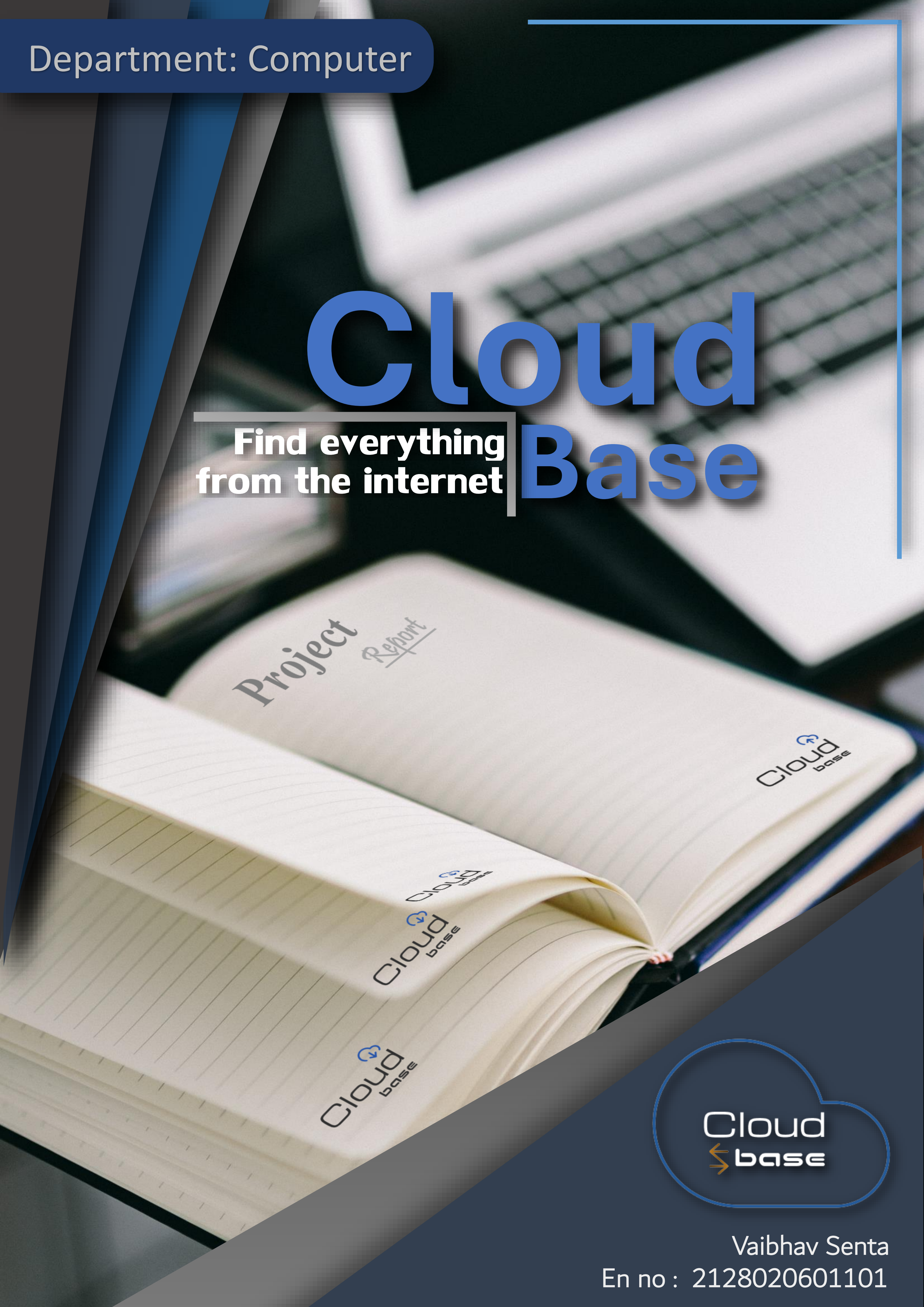
Department: Computer

Cloud Base

Find everything from the internet



Cloud base

Vaibhav Senta
En no : 2128020601101

Abstract

Cloud Base is a Node.js-based web application designed to provide a platform for users to upload, manage, and view movies. It includes functionalities like user authentication with JWT, movie management, and account management. The application uses MongoDB for storing user and movie data, Multer for handling file uploads, and JWT for managing user sessions securely. The website is user-friendly, enabling users to interact easily with the content, upload movies, and manage their profiles.

