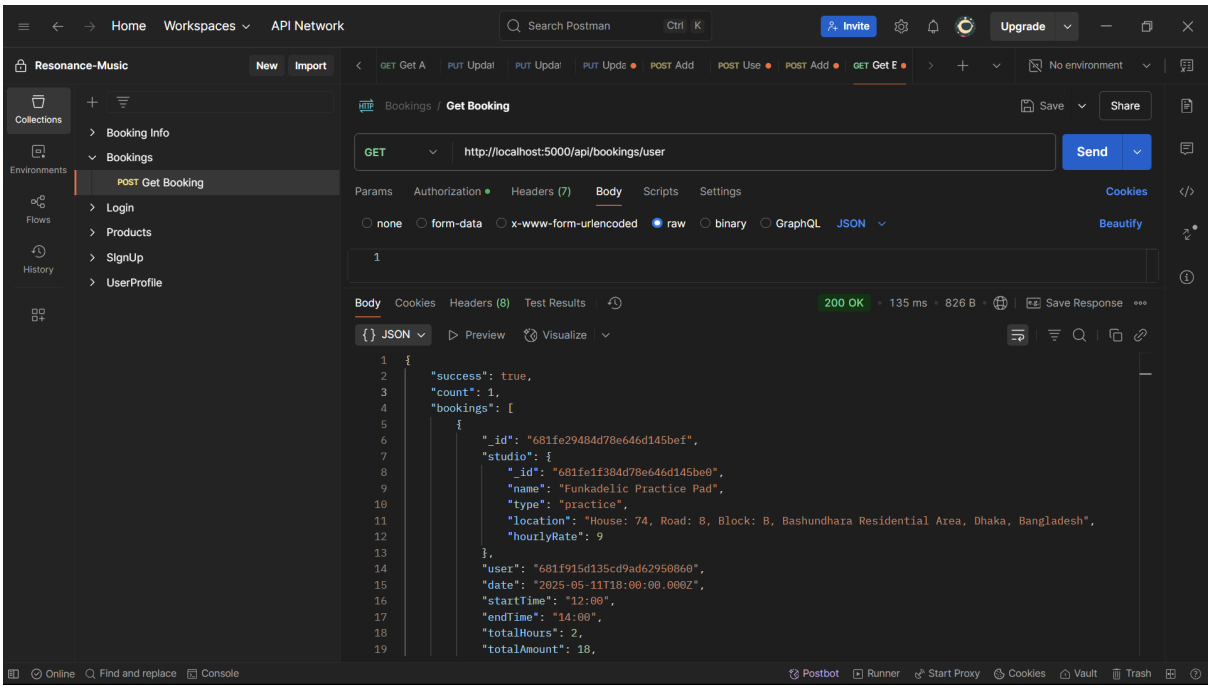


5. Get Booking Info:

```
// Get all bookings for a user
export const getUserBookings = async (req, res) => {
  try {
    const userId = req.user.userId;

    const bookings = await Booking.find({ user: userId })
      .populate('studio', 'name type location hourlyRate')
      .sort({ date: -1, startTime: 1 });

    res.status(200).json({
      success: true,
      count: bookings.length,
      bookings
    });
  } catch (error) {
    console.error('Error fetching user bookings:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to fetch bookings',
      error: error.message
    });
  }
};
```

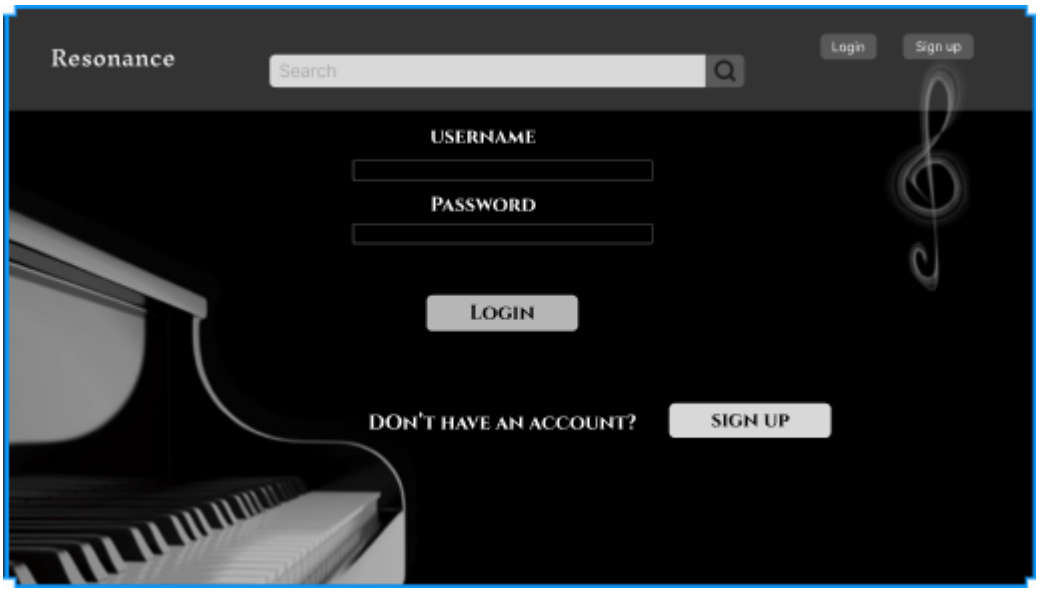
The Get User Bookings API retrieves all studio bookings made by the currently authenticated user. It returns a list of bookings, including details such as studio name, type, location, hourly rate, booking date, start and end time, total duration, amount, and status. Results are sorted

with the most recent bookings first. Authentication via JWT token is required.

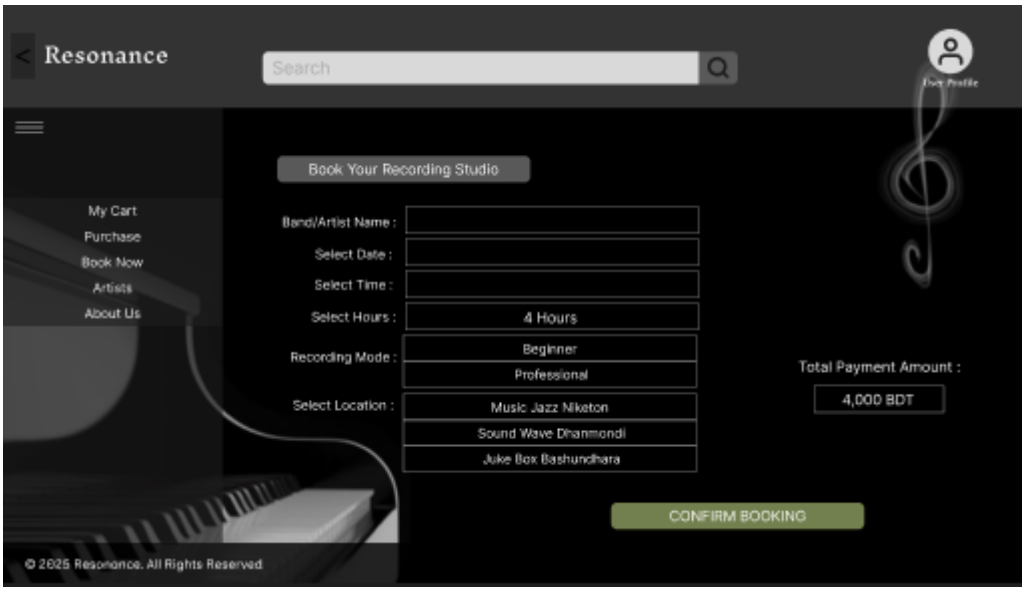


5. User Interface Design

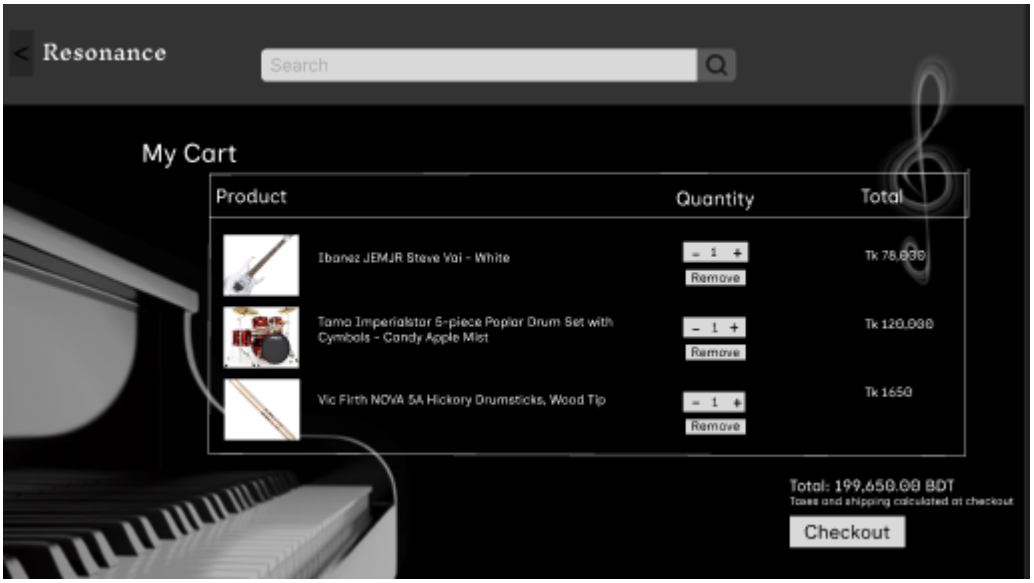
Login Page:



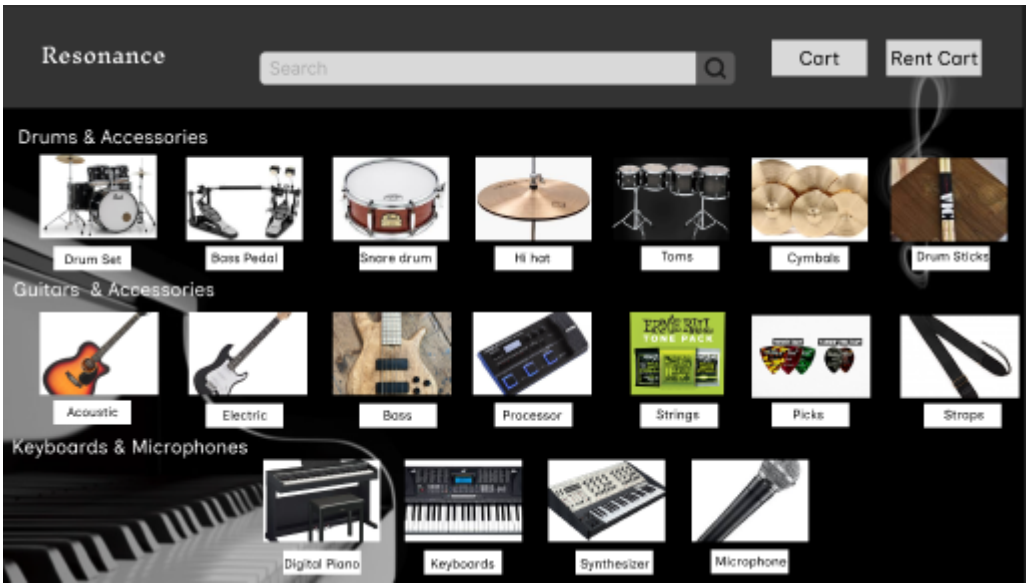
Recording studio booking:



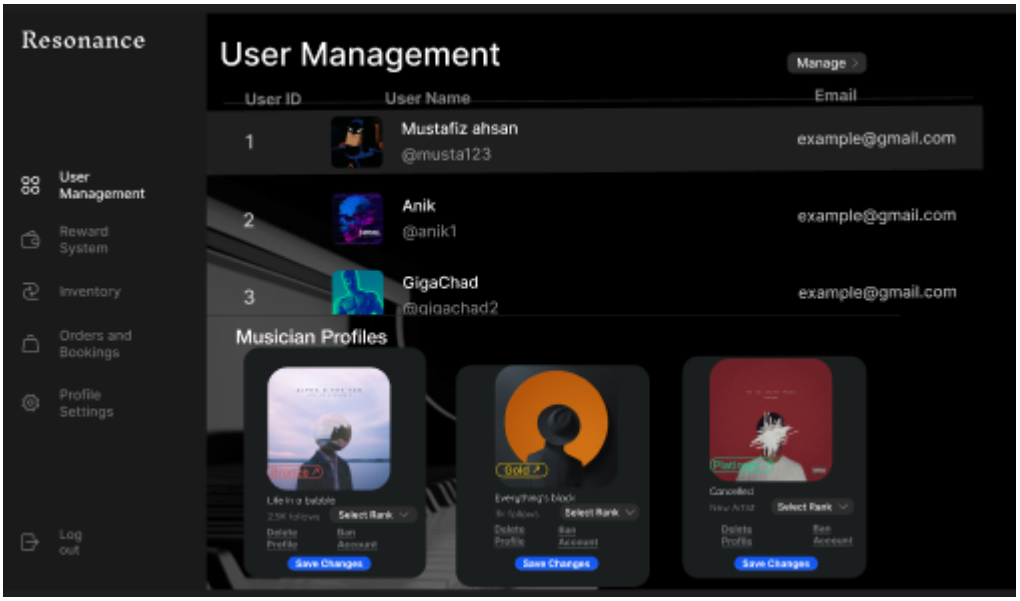
Cart:



