



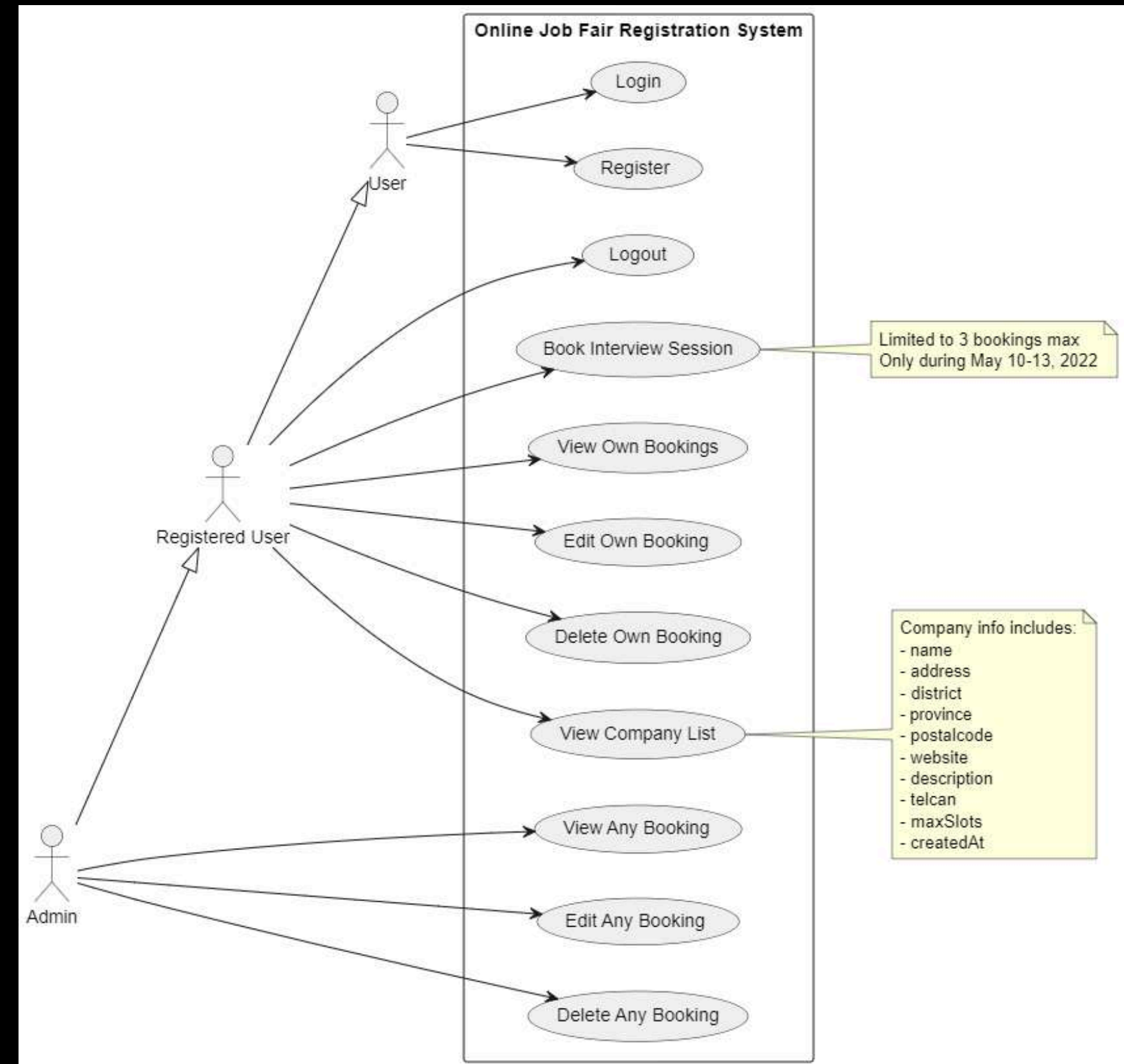
ONLINE JOB FAIR REGISTRATION

Group เมียวเมี้ยวเมี้ยวเมี้ยว
Praeploy Kiatsuksri
Pokkrong Ouangjan