Introduction

Cloud Base is built to simplify movie management for users, allowing them to upload, view, and manage movies in a secure and organized way. The core functionalities of the website include:

- User Signup & Login: Allow users to create accounts, log in using JWT, and securely authenticate their sessions.
- Account Management: Users can edit their profiles and delete their accounts when needed.
- Movie Upload: Users can upload movies with metadata such as title, details, release date, etc., along with movie files and posters.

This report explains the system design, implementation process, libraries used, challenges faced, and the future scope of the project.

System Design

Architecture

The Cloud Base system follows a client-server architecture:

- Client Side: The client side is built using HTML, CSS, and EJS for rendering dynamic pages (like the homepage, movie upload forms, etc.). The front-end interacts with the server via HTTP requests.
- Server Side: The back-end is developed using Node.js with the Express framework, handling routing and managing HTTP requests.
- Database: MongoDB stores data related to users and movies, including metadata and file paths. It has models like movies and user which uses mongoose schemas like user scheme, login request details, delete request details, and movie schema. This schema contains metadata details about models.
- Authentication: The system uses JWT (JSON Web Token) for securely managing user authentication. JWT tokens are generated during login and used for verifying the user in subsequent requests.

Database Design

In MongoDB, a collection is a group of documents, similar to a table in relational databases. Each document has a unique _id field and can have different structures. Collections can be indexed for faster queries. MongoDB supports operations like inserting, querying, updating, and deleting documents and more. This database contains 4 models which are given below.

1) Deleted device details Collection
2) Login device detail Collection.
3) Movies Collection

4) User Collection

This all collections have their own schemas and databases in mongodb.

- User Collection
  - Fields: email, password, username, first name, last name, phone number, status (active/deleted), createdAt, profile picture path, database path, date of birth, country code, phone number, login device array (initially empty), signup device details & updatedAt.
  - Username, email, _if & phone number will be always unique.
  - It stores user detail at mongodb with this all fields and use it by mongodb queries to operations login, authentication, verification etc.
- Movie Collection:
  - Fields: title, description, title name, ucbid, release date, duration, cast, poster path, movie file path, available audio tracks, subtitle tracks, director, size, original file data, released in country, resolutions, uploadedBy, total visits, total ratings, total downloads, createdAt, updatedAt, total online streams etc.
  - Stores movie metadata along with the file paths for movie posters and movie files uploaded by users.
- Login device detail Collection:
  - Fields: _id, hostname, os username, is loginsuccessfull, os machine type, os name, os, os version, home directory, free system memory, total system memory, device Ip, Wi-Fi mac address, CPU architecture, CPU model, CPU speed, CPU cores, temporary directory location, Browser, Browser version, createdAt & updatedAt.
  - It stores all this information in the database as object. This basically tracks browser requests.
- Deleted device details Collection:
  - Fields: _id, hostname, os username, is loginsuccessfull, os machine type, os name, os, os version, home directory, free

system memory, total system memory, device Ip, Wi-Fi mac address, CPU architecture, CPU model, CPU speed, CPU cores, temporary directory location, Browser, Browser version, createdAt & updatedAt.

→ It stores all this information in the database as object. This basically tracks browser requests.

Flowcharts

- User Signup Flow:
  - → User enters details (email, password, etc.) on the signup page (Signup page contain all fields according to the user schema).
  - → After this, server will check that if user is already available or not.
  - → If user already available then user should login else user can create account with the email.
  - → Server validates inputs (checks email format, password strength).
  - → Now server will make a database document using the user schema, model and input data and redirect to the login page.
- User Login Flow:
  - → User will enter email and password to the login page and send those details to server by poste request.
  - → These details will be received by server and server will check the email and password with the database document.
  - → If the email found in the database, then process will be continued otherwise server will tell user to create an account.
  - → If the email found in database, then it will check if the password is correct or not.
  - → If the password is not correct then server tells user to enter correct password.
  - → If the password is right then a JWT token will be generated by the JWT library and it will send to the user with next response as a cookie and render the index.ejs page.
- Movie Upload Flow:

- → User will fill the form which contains all field according to movie schema and send it to server by using POST request.
- → Server will check if movie is uploaded or not
- → If it was uploaded then server tells the user to select different username.
- → If it was not uploaded then server will wrap this all movie-details to an object and send it as cookie with rendering files upload page.
- → Movie poster and movie file will be uploaded by this page and sent to the server by post request where it will handle by multer (File uploading library).
- → With this request server will take the movie data from cookies and they will be stored in MongoDB including file paths, and the server responds with a success message.