Products list:



Admin User Management:



Link: [Prototype Link]

6. Frontend Development

1. Login page :

This React component, LoginPage, renders a styled login form for users to enter their email and password. It:

- Uses useState to manage form data, errors, and a success message.
- Checks for a registration success message from the router location.state

- Validates the form fields before submission.
- Calls a login function (from useAuthStore) to authenticate the user.
- Navigates to the homepage (/) on successful login or shows error messages otherwise.
- Includes UI elements like a navbar, form inputs, error/success alerts, and links to "Forgot Password" and "Sign up" pages.

```
import React, { useState, useEffect } from 'react';
import { Link, useNavigate, useLocation } from 'react-router-dom';
import Navbar from '../components/Navbar';
import useAuthStore from '../store/authStore';

const LoginPage = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const { login } = useAuthStore();
  const [formData, setFormData] = useState({
    email: '',
    password: ''
  });

  const [errors, setErrors] = useState({});
  const [successMessage, setSuccessMessage] = useState('');

  useEffect(() => {
    // Check for success message from registration
    if (location.state?.message) {
      setSuccessMessage(location.state.message);
      // Clear the message from location state
      window.history.replaceState({}, document.title);
    }
  }, [location]);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleSubmit = async (e) => {
```

```
    e.preventDefault();

    const newErrors = {};
    if (!formData.email) newErrors.email = 'Email is required';
    if (!formData.password) newErrors.password = 'Password is
required';

    if (Object.keys(newErrors).length > 0) {
      setErrors(newErrors);
      return;
    }

    try {
      const result = await login(formData.email, formData.password);

      if (result) {
        navigate('/');
      } else {
        setErrors({
          submit: 'Failed to login. Please check your credentials.'
        });
      }
    } catch (error) {
      console.error('Login error:', error);
      setErrors({
        submit: error.response?.data?.message || 'Invalid email or
password'
      });
    }
  };

  return (
    <div className="flex flex-col min-h-screen bg-cover bg-center
text-white" style={{ backgroundImage: "url('piano-background.jpg')" }}>
      <Navbar showLogin={false} />
      <div className="flex-1 flex justify-center items-center px-4">
        <div className="bg-black/70 backdrop-blur-md p-8 rounded-2xl
shadow-2xl w-full max-w-md border border-white/30 text-white">
          <h1 className="text-center text-3xl font-bold mb-8">Welcome
Back</h1>

          {successMessage && (
```

```
            <div className="bg-green-800/50 text-green-200 p-3
rounded-md mb-4 text-center border border-green-500/50">
              {successMessage}
            </div>
          )}

          <form className="flex flex-col gap-4"
onSubmit={handleSubmit}>
            <div className="flex flex-col gap-2">
              <label htmlFor="email"
className="font-medium">Email</label>
              <input
                type="email"
                id="email"
                name="email"
                value={formData.email}
                onChange={handleChange}
                className={`p-3 border rounded-md text-base bg-white/10
text-white placeholder-white/60 ${
                  errors.email ? 'border-red-500' : 'border-white/30'
                }`}
                placeholder="Enter your email"
              />
              {errors.email && <span className="text-red-400 text-sm
font-semibold">{errors.email}</span>}
            </div>

            <div className="flex flex-col gap-2">
              <label htmlFor="password"
className="font-medium">Password</label>
              <input
                type="password"
                id="password"
                name="password"
                value={formData.password}
                onChange={handleChange}
                className={`p-3 border rounded-md text-base bg-white/10
text-white placeholder-white/60 ${
                  errors.password ? 'border-red-500' :
'border-white/30'
                }`}
                placeholder="Enter your password"
              />
```

```
              {errors.password && <span className="text-red-400 text-sm
font-semibold">{errors.password}</span>}
            </div>

            <div className="text-right">
              <Link to="/forgot-password" className="text-white/80
text-sm hover:text-white transition-colors">
                Forgot Password?
              </Link>
            </div>

            {errors.submit && (
              <div className="text-red-400 text-sm
font-semibold">{errors.submit}</div>
            )}

            <button
              type="submit"
              className="bg-white text-black font-semibold p-3
rounded-md text-base cursor-pointer transition-colors hover:bg-gray-300
mt-2"
            >
              Login
            </button>
          </form>

          <div className="mt-6 text-center text-white/80">
            Don't have an account? <Link to="/signup"
className="text-white hover:underline">Sign up</Link>
          </div>
        </div>
      </div>
    </div>
  );
};

export default LoginPage;
```

2. Studio booking

This React component, Booking, displays a page with two interactive booking options: "Book Studio" and "Rent Instrument". It:

- Uses useNavigate from React Router to handle navigation.
- Displays a full-page background with a Navbar at the top.
- Shows two clickable cards: one navigates to /book-studio, the other to /rent-instrument.
- Each card contains an icon, title, and description with hover effects for visual feedback.

```
import React from 'react';
import { useNavigate } from 'react-router-dom';
import Navbar from '../components/Navbar';

const Booking = () => {
  const navigate = useNavigate();

  return (
    <div className="flex flex-col min-h-screen bg-cover bg-center
text-white" style={{ backgroundImage: "url('piano-background.jpg')" }}>
      <Navbar showLogin={true} showSignup={true} />
      <div className="flex-1 p-8 pt-16">
        <div className="max-w-6xl mx-auto">
          <h1 className="text-4xl font-bold text-center mb-12">Booking
Options</h1>

          <div className="grid grid-cols-1 md:grid-cols-2 gap-8">
            {/* Book Studio Option */}
            <div
              onClick={() => navigate('/book-studio')}
              className="bg-black/40 hover:bg-black/60 transition-all
duration-300 rounded-lg p-8 cursor-pointer border border-gray-700
hover:border-purple-500 flex flex-col items-center justify-center h-80"
            >
              <div className="text-6xl mb-4">🎵</div>
              <h2 className="text-3xl font-bold mb-4 text-center">Book
Studio</h2>
              <p className="text-center text-gray-300">
                Reserve studio time for your recording sessions,
rehearsals, or production needs
```

```
              </p>
            </div>

            {/* Rent Instrument Option */}
            <div
              onClick={() => navigate('/rent-instrument')}
              className="bg-black/40 hover:bg-black/60 transition-all
duration-300 rounded-lg p-8 cursor-pointer border border-gray-700
hover:border-purple-500 flex flex-col items-center justify-center h-80"
            >
              <div className="text-6xl mb-4">🎸</div>
              <h2 className="text-3xl font-bold mb-4 text-center">Rent
Instrument</h2>
              <p className="text-center text-gray-300">
                Browse and rent high-quality instruments for your
performances or recordings
              </p>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Booking;
```

3. Cart

This Cart React component displays and manages the user's shopping cart. It:

- Uses a custom useCart context for accessing cart data and functions like updateQuantity, removeItem, and refreshCart.
- Shows a loading spinner while data is being fetched.
- Displays a message if the cart is empty, with a link to browse products.
- Filters and renders unique cart items, displaying product details, quantity controls, subtotal, and a remove button.
- Shows an order summary including subtotal, estimated shipping/tax, and a checkout button.
- Includes a styled layout with a background image and Navbar.

```
import React from 'react';
import { Link } from 'react-router-dom';
import { useCart } from '../context/CartContext';
import Navbar from '../components/Navbar';
import { FaSpinner, FaTrash, FaArrowLeft, FaShoppingCart } from
'react-icons/fa';

const Cart = () => {
  console.log("Cart component rendered");
  const { cart, loading, updateQuantity, removeItem, refreshCart } =
useCart();

  if (loading) {
    return (
      <div className="flex flex-col min-h-screen bg-cover bg-center
text-white" style={{ backgroundImage: "url('/piano-background.jpg')"
}}>
        <Navbar />
        <div className="flex-1 flex justify-center items-center">
          <FaSpinner className="animate-spin text-4xl text-purple-500"
/>
        </div>
      </div>
    );
  }

  if (!cart?.items || cart.items.length === 0) {
    return (
      <div className="flex flex-col min-h-screen bg-cover bg-center
text-white" style={{ backgroundImage: "url('/piano-background.jpg')"
}}>
        <Navbar />
        <div className="container mx-auto px-4 py-8">
          <h1 className="text-3xl font-bold text-white mb-8">Your
Cart</h1>
          <div className="bg-black/70 backdrop-blur-md p-8 rounded-lg
border border-white/20 text-center">
            <FaShoppingCart className="mx-auto text-5xl text-white/50
mb-4" />
            <p className="text-white/70 mb-6">Your cart is empty</p>
            <Link to="/shop" className="inline-block bg-purple-600
hover:bg-purple-700 text-white py-2 px-6 rounded-md">
```

```
              Browse Products
            </Link>
          </div>
        </div>
      </div>
    );
  }

  const handleQuantityChange = async (productId, newQuantity) => {
    if (newQuantity < 1) return;
    await updateQuantity(productId, newQuantity);
  };

  const handleRemoveItem = async (productId) => {
    if (window.confirm('Are you sure you want to remove this item from
your cart?')) {
      await removeItem(productId);
    }
  };

  // Process cart items to handle duplicates and null products
  const processedCartItems = [];
  const productMap = new Map();

  // First pass: Group items by product ID and prioritize ones with
images
  if (cart.items && cart.items.length > 0) {
    cart.items.forEach(item => {
      if (!item || !item.product) return;

      const productId = item.product._id;
      if (!productId) return;

      const existingItem = productMap.get(productId);

      // If this is a new product or the current item has images while
the existing one doesn't
      if (!existingItem ||
          (!existingItem.product.images && item.product.images &&
item.product.images.length > 0)) {
        productMap.set(productId, item);
      }
    });
```

```
    // Convert map values to array
    productMap.forEach(item => {
      processedCartItems.push(item);
    });
  }

  return (
    <div className="flex flex-col min-h-screen bg-cover bg-center
text-white" style={{ backgroundImage: "url('/piano-background.jpg')"
}}>
      <Navbar />
      <div className="container mx-auto px-4 py-8">
        <h1 className="text-3xl font-bold text-white mb-8">Your
Cart</h1>

        <div className="grid grid-cols-1 md:grid-cols-3 gap-8">
          <div className="md:col-span-2">
            {/* Cart Items */}
            <div className="bg-black/70 backdrop-blur-md rounded-lg
border border-white/20 overflow-hidden">
              {processedCartItems.map((item) => (
                <div key={item.product._id} className="p-4 border-b
border-white/20 last:border-b-0">
                  <div className="flex flex-col sm:flex-row">
                    <div className="flex-shrink-0 w-24 h-24 bg-white/5
rounded-md overflow-hidden mr-4 mb-4 sm:mb-0">
                      <img
                        src={item.product.images &&
item.product.images.length > 0
                          ?
`http://127.0.0.1:5000${item.product.images[0]}`
                          : '/placeholder-image.jpg'}
                        alt={item.product.name || 'Product'}
                        className="w-full h-full object-cover"
                        onError={(e) => {
                          e.target.onerror = null;
                          e.target.src = '/placeholder-image.jpg';
                        }}
                      />
                    </div>

                    <div className="flex-grow">
```

```
                    <div className="flex justify-between">
                      <Link to={`/products/${item.product._id}`}
className="text-white hover:text-purple-400">
                        <h3
className="font-medium">{item.product.name || 'Unknown Product'}</h3>
                      </Link>
                      <button
                        onClick={() =>
handleRemoveItem(item.product._id)}
                        className="text-red-400 hover:text-red-300"
                      >
                        <FaTrash />
                      </button>
                    </div>

                    <p className="text-white/70
text-sm">{item.product.brand || ''}</p>
                    <p className="text-purple-400
mt-1">${(item.product.price || 0).toFixed(2)}</p>

                    <div className="flex items-center mt-2">
                      <button
                        onClick={() =>
handleQuantityChange(item.product._id, item.quantity - 1)}
                        disabled={item.quantity <= 1}
                        className="bg-white/20 text-white w-8 h-8
flex items-center justify-center rounded-l-md disabled:opacity-50"
                      >
                        -
                      </button>
                      <input
                        type="number"
                        min="1"
                        value={item.quantity}
                        onChange={(e) =>
handleQuantityChange(item.product._id, parseInt(e.target.value) || 1)}
                        className="w-12 h-8 bg-white/10 text-white
text-center border-y border-white/20"
                      />
                      <button
                        onClick={() =>
handleQuantityChange(item.product._id, item.quantity + 1)}
```

```
                        className="bg-white/20 text-white w-8 h-8
flex items-center justify-center rounded-r-md"
                      >
                        +
                      </button>
                    </div>
                  </div>

                  <div className="text-right mt-4 sm:mt-0 sm:ml-4">
                    <p className="text-white
font-medium">${(item.subtotal || 0).toFixed(2)}</p>
                  </div>
                </div>
              </div>
            ))}
          </div>

          <div className="mt-4">
            <Link to="/products" className="inline-flex items-center
text-white/70 hover:text-white">
              <FaArrowLeft className="mr-2" /> Continue Shopping
            </Link>
          </div>
        </div>

        {/* Order Summary */}
        <div className="bg-black/70 backdrop-blur-md p-6 rounded-lg
border border-white/20 h-fit">
          <h2 className="text-xl font-bold text-white mb-4">Order
Summary</h2>

          <div className="space-y-2 mb-4">
            <div className="flex justify-between text-white/70">
              <span>Subtotal</span>
              <span>${(cart.total || 0).toFixed(2)}</span>
            </div>
            <div className="flex justify-between text-white/70">
              <span>Shipping</span>
              <span>Calculated at checkout</span>
            </div>
            <div className="flex justify-between text-white/70">
              <span>Tax</span>
              <span>Calculated at checkout</span>
```

```
              </div>
            </div>

            <div className="border-t border-white/20 pt-4 mb-6">
              <div className="flex justify-between text-white
font-bold">
                <span>Total</span>
                <span>${(cart.total || 0).toFixed(2)}</span>
              </div>
            </div>

            <Link
              to="/checkout"
              className="block w-full bg-purple-600 hover:bg-purple-700
text-white py-3 rounded-md text-center font-medium"
            >
              Proceed to Checkout
            </Link>

            <Link
              to="/shop"
              className="block w-full bg-transparent border
border-white/30 text-white py-3 rounded-md text-center font-medium mt-4
hover:bg-white/10"
            >
              Continue Shopping
            </Link>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Cart;
```

