**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ**

**КАФЕДРА КОМП’ЮТЕРНИХ НАУК**

# КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

**на тему:**

**«Інформаційна технологія проектування системи розповсюдження відео контенту»**

**Завідувач**

**випускаючої кафедри** **Довбиш А.С.**

**Керівник роботи** **Берест О. Б.**

**Студента групи ІН м.-02** **Яскевич Б.О.**

**СУМИ 2021**

# ЗМІСТ

ЗМІСТ ........................................ 2
1. ОГЛЯД АНАЛОГІВ ТА ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-СЕРВЕРІВ ........................................ 3
1.1 Огляд аналогів ........................................ 3
1.2 Огляд існуючих програмних рішень для розробки WEB - сервера .. 4
2. ВИБІР МЕТОДІВ РІШЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ ........................................ 15
2.1 Вибір Framework для створення веб-застосунку ........................................ 15
2.2 Вибір бази даних для веб-застосунку ........................................ 16
2.3 Постановка задачі ........................................ 16
3. ДИЗАЙН МАЙБУТНЬОГО ДОДАТКУ ........................................ 19
4. ТЕОРЕТИЧНІ ЗАСАДИ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ 25
4.1 Буферизація та потоки ........................................ 25
4.2 Шифрування ........................................ 28
4.3 JWT-токен ........................................ 29
5. РОЗРОБЛЕННЯ ТА ІМПЛЕМЕНТАЦІЯ ........................................ 31
5.1 База даних ........................................ 31
5.2 JWT токен ........................................ 36
5.3 Шифрування відеозапису ........................................ 37
6. ТЕСТУВАННЯ СЕРВІСУ ........................................ 39
6.1 Jest бібліотека для тестування ........................................ 40
ВИСНОВКИ ........................................ 43
СПИСОК ЛІТЕРАТИ ........................................ 44

# ВСТУП

Відео сервіс - це ресурс, на якому користувачі можуть переглядати відео які були створені контент мейкерами, або компаніями які мають змогу знімати фільми та серіали. На даний момент під час самоізоляції ринок таких сервісів є одним з швидко зростаючих в світі. Під час локдауну потрібно скоротити перебування з іншими людьми, тому краще придбати підписку лежачі в себе дома на дивані, ніж ризикувати своїм здоров'ям та сходити в кінотеатр.

Водночас для авторів існує ризик що їх контент може бути скопійований в наслідок хакерської атаки, та викладений в мережу інтернет без їх згоди. Будь який веб-сервіс розміщується на веб-сервері.

Веб-сервер - це комп'ютер під керуванням операційною системою водночас підключений до серверної бази. Будь-яка вразливість у програмах, базі даних, операційній системі чи мережі призведе до атаки на веб-сервер.

Робота присвячена розробці архітектури веб-серверу, бази даних з допомогою якої можна буде зберігати та швидко діставати данні, також потрібно шифрувати файли що будуть знаходитися на цьому відео сервісі, за допомогою алгоритму шифрування який буде забезпечувати збереження даних без ризику бути розшифрованими.

Для роботи веб-серверу потрібно обрати архітектуру для розробки, налаштувати фреймворк та базу даних, підключитися до бази даних, створити endpoints для створення юзера, відео контенту який при потраплянні на сервер буде шифруватися, реалізувати механізм підписок та лайків з коментарями. Реалізувати авторизацію та аутентифікацію за допомогою JWT токену який буде шифрувати в собі деякі данні про користувача.

# 1. ОГЛЯД АНАЛОГІВ ТА ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-СЕРВЕРІВ

## 1.1 Огляд аналогів

YouTube - це платформа що належить компанії Google, надає змогу поширювати відео підписникам, та стежити за творчою діяльністю автора. Запустили даний сервіс 14 лютого 2005 року Чад Херлі, Стів Чен та Джавед Карім. Цей ресурс посідає друге місце за кількістю відвідування веб-сайтів відразу після пошукового сервісу Google. На відео сервіс YouTube заходять понад мільярд користувачів щомісяца, які в свою чергу переглядаю дуже велику кількість контенту щодня. У жовтні 2006 року Google купив право власності на YouTube. Після цього сервіс змінив свою бізнес модель, він більше не приносить дохід тільки з реклами, тепер пропонується платний контент наприклад фільми та ексклюзивний контент. YouTube і схвалені автори беруть участь у програмі Google AdSense, яка приносить більше доходу для обох сторін. З тих пір він перетворився з невеликої платформи для потокового відео до великого сервісу із заявленим доходом у 19,8 мільярдів доларів у 2020 році. З моменту придбання компанією Google YouTube вийшов за межі веб-сайту на мобільні додатки, мережеве телебачення та можливість підключення до інших служб. Категорії відео на сервісі включають музичні відео, відеокліпи, новини, художні фільми, короткометражні фільми, документальні фільми, тизери, аудіозаписи, трейлери фільмів, прямі трансляції, відеоблоги тощо. Більшість контенту створюють окремі особи. Це включає співпрацю між ютуберами та корпоративними спонсорами. З 2015 року створені медіа-корпорації, такі як Disney, ViacomCBS та WarnerMedia, створили та розширили свої корпоративні канали YouTube, щоб рекламувати ширшу аудиторію.

Netflix, Inc. - американська компанія, що займається продюсуванням та росповсюдженням відео контенту, створена 29 серпня 1997 року, пропонує бібліотеку фільмів і телесеріалів. Також розроблює власну продукцію, відому як Netflix Originals. Станом на жовтень 2021 року Netflix має понад 214 мільйонів передплатників. Сервіс доступний у всьому світі, за винятком материкового Китаю (через місцеві обмеження), Сирії,

Північної Кореї та Криму (через санкції США). Netflix відіграє визначну роль у незалежному кінопрокаті та є членом Асоціації кіно (MPA).

## 1.2 Огляд існуючих програмних рішень для розробки WEB - сервера

### Node JS

Node.js — це програмне середовище виконання JavaScript із відкритим вихідним кодом, що використовує технологію движку V8 та виконує код на мові JavaScript за межами веб-браузера. Node.js надає змогу розробникам програмного забезпечення використовувати JavaScript для створення сценаріїв на стороні сервера — це потрібно для того щоб була можливість динамічного змінювати вміст веб-сторінки, перед тим як данні будуть надіслані у веб-браузер користувача. Node.js являє собою парадигму JavaScript всюди, тобто розроблювати на стороні сервера та клієнта на одній мові програмування.

Стандартним розширенням для коду JavaScript є .js, назва "Node.js" не відноситься до конкретного файлу проте також має розширення .js проте в цьому контексті є назвою продукту. Node.js має архітектуру, керовану подіями, з можливістю асинхронного введення-виведення. Такі інструменти сприяють оптимізації масштабованості та пропускної здатності у веб-серверах з багатьма операціям, а також для веб-сервререів які працюють в реальному часі (наприклад, комунікаційні програми в режимі реального часу та браузерні ігри).

Механізм розпаралелювання потоків в JavaScript працює за рахунок циклу подій (event loop), який відповідає за виконання коду, збору і обробки подій і виконання під-задач з черги. Ця модель дуже відрізняється від інших мов програмування, таких як Java або C.

Проект розподіленої розробки Node.js раніше керувався Node.js Foundation, а тепер об’єднався з JS Foundation, щоб утворити OpenJS Foundation, якому сприяє програма Collaborative Projects Linux Foundation. Корпоративні користувачі які в своїх сервісах використовують Node.js включають Netflix, Groupon, Yahoo!, Rakuten, SAP, Walmart, GoDaddy, IBM, Microsoft, LinkedIn, PayPal, Voxer, і Amazon Web Services.

Приклад роботи традиційного сервера, та сервера який написаний на Node JS (Рис. 1)