Implementation Workflow

User Signup Process

- User submits the signup form
    - → The user enters details like email, password, username, first name, last name, date of birth, country code, phone number and submits the form.
    - → A `POST` request is sent to the server's `/signup` route.
- Server receives the request
    - → The server receives the form data.
    - → It will check if the email is already in use or not.
    - → If the email is already in use, then server tells user to add different email.
    - → After this, server will check that the username is already taken or not.

- → If user name is already taken then user should enter unique username.
  - → Third condition is unique phone number
  - → If this all conditions are satisfied then server will ensure that all required fields are provided.
  - → Server will collect device details from the browser request like os name, os version, browser details, CPU details, host name, machine name, os name, home directory, total memory, free memory, temporary directory, and device Ip.
  - → This device details will be wrapped in object and added in the user object with name "SignupDeviceDetails".
- Store User in MongoDB
  - → The server creates a new user document in the MongoDB User by using mongodb queries and operations.
  - → This process is synchronising process so the callback function of the route is async function and database operation will wait for finish earlier processes.
  - → The user object will be passed in mongodb create operation and user will be redirected to the login page after completion of the process.

User Login Process (JWT Authentication)

- User submits login form
  - → The user enters their email and password and submits the form.
  - → A `POST` request is sent to the server's `/login` route.
- Server validates credentials
  - → The server checks if the email exists in the database.
  - → If the email is not found in the database, then server tells the user to create an account at CLOUD BASE.
  - → If email found in the database, then server will match the password with database.

- → If email and password both will match with database then the process will be forwarded.

- Generate JWT Token
    - → If the credentials are correct, the server generates a JWT token using jsonwebtoken.
    - → The token contains the user's email and username.
    - → This token will sign by using passkey so only server can modify it.
    - → The expiry of JWT token will set to eh 1h.
    - → After 1h JWT token will expire and user needs to login again at the website.
- Send JWT Token
    - → The token is sent as a cookie named as "logintoken" to the client for use in subsequent requests.
    - → By default, browser catch cookies and sent them with every request.
- Response
    - → If the login is successful, the user is redirected to the home page or the dashboard. The JWT token is stored in the user's cookies for session management.
    - → With every request, this login token will send to the server and every time server will verify the cookie data and expiration.
    - → After verification server find user with the email received from login token and add that at "req. tokenuser"
    - → If the cookie will not receive by the server or expired then as a response user will be logged out at next response and user need to login again to the website.

User Logout Process (Manually)

- User should be logged in
    - → First user should be logged in to the website
- Visit "/: profile" and send logout request
    - → User will visit to ":/profile" route which is a dynamic route for usernames

- This route is basically profile page which has 3 buttons logout, delete Account & manage Account.
- User will click on the logout button and this button will send request to server.

- Server will receive a GET request
  - The request sent by logout button is GET request to “:/profile/logout”.
  - By receiving this request, server will verify the login token and if the user is logged in then process will be continued.
- Cleare login cookies
  - If user logged in then server sent clear cookie response to the user that clear the cookies from the user browser.
  - And the user will be logged out.