4. Product List

This React component, ProductList, renders a musical instrument product listing page with filtering and shopping cart functionality. It fetches products, categories, and brands from an API and allows users to:

- Search products by keyword
- Filter by category, price range, and brand
- Add products to a cart using an API call

The component uses useState and useEffect to manage and fetch data dynamically when filters change. It features a styled UI with a responsive grid layout and includes loading and error handling states. The toast library is used to show success or error notifications when adding items to the cart.

```
import React, { useState, useEffect } from 'react';
import api from '../api/axios';
import Navbar from '../components/Navbar';
import { toast } from 'react-toastify';

const ProductList = () => {
  const [products, setProducts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [categories, setCategories] = useState([]);
  const [selectedCategory, setSelectedCategory] = useState('all');
  const [searchQuery, setSearchQuery] = useState('');
  const [priceRange, setPriceRange] = useState({ min: 0, max: 10000 });
  const [brands, setBrands] = useState([]);
  const [selectedBrands, setSelectedBrands] = useState([]);

  useEffect(() => {
    fetchCategories();
    fetchBrands();
    fetchProducts();
  }, [selectedCategory, searchQuery, priceRange, selectedBrands]);

  const fetchCategories = async () => {
    try {
      const response = await api.get('/products/categories');
      setCategories(response.data.categories);
    } catch (err) {
      console.error('Error fetching categories:', err);
    }
  };

  const fetchBrands = async () => {
    try {
```

```
      const response = await api.get('/products/brands');
      setBrands(response.data.brands);
    } catch (err) {
      console.error('Error fetching brands:', err);
    }
  };

  const fetchProducts = async () => {
    try {
      setLoading(true);
      let url = '/products';

      // Build query parameters
      const params = new URLSearchParams();
      if (selectedCategory !== 'all') params.append('category',
selectedCategory);
      if (searchQuery) params.append('search', searchQuery);
      params.append('minPrice', priceRange.min);
      params.append('maxPrice', priceRange.max);
      if (selectedBrands.length > 0) {
        selectedBrands.forEach(brand => params.append('brands',
brand));
      }

      const response = await api.get(`${url}?${params.toString()}`);
      setProducts(response.data.products);
      setError(null);
    } catch (err) {
      setError('Failed to fetch products. Please try again later.');
      console.error('Error fetching products:', err);
    } finally {
      setLoading(false);
    }
  };

  const handleCategoryChange = (category) => {
    setSelectedCategory(category);
  };

  const handleSearchChange = (e) => {
    setSearchQuery(e.target.value);
  };
```

```
  const handlePriceChange = (type, value) => {
    setPriceRange(prev => ({
      ...prev,
      [type]: value
    }));
  };

  const handleBrandToggle = (brand) => {
    setSelectedBrands(prev =>
      prev.includes(brand)
        ? prev.filter(b => b !== brand)
        : [...prev, brand]
    );
  };

// Then update the addToCart function
const addToCart = async (productId) => {
  try {
    console.log('Adding to cart:', productId);
    await api.post('/cart/add', { productId, quantity: 1 });
    toast.success('Product added to cart successfully!');
  } catch (err) {
    console.error('Error adding to cart:', err);
    toast.error(err.response?.data?.message || 'Failed to add product
to cart');
  }
};


  if (loading && products.length === 0) {
    return (
      <div className="flex flex-col min-h-screen bg-cover bg-center
text-white" style={{ backgroundImage: "url('piano-background.jpg')" }}>
        <Navbar />
        <div className="min-h-screen flex justify-center items-center">
          <p className="text-white text-xl">Loading products...</p>
        </div>
      </div>
    );
  }

  return (
```

```
    <div className="flex flex-col min-h-screen bg-cover bg-center
text-white" style={{ backgroundImage: "url('piano-background.jpg')" }}>
      <Navbar />
      <div className="flex-1 p-8">
        <div className="container mx-auto">
          <h1 className="text-5xl font-bold text-white mb-8
leading-tight">Musical Instruments</h1>

          {/* Search and Filters */}
          <div className="bg-black/70 backdrop-blur-md p-6 rounded-2xl
shadow-2xl mb-8 border border-white/30 text-white">
            <div className="mb-4">
              <input
                type="text"
                placeholder="Search products..."
                value={searchQuery}
                onChange={handleSearchChange}
                className="w-full p-3 border rounded-md text-base
bg-white/10 text-white placeholder-white/60 border-white/30
focus:outline-none focus:ring-2 focus:ring-blue-500"
              />
            </div>

            <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
              {/* Categories */}
              <div>
                <h3 className="text-white font-semibold
mb-2">Categories</h3>
                <div className="space-y-2">
                  <button
                    onClick={() => handleCategoryChange('all')}
                    className={`px-4 py-2 rounded-md text-sm
font-medium ${
                      selectedCategory === 'all'
                        ? 'bg-blue-600 hover:bg-blue-700 text-white'
                        : 'bg-white/10 hover:bg-white/20 border
border-white/30 text-white backdrop-blur-md'
                    }`}
                  >
                    All
                  </button>
                  {categories.map(category => (
                    <button
```

```
                    key={category}
                    onClick={() => handleCategoryChange(category)}
                    className={`px-4 py-2 rounded-md text-sm
font-medium ${
                      selectedCategory === category
                        ? 'bg-blue-600 hover:bg-blue-700 text-white'
                        : 'bg-white/10 hover:bg-white/20 border
border-white/30 text-white backdrop-blur-md'
                    }`}
                  >
                    {category}
                  </button>
                ))}
              </div>
            </div>

            {/* Price Range */}
            <div>
              <h3 className="text-white font-semibold mb-2">Price
Range</h3>
              <div className="grid grid-cols-2 gap-2">
                <div>
                  <label className="block mb-2 font-medium">Min
($)</label>
                  <input
                    type="number"
                    min="0"
                    value={priceRange.min}
                    onChange={(e) => handlePriceChange('min',
e.target.value)}
                    className="w-full p-3 border rounded-md text-base
bg-white/10 text-white placeholder-white/60 border-white/30"
                  />
                </div>
                <div>
                  <label className="block mb-2 font-medium">Max
($)</label>
                  <input
                    type="number"
                    min="0"
                    value={priceRange.max}
                    onChange={(e) => handlePriceChange('max',
e.target.value)}
```

```
                    className="w-full p-3 border rounded-md text-base
bg-white/10 text-white placeholder-white/60 border-white/30"
                  />
                </div>
              </div>
            </div>

            {/* Brands */}
            <div>
              <h3 className="text-white font-semibold
mb-2">Brands</h3>
              <div className="space-y-2 max-h-40 overflow-y-auto">
                {brands.map(brand => (
                  <div key={brand} className="flex items-center">
                    <input
                      type="checkbox"
                      id={`brand-${brand}`}
                      checked={selectedBrands.includes(brand)}
                      onChange={() => handleBrandToggle(brand)}
                      className="rounded text-blue-600
focus:ring-blue-500 bg-white/10 border-white/30"
                    />
                    <label htmlFor={`brand-${brand}`} className="ml-2
text-white">
                      {brand}
                    </label>
                  </div>
                ))}
              </div>
            </div>
          </div>
        </div>

        {/* Product Grid */}
        {error ? (
          <div className="bg-red-500 text-white p-4 rounded-md mb-6">
            {error}
          </div>
        ) : products.length === 0 ? (
          <div className="bg-black/70 backdrop-blur-md p-6
rounded-2xl shadow-2xl mb-8 border border-white/30 text-white
text-center">
            No products found matching your criteria.
```

```
            </div>
          ) : (
            <div className="grid grid-cols-1 sm:grid-cols-2
md:grid-cols-3 lg:grid-cols-4 gap-6">
              {products.map(product => (
                <div key={product._id} className="bg-black/70
backdrop-blur-md rounded-2xl shadow-2xl overflow-hidden border
border-white/30 transition-transform hover:scale-105">
                  <a href={`/products/${product._id}`}>
                    <img
                      src={`http://127.0.0.1:5000${product.images[0]}`}
                      alt={product.name}
                      className="w-full h-48 object-cover"
                    />
                    <div className="p-4">
                      <h3 className="text-white font-semibold
text-lg">{product.name}</h3>
                      <p className="text-white/70
text-sm">{product.brand}</p>
                      <p className="text-blue-400 font-bold
mt-2">${product.price.toFixed(2)}</p>
                    </div>
                  </a>
                  <div className="px-4 pb-4">
                    <button
                      onClick={() => addToCart(product._id)}
                      className="w-full bg-blue-600 hover:bg-blue-700
text-white font-bold py-2 px-4 rounded"
                    >
                      Add to Cart
                    </button>
                  </div>
                </div>
              ))}
            </div>
          )}
        </div>
      </div>
      <footer className="bg-black bg-opacity-50 text-center p-4">
        <p>© 2025 Resonance. All Rights Reserved</p>
      </footer>
    </div>
  );
```

```
};

export default ProductList;
```

5. Admin Dashboard

- This AdminDashboard React component renders a multi-tab administrative interface for managing different aspects of a platform, such as users, studios, bookings, products, and rewards.
- It uses React's useState to track the active tab and conditionally renders the corresponding management component.
- The UI is styled with Tailwind CSS and enhanced with icons from react-icons, providing a clean, interactive admin panel with persistent navigation via the Navbar.

```
import React, { useState } from 'react';
import UserManagement from '../../components/admin/UserManagement';
import { FaUsers, FaClipboardList, FaMusic, FaShoppingBag,
FaCalendarAlt, FaGift } from 'react-icons/fa';
import Navbar from '../../components/Navbar';
import StudioManagement from '../../components/admin/StudioManagement';
import ProductManagement from
'../../components/admin/ProductManagement';
import BookingManagement from
'../../components/admin/BookingManagement';
import RewardManagement from '../../components/admin/RewardManagement';

const AdminDashboard = () => {
  const [activeTab, setActiveTab] = useState('users');
  console.log('AdminDashboard rendered');

  return (
    <div className="app-background min-h-screen">
      <Navbar />
      <div className="container mx-auto px-4 py-8">
        <h1 className="text-3xl font-bold text-white mb-6">Admin
Dashboard</h1>

        <div className="bg-black/70 backdrop-blur-md p-6 rounded-2xl
shadow-2xl border border-white/30 text-white">
          <div className="flex mb-6 border-b border-white/20
overflow-x-auto">
            <button
```