https://github.com/praeploykiat/SW_Dev_Final_Project

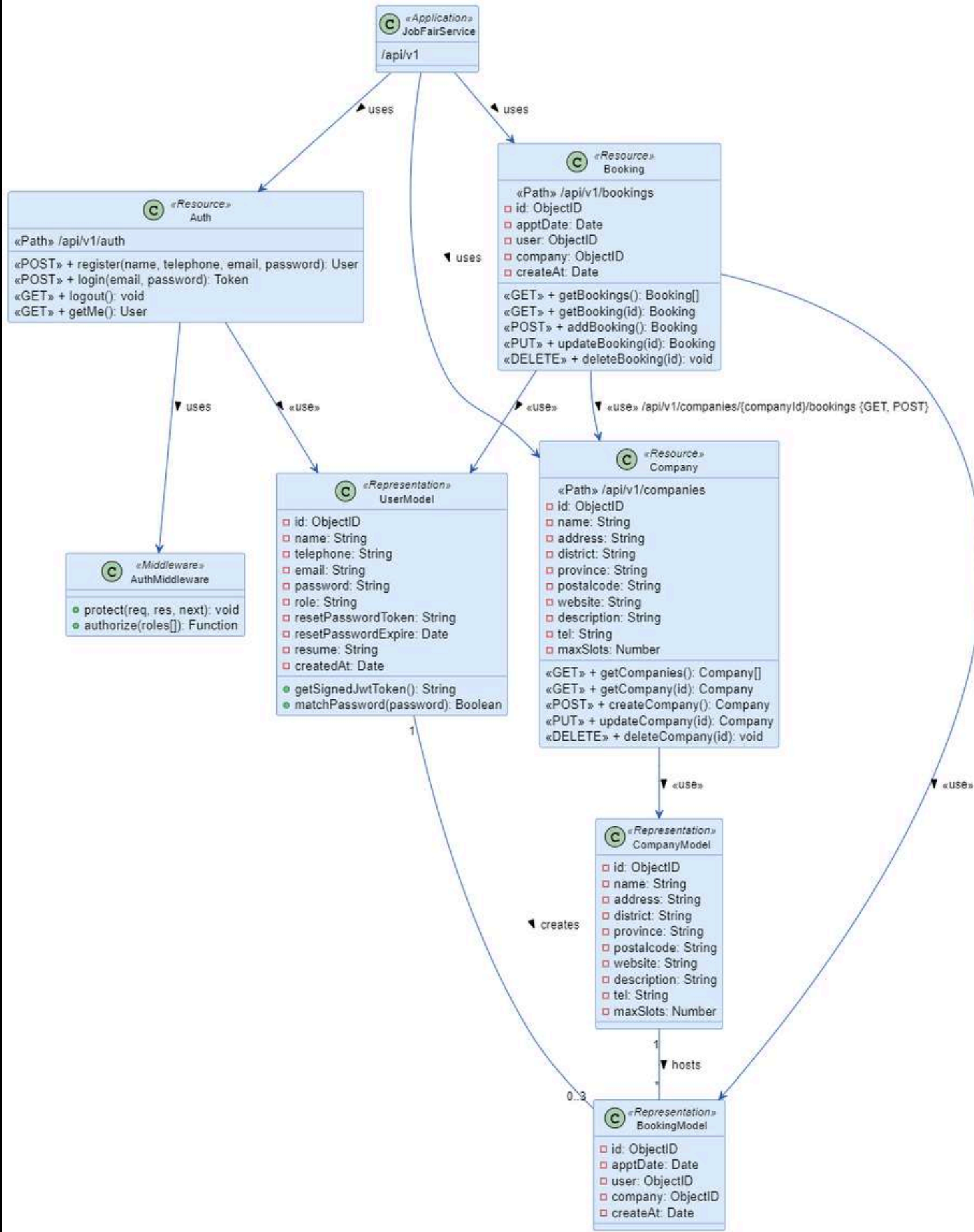


Use Case Diagram



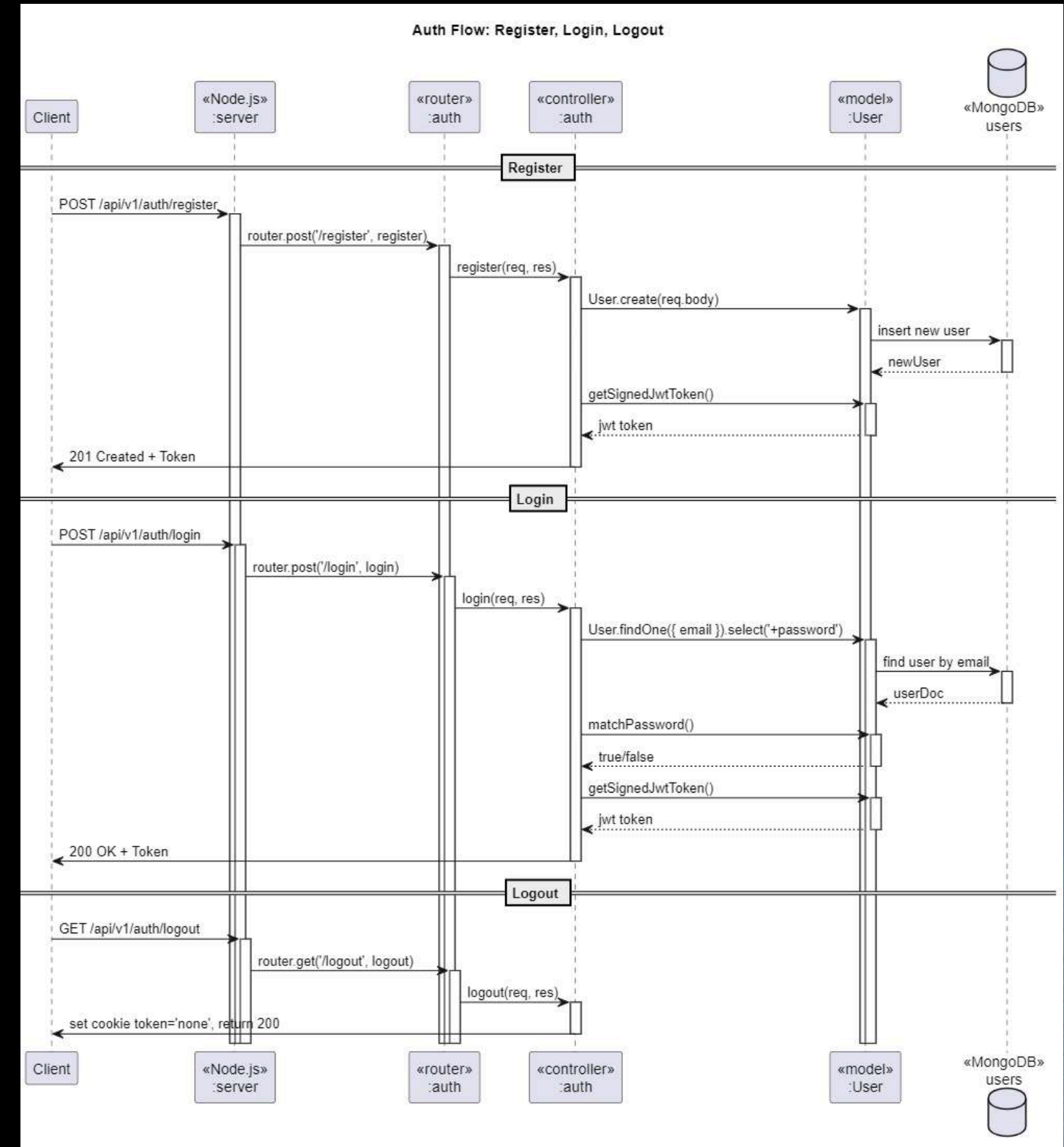