User Delete Process

- Visit “/: profile/delete” by visiting “/: profile”
  - Visit at profile page which is located at “/: profile”
  - Click on the delete button which sent a GET request to the server.
- User should be logged in, check if logged out
  - Server will check that user is logged in or not
  - If the user is not logged in than process will be stope.
  - If the user is logged in then server will send a form that contain some details about your account.
- Fille the form and send request
  - Verify the details of your account and fill the form with account password.
  - This form will send a POST request to the server at button click
- Check if user already deleted and delete user
  - Now the server will check that the user is already deleted or not.
  - If user is mark as delete then server respond that user is already deleted from
- Verify password and delete user

- If user ins not deleted then server will verify the password with database and password match then change the user's account-state from "active" to "deleted"

User or Profile Update Process

- Visit to "/: profile/manage"
  - Visit to profile manage page by clicking on the manage account button which is GET request.
  - When you visit the manage page, server will get login token from "req. logintoken" and find the user by database process and collect the details like email, recovery email, phone number, date of birth, profile picture, first name, last name etc.
  - These details will be passed and sent with the manage page
- Visit to "/: profile/manage/edit"
  - Now, there is a button which sends GET request to the server
  - By clicking this button, server will send a form which contains input fields and select options to update the details.
- Fille the form
  - Select profile image if want to update and fille the detail that user wants to update.
  - On submit, this form sends POST request to the server with form data.
- Update mongodb document
  - This request will be received by the server.
  - Server scans the form and remove empty fields from req.body object by different types of processes.
  - Now, server will wrap all this details to another object by adding additional information.
  - After this, the server executes find and update process and pass the new object in the process
  - Now, profile data will be updated and server will send response to the user.

Movie Upload Process

- Submit movie details
  - → User will fill the form which contains all field according to movie schema and send it to server by using POST request.
  - → Server will check if movie is uploaded or not
  - → If it was uploaded then server tells the user to select different username.
  - → If it was not uploaded then server will wrap this all movie-details to an object and send it as cookie with rendering files upload page.
- User selects movie file and poster
  - → The user selects a movie file and its poster image.
  - → Metadata will receive with the response.
- Movie Data Submission
  - → The movie data and files are submitted to the server as a multipart/form-data request, handled by Multer.
- Server Checks for Existing Movie
  - → The POST request will send to the server.
  - → Movie file and movie poster will receive at "req.files"
  - → Server will receive movie details from cookie named "moviedetails" and store it to object.
  - → Now, server will collect some information from file like file-size, poster-path, database-path and other original data of the files.
  - → This information will be added to the moviedetails object
  - → Now, the files will be uploaded and details will be saved in database by creating new database document.
- File Upload Using Multer
  - → Multer handles the file upload:
  - → The movie file is stored in the `uploads/` directory on the server.
  - → The movie poster image is also uploaded and stored in the `uploads/` folder.
- Store Movie Metadata

- → The movie metadata, including the file paths (for both the movie file and poster), is stored in the Movie collection in MongoDB.
- Success Response
  - → Once the movie is uploaded and stored, the server responds with a success message (e.g., "Movie Uploaded Successfully").

Challenges Faced

- Large File Handling: Uploading large movie files posed a challenge. Multer was used to handle multipart uploads, and file size limits were enforced to prevent excessive storage use.
- JWT Token Security: Managing JWT tokens securely is crucial for preventing unauthorized access. Tokens expire after a certain period, requiring users to log in again. We implemented a secure token storage mechanism via cookies and used HTTPS to protect sensitive data.
- User Data Validation: Ensuring that user input (e.g., email, password) is valid required thorough input validation to prevent invalid data from being entered into the system.

System Requirements

Software Requirements

→ Node.js
→ MongoDB
→ Express
→ Multer
→ jsonwebtoken
→ cookie-parser (v1.x or higher)

Hardware Requirements

- CPU: Dual-Core Processor (minimum)
- RAM: 4GB or higher
- Storage: 50GB or more (for large movie file storage)

NPM libraries used in

This node.js project uses Express with Node.js server-side language. There are some more things that are helpful to built this and make it easier. Npm (Node Package Manager) is a package manager for JavaScript, primarily used for managing libraries and dependencies in Node.js projects. It allows developers to install, update, and manage third-party packages or modules, which can be reused in their applications. npm also helps manage scripts and automation tasks, making it easier to set up, configure, and run projects. It comes bundled with Node.js and provides access to a vast registry of open-source packages through the command line. The npm packages used in this project is given below...