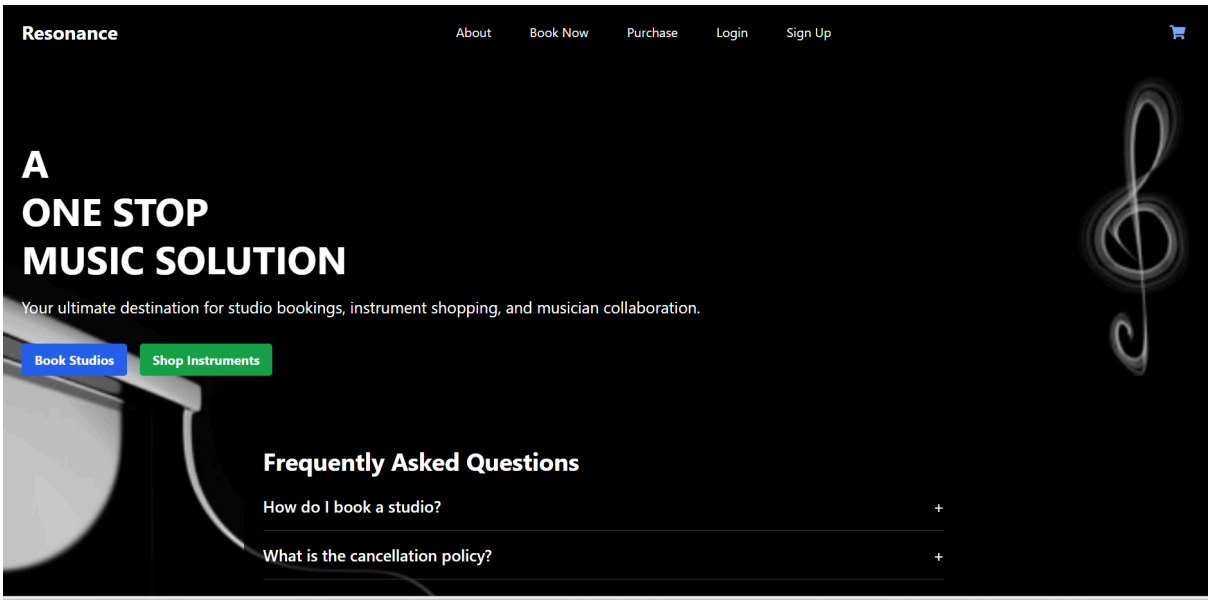
```
              className={`flex items-center px-4 py-3 text-base
transition-colors hover:bg-white/10 ${
                activeTab === 'users' ? 'border-b-2 border-emerald-500
font-medium text-emerald-400' : 'text-white'
              }`}
              onClick={() => setActiveTab('users')}
            >
              <span className="mr-2 flex items-center"><FaUsers
/></span>
              <span>User Management</span>
            </button>
            <button
              className={`flex items-center px-4 py-3 text-base
transition-colors hover:bg-white/10 ${
                activeTab === 'studios' ? 'border-b-2
border-emerald-500 font-medium text-emerald-400' : 'text-white'
              }`}
              onClick={() => setActiveTab('studios')}
            >
              <span className="mr-2 flex items-center"><FaMusic
/></span>
              <span>Studio Management</span>
            </button>
            <button
              className={`flex items-center px-4 py-3 text-base
transition-colors hover:bg-white/10 ${
                activeTab === 'bookings' ? 'border-b-2
border-emerald-500 font-medium text-emerald-400' : 'text-white'
              }`}
              onClick={() => setActiveTab('bookings')}
            >
              <span className="mr-2 flex items-center"><FaCalendarAlt
/></span>
              <span>Booking Management</span>
            </button>
            <button
              className={`flex items-center px-4 py-3 text-base
transition-colors hover:bg-white/10 ${
                activeTab === 'products' ? 'border-b-2
border-emerald-500 font-medium text-emerald-400' : 'text-white'
              }`}
              onClick={() => setActiveTab('products')}
            >
```

```
              <span className="mr-2 flex items-center"><FaShoppingBag
/></span>
              <span>Product Management</span>
            </button>
            <button
              className={`flex items-center px-4 py-3 text-base
transition-colors hover:bg-white/10 ${
                activeTab === 'rewards' ? 'border-b-2
border-emerald-500 font-medium text-emerald-400' : 'text-white'
              }`}
              onClick={() => setActiveTab('rewards')}
            >
              <span className="mr-2 flex items-center"><FaGift
/></span>
              <span>Reward System</span>
            </button>

          </div>

          <div>
            {activeTab === 'users' && (
              <div className="rounded-md">
                <UserManagement />
              </div>
            )}

            {activeTab === 'studios' && (
              <div className="rounded-md">
                <StudioManagement />
              </div>
            )}

            {activeTab === 'bookings' && (
              <div className="rounded-md">
                <BookingManagement />
              </div>
            )}

            {activeTab === 'products' && (
              <div className="rounded-md">
                <ProductManagement />
              </div>
            )}
```

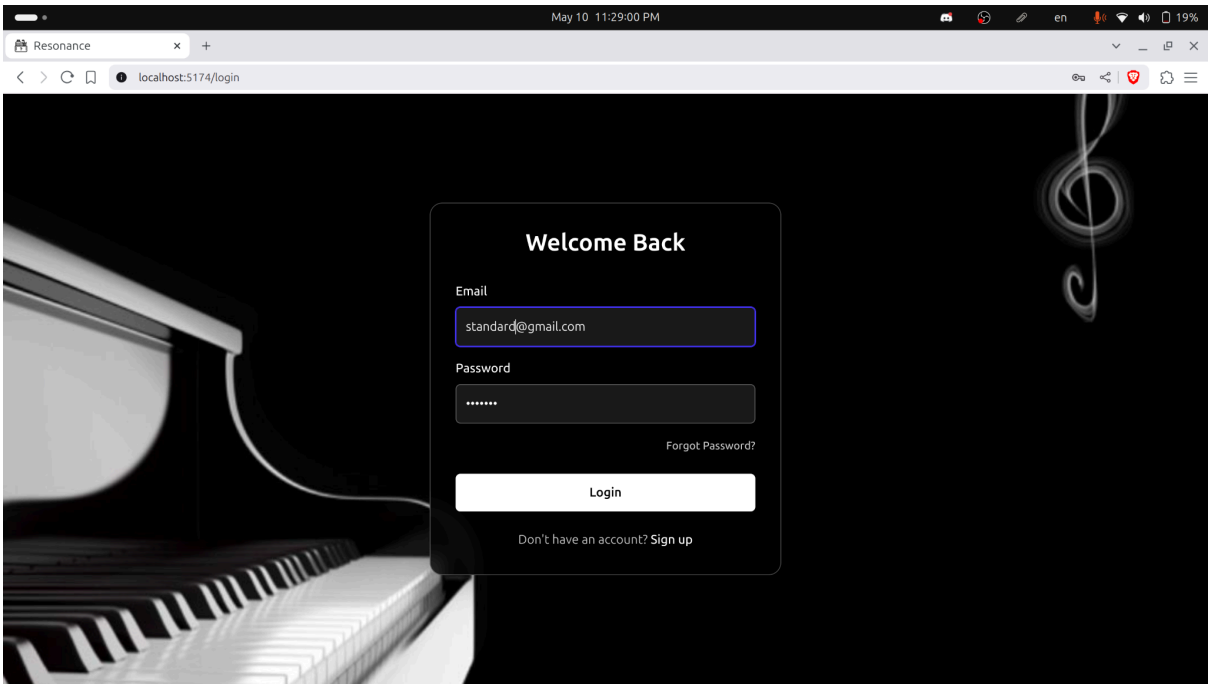
```
            {activeTab === 'rewards' && (
              <div className="rounded-md">
                <RewardManagement />
              </div>
            )}


          </div>
        </div>
      </div>
    </div>
  );
};

export default AdminDashboard;
```

7. User Manual

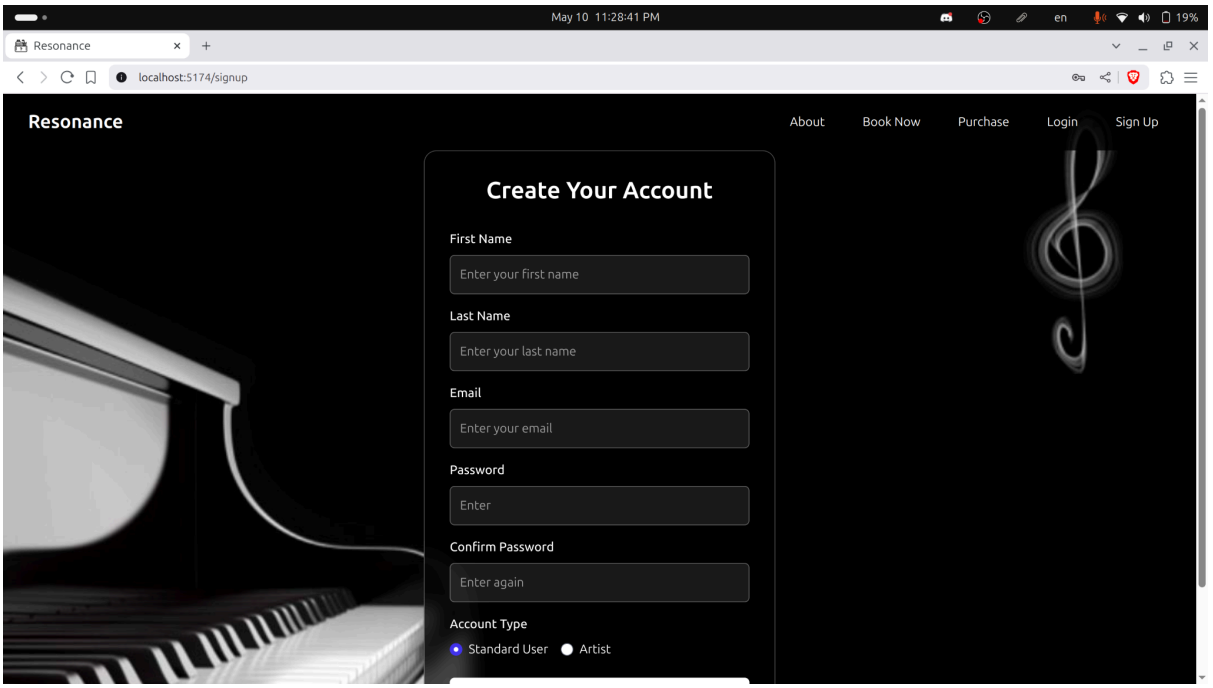
1. At the start of the website, users can see a home page with buttons to log in and sign up.



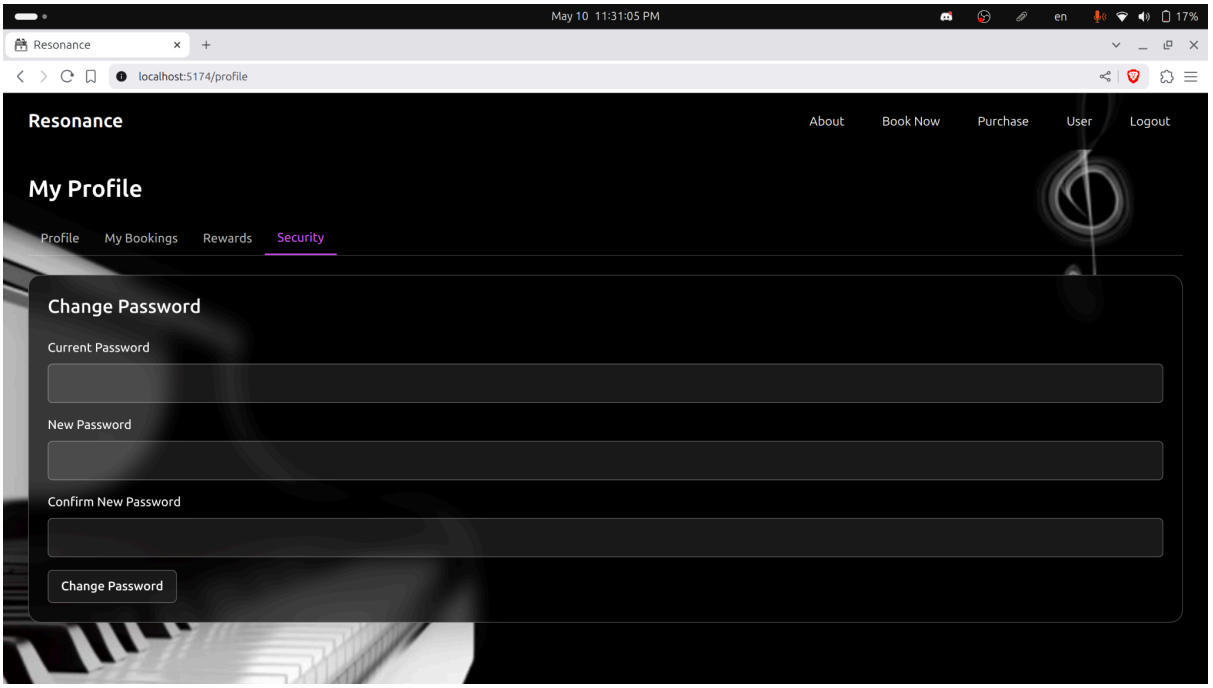
2. After clicking the login button, the user can see this page for logging in, where he/she can fill in the credentials for entering the website.