Class Diagram



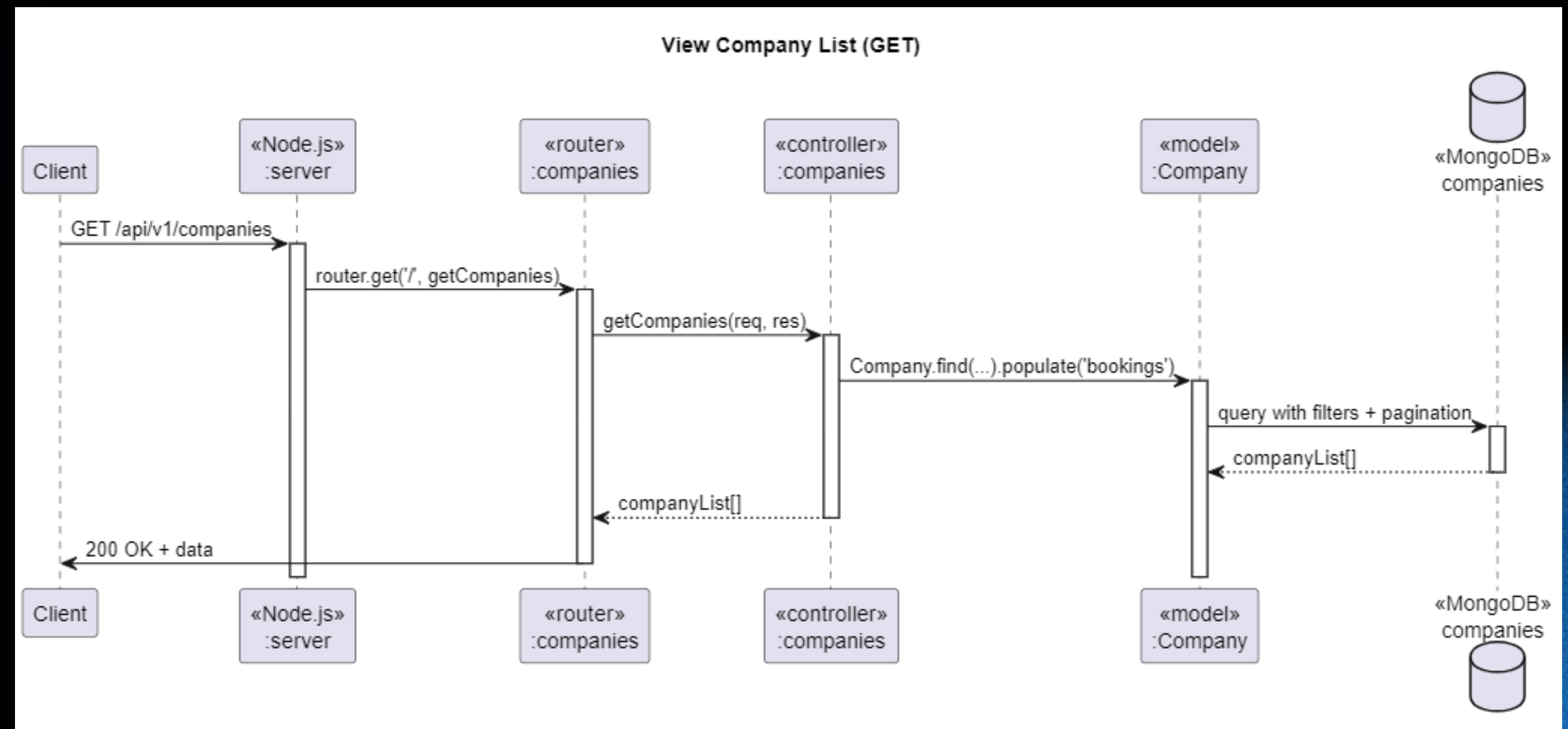


Sequence Diagram

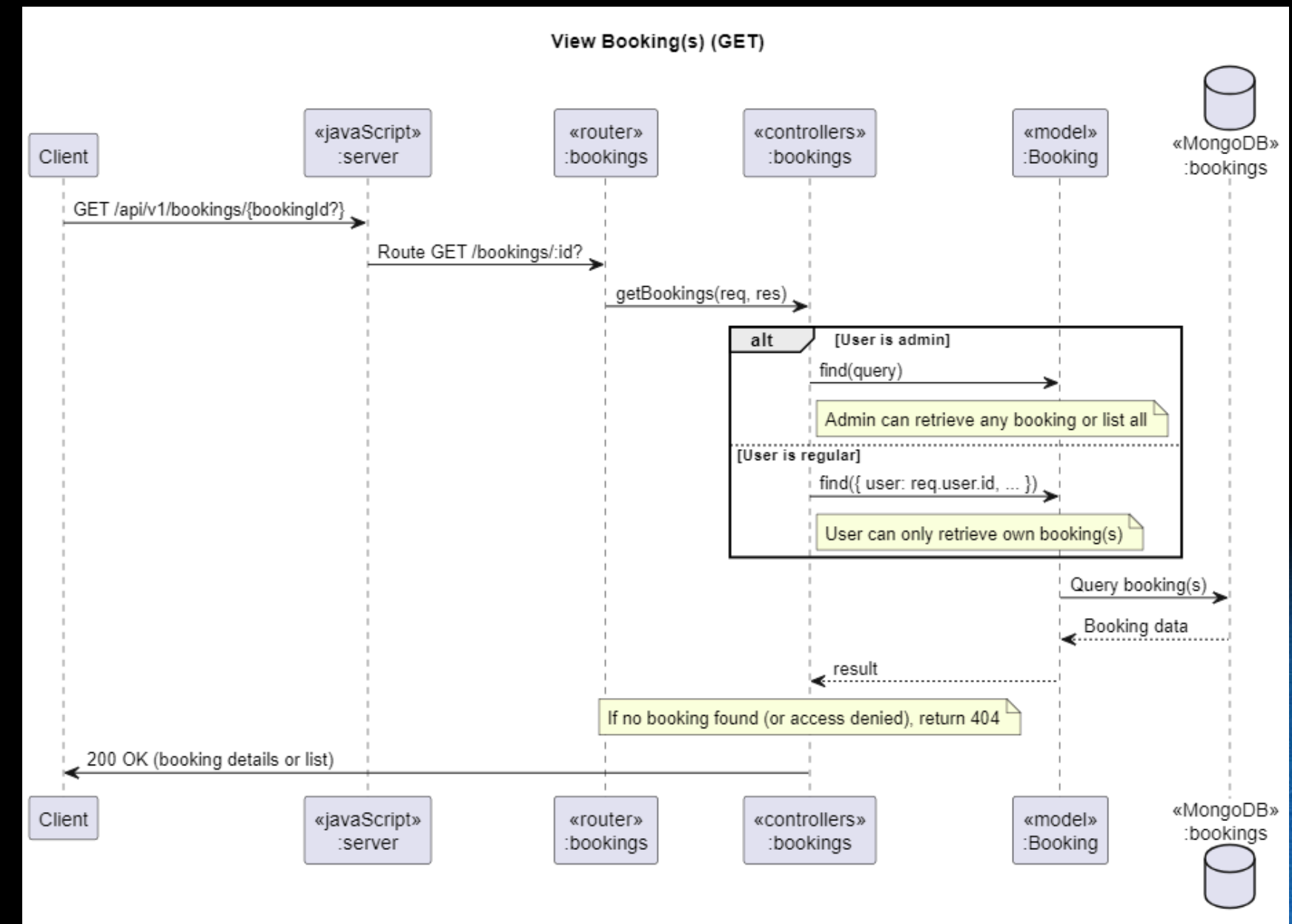
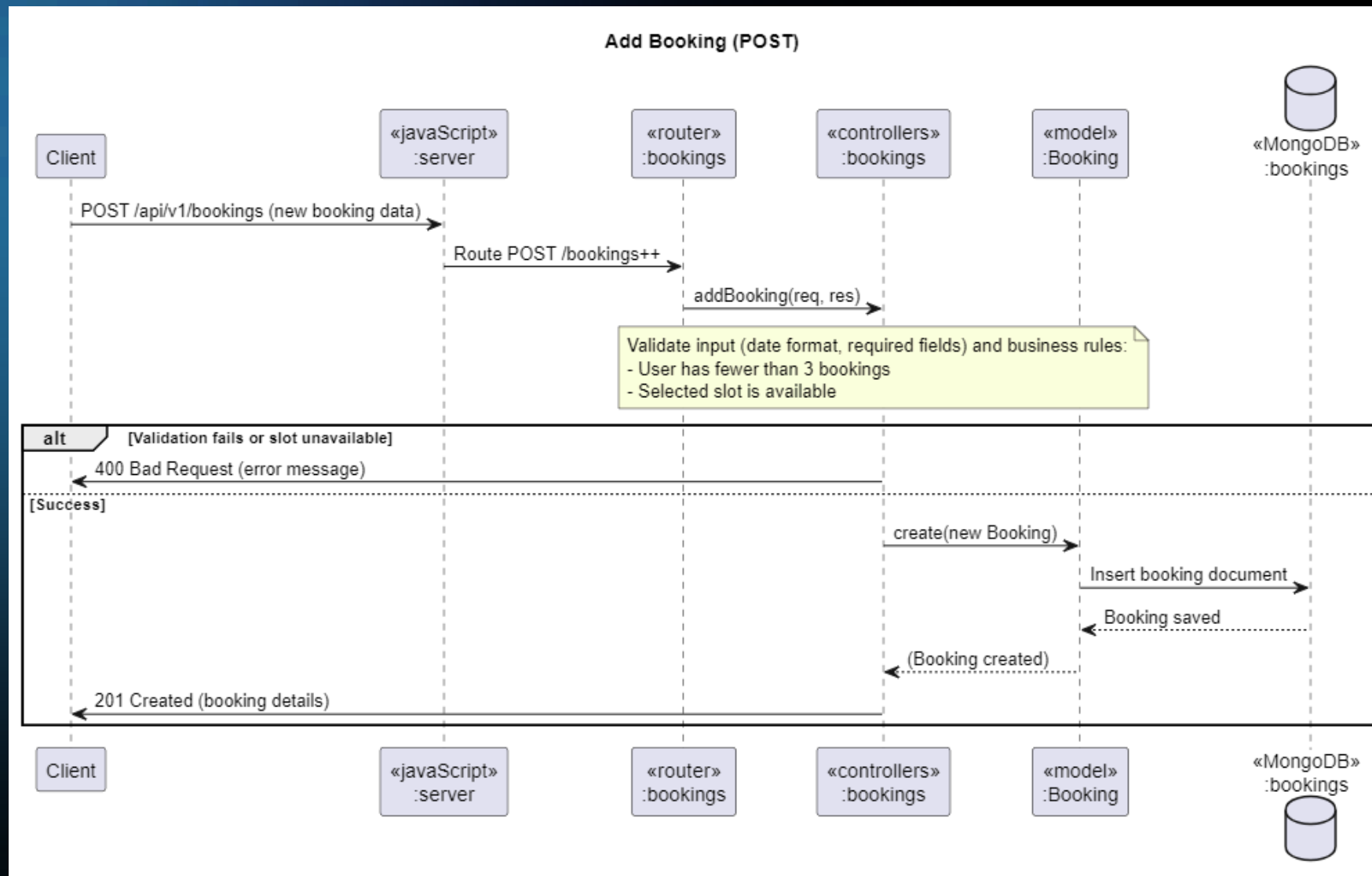