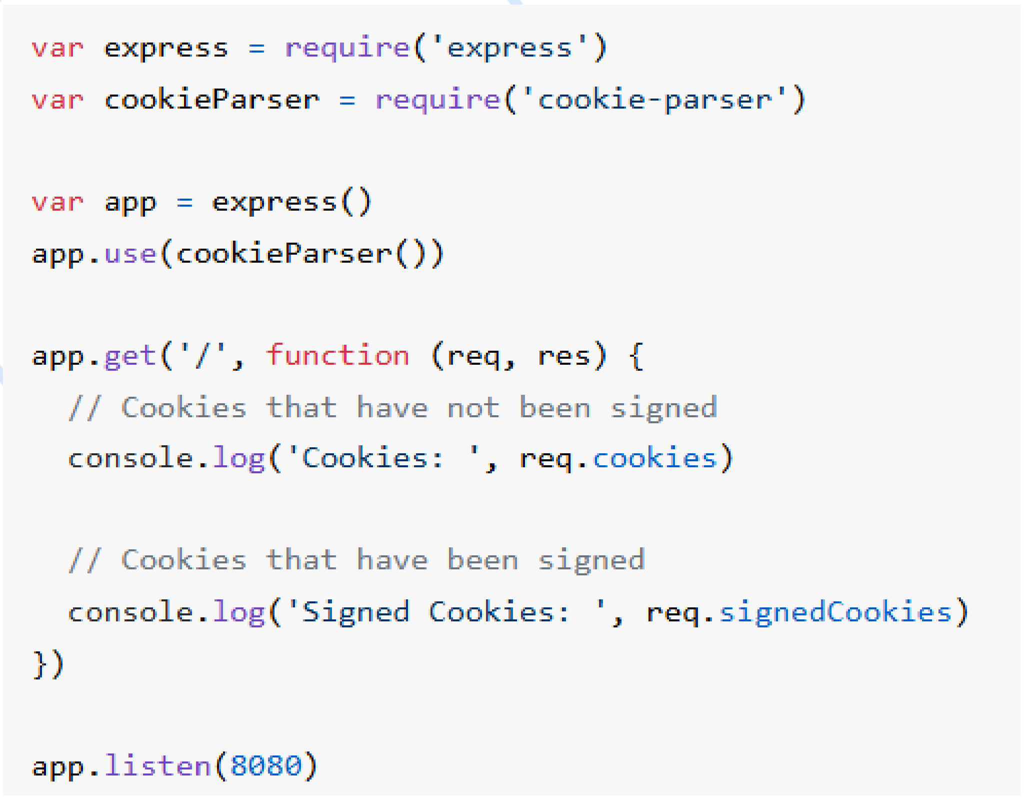
- cookie-parser
- ejs
- express
- jsonwebtoken
- mongoose
- multer
- os
- path
- short-unique-id
- useragent

cookie-parser

Parse Cookie header and populate req.cookies with an object keyed by the cookie names. Optionally you may enable signed cookie support by passing a secret string, which assigns req.secret so it may be used by other middleware. You can pass a secret key to the middleware to sign cookies, providing additional security (e.g., req.signedCookies for signed cookies). This middleware is commonly used in applications that need to manage user sessions or store small pieces of data in cookies.

Installation: npm install cookie-parser
API: var cookieParser = require('cookie-parser')
Use: app.use(cookieParser());

Example:



```
var express = require('express')
var cookieParser = require('cookie-parser')

var app = express()
app.use(cookieParser())

app.get('/', function (req, res) {
  // Cookies that have not been signed
  console.log('Cookies: ', req.cookies)

  // Cookies that have been signed
  console.log('Signed Cookies: ', req.signedCookies)
})

app.listen(8080)
```

ejs

EJS (Embedded JavaScript) is a templating engine for Node.js and Express that allows you to embed JavaScript code into HTML templates. It helps generate dynamic HTML content by rendering data on the server-side before sending it to the client. EJS enables the creation of reusable

HTML components with logic (loops, conditionals) embedded directly into the template. Allows you to embed JavaScript directly within HTML using <%= %> for output and <% %> for logic (e.g., loops, conditionals). You can pass data from your server-side code (like Express routes) to the template for rendering dynamic content.

Installation: npm install ejs

Example:

```
<ul>
  <% users.forEach(function(user){ %>
    <%- include('user/show', {user: user}) %>
  <% }); %>
</ul>
```

express

The Express philosophy is to provide small, robust tooling for HTTP servers, making it a great solution for single page applications, websites, hybrids, or public HTTP APIs.Express does not force you to use any specific ORM or template engine. With support for over 14 template engines via Consolidate.js, you can quickly craft your perfect framework.