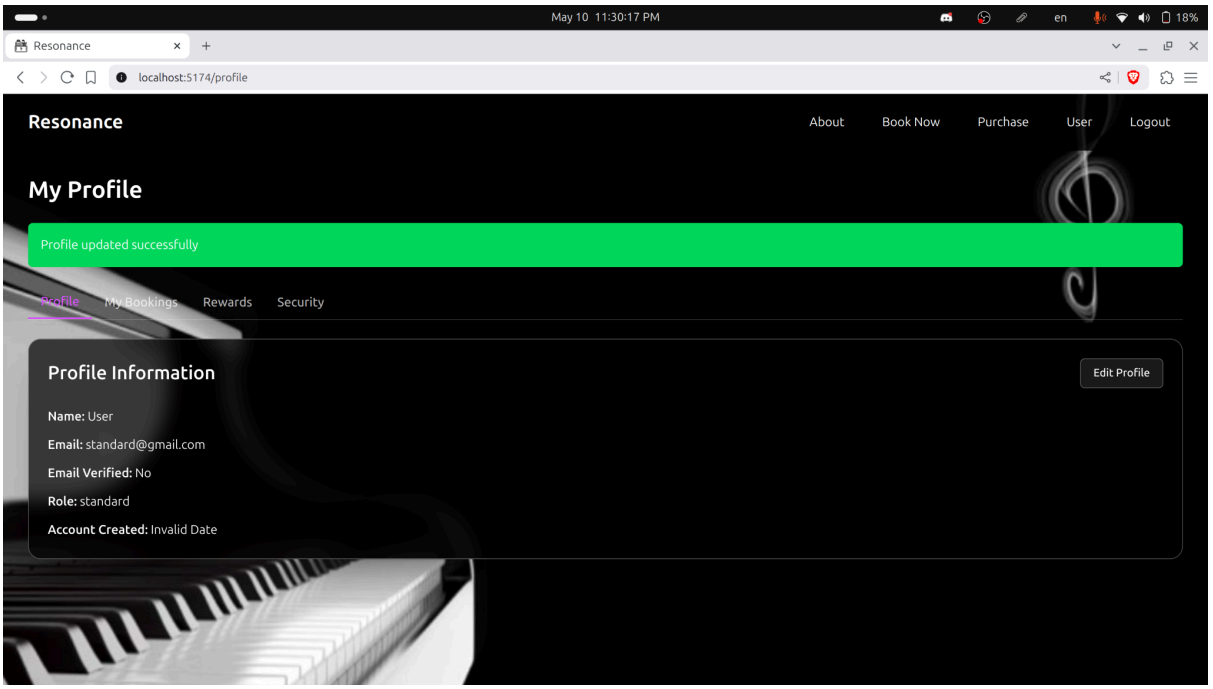
3. In the login page, there is a signup button as well. If the user doesn't have an account, the user can fill up the necessary information to create an account. Otherwise, the user can create an account by clicking sign-up button from the navigation bar of the home page.



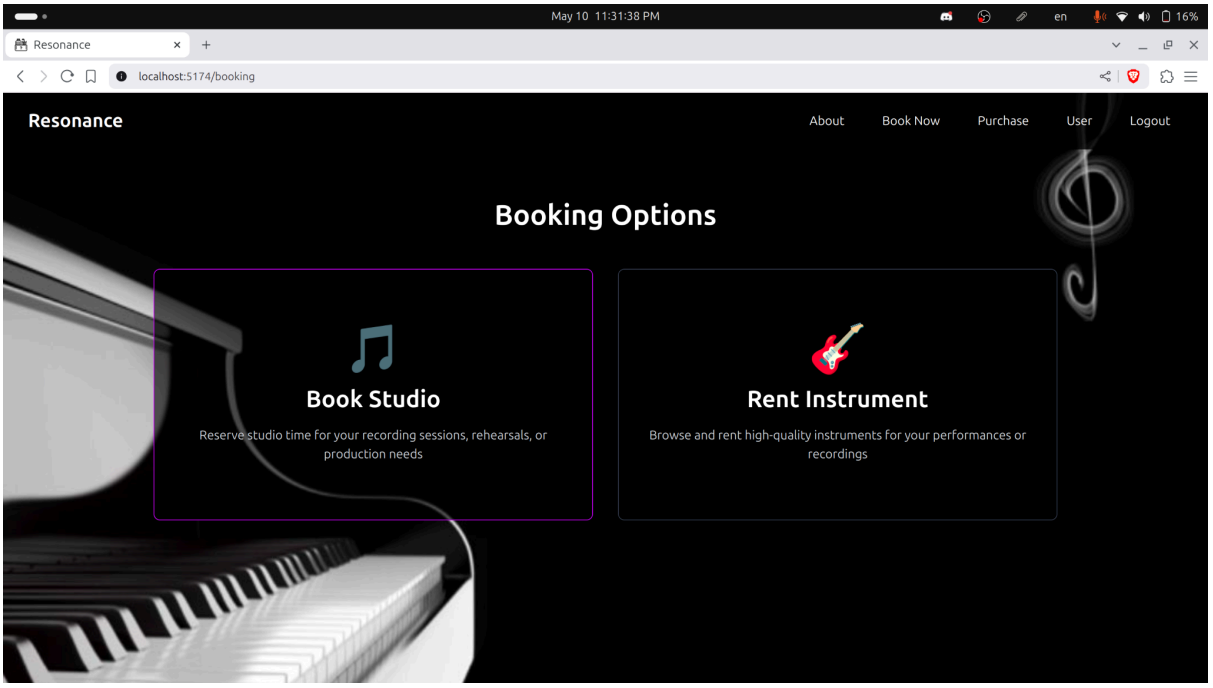
4. After logging in, the user can change the password from the Security tab.



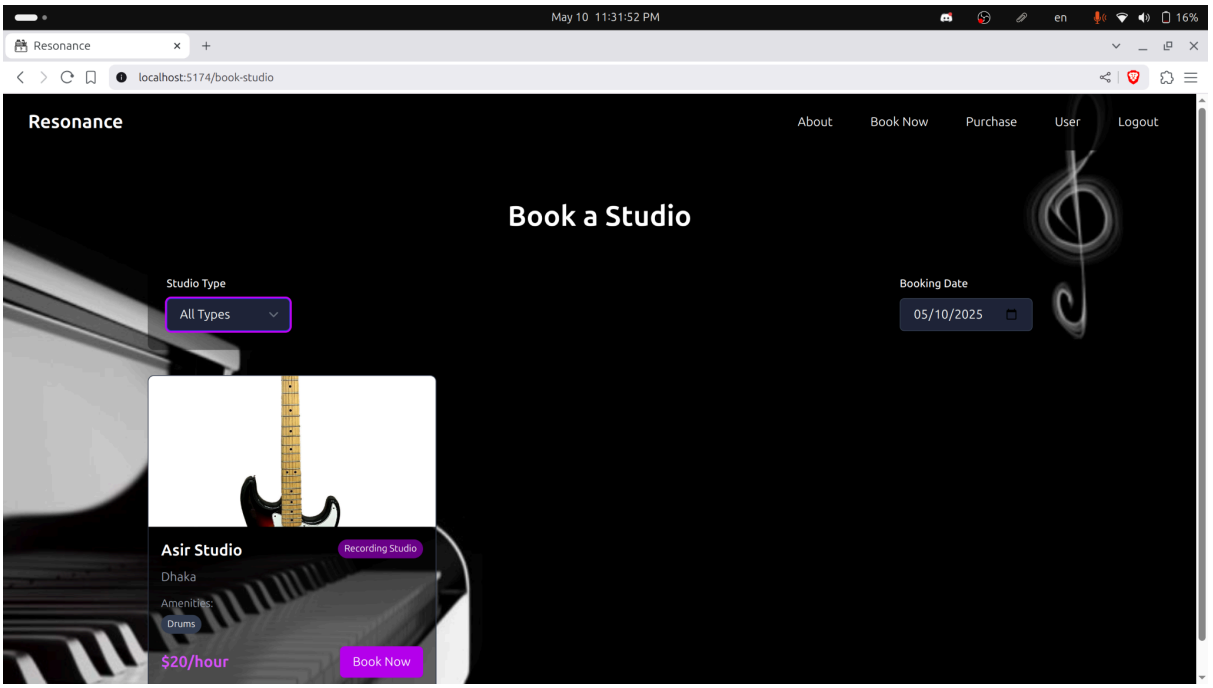
5. In the profile tab in the user dashboard, the user can see the profile information and edit the information if necessary.



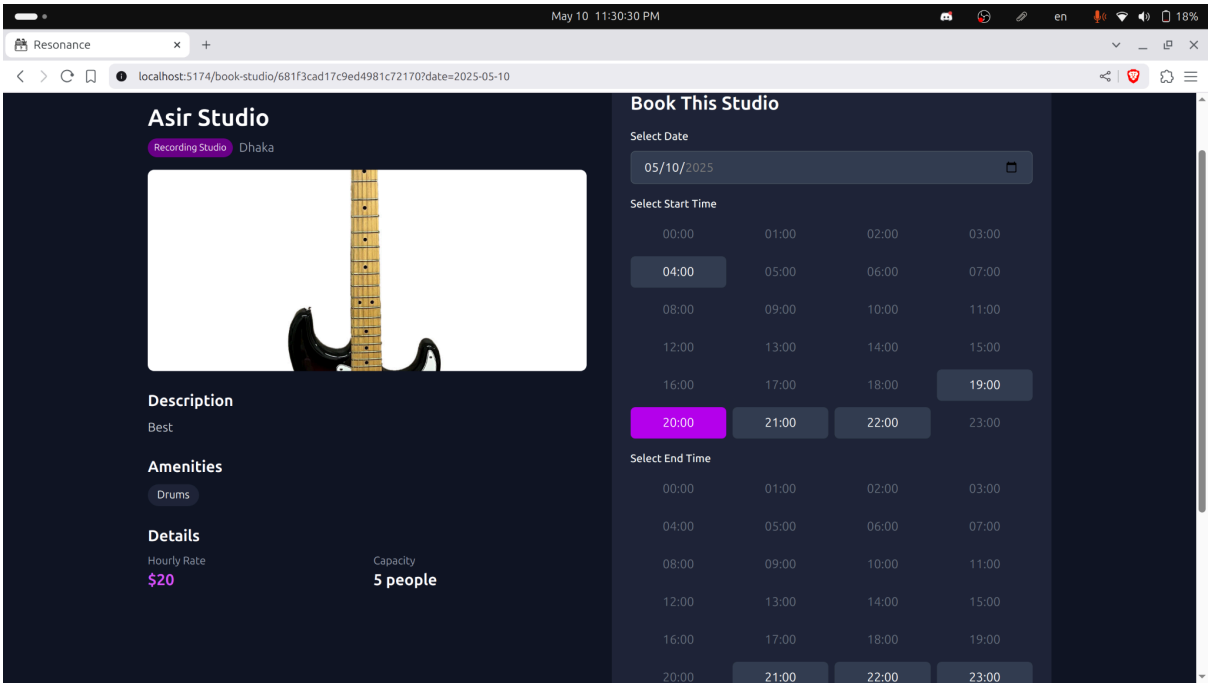
6. From the Book Now button of the navigation bar, users can see the booking options for booking studio or practice pads and renting instruments.



