# HandL Technical Description

---

Dhruv Gupta

**Project Proposal**

I would like to develop a project that allows people to track their day through a quick few things. The project is named HighsAndLows. Every day, they should be able to enter in a score from 1-10, a daily high, and a daily low.

They should be able to friend other users of the app and view their daily highs and lows, as well as charts/analytics about how they are doing generally. People should also be able to backtrack/fill out days they missed in a later time.

People will get formatted emails which recap their week (after they have used the platform for a whole week at least). These recaps should be generated with the help of LLMs.

**Proposed Technical Stack**

- **MongoDB + Express + Node**: We will intend to use MongoDB for storage of zips, agencies, locations, and partners.
- **Vercel:** Both the backend and frontend will be deployed on Vercel (in different deployments of course)
- **Up to you:** Some package that allows us to send texts every day around 10:30 PM to users to remind them to fill out for that day
- **MailJS:** For sending out emails that recap a person's week
- **OpenAI API:** We should use ChatGPT 4o-mini to generate these recaps above (given quite a lot of structure)
- **Cloudinary**: For image hosting
- **React:** The frontend will be all react. The app has been created using npx create-react-app to begin with.

**Backend Setup**

We have begun by creating a new react project. In the root of that, we have created a new folder called "backend". In the backend, we have run npm init -y

**User Model**

Username (String)
Name (String)

Phone Number (String)
Email (String)
Profile_Picture_URL (String)
Friends[] (Foreign Key)
Days[] (Foreign Key)

**Day Model**
Date (Date)
Score (Number)
High (String)
Low (String)

**Backend Development Steps**

These steps are to be followed by Cursor Agent running Claude 3.7 Sonnet. Each step should only be completed one at a time, and after each step is completed, the readme file should be updated accordingly. Do NOT go ahead at all and do not set up extra steps in advance

1. Set up the file directory and all necessary introductory files for the project. Install any necessary components.
2. Create and define our different file models
3. Build out authentication and sign up/login framework. Authentication should last for 1 year when done, and rely on email. Phone number is also required.
4. Build the controllers and routes needed to add and remove each of our models. Then, create routes to query the lists of them.
5. Build out framework and middleware necessary for sending texts, adding friends, viewing friends, etc
6. Build out framework and middleware necessary for uploading pictures to Cloudinary and then receiving back the link.
7. Build out framework for summarizing weeks and sending out emails about them using MailJS

**Frontend Development Steps**

1. Create a file directory with images, components, data and pages. Create a global API variable that is set and can be edited for where the server is hosted
2. Develop a header, footer, and landing page which explain what this project is about and how it works generally.
3. Develop a sign up/log in page which follows the authentication guidelines laid out above

4. Develop a dashboard from which users can view their own stats, fill out days they've missed and view previous days through a calendar feature and/or fill out the current day
5. Develop a tab on the dashboard through which users can manage friend requests or search for new friends
6. Develop a public view for each user which will show some aggregated stats as well as their 5 most recent days
7. Build a feed feature which will display the person's friend's "Days", with the most recent ones at the top. You should be able to view their profiles from there as well

**Frontend Considerations**

1. We want the frontend to be as clean and modern as possible, considering our target audience is 16-24 year olds. Take heavy inspiration from the UI of Notion
2. We want the frontend to feel responsive and provide micro feedback
3. This web application will most likely be used predominantly on mobile devices. Mobile compatibility is a top priority for this project.

**Necessary Keys**

```
JWT_SECRET=fde5b….
CLOUDINARY_CLOUD_NAME="djt4g…."
CLOUDINARY_API_KEY="55162….."
CLOUDINARY_API_SECRET="d6HCn….."
OPENAI_API_KEY="sk-proj-v7u…."
```

**Cloudinary Example File**

```
const express = require('express');
const router = express.Router();
const upload = require('../middleware/upload');
const { uploadImage } = require('../utils/cloudinary');
const { auth } = require('../middleware/auth');

// Handle image uploads
router.post('/image', auth, upload.single('image'), async (req, res) => {
  try {
    if (!req.file) {
      return res.status(400).json({
        success: false,
        error: 'No image file provided'
      });
    }

    // Validate file type
    const allowedTypes = ['image/jpeg', 'image/png', 'image/jpg', 'image/webp'];
    if (!allowedTypes.includes(req.file.mimetype)) {
      return res.status(400).json({
        success: false,
        error: 'Invalid file type. Only JPEG, PNG, and WebP images are allowed.'
      });
    }

    // Convert buffer to base64
    const b64 = Buffer.from(req.file.buffer).toString('base64');
    const dataURI = `data:${req.file.mimetype};base64,${b64}`;

    // Determine folder based on route or query param
    const folder = req.query.type === 'profile' ? 'profiles' : 'articles';
    console.log(`Uploading image to ${folder} folder`);

    // Upload to Cloudinary using our utility
    const imageUrl = await uploadImage(dataURI, folder);

    res.json({
      success: true,
      data: {
        url: imageUrl
      }
    });
  } catch (error) {
    console.error('Image upload error:', error);
    res.status(500).json({
      success: false,
      error: 'Failed to upload image. Please try again.'
    });
  }
});

module.exports = router;
```