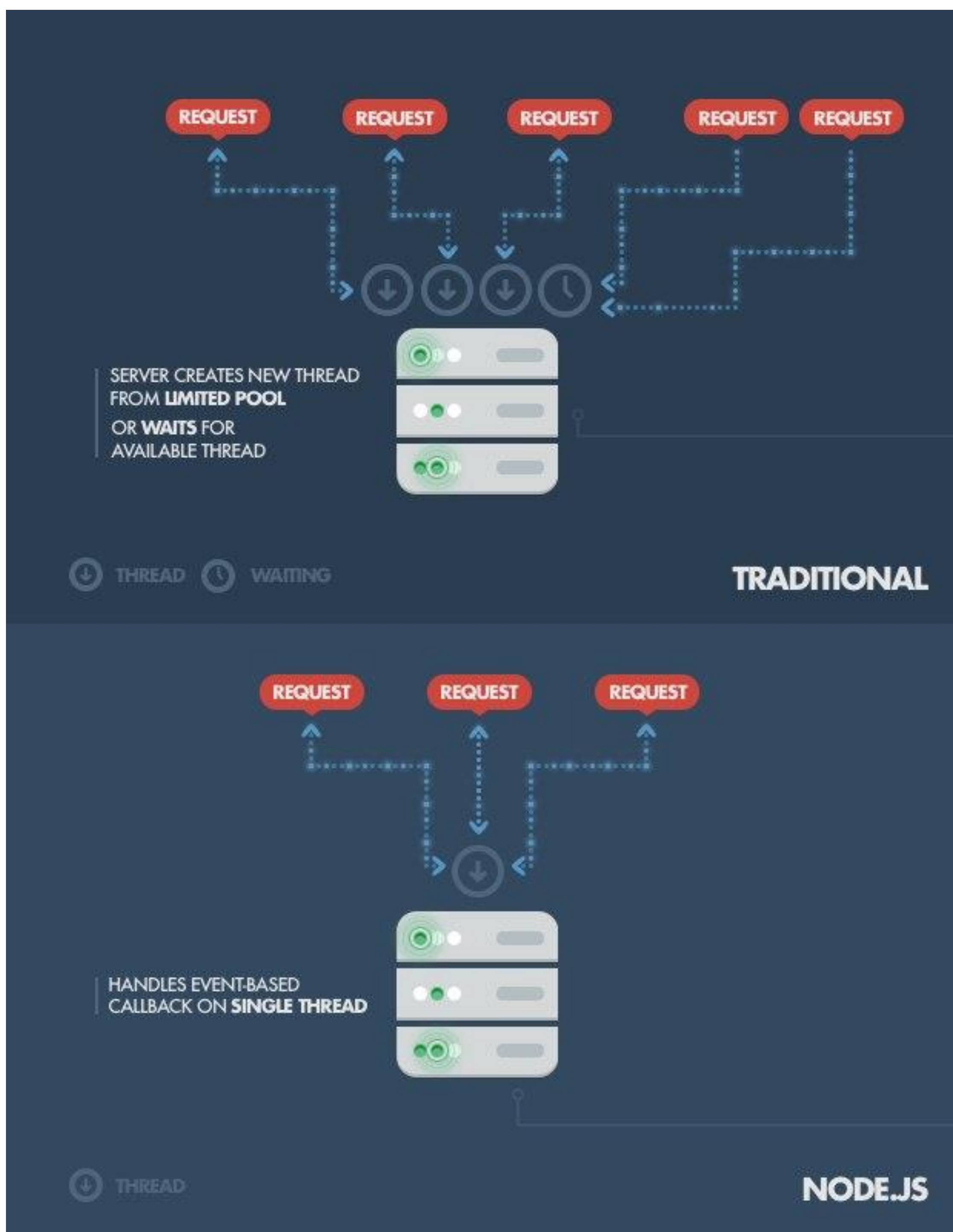


Рисунок 1 – Принцип роботи традиційного сервера і сервера написаного на Node JS

Node JS з точки зору веб-серверної розробки має ряд переваг:

- Сервер або декстопний додаток буде написаний на звичайному JavaScript, це означає що не потрібно втрачати час для вивчення окремої мови щоб створити додаток для браузера.
- Node JS має чудову продуктивність, він був розроблений для того щоб оптимізувати пропускну здатність та гнучку масштабованість веб-застосунків.
- Пакетний менеджер Node – це інструмент без якого було би дуже складно розробляти додатки, він дозволяє поширювати та завантажувати модулі які є самі по собі інструментами зі своїми внутрішніми зв'язками, тож npm налаштовує їх щоб користувач мав змогу одразу після завантаження користуватися модулем в повній мірі не чіпаючи уже існуючу екосистему додатка.
- Node портативний на всі платформи, має версію для Linux, Microsoft Windows та MacOS.
- Активне ком'юніті, мова розвивається, вводяться нові стандарти які покращують життя розробника та швидко виправляють критичні помилки.

## HTTP модуль

HTTP інтерфейси в Node JS мають мету підтримувати багатьох функцій протоколу, які в традиційному було важко використовувати. Наприклад, великі повідомлення або закодовані по частинам. Даний інтерфейс намагається не буферизувати одразу цілий запит або відповідь, у користувача для цього є можливість передавати данні в потоці.

Щоб підтримувати всі можливості HTTP-додатка, інтерфейс в HTTP є дуже низько рівеневим. Займається він тільки тим що обробляє потоки і аналізує повідомлення які приходять.

Повідомлення розбивається на дві частини. Сирі заголовки в тому виді в якому вони прийшли в повідомленні, зберігаються в масиві заголовків. А потім сервер зчитує заголовки, які сигналізують йому що потрібно робити з тілом запита.

Якщо ви хочете добавити в свій веб-застосунок підтримку різних HTTP методів, ваш сервер відавав статичні файли або використовувати шаблони для створення динамічних відповідей, то тоді потрібно весь код

писати самим, проте можна відмовитися від цього використовуючи фреймоврк.

Приклад роботи HTTP серверу (рис. 2).

```
// Завантажуемо HTTP модуль
const http = require("http");

const hostname = '127.0.0.1';
const port = 8000;

// Створюємо HTTP-сервер
const server = http.createServer( requestListener: (req :IncomingMessage , res :ServerResponse ) => {

  // Встановлюємо HTTP-заголовок відповіді з HTTP статусом и Content type
  res.writeHead( statusCode: 200,  headers: {'Content-Type': 'text/plain'});

  // Відсилаємо тіло відповіді "Hello World"
  res.end( chunk: 'Hello World\n');
});

// Виводимо в консолі коли сервер запущено
server.listen(port, hostname,  backlog: () => {
  console.log(`Server running at http://${hostname}:${port}/`);
})
```

Рисунок 2 – HTTP сервер на Node JS

Переваги:

- HTTP модуль є влаштований в Node, тобто його не потрібно завантажувати, також він підтримується офіційно розробниками Node JS.
- Дозволяє керувати сервером на достатньо низькому рівні.

Недоліки:

- Не має власної екосистеми, тому потрібно налаштовувати кожний елемент серверу самому розробнику.
- Модуль HTTP складний в масштабування та підтримки.

- Вважається застрілим в продакшені.

## Express Framework

Express – найпопулярніший веб-фреймворк для Node. Він є базовою бібліотекою для інших популярних веб-фреймворків Node. Він надає такі механізми:

- Написання обробників для запитів з різними HTTP-методами у різних URL-адресах (маршрутах).

- Інтеграцію з механізмами рендерингу "view", для створення відповідей, вставляючи дані в шаблони.

- Встановлення загальних параметрів веб-застосунка, таких як порт для підключення, та розташування шаблонів, які використовуються для відображення відповіді.

- "проміжне ПЗ" для додаткової обробки в будь який момент часу у конвеєрі обробки запитів.

У той час як сам Express досить мінімальний, розробники створили сумісні пакети проміжного програмного забезпечення для того щоб вирішувати проблеми які можуть виникнути під час розробки додатка. Існують вже готові рішення для роботи з файлами сеансами, параметрами URL, входами користувачів, даними POST,  заголовками безпеки та багатьма іншими.

Фреймворки часто прийнято ділити на "обмежуючі" та "не обмежуючі". Обмежують фреймворки вважаються фреймворки, які випливають " належним " обмеженням під час виконання окремих завдань. Досить часто вони орієнтовані на прискорену розробку в конкретній галузі, оскільки належний підхід до довільно обраного завдання буває не простим для розуміння і погано документований. При цьому вони позбавляються гнучкості при вирішенні завдань, що виходять за сферу їх звичайного застосування, а також виявляють тенденцію до обмеження вибору компонентів та підходів свого застосування.

Навпаки, фреймворки, що не обмежують, мають набагато менше обмежень для зв'язку компонентів, щоб досягти мети або обмежень у виборі використовуваних компонентів. Вони полегшують розробникам використання найбільш підходящих інструментів для виконання конкретного завдання, але платою за це буде те, що ви самостійно маєте знайти такі компоненти. Express не обмежує. Ви можете вставити в ланцюжок обробки запитів будь-які сумісні проміжні компоненти, які вам подобаються. Ви можете структурувати програму в одному файлі або в декількох, використовуючи будь-яку структуру каталогів.

У традиційних динамічних веб-сайтах, веб-застосунок очікує HTTP-запиту від веб-браузера (або іншого клієнта). Коли запит отримано, програма визначає, яку дію необхідно виконати на основі URL шаблону і можливо пов'язаної інформації, що міститься в даних POST або GET. Залежно від того, що потрібно, Express може потім читати або записувати дані з бази даних або виконувати інші завдання, відповідно до отриманого запиту. Потім програма повертає відповідь у веб-браузер, часто динамічно створюючи HTML-сторінку для відображення браузером, вставляючи витягнуті дані в заповнювачі HTML шаблону.

В Express є підтримка HTTP методів, що дозволяють вказати, яка функція викликається для конкретного HTTP запиту POST, GET, SET, або інші, і URL шаблон, а також методи, що дозволяють вказати, який механізм шаблону використовується, де знаходяться шаблони файлів і який шаблон використовуватиме виведення відповіді. Ви можете використовувати Express проміжні функції для додавання підтримки файлів, сеансів, cookie, і користувачів, отримання POST/GET параметрів. Ви можете використовувати будь-який механізм бази даних, який підтримується Node.

Приклад роботи серверу написаного за допомогою Express (рис. 3)

```
// Завантажуэмо Express модуль
const express = require('express');

// Створюємо express додаток
const app = express();

// Чекаємо GET запити
app.get('/', function(req, res) {
  res.send('Hello World!');
});

// Прослуховуємо порт додатка
app.listen(3000, function() {
  console.log('Example app listening on port 3000!');
});
```

Рисунок 3 – Ініціалізація серверу за допомогою Express

Переваги:

- Express має хорошу документацію.
- Багато проектів розроблені на цьому фреймворку, тому ком’юніті дуже розвинуте, можна завжди знайти уже готові рішення проблем які виникнуть під час розробки.
- Містить в собі вже готові інструменти які не потрібно буде встановлювати окремо.

Недоліки:

- Все тіло запиту проходить через проміжні функції, може вплинути на продуктивність серверу.
- Складно знайти помилку в ланцюгі проміжних функцій.

## Nest JS Framework

Першим супер популярним веб-фреймворком для Node.js був express.js. Nest.js значно розширює його функціональність, додає декларативності, а також допомагає розробнику будувати програму відповідно до кращих архітектурних практик.

В останні роки, завдяки Node.js, JavaScript став лінгва-франка Інтернету як для передніх, так і для серверних додатків. Це породило чудові

проекти, такі як React, Angular і Vue, які підвищують продуктивність розробників і дозволяють створювати швидкі, тестовані та розширювані інтерфейсні додатки. Проте, незважаючи на те, що для Node існує безліч чудових бібліотек, допоміжних засобів та інструментів, жодна з них ефективно не вирішує головну проблему — архітектури. Nest надає готову архітектуру додатків, яка дозволяє розробникам і командам створювати добре тестовані, масштабовані, слабо пов'язані та легко обслуговувані програми. Архітектура в значній мірі натхнена Angular.. Також обов'язковою частиною Nest JS є TypeScript.