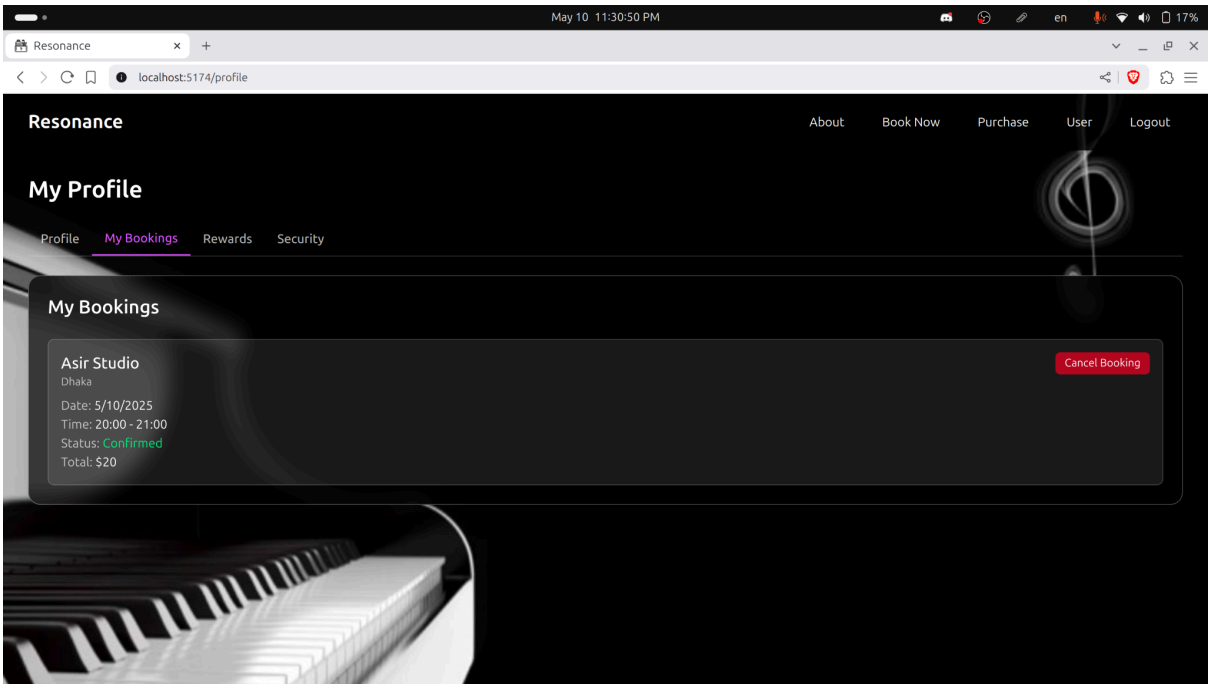
7. After clicking the Book studio option, the user can see the studio names and options for choosing the studio location.



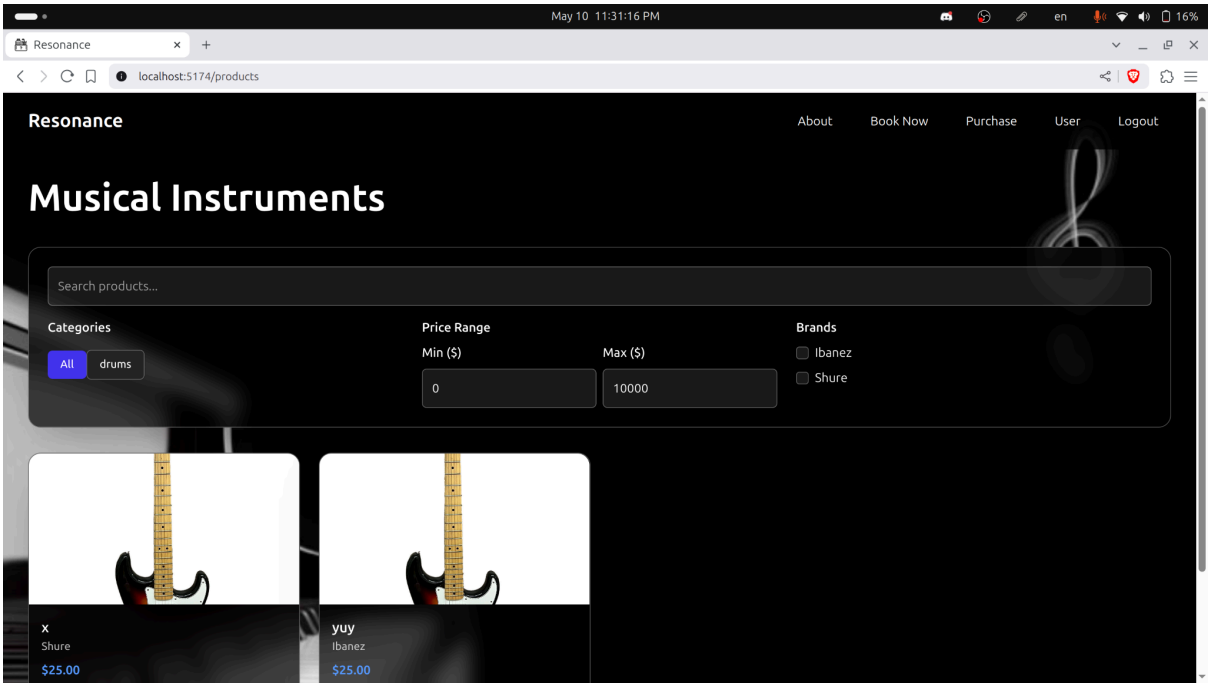
8. After choosing the studio, the user can select a time for their booking. Then the user can confirm their booking.



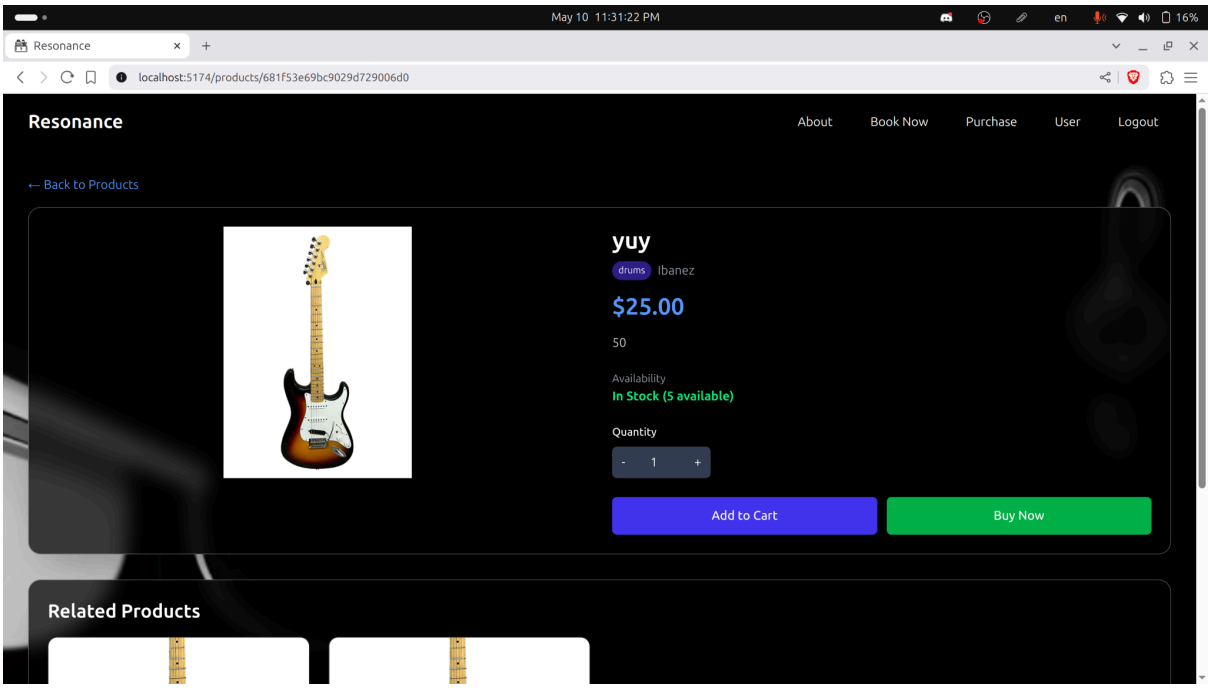
9. Then, the booking will show up in the dashboard. User can see all their bookings in the My Bookings tab on the dashboard.



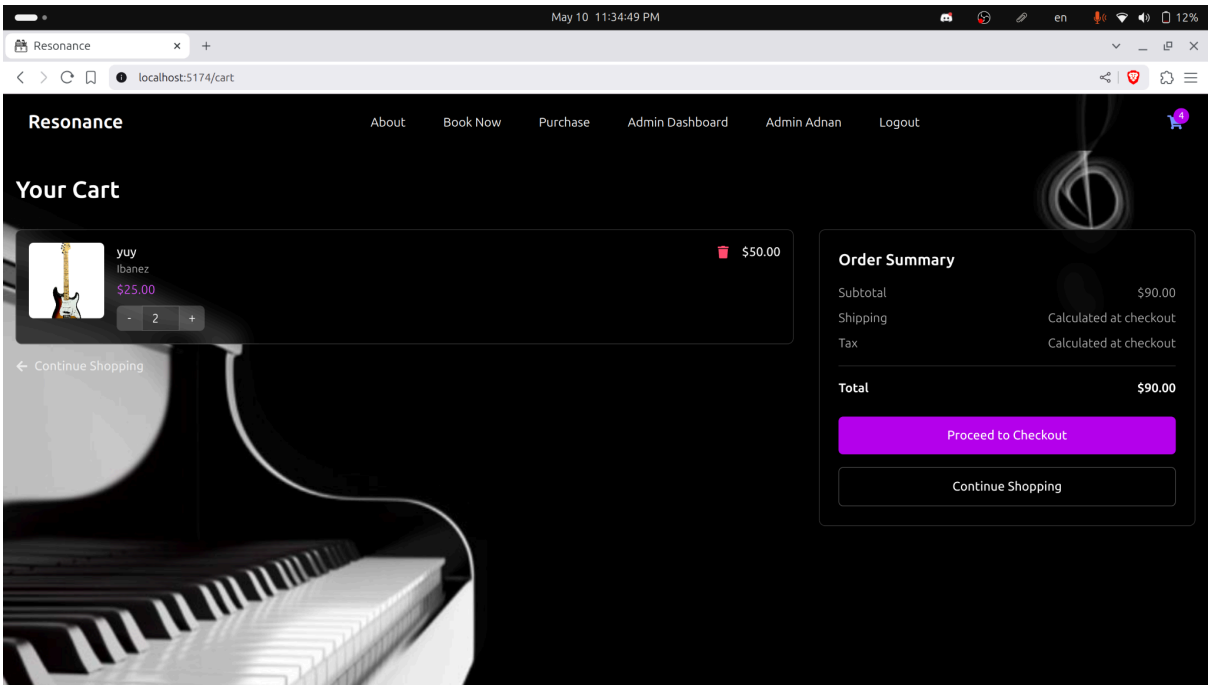
10. From the dashboard Purchase button, the user can see the e-commerce page for buying instruments. The products will be categorized, and users can use the filter to search for products.



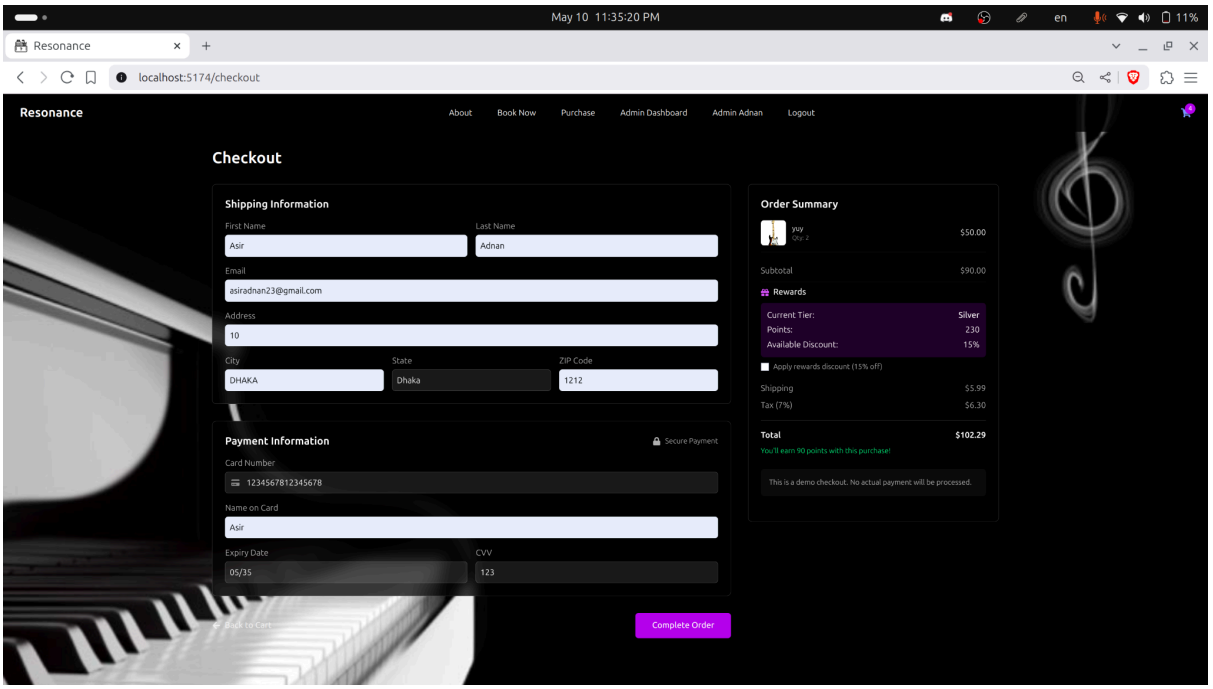
11. After clicking a product, an individual product page will show up with a detailed explanation.



