A

Practical Activity Report Submitted for

UI & UX SPECIALIST-(UCS542)

END-Semester Lab Evaluation

DevTinder

Submitted to Ms. KANUPRIYA

BE Third Year

Submitted by-

Lovish Goyal(102315165)

ECED Department

TIET, Patiala

August-December 2025

Project Link: https://devtinder-web-three.vercel.app/

Index

1. Introduction

DevTinder is a full-stack web application designed to help developers connect, collaborate, and network. Similar to modern matching platforms, DevTinder enables users to create profiles, view other developers in a feed, send connection requests, accept or reject requests, chat in real-time, and manage their professional presence.

---

2. Problem Statement

Developers often struggle to find meaningful technical connections or collaborators. Existing platforms mix general-purpose networking, making it harder to discover relevant people. DevTinder solves this by offering an exclusive developer-matching platform with authentication, profile management, connection requests, and real-time chat.

---

3. Specific Requirements

Functional Requirements

- User authentication (Signup/Login using JWT Cookies)

- Profile creation & editing with a Cloudinary image upload

- Feed page displaying other developers

- Connection request system (Send/Accept/Reject)

- Protected routes using token validation

- Real-time chat using Socket.io

- Redux-powered global state management

- Pagination for feed API

Non-functional Requirements

- Responsive UI using Tailwind + DaisyUI

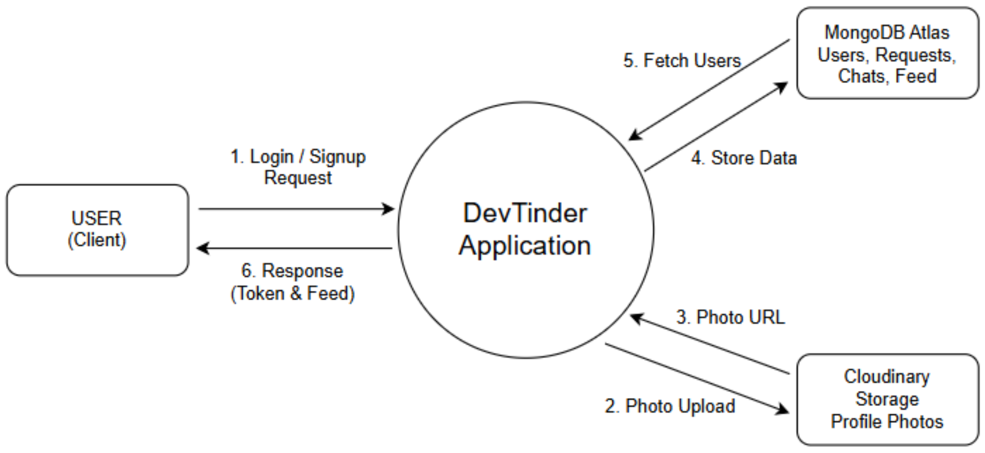
- Secure APIs with JWT & bcrypt hashing

- Cloud-based image storage (Cloudinary)

- Scalable NoSQL storage using MongoDB Atlas

4. System Architecture

Data Flow Diagram –


USER
(Client)
1. Login / Signup Request
6. Response (Token & Feed)
DevTinder Application
5. Fetch Users
4. Store Data
MongoDB Atlas Users, Requests, Chats, Feed
3. Photo URL
2. Photo Upload
Cloudinary Storage Profile Photos

---

5. System Specifications

Hardware Requirements

• Intel i7
• 8–16GB RAM
• SSD Storage

Software Requirements

• Vite + React
• Node.js + Express.js
• MongoDB Atlas
• Tailwind, DaisyUI
• Redux Toolkit
• Socket.io

---

6. Tools Used

- Vite + React for frontend

- Tailwind CSS & DaisyUI for UI

- Redux Toolkit for state management

- Axios for API handling

- Node.js + Express for backend

- Mongoose for schema modelling

- Cloudinary for image uploads

- MongoDB Atlas for cloud database

- Socket.io for real-time messaging

---

7. Implementation Summary

The frontend was built using Vite + React with modular components such as NavBar, Footer, Body, Feed, Login, Profile, and Chat. React Router was used for navigation while Redux Toolkit handled global state management. Axios communicated with backend APIs using credentials.