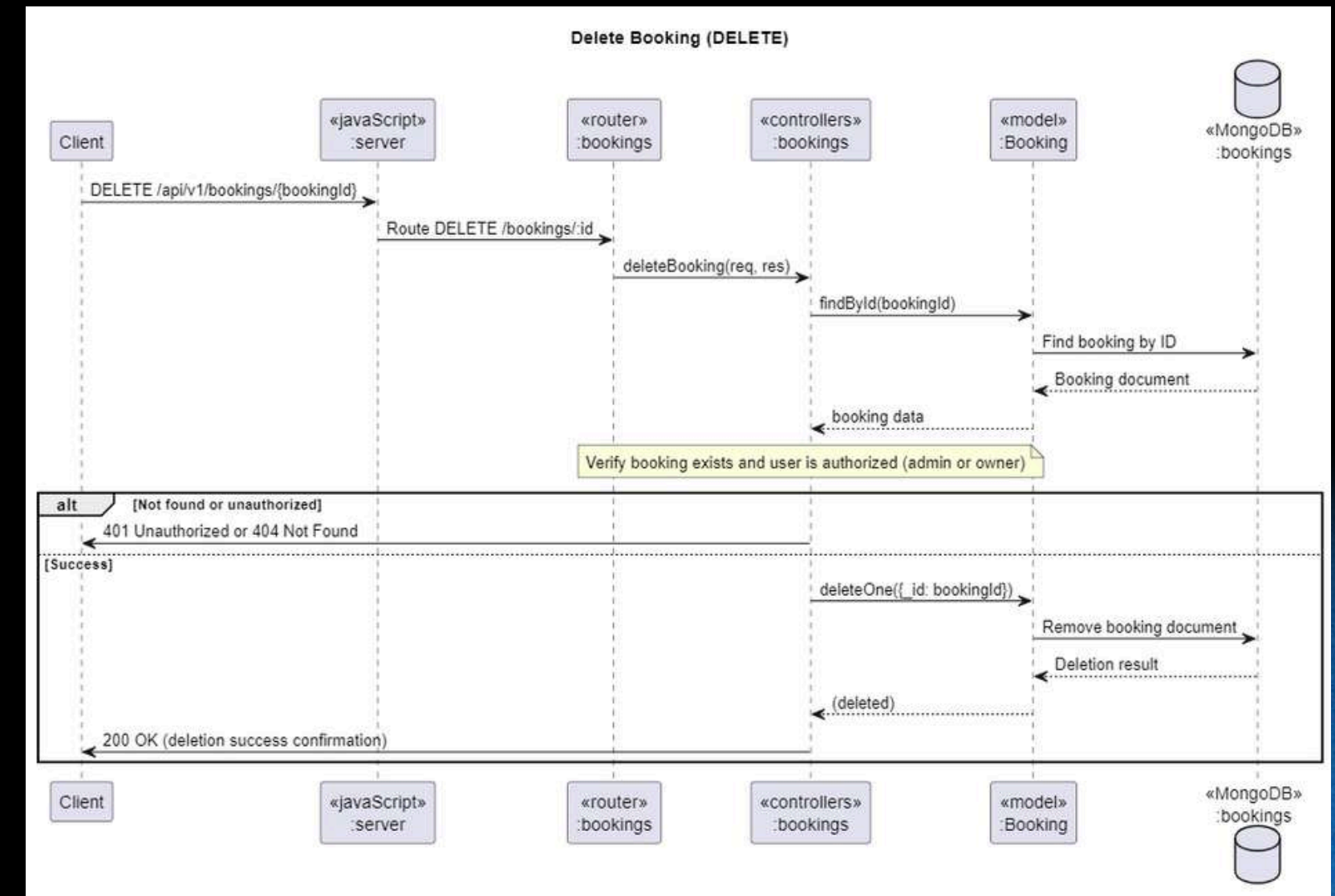
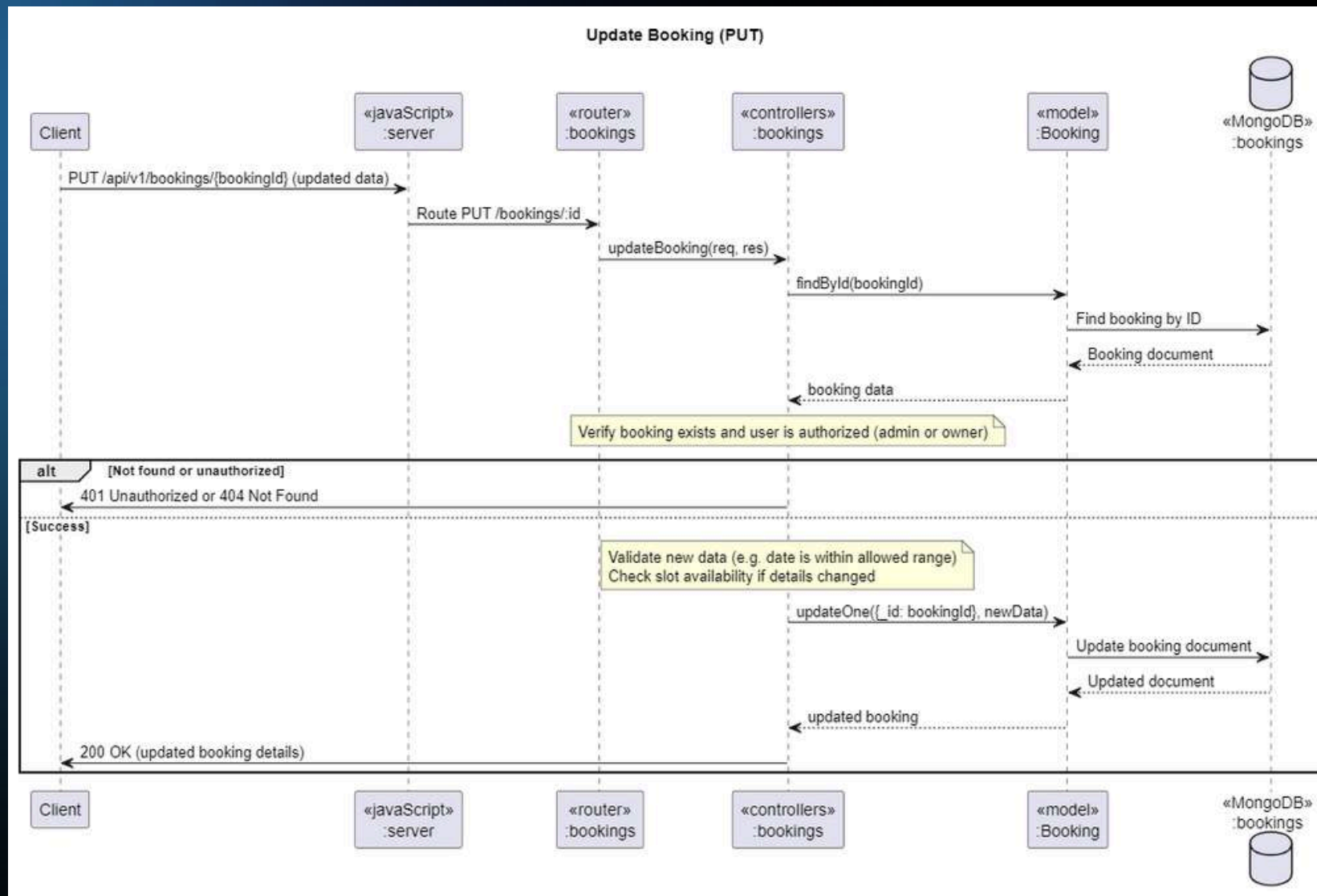
Sequence Diagram