TypeScript - це мова програмування, в якій виправлено багато недоліків JavaScript. Код на TypeScript виглядає майже так само, як і код на JS, і якщо у вас є досвід в інших мовах програмування, вивчити TypeScript досить просто. Особливо з огляду на те, що ви можете писати JS-код прямо в TS-скриптах. TypeScript доповнює JS тим що додає нормальну типізацію та розширює ООП.

Приклад написання серверу за допомогою Nest JS (рис. 4)

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

const PORT = process.env.PORT || 8080;

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  await app.listen(PORT, callback: () => {
    console.log(`Server has been started on PORT ${PORT}`);
  });
}
bootstrap().catch((e) => console.log(e));
```

Рисунок 4 – Ініціалізація серверу за допомогою Nest JS

Переваги:

- Nest має хорошу документацію.
- Nest перейняв всі плюсі Express, так як він побудований навколо нього.
- Має налаштований фреймоворк для тестування, та сам по собі легко тестується.
- Його архітектура дозволяє легко маштабовуватися та не втрачати якість написаного коду.

Недоліки:

- Високий поріг входу.
- Архітектура може здатися занадто громіздкою.

## Postgre SQL

PostgreSQL є однією з найпопулярніших систем управління базами даних. Сам проект postgresql еволюціонував з іншого проекту, який називався Ingres. Формально розвиток postgresql почався ще 1986 року. Тоді він називався POSTGRES. А в 1996 році проект був перейменований на PostgreSQL, що відображало більший акцент на SQL. І саме 8 липня 1996 року відбувся перший реліз товару. З того часу вийшло безліч версій postgresql. Поточною версією є версія 14.

Однак, регулярно також виходять підверсії. PostgreSQL підтримується всім основних операційних систем - Windows, Linux, MacOS.

Надійність PostgreSQL є перевіреним та доведеним фактом та забезпечується такими можливостями:

- повна відповідність принципам ACID - атомарність, несуперечність, ізольованість, збереження даних.
- MVCC означає, зроблена транзакція бачить копію даних які були на початку транзакції, не дивлячись на те, що стан бази вже міг змінитися. Це захищає транзакцію від того що дані були змінені не узгоджено, така поведінка могла бути викликана конкурентною транзакцією, та забезпечує ізоляцію транзакцій.
- Write Ahead Logging – механізм логування транзакцій, що системі відновитися після можливих збоїв.
- Point in Time Recovery – можливість відкату бази даних на будь-який момент у минулому.

- Цілісність даних є серцем PostgreSQL. Крім MVCC, PostgreSQL підтримує цілісність даних лише на рівні схеми -  це обмеження та зовнішні ключі.

Розширюваність PostgreSQL означає, що користувач може налаштовувати систему шляхом визначення нових функцій, мов, типів, агрегатів, індексів та операторів. Об'єктно-орієнтованість PostgreSQL дозволяє перенести логіку програми на рівень бази даних, що спрощує розробку клієнтів, оскільки вся бізнес логіка знаходиться в базі даних. В PostgreSQL функції однозначно визначаються назвою, кількістю та типами аргументів.

## Mongo DB

MongoDB реалізує новий підхід до того як будувати бази даних, де немає схем, запитів SQL,  таблиць, зовнішніх ключів та інших речей які існують в реляційних базах даних. За звичай було нормально зберігати дані в реляційних базах, не було великої різниці які дані потрібно було зберегти. MongoDB пропонує документо-орієнтовану модель даних, дякуючи цьому вона працює швидше, її легше використовувати, та має кращу масштабованість,.

Але, навіть з огляду на переваги MongoDB та недоліки реляційних баз, потрібно мати на увазі, що проблеми бувають різні так же як і способи їх вирішення. Якщо потрібно зберегти дані зі складною структурою то краще для цього обрати MongoDB. В іншій ситуації потрібно використовувати традиційні реляційні бази даних. Зараз також є можливість використовувати змішаний підхід: частину яка має велику вкладеність, та не потребує відношень зберігати в MongoDB, а все інше в реляційних базах.

MongoDB має можливість знаходитися не на одному фізичному сервері. Її функціональність дає змогу розташовуватися да декількох фізичних серверах які можуть спілкуватися між собою та зберігати цілісність.

JSON є одним із самих популярних стандартів в світі. Він може ефективно описує складні структури даних. Спосіб зберігання даних в

Mongo DB дуже схожий на структуру JSON. BSON – це формат в якому зберігається MongoDB. BSON надає змогу швидко працювати з даними: швидше виконується пошук та обробка. Проте можна відмітити, що BSON на відміну від JSON має деякий недолік: дані які записані в JSON форматі займають в якусь міру менше пам’яті ніж в BSON, проте з іншого боку, швидкість маніпулювання з даними швидша.

# 2. ВИБІР МЕТОДІВ РІШЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ

## 2.1 Вибір Framework для створення веб-застосунку

Для розробки веб-застосунку було обрано такий фреймоврк як Nest JS, бо з коробки він пропонує заздалегідь визначену архітектуру, яка заточена під максимально зручну підтримку та масштабованість нашого сервісу. Закладені архітектурні підходи перевірені часом і давно використовуються в інших, більш зрілих платформах.

Перевірка типів. Nest.js реалізований на базі TypeScript. TypeScript забезпечить застосунку максимальну захищеність від помилок, які б могли виникнути при написанні коду, тим більше можна буде використовувати ООП в повній мірі. При цьому не заборонено використовувати стандартний JavaScript.

Використання відносин дозволяє тримати послуги ізольованими від решти логіки, такий код буде більш підтримуваним і читаним.

Тестування. З коробки Nest.js надає повністю інтегроване та налаштоване середовище для написання та запуску модульних unit та e2e-тестів, якщо б був обраний Express то потрібно було б налаштовувати екосистему з різних бібліотек для тестування.

## 2.2 Вибір бази даних для веб-застосунку

В ролі бази даних була вибрана PostgreSQL. Postgres неймовірно гнучка та надійна. Серед іншого, віна уміє створювати, зберігати та витягувати складні структури даних. Postgres підтримує велику кількість типів даних. Крім числових, з плаваючою точкою, булевих, текстових та інших очікуваних типів даних, PostgreSQL має підтримку uuid, грошового, геометричного, бінарного типів, перерахованого, мережевих, бітових рядків, адрес, текстового пошуку, масивів, xml, композитних типів та діапазонів, а також деяких внутрішніх типів для розпізнання об'єктів та знаходження логів.

## 2.3Постановка задачі

API - це набір протоколів та визначень для створення та інтеграції прикладного програмного забезпечення. Інколи такий вид взаємодій можуть назвати контрактом між тим хто постачає інформацію та

користувачем інформації, який встановлює вміст, запит який вимагається від споживача і контент який міститься в ньому, та відповідь яка вимагається виробником. Наприклад, у дизайні API для служби погоди можна було б вказати, що користувач надає поштовий індекс, а виробник відповідає з двома частинами відповіді, перша – висока температура, а друга – низька.

REST - це набір архітектурних обмежень, а не стандарт чи протокол. Коли клієнт здійснює запит через API RESTful, він передає уявлення про стан клієнта або кінцевої точки. Ця інформація надається в одному з кількох форматів через JSON, HTML, XLT або звичайний текст. Найбільш популярним форматом файлів є JSON, оскільки він не залежить від мови, а також його можна читати як людьми, так і машинами.

Щоб API вважався повністю RESTful, йому потрібно відповідати такм критеріям як:

- Архітектура клієнт-сервер, що складається з деяких клієнтів, серверів та ресурсів, із запитами, які керуються через HTTP.
- Зв’язок клієнт-сервер без стану, інформація про стан клієнту не зберігається між запитами.
- Дані з кешуванням, мають за мету спростити взаємодію між клієнтом-сервером.
- Певний єдиний стандарт інтерфейсу між компонентами, щоб передача інформації здійснювалася за стандартними формами.
- Багатошарова система, яка має на меті організувати кожний з типів можливих северів (відповідальних за безпеку, балансування навантаження тощо), передбачала отримання запитуваної інформації в ієрархії, невидимі для клієнта.
- Code-on-demand (опціонально): можливість надсилати виконуваний код із сервера клієнту за запитом, що розширює функціональні можливості клієнта.

Приклад спілкування між клієнтом і сервером (рис. 5)

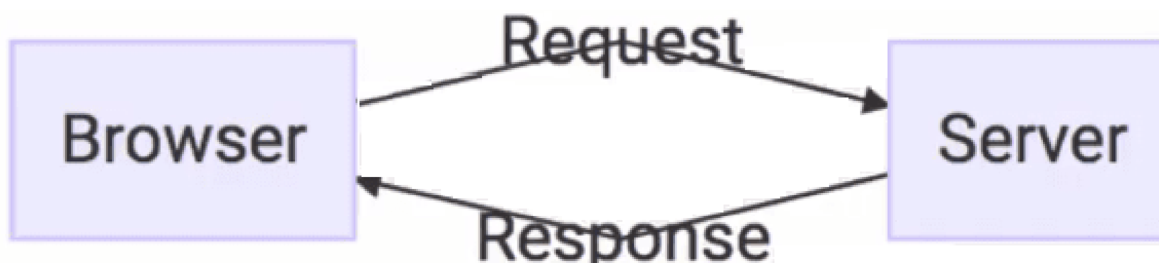


Рисунок 5 – Спілкування між клієнтом і сервером REST

Для нормального функціонування відео сервісу потрібно створити RESTfull API. На ньому буде знаходитися вся логіка додатку. Щоб створити такий сервер потрібно спочатку спроектувати архітектуру за допомогою бібліотеки Nest JS також знадобиться база даних на якій буде знаходитися інформація яка потрібна бізнес логіки для нормальної роботи сервісу.

База даних буде складатися з таких сутностей як:

- Користувачі в цій таблиці будуть знаходитися дані про людину яка зареєструвалася в сервісі.
- Канали потрібні для того щоб користувач мав змогу зберігати відео записи в колекції, яку в свою чергу можна би було легко ідентифікувати за назвою каналу.
- Підписки ця сутність потрібна щоб користувачі мали змогу закріпитися за каналом.
- Відео таблиця потрібна щоб описати відеозапис який буде завантажений на сервіс.
- Коментарі дають можливість коментувати відеозаписи
- Вподобання потрібні для того щоб користувач міг оцінити відеозапис.

Також потрібно створити REST API сервер який буде в свою чергу відповідати на запити клієнта та маніпулювати бізнес логікою додатку. Цей сервер повинен спілкуватися з базою даних та вносити в неї зміни.

На сервері повинні знаходитися кінцеві маршрути, такі як:

**Клієнтські**

- Авторизація
- Аутентифікація
- Створення клієнта
- Редагування клієнта
- Видалення клієнта
- Вибірка клієнтів
- Добавити та видалити коментар
- Поставити вподобання

**Для каналу**

- Створення каналу
- Видалення каналу
- Редагування каналу
- Вибірка каналів

**Для відеозаписів**

- Завантаження відео
- Видалення відео
- Редагування даних про відео
- Пошук відео

Відеозаписи які потрапляють на сервер потрібно шифрувати за допомогою алгоритму, також користувача потрібно ідентифікувати при запиті на сервер тому що не всі кінцеві точки будуть публічні.

# 3. ДИЗАЙН МАЙБУТНЬОГО ДОДАТКУ

Model View Controller - це архітектурний шаблон, що розділяє програму на три основні логічні компоненти: модель, представлення та контролер. Кожен з цих компонентів створений для обробки конкретних аспектів розробки програми. Цей шаблон є однією з найбільш часто використовуваних стандартних платформ веб-розробки для створення масштабованих і розширюваних проектів.

Приклад роботи MVC (рис. 6)

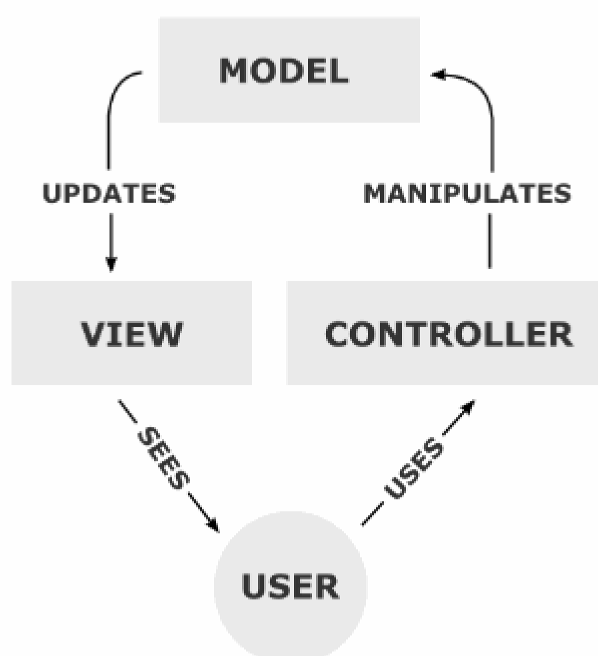


Рисунок 6 – графічне представлення роботи MVC

Точкою входу в сервер на Nest JS, як і в будь-якому іншому MVC-подібному фреймворку, є контролери. Шар контролерів відповідає за обробку вхідних запитів та повернення відповіді клієнту.

Простий приклад контролеру (рис. 7)