The backend used Express.js, JWT authentication, Mongoose schemas, connection request logic, and pagination for the feed. Real-time chat was achieved using Socket.io with WebSocket auth.

---

8. API & Database Overview

MongoDB Schema (User, ConnectionRequest, Message)

User Schema

```
const mongoose = require("mongoose");
const validator = require("validator");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");

const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true,
    minLength: 3,
    maxLength: 50,
  },
  lastName: { type: String },
  emailId: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    trim: true,
    validate(value) {
      if (!validator.isEmail(value)) {
        throw new Error("Not valid email");
      }
    },
  },
  password: { type: String, required: true },
  age: { type: Number, min: 18, max: 70 },
  gender: {
    type: String,
    enum: ["others", "Male", "Female"],
  },
  photoURL: {
    type: String,
    default:
"https://static.vecteezy.com/system/resources/previews/036/594/092/original/man-
empty-avatar-photo-placeholder-for-social-networks-resumes-forums-and-dating-sites-
male-and-female-no-photo-images-for-unfilled-user-profile-free-vector.jpg",
    validate(value) {
      if (!validator.isURL(value)) {
        throw new Error("Not valid photoURL");
      }
    },
  },
  about: {
    type: String,
    default: "This is default about of user",
    validate(value) {
      if (value.length > 125) {
        throw new Error("Short the about");
      }
    },
  },
  skills: {
    type: [String],
    validate(value) {
      if (value.length > 5) {
        throw new Error("Skills can't be more than 5");
      }
    },
  },
}, { timestamps: true });

userSchema.methods.getJWT = async function () {
  return jwt.sign({ _id: this._id }, "DEV@Tinder$1505", { expiresIn: "7d" });
};

userSchema.methods.validatePassword = async function (passwordInputByUser) {
  return bcrypt.compare(passwordInputByUser, this.password);
};

const User = mongoose.model("User", userSchema);
module.exports = User;
```

Connection Request Schema

```
const mongoose = require("mongoose");

const connectionRequestSchema = new mongoose.Schema({
  fromUserId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  toUserId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  status: {
    type: String,
    required: true,
    enum: ["ignored", "interested", "accepted", "rejected"],
  },
}, { timestamps: true });

connectionRequestSchema.pre("save", function (next) {
  if (this.fromUserId.equals(this.toUserId)) {
    throw new Error("Cannot send user request to yourself!!!");
  }
  next();
});

connectionRequestSchema.index({ fromUserId: 1, toUserId: 1 });

const ConnectionRequest = mongoose.model("Connection Request",
connectionRequestSchema);
module.exports = ConnectionRequest;
```

Chat & Message Schema

```
const mongoose = require("mongoose");

const messageSchema = new mongoose.Schema({
  senderId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  text: {
    type: String,
    required: true,
  },
}, { timestamps: true });

const chatSchema = new mongoose.Schema({
  participants: [
    { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
  ],
  messages: [messageSchema],
});

const Chat = mongoose.model("Chat", chatSchema);
module.exports = { Chat };
```

Key APIs:

- POST /signup

- POST /login

- GET /feed

- POST /request/send/:status/:toUserId

- POST /request/review/:status/:requestId

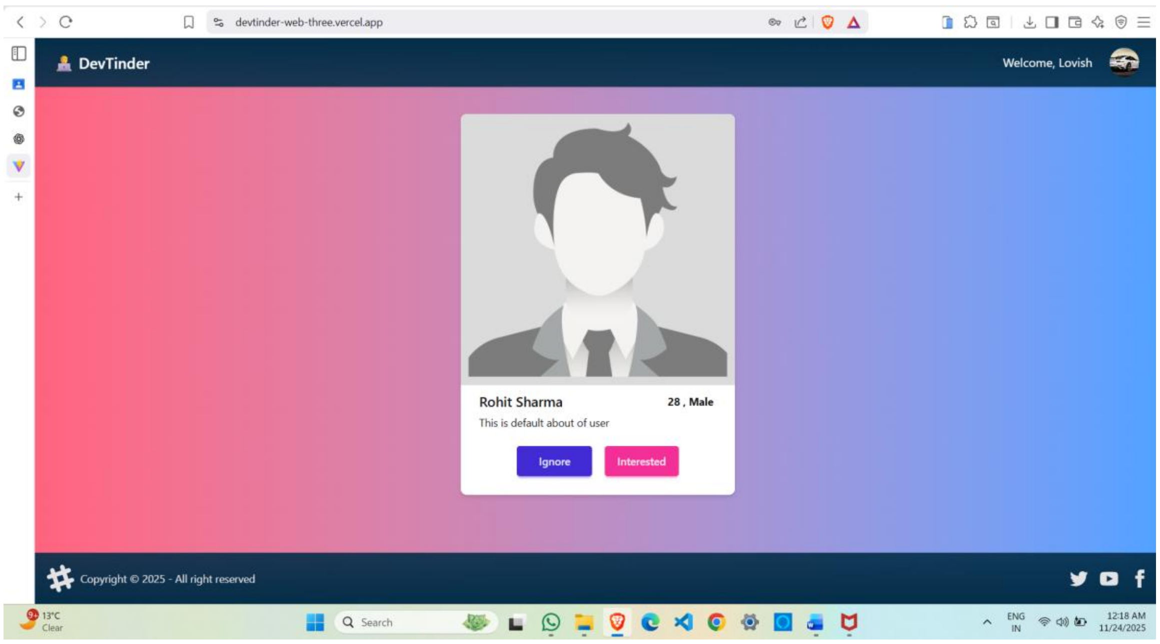
- GET /user/connections

- PATCH /profile/edit

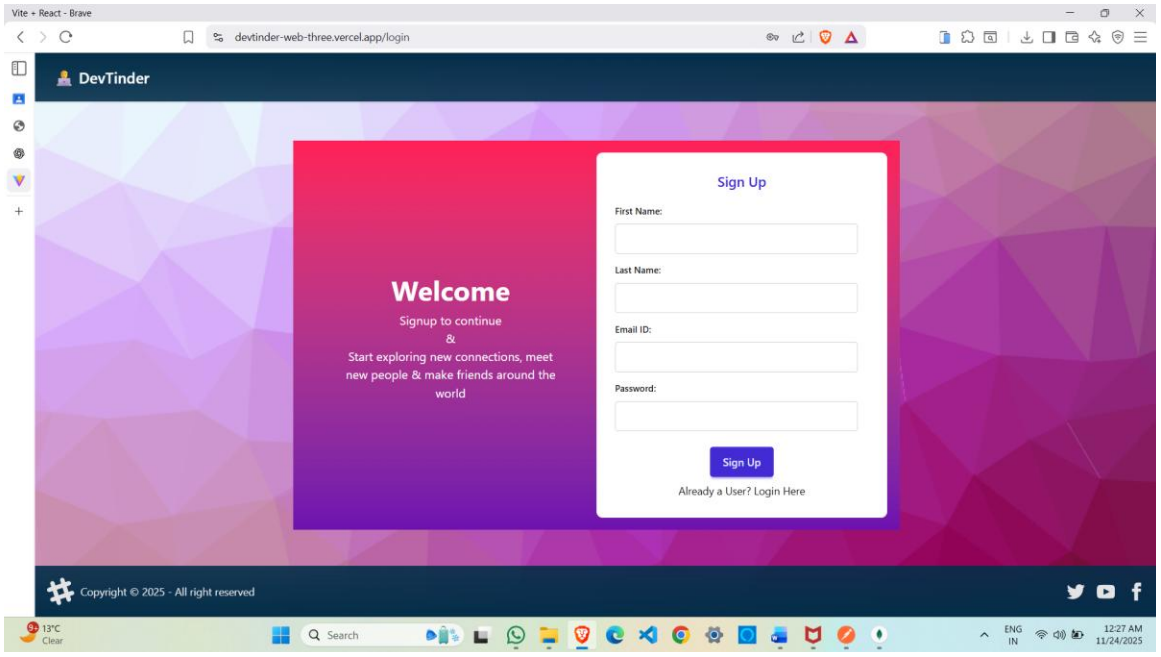
---

9. Sample Screenshots

• **Feed Page**


devtinder-web-three.vercel.app
DevTinder
Welcome, Lovish
Rohit Sharma
28 , Male
This is default about of user
Ignore
Interested
Copyright © 2025 - All right reserved