Sequence Diagram



Sequence Diagram





Extra Credits



Tutorials

<https://www.youtube.com/watch?v=fvP6Ls8NlYk&t=3s>

INTERVIEW SESSION AVAILABILITY

The system shall allow admins to set and update the maximum number of interview slots for each company.

FORGOT PASSWORD

The system shall allow users to reset their password if they forget it by sending a Password Reset Email containing a reset link.

BOOKMARK

The system shall allow registered users to bookmark preferred companies for quick access. Users can add, view, and remove bookmarks.

RESUME UPLOAD

The system shall allow a registered user to upload a PDF resume after registration.

AI INTERVIEW ASSISTANCE

The system shall allow a registered user to receive AI-generated interview tips, question suggestions, or resume feedback using an LLM API.

CONTRIBUTIONS

SET UP MONGODB & GITHUB REPO

PRAEPLY



FUNCTIONAL REQUIREMENT

1-3

PRAEPLY



4-9

POKKRONG



ADDITIONAL REQUIREMENT

Interview Session Availability

POKKRONG



Forgot Password

PRAEPLY



Bookmark

PRAEPLY



Resume Upload

PRAEPLY



AI Interview Assistance

POKKRONG



GitHub Repo

https://github.com/praeploykiat/SW_Dev_Final_Project



THANK YOU

APPENDIX



Extra Credits



Tutorials

<https://www.youtube.com/watch?v=fvP6Ls8NlYk&t=3s>

INTERVIEW SESSION AVAILABILITY

The system shall allow admins to set and update the maximum number of interview slots for each company.

FORGOT PASSWORD

The system shall allow users to reset their password if they forget it by sending a Password Reset Email containing a reset link.

BOOKMARK

The system shall allow registered users to bookmark preferred companies for quick access. Users can add, view, and remove bookmarks.

RESUME UPLOAD

The system shall allow a registered user to upload a PDF resume after registration.

AI INTERVIEW ASSISTANCE

The system shall allow a registered user to receive AI-generated interview tips, question suggestions, or resume feedback using an LLM API.

Interview Session Availability

The system shall allow admins to set and update the maximum number of interview slots for each company

models/Company.js

```
tel:{
  type:String,
},
maxSlots: {
  type: Number,
  required: [true, 'Please add the maximum number of interview slots'],
  default: 10,
  validate: {
    validator: function(val) {
      return val > 0;
    },
    message: 'Maximum slots must be a positive number'
  }
},
```

Add maxSlots field into CompanySchema

Add constraint maxSlots always >0

routes/companies.js

Add updateMaxSlots

```
const {getCompanies,getCompany,createCompany,updateCompany,deleteCompany, updateMaxSlots} = require('../controller/companies');
```

```
/ Add this new route for updating max slots
router.route('/:id/slots').put(protect, authorize('admin'), updateMaxSlots);
```


Interview Session Availability

controller/bookings.js

```
if(!company){
  return res.status(404).json({success:false,msg:`No company with the id of ${req.params.companyId}`});
}

const requestedDate = new Date(req.body.apptDate);

const startOfDay = new Date(requestedDate);
startOfDay.setHours(0, 0, 0, 0);

const endOfDay = new Date(requestedDate);
endOfDay.setHours(23, 59, 59, 999);

const bookingsOnThisDate = await Booking.countDocuments({
  company: req.params.companyId,
  apptDate: {
    $gte: startOfDay,
    $lte: endOfDay
  }
});

if(bookingsOnThisDate >= company.maxSlots) {
  return res.status(400).json({
    success: false,
    msg: `Company ${company.name} has reached its maximum number of interview slots for ${startOfDay.toDateString()}`
  });
}
```


Interview Session Availability

controller/companies.js

```
/**
 * @desc    Update maximum slots for a company
 * @route    PUT /api/companies/:id/slots
 * @access   Private/Admin
 */
exports.updateMaxSlots = async (req, res, next)=>{
  try{
    const {maxSlots} = req.body;
    //validate input
    if(!maxSlots || !Number.isInteger(maxSlots) || maxSlots <1){
      return res.status(400).json({
        success: false,
        error : 'Please provide a valid positive integer for maximum slots'
      });
    }

    const company = await Company.findByIdAndUpdate(
      req.params.id,
      {maxSlots},
      {new: true, runValidators: true}
    );
    if(!company){
      return res.status(404).json({
        success: false,
        error: `Company with id ${req.params.id} not found`
      });
    }
    res.status(200).json({
      success: true,
      data: company
    });
  }catch (err) {
    console.log(err.stack);
    res.status(500).json({
      success: false,
      error: 'Server error updating maximum slots'
    });
  }
};
```