```
import { Controller, Get } from '@nestjs/common';
import { UsersService } from './users.service';

@Controller( prefix: 'users')
export class UsersController {
  constructor(private userService: UsersService) {}

  @Get( path: '')
  getUsers() {
    return this.userService.getUsers();
  }
}
```

Рисунок 7 – приклад контролеру в Nest JS

Провайдери – це елементи додатку які можуть впроваджуватися та розширяти функціонал класів або інших провайдерів.

Простий приклад провайдеру (рис. 8)

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { UsersRepository } from './users.repository';

@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(UsersRepository)
    private usersRepository: UsersRepository,
  ) {}

  getUsers() {
    return this.usersRepository.findUsers();
  }
}
```

Рисунок 8 – провайдер в Nest JS

Модуль – це Injectable клас з декоратором @Module(). Декоратор @Module() надає метадані, які Nest використовує для організації структури програми. Кожна програма Nest має як мінімум один модуль, кореневий модуль. Кореневий модуль це місце, де Nest починає впорядковувати дерево додатків. Фактично, кореневий модуль може бути єдиним модулем у вашому додатку, особливо коли програма маленька, але це не має сенсу. У більшості випадків у вас буде кілька модулів, кожен із яких має тісно пов'язаний набір можливостей. У Nest модулі за замовчуванням є синглетонами, тому ви можете легко ділити один і той самий екземпляр компонента між двома і більше модулями.

Простий приклад модулю (рис. 9)

```
import { Module } from '@nestjs/common';
import { UsersController } from './users.controller';
import { UsersService } from './users.service';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UsersRepository } from './users.repository';

@Module( metadata: {
  imports: [TypeOrmModule.forFeature( entities: [UsersRepository])],
  controllers: [UsersController],
  providers: [UsersService],
})
export class UsersModule {}
```

Рисунок 9 – модуль в Nest JS

Guards реалізують інтерфейс CanActivate. Вони мають єдину відповідальність також визначають, чи повинен запит оброблятись обробником маршруту чи ні.

Приклад Guard (рис. 10)

```
@Injectable()
export class RolesGuard implements CanActivate {
  canActivate(
    context: ExecutionContext,
  ): boolean | Promise<boolean> | Observable<boolean> {
    return true;
  }
}
```

Рисунок 10 – guard в Nest JS

TypeORM підтримує шаблон проектування сховища, тому кожна сутність має свій власний репозиторій. Ці репозиторії можна отримати з підключення до бази даних.

Приклад реалізації патерна Repository (рис. 11)

```
import { EntityRepository, Repository, createQueryBuilder } from 'typeorm';
import { UsersEntity } from '../models/users.entity';

@EntityRepository(UsersEntity)
export class UsersRepository extends Repository<UsersEntity> {
  constructor() {
    super();
  }
  public async findUsers() {
    const users = await createQueryBuilder( entityClass: 'users').getMany();

    return users;
  }
}
```

Рисунок 11 – патерн Repository

# 4. ТЕОРЕТИЧНІ ЗАСАДИ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

## 4.1 Буферизація та потоки

Буфер - це область пам'яті. Більшість розробників JS знайомі набагато менше ця концепція, у порівнянні з програмістами, які використовують системні мови програмування, які щодня безпосередньо взаємодіють з пам'яттю. Він являє собою шматок пам'яті фіксованого розміру (розмір не може бути змінений), виділений за межами V8 Движок JavaScript. Ви можете уявити буфер як масив цілих чисел, кожне з яких представляє байт даних.

Буфери були введені, щоб допомогти розробникам працювати з двійковими даними в екосистемі, яка традиційно розглядаються лише рядки, а не двійкові файли. Буфери в Node.js не пов'язані з концепцією буферизації даних. Ось що відбувається, коли потоковий процесор отримує дані швидше, ніж вони можуть переварити.

Приклад принципу роботи буферу (рис. 12)

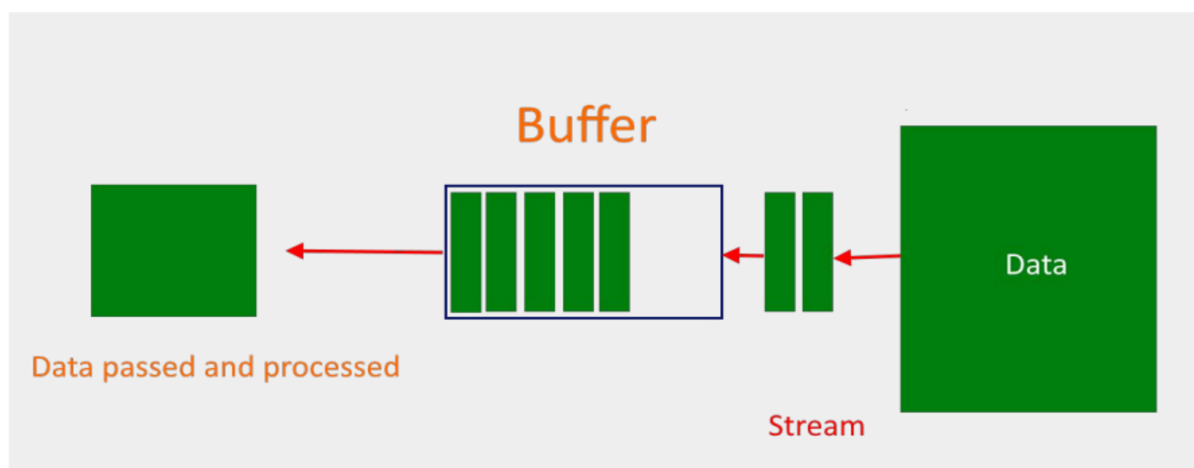


Рисунок 12 – буфер в Node JS

Потоки є однією з основоположних концепцій, які забезпечують додатки Node.js. Вони є методом обробки даних і використовуються для послідовного читання або запису вхідних даних у вихідні дані. Потоки – це спосіб обробки читання/запису файлів, мережевого зв'язку тощо вид

наскрізного обміну інформацією ефективним способом. Що робить потоки унікальними, так це те, що замість програми читає файл у пам'ять все одночасно, як традиційно, потоки зчитують шматки даних по частинах, обробляти його вміст без збереження всього цього в пам'яті.

Чому потоки?

