# Making of Psychological Calendar

**Initiation Phase:**

Minimally: motivational quotes, daily mood tracking, daily journal entries, daily goal setting, and visualization of data on a calendar. Ideally retrospective analysis of positives and negatives throughout the day. – The main part was done, the following was dropped. Maximally (harder to implement): user accounts and adding data to the days in the past, with subsequent review of data logged in the past. Analysis of data stored in the database with weekly notifications of the past week's summary. – Adding/Reviewing data is possible, no analysis.

**Artifacts that were created and technologies used:**

Server.js – used Node.js and express to run a server, configured backend for efficient communication.

Index.html, register.html, login.html, new-password.html, reset-password.html – main pages.

Script.js, auth.js – first script covers the index.html and the other one covers authentication.

Styles.css – a single style-setting file that makes sure that the website has a unified look.

MongoDB database – contains all user information (username, email, passwords (hashed))

SQL database – contains all user-related data about their day

**Implementation Phase:**

1) Re-writing logged data into the SQLite database - now it is possible to rewrite the data in the already logged day. Reviewing of any day works as well, as well as correctly.
2) Hashed passwords (stored on MongoDB) for the sake of security - the passwords are now hashed using bcrypt. Improved security for the users.
3) Regex for checking the email format – regex was introduced to account for a capital letter, small letter, more than 8 characters and at least one number.
4) Server.js can be extended with try{}-catch{} sequences for common HTTP status codes – the statements appeared naturally where the code was prone to crashing,
5) The website is responsive enough, but maybe something is missing - the website is now mobile-first oriented, with two media queries – for a small phone and for anything above it until 1900px. After it, the application is supposed to take the central part of screen.
6) Add more console.log() statements for debugging – console.log debug items were mostly re-moved, due to the fact that they were no longer needed.

**Main lessons learned in the process:**

1) Setting up and running an Express server framework in Node.js, as well as importance of specifying the folder with source files using static file serving. The need for the backend functions to be asynchronous and to be able to await the data to be correctly handled before being processed. What is a REST API design and how the "app.post" and "app.get" facilitate the communication  between frontend and backend via POST/GET requests. Error handling on the server side and the need for it to avoid entire web application crashes.
2) The importance of having security features such as password hashing using bcrypt, as well as password-reset token generation via crypto library. How the password validation is configured and enforced with regex.
3) Externally created non-relational database setup and operation: connecting to the database via mongoose and being able to perform CRUD operations with the database.
4) Internally created relational database setup from scratch: using SQL in order to log/change the data in a relational database, as well as define schemas for them.
5) Handling of JSON packages in HTTP requests via bodyParser, which extracts and parses incoming payloads so that the data can be accessed via req.body in Express.
6) CORS issues that allow requests from specific domains, such as unauthorized access. This also involved an extensive research and conclusion of using a /proxy-quote, async function in "app.get" statement to resolve the issue when fetching data from external APIs.
7) Nodemailer library that can automatically handle working with emails and sending them from the application to the users with custom content.
8) Dynamically creating elements on page via DOM-manipulation (e.g generateCalendarHTML()). The hardest piece of the task was creating the calendar that would be able to adjust itself based on the month and days of the week. It was not possible to do it statically, inspiration was taken from here.
9) Fetching data from a local storage (e.g. loadEntries()). This function kept crashing or failing to work, so it was necessary to introduce the function "then"  on the "fetch" call to handle the response correctly, parsing the JSON received. Chaining "then" statements is extremely useful in async operations and allows for easier debugging.
10) Using „response" that is received after a „fetch()" API call in order to verify if the response was returned from the fetch call and if it was failed or was successful (response.ok).
11) The hardest part of working with the styles.css was by far setting the heights and widths of elements. Flexbox was revised as it is a very useful feature, although enabling media queries proved to be a harder task, so the styles were adapted to fit the smallest phone in firefox debug tool (IPhone SE) and higher resolutions were also checked to be well-fitted.