• **LOGIN Page**



Welcome

Signup to continue
&
Start exploring new connections, meet new people & make friends around the world

Sign Up

First Name:

Last Name:

Email ID:

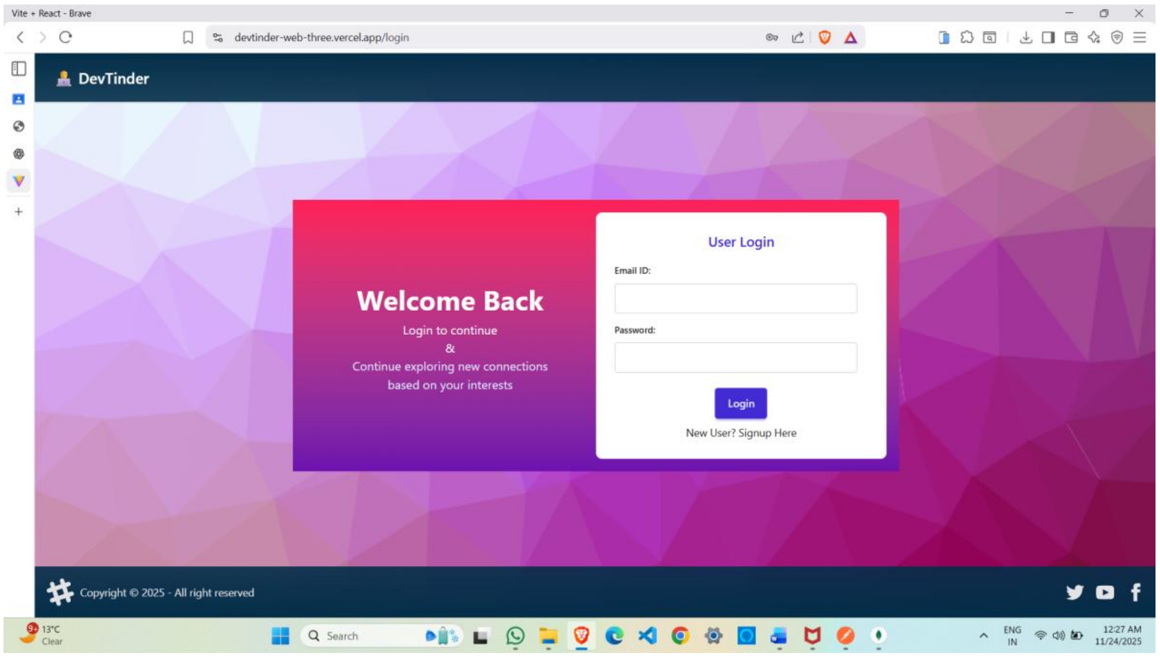
Password:

Sign Up

Already a User? Login Here

Copyright © 2025 - All right reserved

• **SIGNUP Page**



Welcome Back

Login to continue
&
Continue exploring new connections based on your interests

User Login

Email ID:

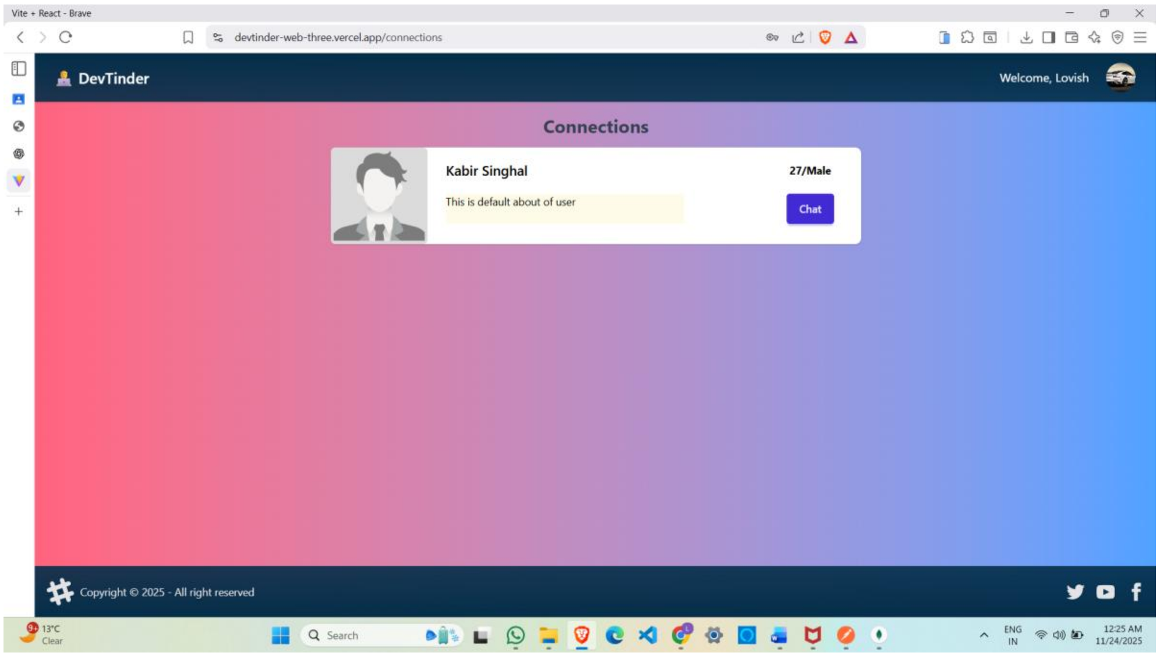
Password:

Login

New User? Signup Here

Copyright © 2025 - All right reserved

•CONNECTIONS Page


Vite + React - Brave

devtinder-web-three.vercel.app/connections

DevTinder

Welcome, Lovish

Connections

Kabir Singhal

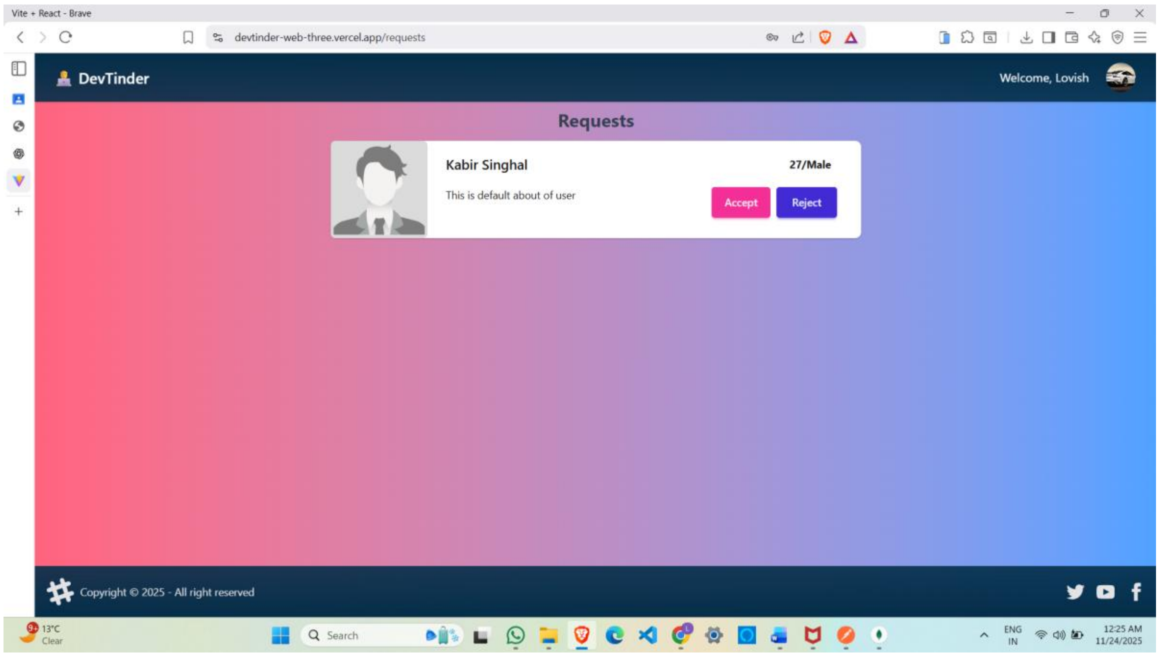
27/Male

This is default about of user

Chat

Copyright © 2025 - All right reserved

• CONNECTION REQUESTS Page


Vite + React - Brave

devtinder-web-three.vercel.app/requests

DevTinder

Welcome, Lovish

Requests

Kabir Singhal

27/Male

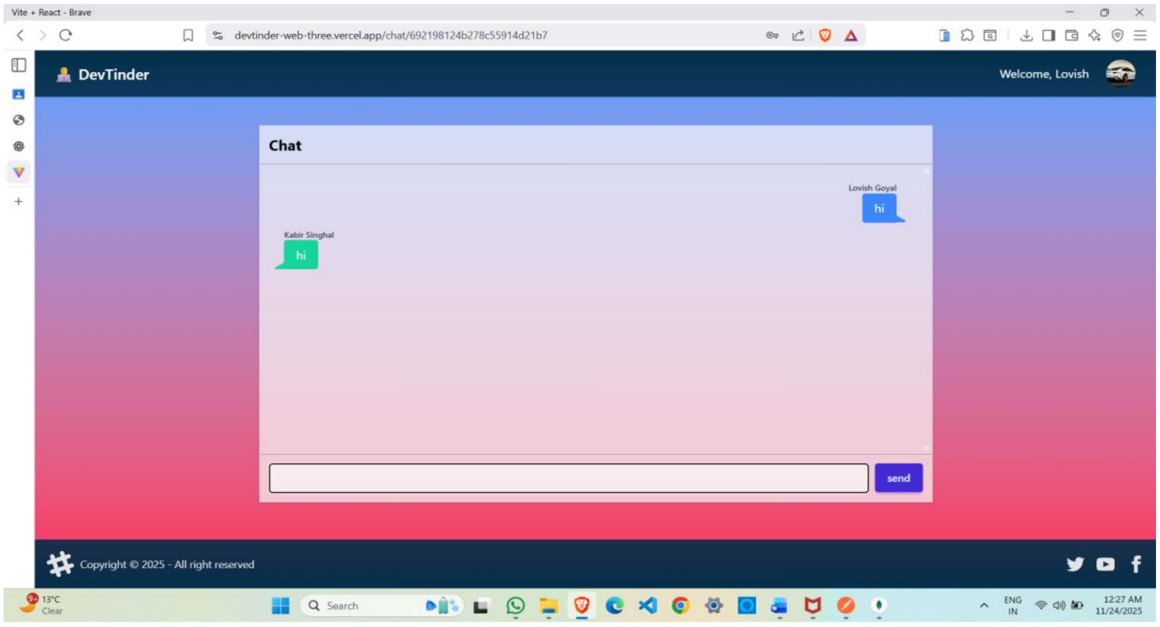
This is default about of user

Accept

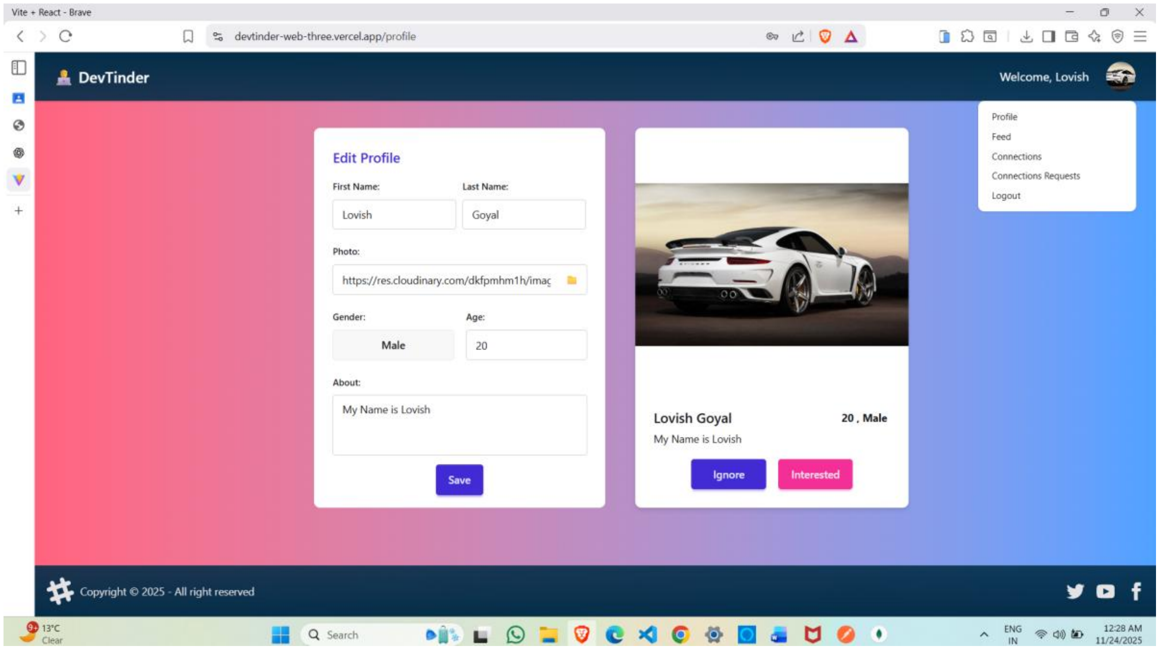
Reject

Copyright © 2025 - All right reserved

• **CHAT Page**


Vite + React - Brave
devtinder-web-three.vercel.app/chat/692198124b278c55914d21b7
DevTinder
Welcome, Lovish
Chat
Lovish Goyal
hi
Kabir Singhal
hi
send
Copyright © 2025 - All right reserved

• **EDIT PROFILE Page**



Output Reports

During the development of DevTinder, multiple layers of testing and validation were performed to ensure the reliability and accuracy of the system. These included API validation through Postman, database verification in MongoDB Atlas, and frontend state debugging using Redux DevTools.

a. API Testing (Postman)

A dedicated Postman collection was created to test all backend routes. Each endpoint was validated for:

- **Correct request/response formats**
- **Proper status codes** (200, 201, 400, 401, 404, 500)
- **Authentication checks** using JWT cookies
- **Error handling** for invalid inputs