- Ефективність пам’яті: вам не потрібно завантажувати великі обсяги даних у пам’ять перш ніж ви зможете його обробити
- Ефективність часу: потрібно значно менше часу, щоб почати обробку даних якнайшвидше як у вас, замість того, щоб чекати з обробкою до повного використання було передано Чому потоки

Типи потоків:

- З можливістю запису: потоки, в які ми можемо записувати дані. Наприклад, fs.createWriteStream() дозволяє нам записувати дані у файл за допомогою потоків.
- Читається: потоки, з яких можна зчитувати дані. Наприклад: fs.createReadStream() дозволяє нам читати вміст файлу.
- Дуплекс: потоки, які можна як читати, так і записувати. Наприклад, net.Socket
- Трансформація: потоки, які можуть змінювати або трансформувати дані під час їх запису та читати. Наприклад, у випадку стиснення файлів ви можете написати стислі дані та читання розпакованих даних у файл та з файлу.

Приклад принципу роботи потоку зчитування (рис. 13)

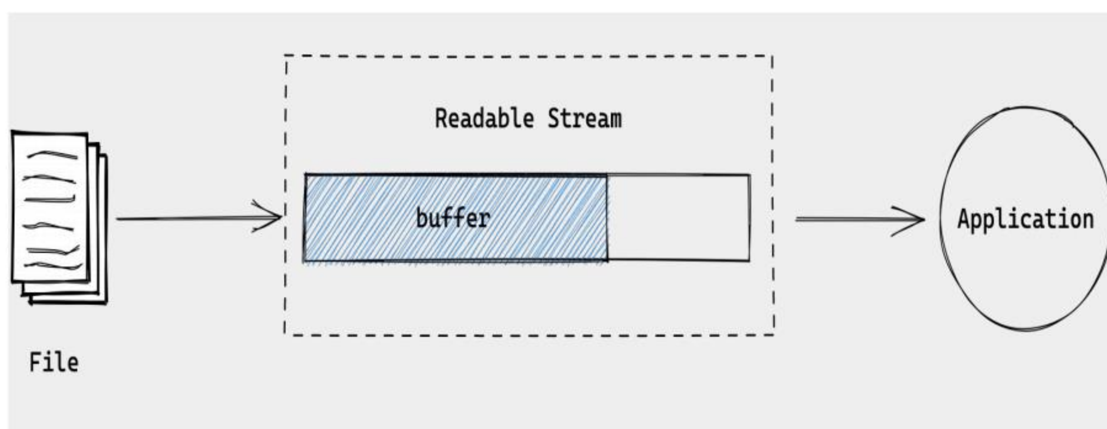

Рисунок 13 – потік зчитування

Приклад роботи потоку запису (рис. 14)

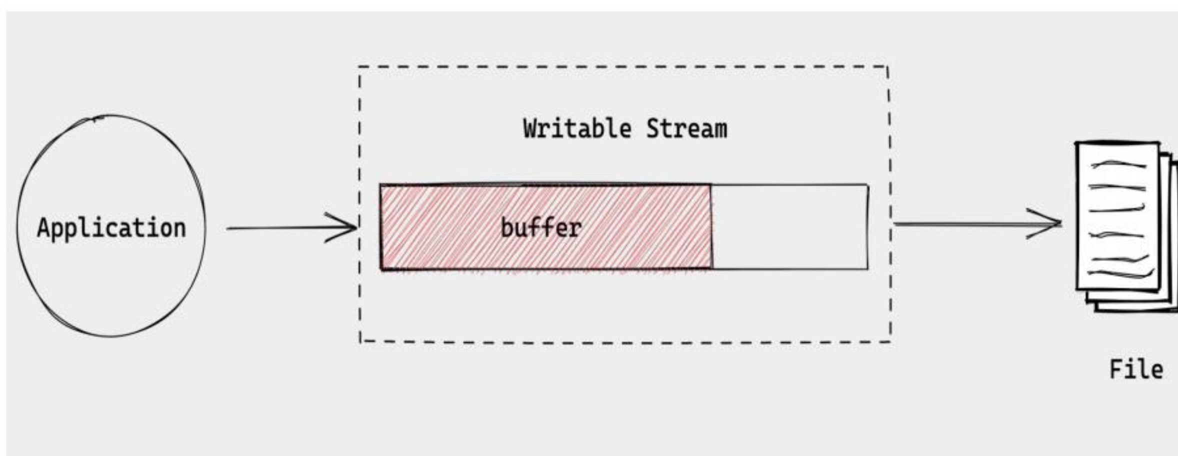


Рисунок 14 – потік запису

Приклад роботи потоку зчитування і запису (рис. 15)

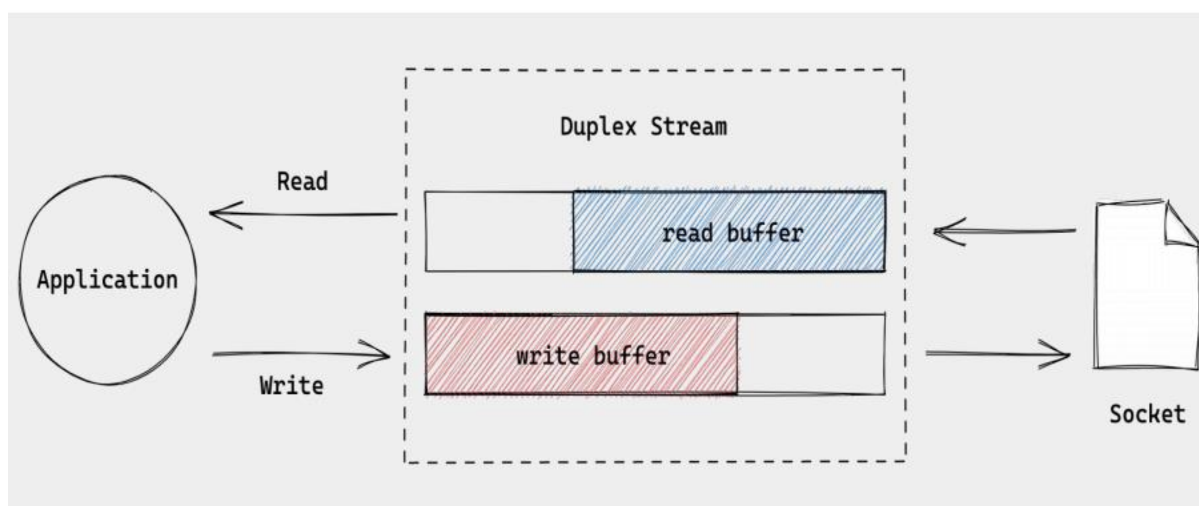


Рисунок 15 – потік зчитування і запису

Приклад роботи потоку мутуючого потоку (рис. 16)

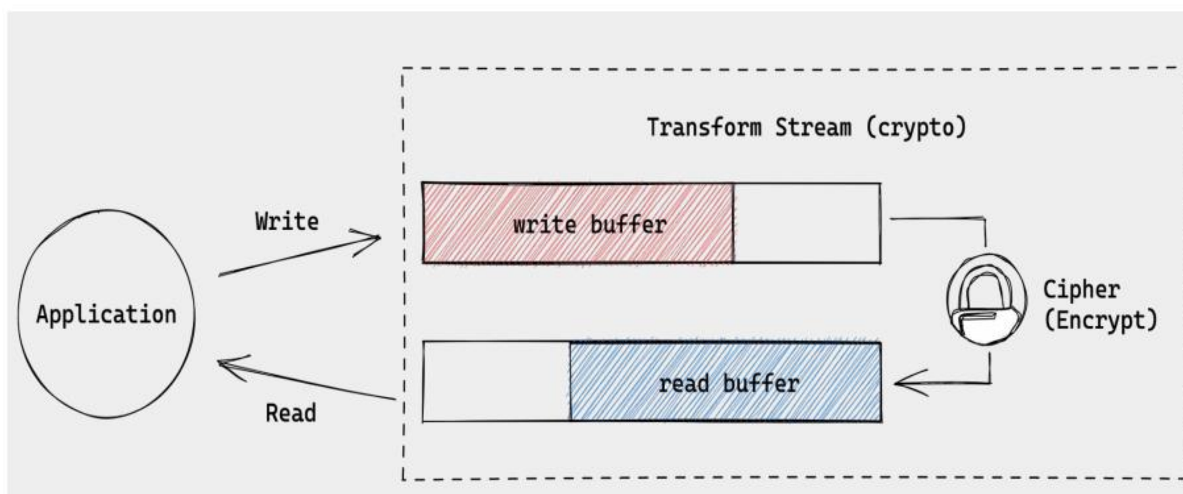


Рисунок 16 – потік з мутацією

## 4.2 Шифрування

SHA-256 є однією з наступних хеш-функцій SHA-1 (сукупно називають SHA-2) і є однією з найсильніших доступних хеш-функцій. SHA-256 не набагато складніший для кодування, ніж SHA-1, і ще жодним чином не був скомпрометований. 256-бітний ключ робить його хорошим партнером для AES. Це визначено в стандарті NIST (Національного інституту стандартів і технологій) «FIPS 180-4». NIST також надає ряд тестових векторів для перевірки правильності реалізації

Хеш-функції сімейства SHA-2 створені на основі структури Меркла — Дамгарда. Повідомлення яке виходить роздроблюється на блоки. Кожен блок повідомлення проходить через цикл в якому 64 операції. На кожній ітерації дані перетворюються в хеш. Вихідні результати обробки кожного блоку складуються, сума цих блоків є значення хеш-функції.