Installation: npm install express

Example:

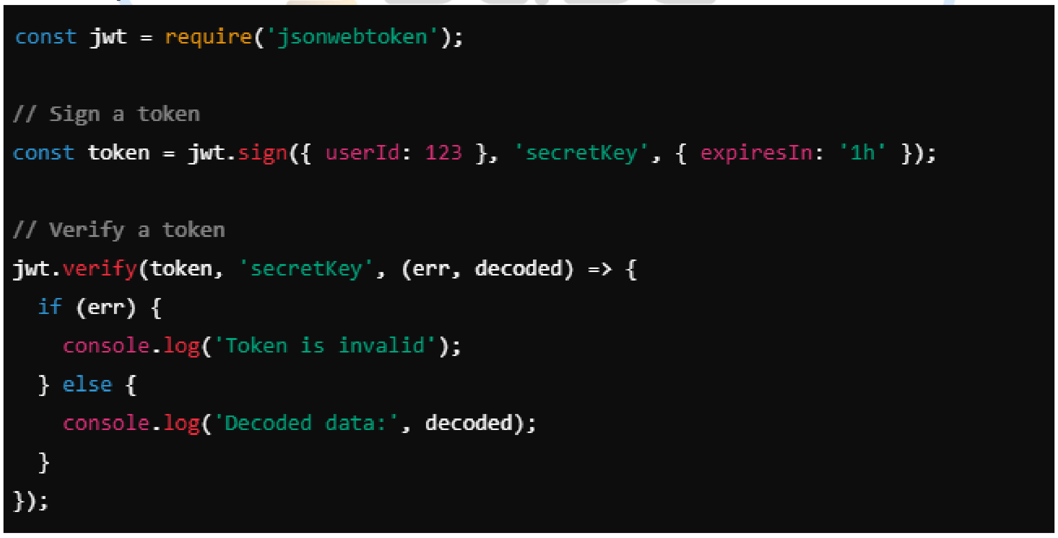
```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

jsonwebtoken

JSON Web Tokens (JWT) are a compact, URL-safe way to represent claims (data) between two parties, commonly used for authentication and authorization in web applications. A JWT consists of three parts: a header (containing metadata), a payload (carrying the claims or data), and a signature (used to verify the integrity of the token). Once a user logs in, the server generates a JWT and sends it to the client, which stores it and sends it in subsequent requests. The server verifies the token and grants access to protected resources if valid. JWTs are stateless and self-contained, meaning they don't require server-side session storage, making them scalable and ideal for microservices. They can also carry user roles or permissions for authorization. Popular libraries like jsonwebtoken make it easy to create and verify JWTs in Node.js. However, it's important to secure the secret key and use HTTPS to protect tokens during transmission.

Installation: npm install jsonwebtoken

Example:



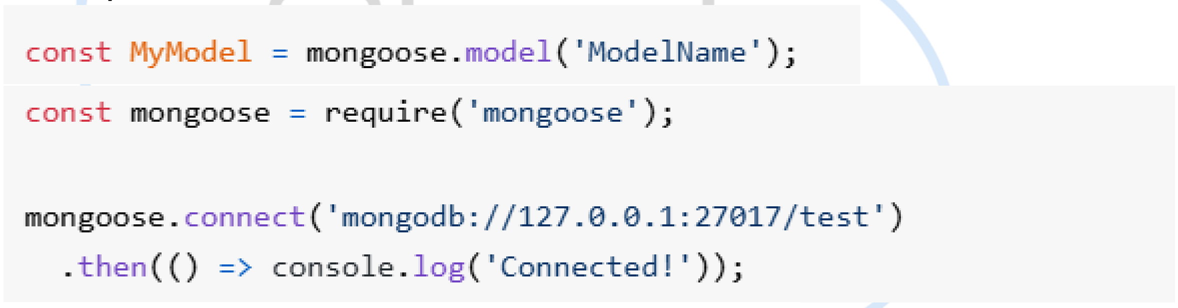
```
const jwt = require('jsonwebtoken');