AI Interview Assistance

The system shall allow a registered user to receive AI-generated interview tips, question suggestions, or resume feedback using an LLM API.

config/config.env

```
GEMINI_API_KEY=Your API KEY
```

Change to your gemini API key

routes/aiController.js

```
const express = require('express');
const router = express.Router();
const { generateAIResponse } = require('../controller/aiController');
const { protect } = require('../middleware/auth');

router.post('/generate', protect, generateAIResponse);

module.exports = router;
```

server.js

```
const aiRoutes = require('./routes/aiController');
```

```
app.use('/api/ai', aiRoutes);
```


AI Interview Assistance

```
const axios = require('axios'); //need to npm install axios
```

don't forget to npm install axios

utils/ai.js

```
/**
 * @param {string} apiKey - Gemini API key
 * @param {string} prompt - The text prompt to send to the model
 * @param {string} model - The model to use (default: 'gemini-2.0-flash')
 * @returns {Promise<object>} - The API response
 */
async function generateContent(apiKey, prompt, model = 'gemini-2.0-flash') {
  try {
    const response = await axios.post(
      `https://generativelanguage.googleapis.com/v1beta/models/${model}:generateContent?key=${apiKey}`,
      {
        contents: [{
          parts: [{ text: prompt }]
        }],
      },
      {
        headers: {
          'Content-Type': 'application/json'
        }
      }
    );

    return response.data;
  } catch (error) {
    throw new Error(`Request failed: ${error.message}`);
  }
}
```

```
/**
 * Extract text content from Gemini API response
 * @param {object} response - The API response
 * @returns {string} - The extracted text
 */
function extractTextFromResponse(response) {
  try {
    return response.candidates[0].content.parts[0].text;
  } catch (error) {
    throw new Error(`Failed to extract text from response: ${error.message}`);
  }
}

module.exports = {
  generateContent,
  extractTextFromResponse
};
```


AI Interview Assistance

controller/aiController.js

Add on the right first

```
/**
 * Format AI response text
 * @param {string} text - Raw text from AI response
 * @returns {string} - Formatted text
 */
function formatTextResponse(text) {
  if (!text) return "No response generated.";

  // Replace multiple newlines with just two
  text = text.replace(/\n{3,}/g, '\n\n');

  // Add proper spacing after punctuation if missing
  text = text.replace(/([.!?])\s*([A-Z])/g, '$1 $2');

  // Add horizontal line for separation
  const separator = '\n' + '-'.repeat(60) + '\n';

  return separator + text + separator;
}
```

```
const { generateContent, extractTextFromResponse } = require('../utils/ai');
require('dotenv').config();

/**
 * @desc    Get AI response for a prompt
 * @route   POST /api/ai/generate
 * @access  Private
 */
exports.generateAIResponse = async (req, res) => {
  try {
    const { prompt } = req.body;

    // Validate input
    if (!prompt || typeof prompt !== 'string') {
      return res.status(400).json({
        success: false,
        error: 'Please provide a valid prompt string'
      });
    }

    const apiKey = process.env.GEMINI_API_KEY;
    if (!apiKey) {
      return res.status(500).json({
        success: false,
        error: 'Server configuration error: API key not found'
      });
    }

    const response = await generateContent(apiKey, prompt);
    let textOutput = extractTextFromResponse(response);

    //format text
    textOutput = formatTextResponse(textOutput);

    return res.status(200).send(textOutput);
  } catch (error) {
    console.error('AI generation error:', error);
    return res.status(500).json({
      success: false,
      error: 'Error generating AI response',
      message: error.message
    });
  }
};
```


Forgot Password

The system shall allow users to reset their password if they forget it by sending a Password Reset Email containing a reset link.

config/config.env

```
EMAIL_USER = swdevjobfair@gmail.com  
EMAIL_PASS =
```

App Password, not your actual Gmail password !!

routes/passwordReset.js

```
const express = require('express');  
const router = express.Router();  
const { requestPasswordReset, resetPassword } = require('../controller/passwordReset');  
  
// Route to request password reset  
router.post('/requestPasswordReset', requestPasswordReset);  
  
// Route to reset password using token and user ID  
router.put('/resetPassword/:id/:token', resetPassword);  
  
module.exports = router;
```

server.js

```
const passwordReset = require('./routes/passwordReset');  
  
app.use('/api/v1/auth', passwordReset);
```


Forgot Password

controller/passwordReset.js
requestPasswordReset

```
const jwt = require('jsonwebtoken'); // For generating a secure, time-limited token
const nodemailer = require('nodemailer'); // For sending password reset emails
const User = require('../models/User'); // User model (MongoDB)
const fs = require('fs'); // For reading the HTML email template
const path = require('path'); // To help build cross-platform file paths
```

```
// @desc    Send password reset link via email
// @route   POST /api/v1/auth/requestPasswordReset
// @access  Public
```

```
exports.requestPasswordReset = async (req, res) => {
```

```
  const { email } = req.body;
```

Gets the user's email from the request body.

```
  try {
```

```
    const user = await User.findOne({ email });
```

Looks up the user in the database using their email.

```
    if (!user) {
```

```
      return res.status(404).json({ success: false, message: "User doesn't exist" });
```

If the user isn't found, return an error.

```
    }
```

```
    const secret = process.env.JWT_SECRET + user.password;
```

Create a secure JWT reset token.

```
    const token = jwt.sign({ id: user._id, email: user.email }, secret, { expiresIn: '1h' });
```

Create a reset URL.

```
    const resetURL = `http://localhost:5000/api/v1/auth/resetPassword/${user._id}/${token}`;
```

```
    const transporter = nodemailer.createTransport({
```

```
      service: 'gmail',
```

```
      auth: {
```

```
        user: process.env.EMAIL_USER,
```

Set up the email transporter.

```
        pass: process.env.EMAIL_PASS,
```

```
      },
```

```
    });
```


Forgot Password

utils/resetPasswordEmail.html

controller/passwordReset.js
requestPasswordReset

```
<div style="font-family: Arial, sans-serif; padding: 24px; border-radius: 10px; background: #f9f9f9; border: 1px solid #ccc;">
  <h2 style="color: #2c3e50;">Reset Your Password</h2>
  <p>Hello <strong>{{name}}</strong>,</p>
  <p>We received a request to reset your password. Use the following URL as the <strong>PUT request path</strong> in Postman:</p>
  <div style="margin: 20px 0;">
    <code style="background: #f0f0f0; padding: 12px; display: block; border-radius: 6px; font-family: monospace; font-size: 14px; color: #333;">
      PUT {{resetURL}}
    </code>
  </div>
  <p>Include this JSON body in the request:</p>
  <pre style="background: #f0f0f0; padding: 12px; border-radius: 6px; font-family: monospace; font-size: 14px;">
  {
    "password": "yourNewPasswordHere"
  }
  </pre>
  <p>If you didn't request this reset, you can safely ignore this email.</p>
  <p style="margin-top: 24px;">Warm regards,<br><strong>Online Job Fair Team</strong></p>
</div>
```

```
const filePath = path.join(__dirname, '../utils/resetPasswordEmail.html');
let emailHTML = fs.readFileSync(filePath, 'utf8');
emailHTML = emailHTML
  .replace('{{name}}', user.name)
  .replace('{{resetURL}}', resetURL);

const mailOptions = {
  to: user.email,
  from: process.env.EMAIL_USER,
  subject: 'Reset Your Password - Online Job Fair',
  html: emailHTML,
};

await transporter.sendMail(mailOptions);

res.status(200).json({ success: true, message: 'Password reset link sent' });
} catch (err) {
  console.error(err.stack);
  res.status(500).json({ success: false, message: 'Server error' });
}
```

Load and personalize email template.