Приклад однієї ітерації обробки блоку даних (рис. 17)

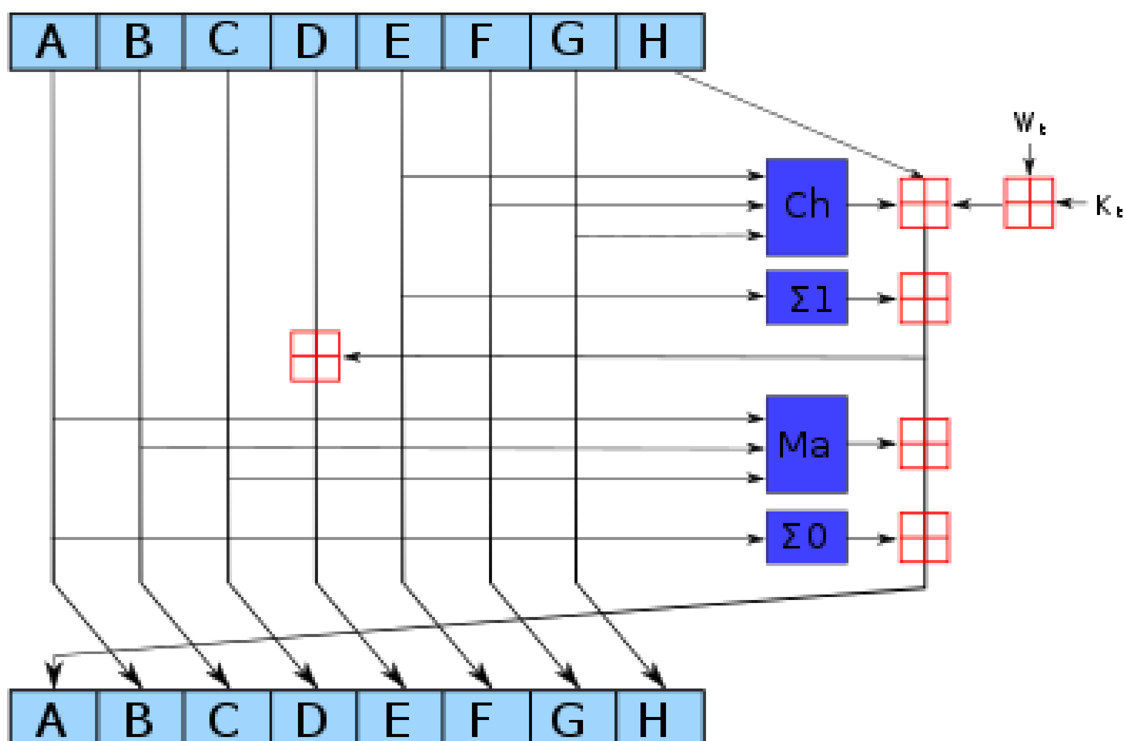


Рисунок 17 – представлення ітерації шифрування

## 4.3 JWT-токен

JSON Web Token - це об'єкт в JSON, який був описаний у відкритому стандарті RFC 7519. Він виступає як один з варіантів безпечного спілкування між користувачами. Для створення необхідно визначити заголовок із загальною інформацією по токену, корисні дані, такі як унікальний ідентифікатор користувача, його роль тощо. та підписи.

Приклад роботи JWT токену (рис. 18)

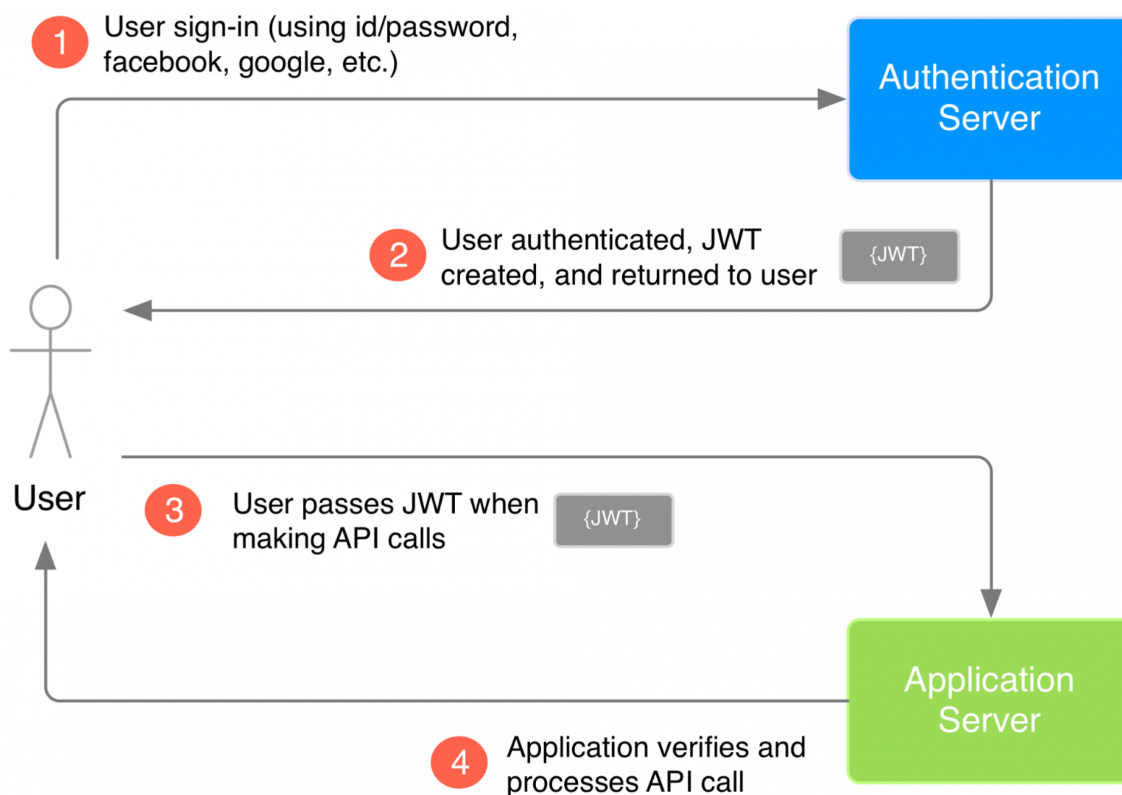


Рисунок 18 – принцип роботі JWT

Додаток використовує JWT для перевірки аутентифікації користувача таким чином:

1. Спочатку користувач заходить на сервер аутентифікації за допомогою аутентифікаційного ключа (це може бути пара логін/пароль, або Facebook ключ, або Google ключ, або ключ від іншого обліку).
2. Потім сервер аутентифікації створює JWT та відправляє його користувачеві.
3. Коли користувач робить запит до програми API, він додає до нього отриманий раніше JWT.
4. Коли користувач робить API запит, програма може перевірити за переданим із запитом JWT чи є користувач тим, за кого себе видає. У цій схемі сервер програми налаштований так, що зможе перевірити, чи є вхідний JWT саме тим, що був створений сервером аутентифікації (процес перевірки буде пояснений пізніше більш детально).

# 5. РОЗРОБЛЕННЯ ТА ІМПЛЕМЕНТАЦІЯ

## 5.1 База даних

Для нормального функціонування серверу була створена база даних на основі PostgreSQL, також ця база була описана в вигляді схеми в TypeORM.

TypeORM — це ORM, яка може працювати на платформі Node JS та використовується з такими мовами як JavaScript та TypeScript. Мета цієї ORM полягає в тому, щоб завжди підтримувати найновіші функції JavaScript і надавати додаткові функції, які допоможуть вам розробляти будь-які програми, які використовують бази даних - від невеликих програм з кількома таблицями до великих корпоративних програм з кількома базами даних.

Реалізація таблиць в PostgreSQL (рис. 19)

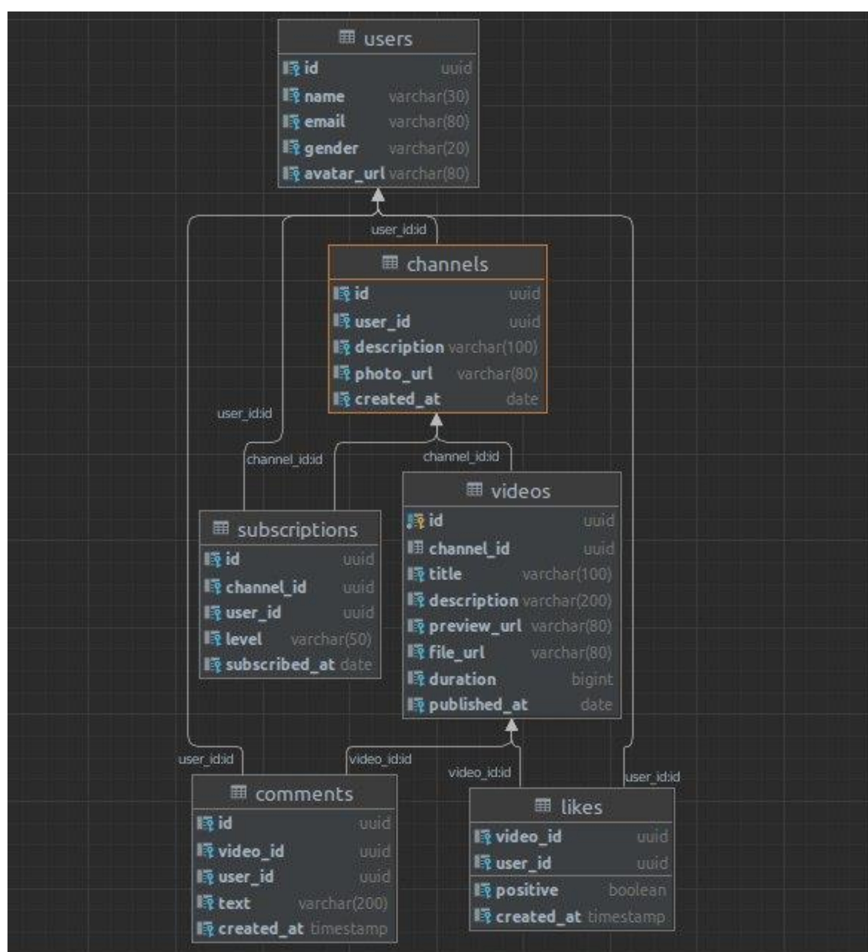


Рисунок 19 – реалізація таблиць

## Створення схеми користувача

```
@Entity('users')
export class UsersEntity extends BaseEntity {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @OneToOne(() => ChannelsEntity, channel => channel.user)
  channel: ChannelsEntity;

  @OneToMany(() => SubscriptionsEntity, subscription => subscription.user)
  subscriptions: SubscriptionsEntity[];

  @OneToMany(() => CommentsEntity, comment => comment.user)
  comments: CommentsEntity[];

  @OneToMany(() => LikesEntity, like => like.user)
  likes: LikesEntity[];

  @Column({
    length: 30
  })
  name: string;

  @Column({
    length: 80
  })
  email: string;

  @Column({
    length: 20
  })
  gender: string;

  @Column({
    length: 80,
    nullable: true
  })
  avatar_url?: string;
}
```