// Sign a token
const token = jwt.sign({ userId: 123 }, 'secretKey', { expiresIn: '1h' });

// Verify a token
jwt.verify(token, 'secretKey', (err, decoded) => {
  if (err) {
    console.log('Token is invalid');
  } else {
    console.log('Decoded data:', decoded);
  }
});
```

mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js that provides a higher-level abstraction for interacting with MongoDB databases. It allows developers to define schemas for their data, providing structure and validation to the otherwise schema-less MongoDB collections. With Mongoose, you can define models based on these schemas, which represent documents in a MongoDB collection. It simplifies tasks such as querying, data validation, and data manipulation by providing an easy-to-use API. Mongoose also supports middleware, enabling developers to define hooks for actions like document validation or saving. It integrates seamlessly with MongoDB and is widely used in Node.js applications to manage database interactions in a more organized and efficient manner.

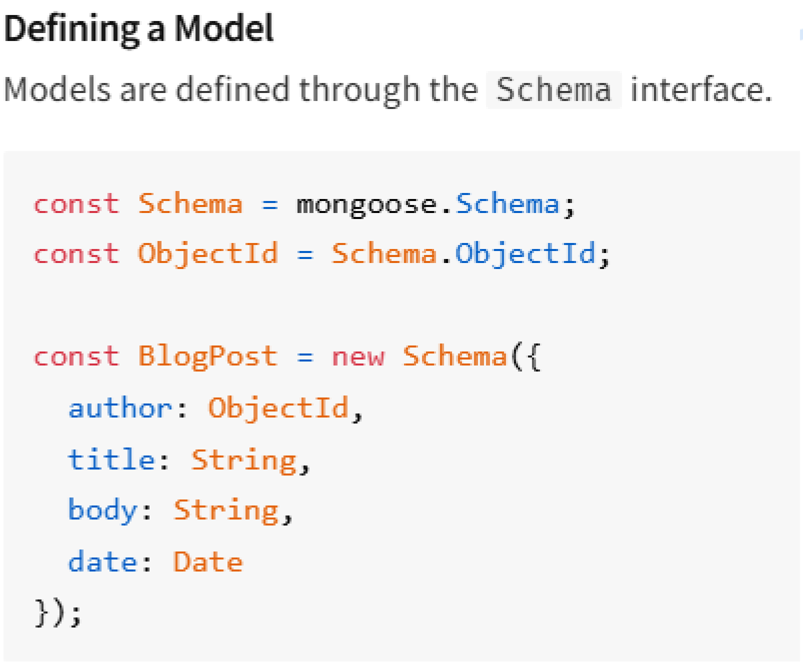
Installation: npm install mongoose

Example:



```
const MyModel = mongoose.model('ModelName');
```

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

mongoose.connect('mongodb://127.0.0.1:27017/test')
  .then(() => console.log('Connected!'));
```



Defining a Model

Models are defined through the `Schema` interface.