Send the email.

Send back a success message.

Forgot Password

controller/passwordReset.js
resetPassword

```
// @desc   Reset user password using token from email
// @route  PUT /api/v1/auth/resetPassword/:id/:token
// @access Public
exports.resetPassword = async (req, res) => {
  const { id, token } = req.params;
  const { password } = req.body;

  try {
    const user = await User.findById(id);
    if (!user) {
      return res.status(404).json({ success: false, message: 'User not found' });
    }

    const secret = process.env.JWT_SECRET + user.password;

    try {
      jwt.verify(token, secret);
    } catch (err) {
      return res.status(400).json({ success: false, message: 'Invalid or expired token', detail: err.message });
    }

    user.password = password; // Will be hashed by Mongoose pre-save hook
    await user.save();

    console.log('✅ Password reset for:', user.email);

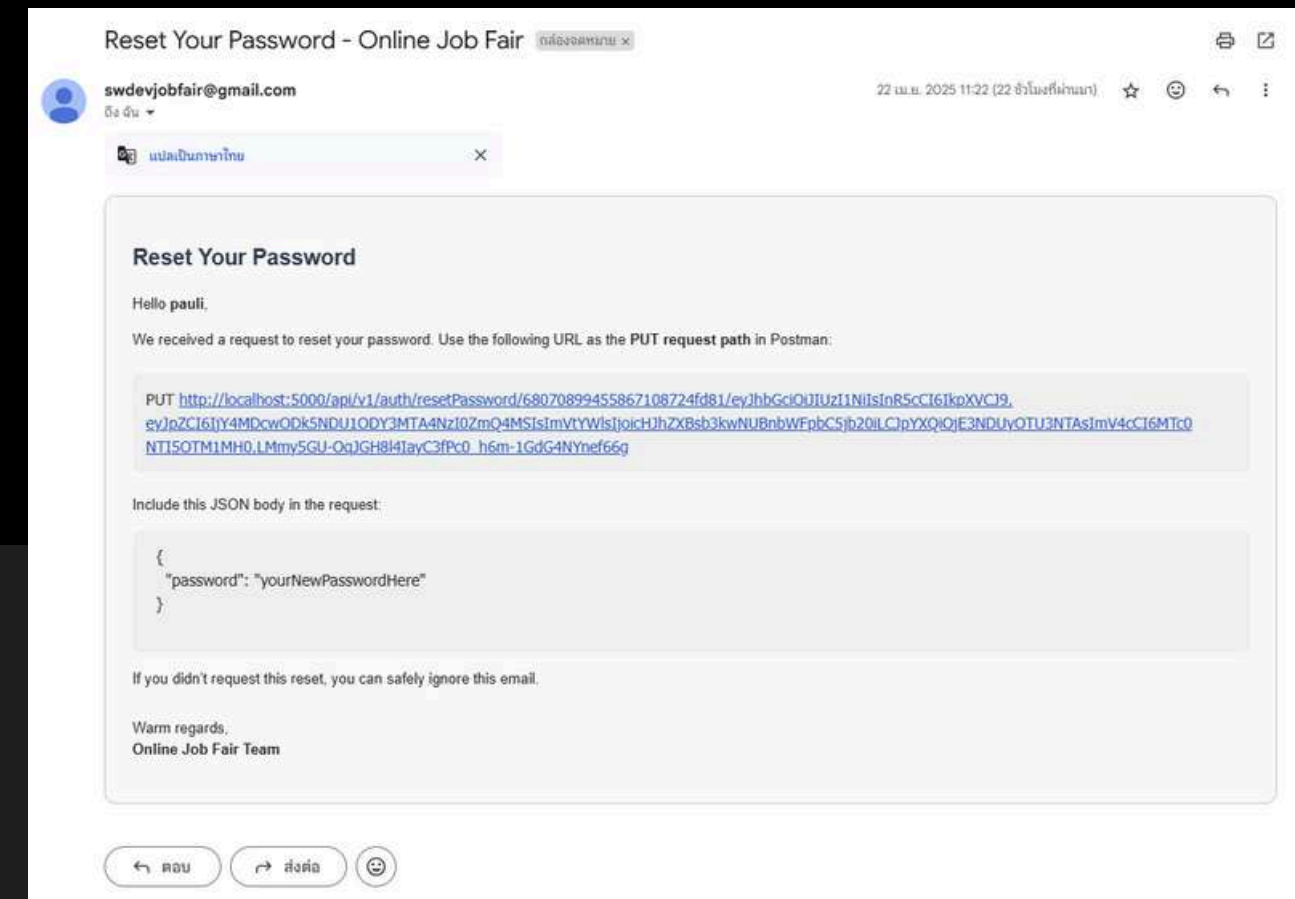
    res.status(200).json({ success: true, message: 'Password has been reset' });
  } catch (err) {
    console.error(err.stack);
    res.status(500).json({ success: false, message: 'Server error' });
  }
};
```

Gets the user ID and token from the URL, and new password from the request body.

Constructs the same secret that was used to generate the token earlier.

Save new password.

Success response.



Verifies the token is valid and has not expired.

Bookmark

The system shall allow registered users to bookmark preferred companies for quick access. Users can add, view, and remove bookmarks.

model/Bookmark.js

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

const BookmarkSchema = new mongoose.Schema({
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  company: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Company',
    required: true
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
});

BookmarkSchema.index({ user: 1, company: 1 }, { unique: true }); // prevent duplicates

module.exports = mongoose.model('Bookmark', BookmarkSchema);
```

Adds a unique index to prevent a user from bookmarking the same company more than once.

Bookmark

The system shall allow registered users to bookmark preferred companies for quick access. Users can add, view, and remove bookmarks.

controller/bookmark.js

```
const Bookmark = require('../models/Bookmark');
const Company = require('../models/Company');

// ✅ Add bookmark using companyId only
exports.addBookmark = async (req, res) => {
  try {
    const { companyId } = req.body;

    if (!companyId) {
      return res.status(400).json({ success: false, message: 'companyId is required' });
    }

    const company = await Company.findById(companyId);
    if (!company) {
      return res.status(404).json({ success: false, message: 'Company not found' });
    }

    const bookmark = await Bookmark.create({
      user: req.user.id,
      company: company._id
    });

    res.status(201).json({ success: true, data: bookmark });
  } catch (err) {
    if (err.code === 11000) {
      return res.status(400).json({ success: false, message: 'Already bookmarked' });
    }

    console.error('❌ Bookmark creation failed:', err);
    res.status(500).json({ success: false, message: 'Failed to bookmark' });
  }
};
```

Check if companyId was provided in the request body.

Validate that the company exists.

Create a new bookmark linked to the logged-in user.

Find all bookmarks of the logged-in user and populate company info.