## Створення схеми каналу

```
@Entity('channels')
export class ChannelsEntity extends BaseEntity {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @OneToOne(() => UsersEntity, user => user.channel)
  @JoinColumn({ name: 'user_id', referencedColumnName: 'id' })
  user: UsersEntity;

  @OneToMany(() => VideosEntity, video => video.channel)
  videos: VideosEntity[];

  @OneToMany(() => SubscriptionsEntity, subscription => subscription.channel)
  subscriptions: SubscriptionsEntity[];

  @Column({
    length: 100
  })
  description: string;

  @Column({
    length: 80
  })
  photo_url: string;

  @Column({
    default: () => 'CURRENT_DATE'
  })
  created_at: Date;
}
```

## Створення схеми відео

```
@Entity('videos')
export class VideosEntity extends BaseEntity {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ManyToOne(() => ChannelsEntity, channel => channel.videos)
  @JoinColumn({ name: 'channel_id', referencedColumnName: 'id' })
  channel: ChannelsEntity;

  @OneToMany(() => CommentsEntity, comment => comment.video)
  comments: CommentsEntity[];

  @OneToMany(() => LikesEntity, like => like.video)
  likes: LikesEntity[];

  @Column({
    length: 100
  })
  title: string;
```

```
  @Column({
    nullable: true,
    length: 200
  })
  description?: string;

  @Column({
    length: 200
  })
  preview_url: string;

  @Column({
    length: 80
  })
  file_url: string;

  @Column()
  duration: Number;

  @Column({
    default: () => 'CURRENT_DATE'
  })
  published_at: Date;
}
```

## Створення схеми підписок

```
@Entity('subscriptions')
export class SubscriptionsEntity extends BaseEntity {
  @PrimaryGeneratedColumn({ name: 'id' })
  id: string;

  @ManyToOne(() => UsersEntity, user => user.subscriptions)
  @JoinColumn({ name: 'user_id', referencedColumnName: 'id' })
  user: UsersEntity;

  @ManyToOne(() => ChannelsEntity, channel => channel.subscriptions)
  @JoinColumn({ name: 'channel_id', referencedColumnName: 'id' })
  channel: ChannelsEntity;

  @Column({
    length: 50,
    default: 'standard'
  })
  level: string;

  @Column({
    default: () => 'CURRENT_DATE'
  })
  subscribed_at: Date;
};
```

## Створення схеми коментарів

```
@Entity('comments')
export class CommentsEntity extends BaseEntity {
  @PrimaryGeneratedColumn({ name: 'id' })
  id: string;

  @ManyToOne(() => UsersEntity, user => user.comments)
  @JoinColumn({ name: 'user_id', referencedColumnName: 'id' })
  user: UsersEntity;

  @ManyToOne(() => VideosEntity, video => video.comments)
  @JoinColumn({ name: 'video_id', referencedColumnName: 'id' })
  video: VideosEntity;

  @Column({
    length: 200
  })
  text: string;

  @Column({
    default: () => 'CURRENT_TIMESTAMP'
  })
  created_at: Date;
}
```

## Створення схеми вподобань

```
@Entity('likes')
export class LikesEntity extends BaseEntity {
  @ManyToOne(() => UsersEntity, user => user.likes, { primary: true })
  @JoinColumn({ name: 'user_id', referencedColumnName: 'id' })
  user: UsersEntity;

  @ManyToOne(() => VideosEntity, video => video.likes, { primary: true })
  @JoinColumn({ name: 'video_id', referencedColumnName: 'id' })
  video: VideosEntity;

  @Column({
    default: true
  })
  positive: boolean;

  @Column({
    default: () => 'CURRENT_TIMESTAMP'
  })
  created_at: Date;
}
```

## 5.2 JWT токен

Приклад генерації JWT під час реєстрації (рис. 20)

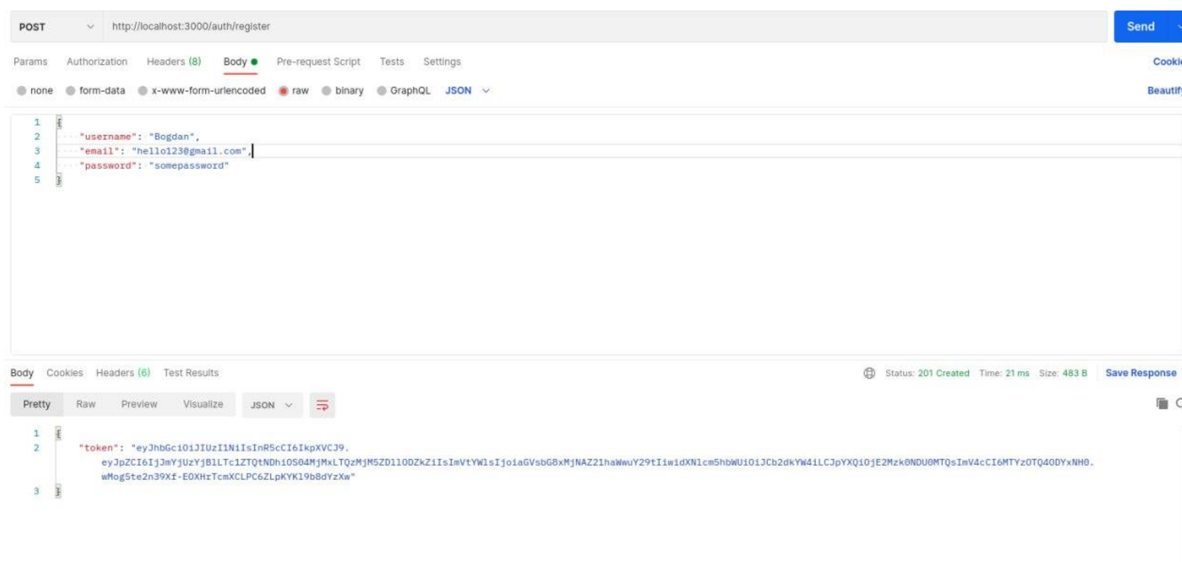


Рисунок 20 – реєстрація користувача в системі

При реєстрації на потрібно створити jwt токен за домогою даних які користувач надав при реєстрації.

```
private _generateJWT(user: UserCreateDto) {
  const payload: PayloadDto = { id: user.id, email: user.email, username:
user.username };

  return {
    token: this.jwtService.sign(payload)
  }
}
```

Коли користувач буде переходити на захищену кінцеву точку, то потрібно його перевіряти.

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(
    private readonly authService: AuthService,
    private readonly configService: ConfigService
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
```

```
      secretOrKey: configService.get<string>('SECRET')
    });
  }

  validate(payload: PayloadDto): UserCreateDto {
    const user = this.authService.validateUser(payload);
    if (!user) {
      throw new HttpException('Invalid token', HttpStatus.UNAUTHORIZED);
    }
    return user;
  }
```

## 5.3 Шифрування відеозапису

Приклад шифрування відеозапису (рис. 21)

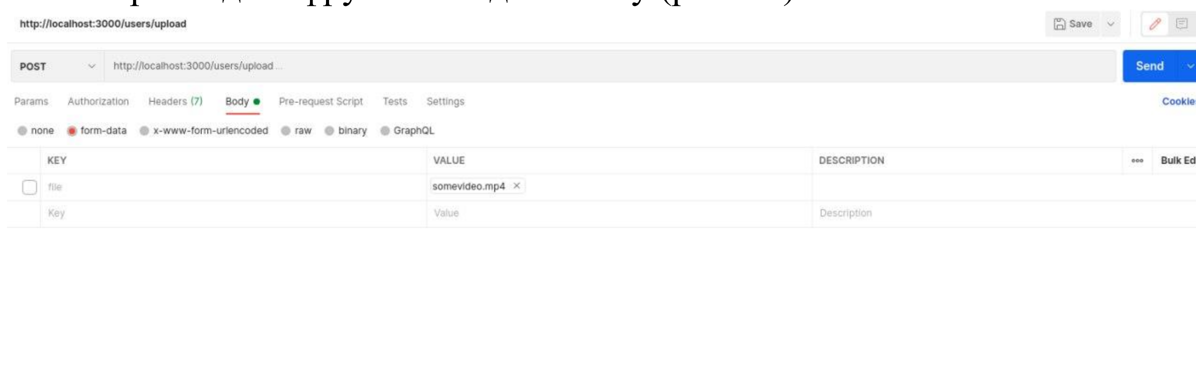


Рисунок 21 – завантаження відеозипису

Для того щоб шифрувати файл, було вирішено за допомогою потоку мутації розбити файл на блоки і по частинам зчитувати та шифрувати, а поті зберігати в файловій системі серверу.