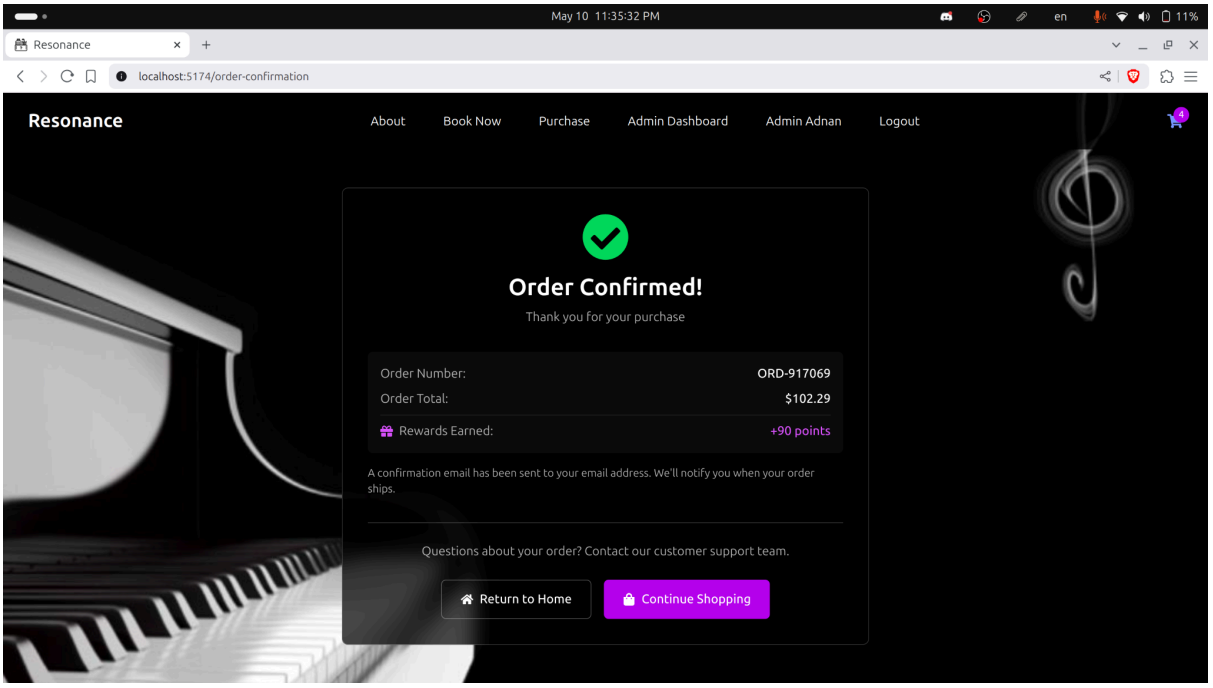
12. After clicking the add cart button from the product page, the user can see the cart.



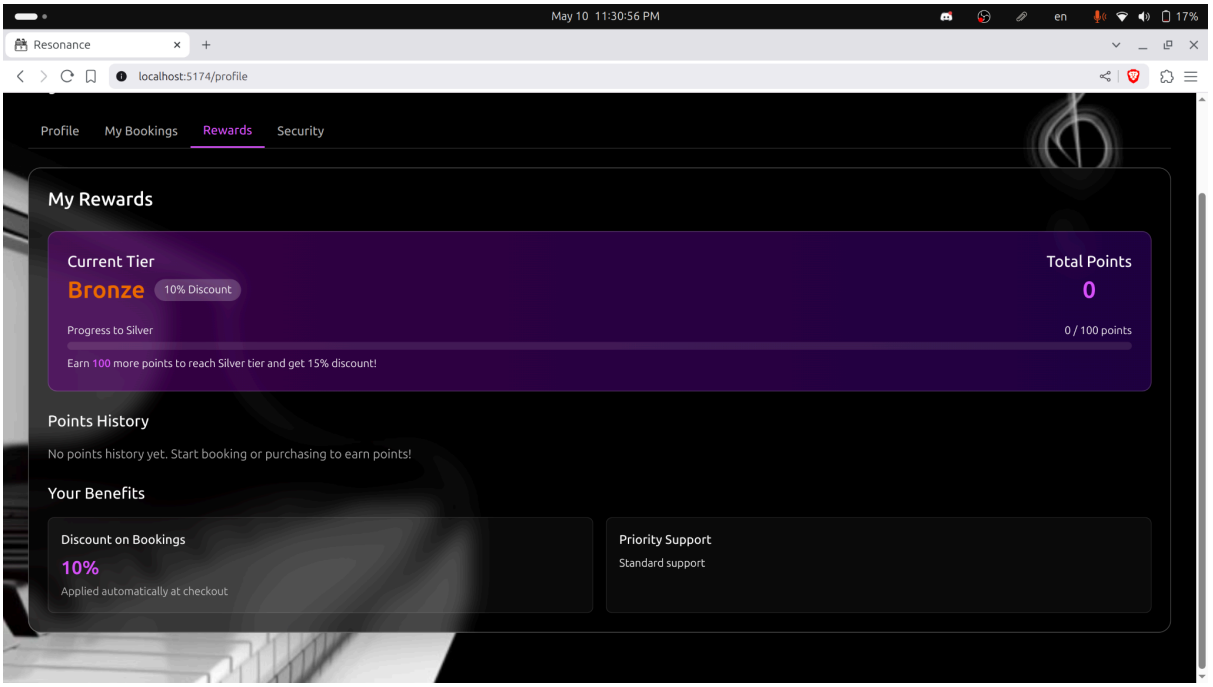
13. From the cart, the user can see the payment gateway by clicking the Proceed to payment button. On the payment page, the order summary will show up.



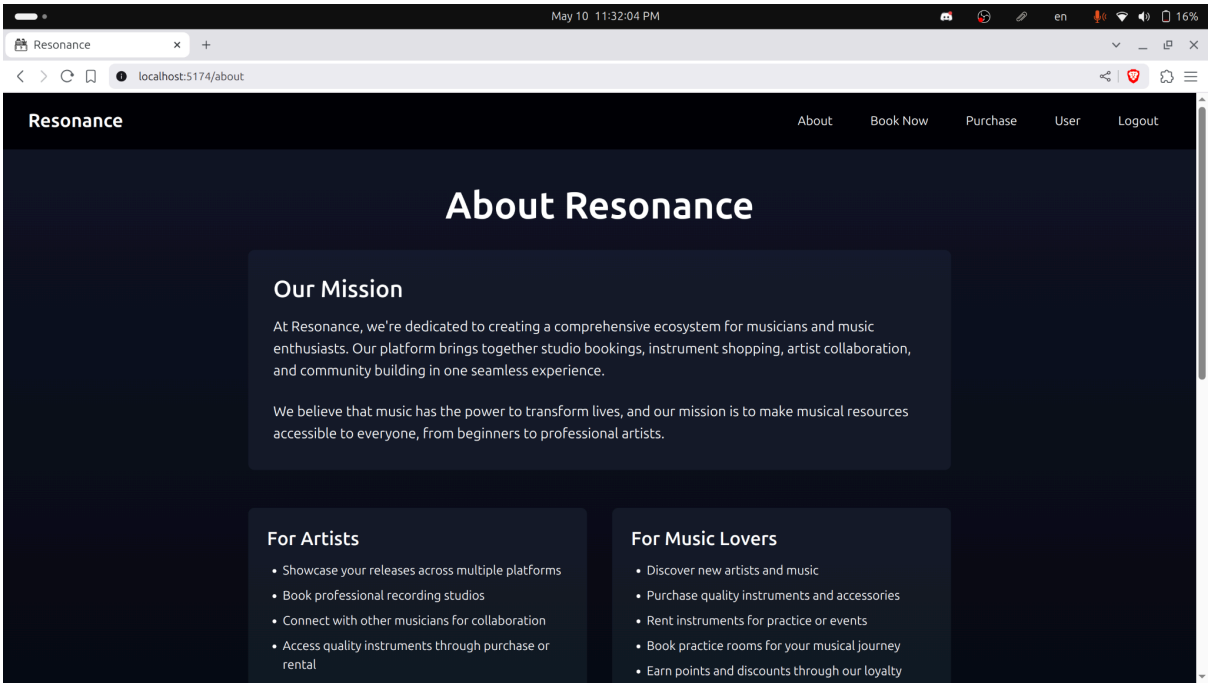
14. In the payment gateway, the user can complete the order, and then an order confirmation will be shown to the user.



15. User rewards can be seen from the Rewards tab of the dashboard. After taking services from the website, users will be rewarded with points, which will give them some special benefits according to their points.

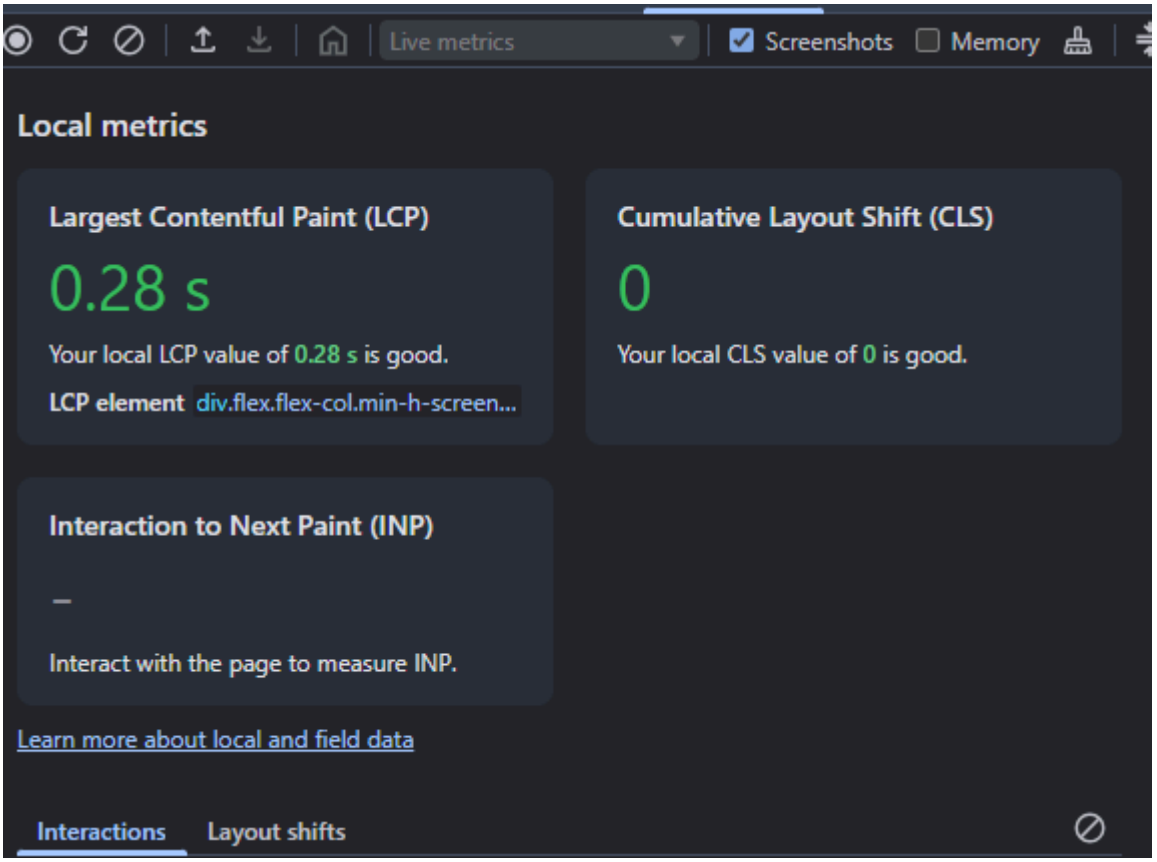


16. Users can go to the About page by clicking the About button from the navigation bar. On that page, users can see details and inspiration behind developing the website.