```
// ✅ Get all bookmarks for current user
exports.getBookmarks = async (req, res) => {
  try {
    const bookmarks = await Bookmark.find({ user: req.user.id })
      .populate('company', 'name address tel website description');

    res.status(200).json({ success: true, data: bookmarks });
  } catch (err) {
    console.error(err);
    res.status(500).json({ success: false, message: 'Failed to retrieve bookmarks' });
  }
};

// ✅ Remove bookmark using companyId only (via URL param)
exports.removeBookmark = async (req, res) => {
  try {
    const companyId = req.params.companyId;

    const result = await Bookmark.findOneAndDelete({
      user: req.user.id,
      company: companyId
    });

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

    res.status(200).json({ success: true, message: 'Bookmark removed' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ success: false, message: 'Failed to remove bookmark' });
  }
};
```

Delete a bookmark where the user and company match.

Bookmark

The system shall allow registered users to bookmark preferred companies for quick access. Users can add, view, and remove bookmarks.

routes/bookmark.js

```
const express = require('express');
const router = express.Router();
const { addBookmark, getBookmarks, removeBookmark } = require('../controller/bookmark');
const { protect } = require('../middleware/auth');

router.post('/', protect, addBookmark);
router.get('/', protect, getBookmarks);
router.delete('/:companyId', protect, removeBookmark);

module.exports = router;
```

server.js

```
const bookmark = require('./routes/bookmark');
```

```
app.use('/api/v1/bookmarks', bookmark);
```

Bookmark

The system shall allow registered users to bookmark preferred companies for quick access. Users can add, view, and remove bookmarks.

controller/companies.js

```
const Bookmark = require('../models/Bookmark');
```

```
exports.deleteCompany = async (req,res,next) => {
  try {
    const company = await Company.findById(req.params.id);
    //const company = await Company.findByIdAndDelete(req.params.id);
    if(!company){
      return res.status(400).json({success:false,msg:`Company not found with id of ${req.params.id}`});
    }
    await Booking.deleteMany({ company: req.params.id });
    await Bookmark.deleteMany({ company: req.params.id });
    await Company.deleteOne({_id:req.params.id});

    res.status(200).json({success:true,data:{}})
  } catch (err) {
    res.status(400).json({success:false});
  }
};
```

When a company is deleted, every Bookmark made to that company should also be deleted.

Resume Upload

The system shall allow a registered user to upload a PDF resume after registration.

models/User.js

```
resume: {  
  type: String,  
  default: null  
},
```

middleware/resume.js

```
const multer = require('multer');  
const path = require('path');  
const fs = require('fs');
```

```
const storage = multer.diskStorage({  
  destination: function (req, file, cb) {  
    const dir = 'uploads/resumes/';  
    if (!fs.existsSync(dir)) fs.mkdirSync(dir, { recursive: true });  
    cb(null, dir);  
  },  
  filename: function (req, file, cb) {  
    const ext = path.extname(file.originalname);  
    const filename = `resume-${req.user.id}${ext}`;  
    const filePath = path.join('uploads/resumes/', filename);  
  
    if (fs.existsSync(filePath)) fs.unlinkSync(filePath); // ✅ ลบไฟล์เก่าก่อน  
  
    cb(null, filename);  
  }  
});
```

Creates the folder uploads/resumes/ if it doesn't exist.

Renames the uploaded file to a consistent format.

If a file already exists, it deletes the old one.

```
const upload = multer({  
  storage,  
  fileFilter: (req, file, cb) => {  
    if (file.mimetype === 'application/pdf') cb(null, true);  
    else cb(new Error('Only PDF files are allowed'));  
  },  
  limits: {  
    files: 1 // ✅ ไม่ให้เกิน 1 ไฟล์  
  }  
});
```

Only allows PDF file.

Restricts to 1 file per upload.

```
module.exports = upload;
```

Resume Upload

The system shall allow a registered user to upload a PDF resume after registration.

controller/resumeController.js

```
const fs = require('fs');
const path = require('path');
const User = require('../models/User');

exports.updateResume = async (req, res) => { Saves the uploaded file path to the user's record..
  try {
    if (!req.file) {
      return res.status(400).json({ success: false, message: 'No file uploaded' });
    }

    const user = await User.findById(req.user.id);
    user.resume = req.file.path;
    await user.save();

    res.status(200).json({ success: true, message: 'Resume uploaded successfully', resume: user.resume });
  } catch (err) {
    console.error(err);
    res.status(500).json({ success: false, message: 'Upload failed' });
  }
};
```

```
exports.deleteResume = async (req, res) => { Deletes the file from disk and clears the path in the DB.
  try {
    const user = await User.findById(req.user.id);
    if (!user.resume) return res.status(400).json({ success: false, message: 'No resume to delete' });

    const resumePath = path.resolve(user.resume);
    if (fs.existsSync(resumePath)) fs.unlinkSync(resumePath);

    user.resume = null;
    await user.save();

    res.status(200).json({ success: true, message: 'Resume deleted' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ success: false, message: 'Failed to delete resume' });
  }
};
```

```
exports.getResume = async (req, res) => { Sends the PDF file back to the user.
  try {
    const filePath = path.join(__dirname, '..', 'uploads/resumes', `resume-${req.user.id}.pdf`);
    if (!fs.existsSync(filePath)) return res.status(404).json({ success: false, message: 'No resume uploaded yet' });

    res.sendFile(path.resolve(filePath));
  } catch (err) {
    console.error(err);
    res.status(500).json({ success: false, message: 'Failed to retrieve resume' });
  }
};
```


Resume Upload

The system shall allow a registered user to upload a PDF resume after registration.

routes/resume.js

```
const express = require('express');
const router = express.Router();
const { protect } = require('../middleware/auth');
const upload = require('../middleware/resume');

const {
  updateResume,
  deleteResume,
  getResume
} = require('../controller/resumeController');

router.put('/', protect, upload.single('resume'), updateResume);
router.delete('/', protect, deleteResume);
router.get('/', protect, getResume);

module.exports = router;
```

server.js

```
const resume = require('./routes/resume.js');
app.use('/api/v1/auth', resume);
```

```
// Global error handler – this should be placed **AFTER** all routes
app.use((err, req, res, next) => {
  console.error(err.stack);

  if (err.name === 'MulterError') {
    // e.g. too many files, unexpected field
    return res.status(400).json({
      success: false,
      message: `Upload error: ${err.message}`
    });
  }

  res.status(500).json({
    success: false,
    message: err.message || 'Server Error'
  });
});
```