```
export class CustomTransform extends Transform {
  public iv: Buffer;
  private appended: boolean;

  constructor(iv, options?) {
    super(options);

    this.iv = iv;

    this.appended = false;
  }

  public _transform(
    chunk: Buffer,
    encoding: string,
```

```
    callback: (error?: Error, data?: Buffer) => void,
  ) {
    if (!this.appended) {
      this.push(this.iv);

      this.appended = true;
    }

    this.push(chunk);

    callback();
  }
}
```

Приклад в якому форматі буде зберігатися файл на сервері (рис. 22)

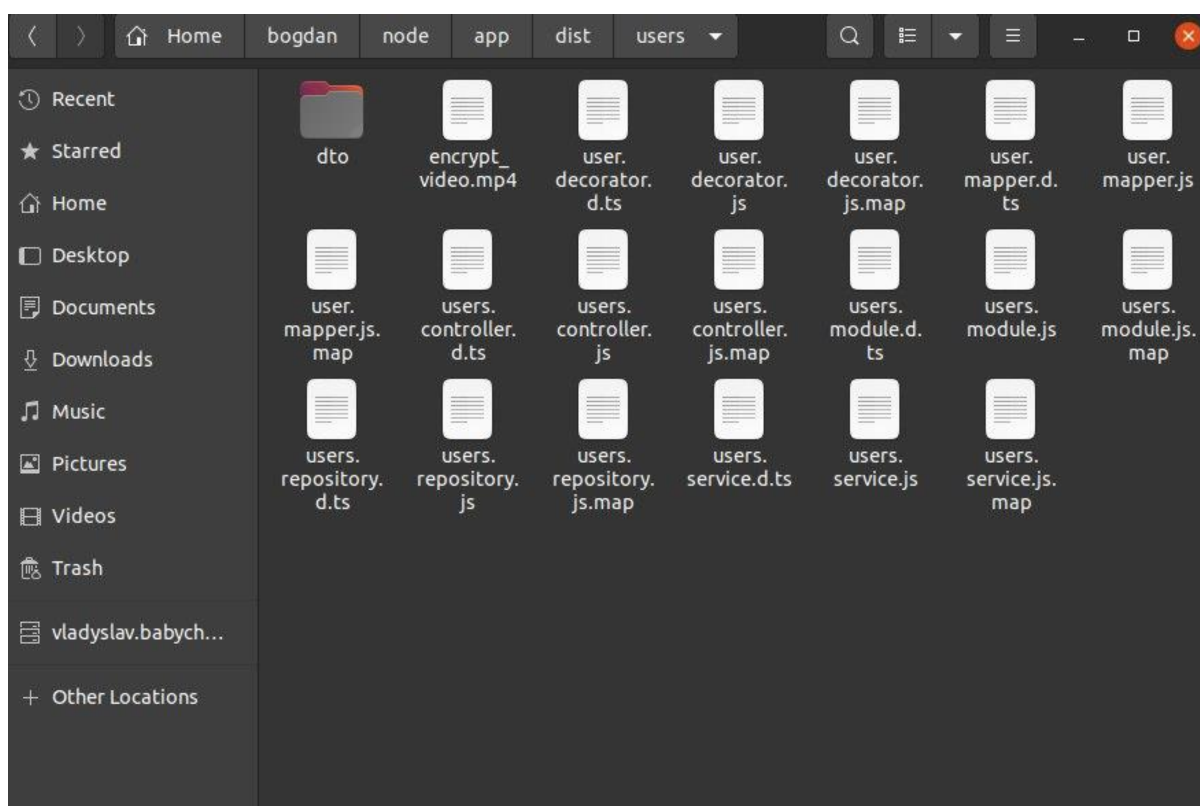


Рисунок 22 – зберігання зашифрованого файлу

# 6. ТЕСТУВАННЯ СЕРВІСУ

Код, покритий тестами, надійніший за код без нього. Якщо майбутня зміна зламається в коді, розробники зможуть правильно визначити корінь проблеми замість того, щоб шукати проблему через громіздку базу коду. Найкращі практики передбачають, що розробники спочатку запускають всі модульні тести або групу тестів локально, щоб переконатися, що будь-які зміни коду не порушують існуючий код.

І тому використовуються інструменти безперервної інтеграції, дозволяють розробникам запускати модульні тести. автоматично. Таким чином, будь-які небажані зміни в коді будуть виявлені холодною, логічною машиною. Швидкість виявлення непрацюючого коду залежить від інструментів, що використовуються для безперервної інтеграції. Тести можуть бути налаштовані на одноразову перевірку у певний час. інтервал або може бути запущений негайно в режимі реального часу, щоб переглянути зміни.

Види тестування:

- Чорна скринька Тестування без знання внутрішньої роботи програми. По суті, ви пройшли введіть в чорний ящик і перевірте, чи є результат таким, яким ви очікували.
- Біла коробка Протилежність чорного ящика, тестування коду програмного забезпечення. Білий має означати коробка прозора, тому насправді це мало бути тестування прозорої коробки.

## Unit тести

Модульні тести, як правило, є автоматизованими тестами, написаними та запущеними розробниками програмного забезпечення переконайтеся, що розділ програми (відомий як "монітор") відповідає його дизайну та поводиться за призначенням. Щоб виділити проблеми, які можуть виникнути, кожен тестовий випадок має бути перевірений незалежно. Написання та підтримка модульних тестів можна прискорити за допомогою параметризованих тестів. Таким чином, вони дозволяють виконувати один тест кілька разів з різними вхідними наборами зменшення дублювання тестового коду. На відміну від традиційних модульних тестів,

які зазвичай закриті Методи та інваріантні умови тесту, параметризовані тести беруть будь-який набір параметри.

## 6.1 Jest бібліотека для тестування

Jest — це тестова платформа для клієнтських додатків JavaScript і спеціально додатків React.

### Тестування відеокотроллеру

```
describe('Videos Controller', () => {
  let controller: VideosController;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      controllers: [VideosController],
      providers: [
        VideosService, VideosRepository
      ]
    }).compile();

    controller = module.get<VideosController>(VideosController);
  })

  it('should be defined', () => {
    expect(controller).toBeDefined();
  })
})
```

### Створення mock для тестування

```
export class VideosRepositoryMock extends
VideosRepositoryMockAbstract<DsFindVideos, DsTopVideos> implements
VideosRepositoryInterface {
  createFake(): DsFindVideos {
    const fake = new DsFindVideos();
    fake.id = faker.datatype.uuid();
    fake.preview_url = faker.internet.url();
    fake.duration = faker.datatype.number();
    fake.title = faker.lorem.word();
    fake.published_at = faker.date.past();
    fake.name = faker.name.firstName();
    return fake;
  }

  createFakeTop(): DsTopVideos {
    const fake = new DsTopVideos();
    fake.id = faker.datatype.uuid();
    fake.title = faker.lorem.word();
    fake.description = faker.lorem.text();
```

```
    fake.positive = faker.number();
    return fake;
  }

  findVideoByName(name): Promise<DsFindVideos[]> {
    return Promise.resolve(this.items.filter(videos => videos.name === name));
  }

  findTopVideos(): Promise<DsTopVideos[]> {
    return Promise.resolve(this.items);
  }
  findMostRateVideo(): Promise<DsTopVideos[]> {
    return Promise.resolve(this.items);
  }
}
```

## Тестування відео сервісу

```
describe('Videos Service', () => {
  let service: VideosService;
  let repository: VideosRepositoryMock;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        VideosService,
        {
          provide: VideosRepository,
          useClass: VideosRepositoryMock
        }
      ]
    }).compile();

    service = module.get(VideosService);
    repository = module.get(VideosRepository);
  })

  it('should be defined', () => {
    expect(service).toBeDefined();
  })

  it('getVideoByName: should return array of dto filter by name', async () => {
    const input = repository.createFake();

    repository.putFake(input);

    repository.putFake(repository.createFake());

    repository.putFake(repository.createFake());

    const dto = await service.getVideoByName(input.name);

    expect(dto).toBeDefined();
```

```
    expect(dto).toHaveLength(1)
  });

  it('getMostRateVideos: should return array dto', async () => {
    const input = repository.createFake();

    repository.putFake(input);

    repository.putFake(input);

    repository.putFake(input);

    const dto = await service.getMostRateVideos();

    expect(dto).toBeDefined();
    expect(dto).toHaveLength(3)
  });

  it('getTopFiveVideos: should return array dto', async () => {
    const input = repository.createFake();

    repository.putFake(input);

    repository.putFake(input);

    repository.putFake(input);

    repository.putFake(input);

    const dto = await service.getTopFiveVideos();

    expect(dto).toBeDefined();
    expect(dto).toHaveLength(4)
  });
})
```

# ВИСНОВКИ

Дана робота була присвячена розробці RestFull API відео сервісу. Користувачі такого продукту будуть в безпеці від атак на дані які зберігаються на фізичних носіях сервера. Такий сервіс забезпечить конфіденційність, цілісність і доступність інформації. Також я хотів звернути увагу на Nest.js. На мою думку, найближчим часом його популярність серед розробників зростатиме. Інструмент привносить системність підходи до розробки сервісів на Node.js. І завдяки тому, що він перший рушив у бік вирішення архітектурних проблем, ми можемо очікувати, що через кілька років він займатиме вагому частку цієї ніші.

Даний сервер був створений для того, щоб звернути увагу на не досконалість сучасних систем, які не переймаються про зберігання контенту своїх користувачів.

# СПИСОК ЛІТЕРАТИ


1. Кантелон М, Хартер М, Райліх Н, Головайчук Node.js у дії.: Видавництво Петербург, 2018. – 432 с.
2. Херрон Д. Node.js. Розробка серверних веб-додатків на JavaScript. – Видавництво ДМК Пресс, 2012. – 2012 с.
3. Mario Casciaro , Luciano Mammino Node.js Design Patterns.: Publisher Packt, 2020. – 660 p.
4. Браун И. Веб-мережі з Node і Express. Полноценное використання стека JavaScript.: Видавництво Петербург, 2017. – 336 с.
5. Марейн Хавербек Выразительный Javascript, 2-е издание.: Издательство Петербург, 2019. – 480 с.
6. John Resig, Bear Bibeault, and Josip Maras Secrets of the JavaScript Ninja, Second Edition.: Publisher black & white, 2016 – 464 p.
7. Bell, Jay, Magolan, Greg, Guijarro, David, de Peretti, Adrien, Housley, Patrick Nest.js: A Progressive Node.js Framework.: Publisher Bleeding Edge Press, 2018. – 350 p.
8. Evan M. Hahn Express in Action.: Publisher black & white, 2016 – 256 p.