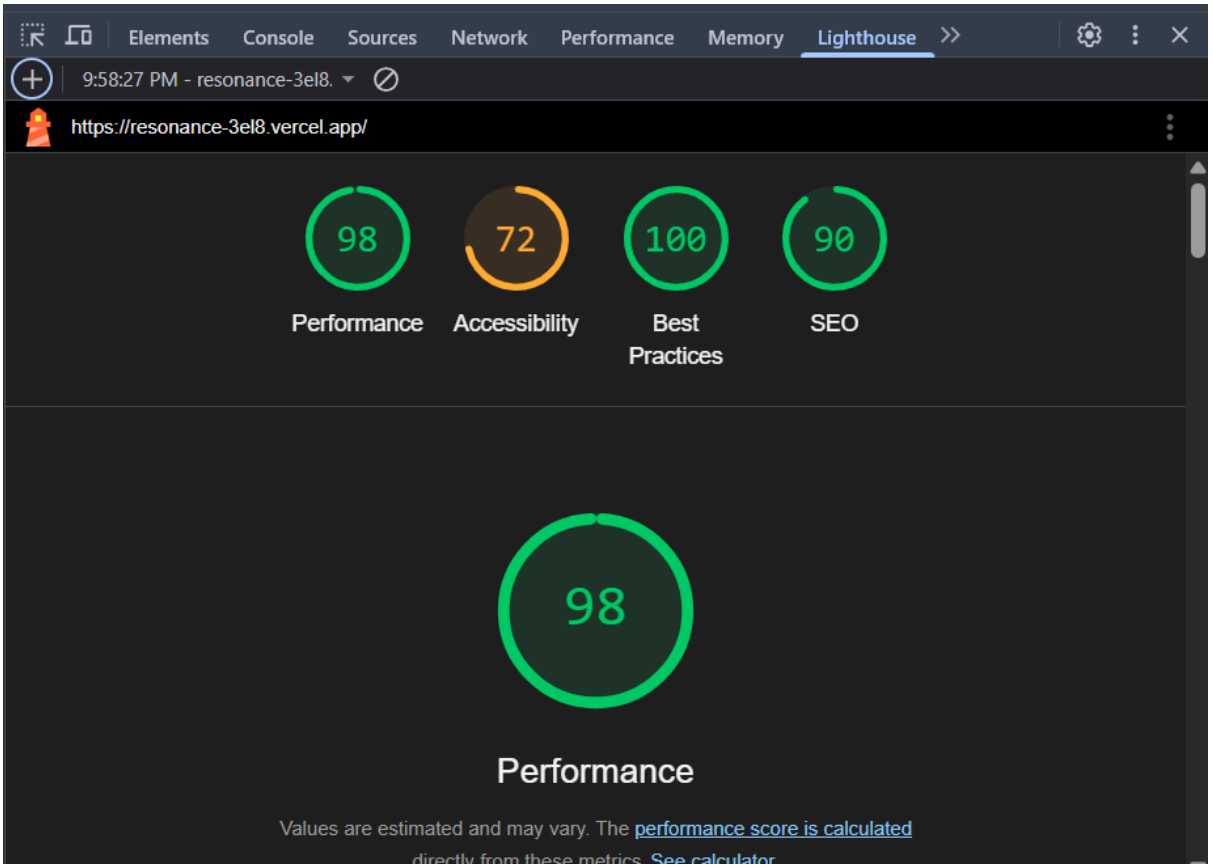


8. Performance and Network Analysis

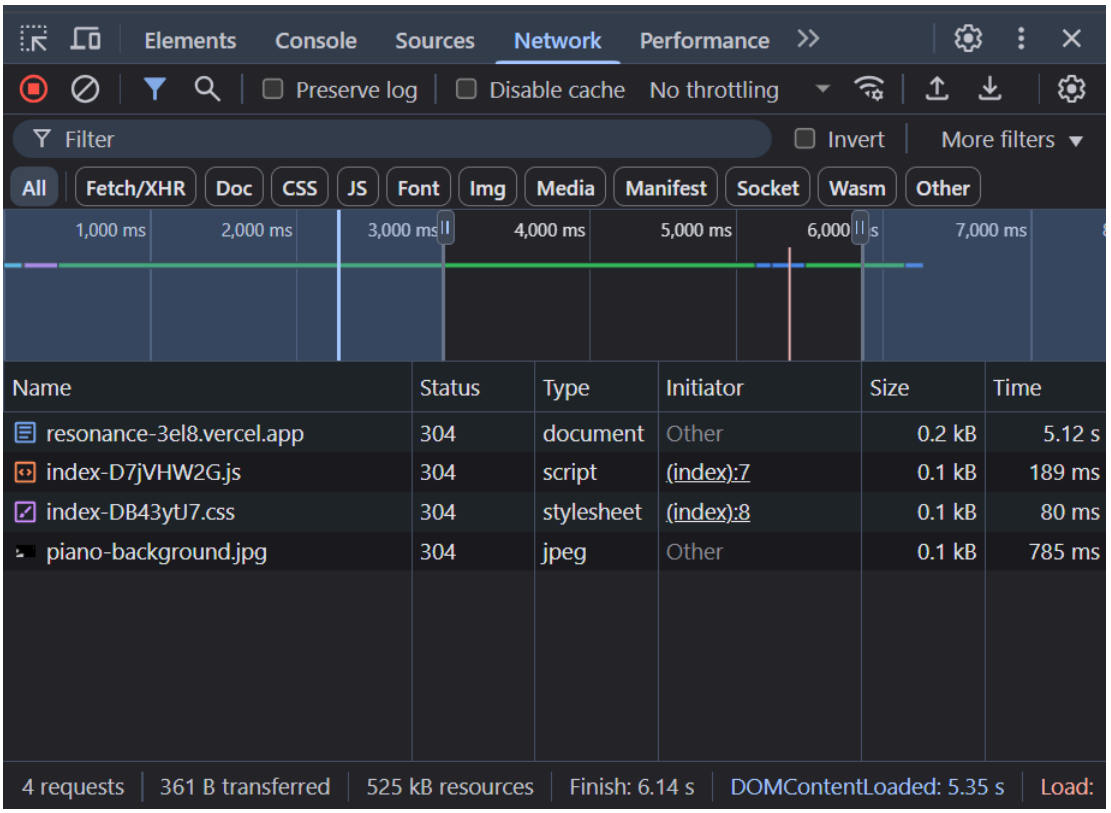
Performance:



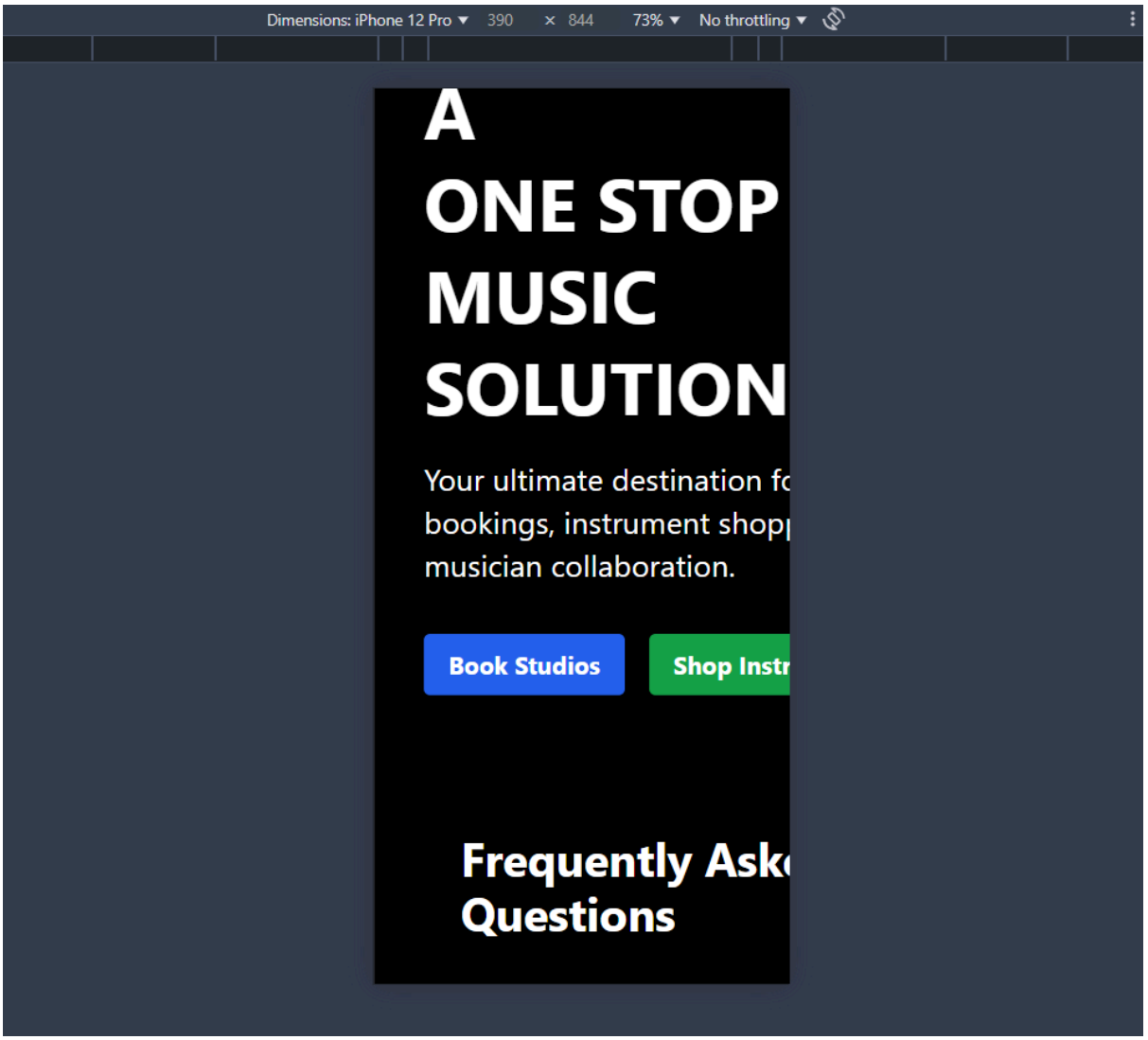
Lighthouse:



NetworkAnalysis:



System:



9. Github Repo [Public] Link

https://github.com/arpitasaha009/RESONANCE

10. Link of Deployed Project

https://resonance-3el8.vercel.app/

11. Individual Contribution

Group member - 01	
Name: Maisha Fairooz	**Student ID: 21201031**
Functional Requirements which are developed by this member:	
1. Users must receive a booking confirmation and a digital receipt via email (sent through Nodemailer).	
2. The system must include an FAQ section where users can find information.	
3. The website must include an About Us page with links to social media platforms like Instagram and YouTube.	
4. Admin can manage rentals.	
5. Users must be able to review products.	

Group member - 02	
Name: Yamin Adnan	**Student ID: 21201356**
Functional Requirements which are developed by this member:	
1. Admins must be able to manage inventory, including adding and editing products for the store.	
2. Users must be able to book studio/practice rooms based on location, date, time, and room size.	
3. The system must provide an availability check for studios and practice rooms.	
4. Users must be able to cancel bookings, following a basic cancellation policy (e.g., "Cancel 24 hours before booking").	
5. The platform must support an instrument rental option, allowing users to rent musical instruments for practice pad sessions.	
6. Users will be able to accumulate points based on their activities (Booking practice or recording sessions, purchasing or renting musical instruments or accessories) and will be allocated to different categories (Bronze, Silver, Gold, Platinum).	
7. Users will get discounts based on their points categories (eg: 5% for Bronze, 10% for Silver, 15% for Gold, 20% for Platinum).	

Group member - 03	
Name: Aparup Chowdhury	**Student ID: 22101229**
Functional Requirements which are developed by this member:	
1. Artist profile users can edit personal details (name, phone, address, releases, etc.)	
2. Users (both Standard and Artist profile users) can see their bookings, place order details, and earn points on their dashboard.	
3. Artists can showcase their releases (several platforms might be integrated eg: YouTube, Spotify) on their artist profile.	
4. The system will allow artists to create and share posts in the Collaboration Hub to express ideas and seek collaboration.	
5. The system must allow artists to comment on posts to engage in discussions and connect with other artists.	
6. Users will get a pop-up notification for their respective tasks and receive a booking confirmation and a digital receipt via email (sent through Nodemailer).	
7. Users will be able to search for products and apply filters such as price range, instrument type, and brand.	

Group member - 04	
Name: Arpita Saha	**Student ID: 22101460**
Functional Requirements which are developed by this member:	
1. The system must allow users to sign up, log in, and reset their passwords via email.	
2. The system must allow users to browse product listings categorized by type (guitars, drums, etc.).	
3. Each product must have a detailed page displaying images, descriptions, and other relevant information.	
4. Users must be able to add products to a shopping cart.	
5. Users (Standard profile users) can edit personal details (name, phone, address, releases, etc.).	

Group member - 05	
Name: Mustafiz Ahsan	**Student ID: 22101486**
Functional Requirements which are developed by this member:	
1. Admin can manage and delete user profiles.	
2. Admin can cancel bookings and orders.	
3. Admin can control and customize the point-based reward system (Configure point values for different activities, modify discounts and benefits for each tier, adjust point expiry rules and redemption rates).	
4. Admin can manage studios.	
5. Admin can reward points.	
6. The system must provide a checkout page with a payment gateway for purchasing items.	

12. References