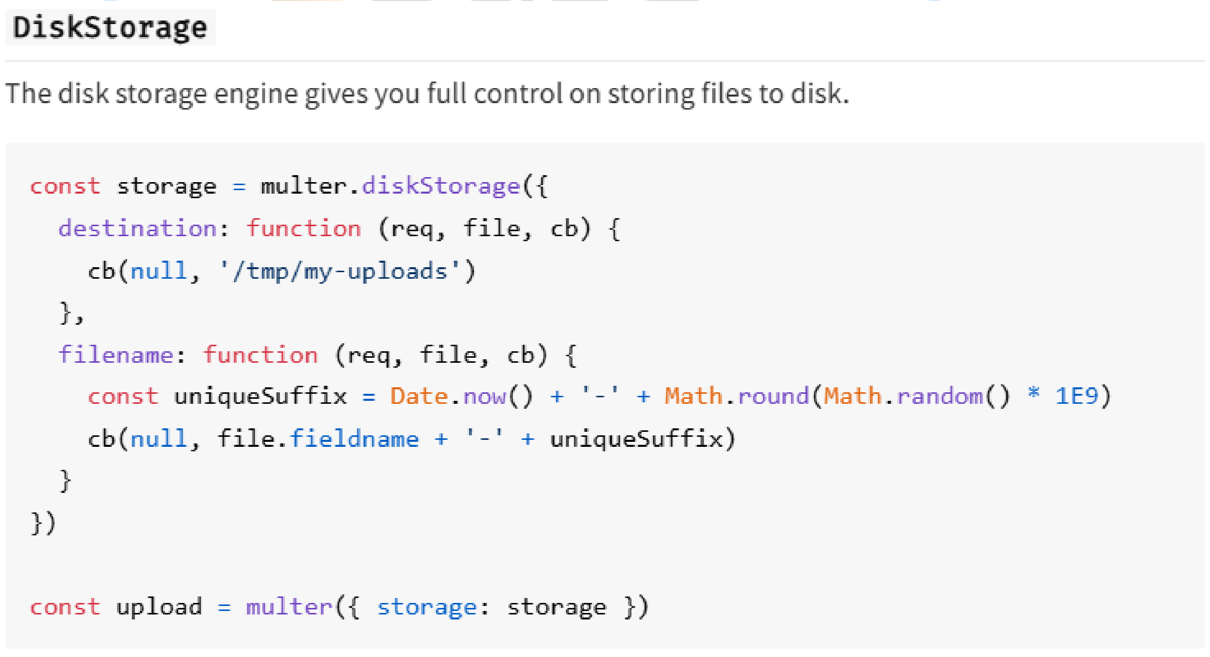
```
const Schema = mongoose.Schema;
const ObjectId = Schema.ObjectId;

const BlogPost = new Schema({
  author: ObjectId,
  title: String,
  body: String,
  date: Date
});
```

multer

Multer is a middleware for handling multipart/form-data, which is commonly used for uploading files in Node.js applications. It is typically used with Express to simplify the process of receiving files from HTTP requests. Multer processes incoming form data, storing the uploaded files either in memory or on disk, depending on the configuration. It allows developers to define custom storage locations and manage file naming conventions. Additionally, Multer provides the ability to validate files by setting constraints such as file type, size limits, and more. It is easy to integrate into an existing application and offers features like support for multiple file uploads and the ability to handle large file transfers efficiently. Multer is widely used in web applications that require file upload functionality, such as for image galleries, document management systems, and media platforms.

Installation: npm install multer

Example:



`DiskStorage`

The disk storage engine gives you full control on storing files to disk.

```
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, '/tmp/my-uploads')
  },
  filename: function (req, file, cb) {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9)
    cb(null, file.fieldname + '-' + uniqueSuffix)
  }
})

const upload = multer({ storage: storage })
```

Os

The os module in Node.js is a built-in module, meaning it comes with Node.js by default and does not require installation via npm. It provides a set of operating system-related utility methods for retrieving information about the computer system, such as CPU details, memory usage, network interfaces, and the operating system platform.

Built in module

Path

The path module in Node.js is a built-in module used to work with and manipulate file and directory paths. It provides utilities to handle different aspects of file paths in a platform-independent way, such as joining, resolving, or normalizing paths, which is especially useful for cross-platform applications (Windows, Linux, macOS) where file path formats differ.

Installation: npm install path

Const path = require('path')

short-unique-id

short-unique-id is a lightweight npm package designed for generating short, unique, and random strings. It's commonly used in applications that require concise, URL-safe identifiers, such as generating user session tokens, random keys, or short links. The package provides an easy way to create IDs with customizable lengths, which can include alphanumeric characters and be used in various scenarios, from database keys to temporary tokens.

Installation: npm install short-unique-id

Example:

```
//Instantiate
const uid = new ShortUniqueId();

// Random UUID
console.log(uid.rnd());
```

```
// Node.js require
const ShortUniqueId = require('short-unique-id');
```

useragent

useragent is a npm package used to parse and analyze User-Agent strings, providing detailed information about the client's environment. User-Agent strings are sent by browsers or other HTTP clients to identify themselves, and they contain data about the operating system, browser type, version, device, and more. The useragent package helps developers extract and interpret this information in a structured way, allowing for better handling of user requests based on their browser or device.

Installation: npm install useragent

Example:

```
var agent = useragent.parse(req.headers['user-agent']);
```

```
var useragent = require('useragent');
```