Key API tests performed:

- **Signup API:** Verified schema validations and duplicate email handling.
- **Login API:** Ensured correct password hashing check and cookie creation.
- **Profile API:** Checked that only authenticated users can access profile data.
- **Feed API:** Validated pagination, user filtering, and secure access.
- **Connection Request API:** Tested all request states (interested, ignored, accepted, rejected).
- **Chat API / socket handshake:** Ensured only connected users can initiate a chat.

All APIs responded correctly and handled edge cases such as invalid tokens, expired cookies, wrong credentials, and unauthorized access.

---

b. Backend Logs & Debugging

Backend debugging was done through:

- **console.log()** tracking
- **Express error handlers**
- **Mongoose validation error outputs**
- **Socket.io connection logs**
- **User action trails (sent request, accepted request, message sent)**

These logs helped identify issues like missing tokens, invalid request flows, and incorrect DB updates.

---

c. Frontend Testing

Frontend behavior was validated using:

- **Redux DevTools**
    - Verified state updates for auth, feed, profile, and chat
- **React Developer Tools**
    - Component rendering and state changes
- **Network Tab (Browser DevTools)**
    - Monitoring axios requests
    - Ensuring cookies (token) were being sent properly
- **UI/UX validation**
    - Mobile responsiveness testing
    - Component layout and Tailwind styling checks

---

d. End-to-End Testing

End-to-end tests validated the entire user flow:

- User signs up → logs in → fetches feed → sends request → accepts request → opens chat → messages.

This ensured the platform behaved consistently across all modules.

Conclusion

DevTinder stands as a fully functional, modern, and scalable developer-matching platform inspired by real-world social and collaboration tools. It demonstrates practical application of key full-stack concepts including authentication, state management, WebSocket-based real-time communication, and clean UI/UX design using Tailwind and DaisyUI.

By integrating Cloudinary for media handling, Redux Toolkit for efficient global state management, Express for robust API creation, and MongoDB's flexible schema system, the project successfully delivers a seamless and responsive user experience.

The platform enables:

- Smooth onboarding through secure JWT login
- Intelligent user discovery via feed & pagination
- Meaningful connections through a structured request system
- Real-time chatting with Socket.io
- Clean user profile management
- Scalable backend that can support more features in future

This project not only fulfills the requirements of a UI/UX-centric full-stack application but also lays a solid foundation for future extensions such as AI-based recommendations, profile analytics, group chats, and mobile app integration.

Overall, DevTinder proves to be a strong prototype for a developer-focused networking app that balances functionality, usability, and performance.