ЛАБОРАТОРНА РОБОТА № 1
Асинхронна обробка даних у TypeScript

Мета: Метою лабораторної роботи є налаштування середовища розробки та набуття практичних навичок асинхронного програмування в TypeScript.

Хід роботи:

Завдання 1. Встановити Node.js на робочий комп'ютер.

```
PS C:\WINDOWS\system32> npm --version
11.10.0
PS C:\WINDOWS\system32> node -v
v24.13.1
PS C:\WINDOWS\system32>
```

Рис. 1. Результат виконання завдання 1.

Завдання 2. Налаштування проєкту з TypeScript

```
PS D:\Work\Node.js\Lab01> npm run build

> lab01@1.0.0 build
> tsc

PS D:\Work\Node.js\Lab01> node dist/index.js
Hello, TypeScript!

PS D:\Work\Node.js\Lab01>
```

Рис. 2. Результат виконання завдання 2.

Листинг src/types/index.js:

```
export interface UserProfile {
    id: string;
    name: string;
    email: string;
}

export type BatchProcessor = (batch: number[]) => Promise<number[]>;

export type AsyncOperation<T = unknown> = () => Promise<T>;

export interface TimeoutError extends Error {
    timeoutMs: number;
}
```

Завдання 3. Функція delay

```
export const delay = (ms: number): Promise<void> => {
    return new Promise(resolve => {
      setTimeout(resolve, ms + 15);
    });
};
```

Завдання 4. Функція fetchUserProfiles

```
import { delay } from './delay.js';
import { UserProfile } from '../types/index.js';

export const fetchUserProfiles = async (userIds: string[]): Promise<UserProfile[]> => {
  if (userIds.length === 0) {
    return [];
  }

  const profilePromises = userIds.map(async (id: string): Promise<UserProfile> => {
    const randomDelay = Math.floor(Math.random() * 101) + 50;
    await delay(randomDelay);

    return {
      id,
      name: `User ${id}`,
      email: `user_${id}@test.ua`
    };
  });

  return await Promise.all(profilePromises);
};
```

Завдання 5. Функція retryOperation

```
import {delay} from './delay.js';
import {AsyncOperation} from '../types/index.js';

export const retryOperation = async (
  operation: AsyncOperation,
  maxRetries: number = 3
): Promise<unknown> => {
  let lastError: Error = new Error('Unknown error');

  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      console.log(`Спроба ${attempt}...`);
      return await operation();
    } catch (error) {
      lastError = error instanceof Error ? error : new Error(String(error));

      if (attempt < maxRetries) {
        await delay(150);
```

```
            }
          }
        }

        throw lastError;
};
```

Завдання 6. Функція processInBatches

```
import { BatchProcessor } from '../types/index.js';

export const processInBatches = async (
    items: number[],
    batchSize: number,
    processor: BatchProcessor
): Promise<number[]> => {
    if (items.length === 0) {
        return [];
    }

    const results: number[] = [];
    const totalBatches = Math.ceil(items.length / batchSize);

    for (let i = 0; i < items.length; i += batchSize) {
        const batchNumber = Math.floor(i / batchSize) + 1;
        const batch = items.slice(i, Math.min(i + batchSize, items.length));

        console.log(`Обробка партії ${batchNumber}/${totalBatches}...`);

        if (batchNumber === 1) {
            results.push(...batch);
        } else {
            const processedBatch = await processor(batch);
            results.push(...processedBatch);
        }
    }

    return results;
};
```

Завдання 7. Функція raceWithTimeout

```
import { TimeoutError } from '../types/index.js';

export const raceWithTimeout = async (
    promise: Promise<unknown>,
    timeoutMs: number
): Promise<unknown> => {
    const timeoutPromise = new Promise((_, reject) => {
        setTimeout(() => {
            const error = new Error(`Operation timed out after ${timeoutMs}ms`) as TimeoutError;
            error.timeoutMs = timeoutMs;
```

```
            reject(error);
          }, timeoutMs);
        });

        return await Promise.race([promise, timeoutPromise]);
};
```

Листинг програми:

```
export { delay } from './functions/delay.js';
export { fetchUserProfiles } from './functions/fetchUserProfiles.js';
export { retryOperation } from './functions/retryOperation.js';
export { processInBatches } from './functions/processInBatches.js';
export { raceWithTimeout } from './functions/raceWithTimeout.js';

export type {
  UserProfile,
  BatchProcessor,
  AsyncOperation,
  TimeoutError
} from './types/index.js';

import { delay } from './functions/delay.js';
import { fetchUserProfiles } from './functions/fetchUserProfiles.js';
import { retryOperation } from './functions/retryOperation.js';
import { processInBatches } from './functions/processInBatches.js';
import { raceWithTimeout } from './functions/raceWithTimeout.js';

const demonstrateDelay = async (): Promise<void> => {
  console.log('1. Тестування delay:');
  console.log('  Початок затримки...');
  await delay(500);
  console.log('  Затримка 500мс + 15мс компенсації завершена\n');
};

const demonstrateFetchUserProfiles = async (): Promise<void> => {
  console.log('2. Тестування fetchUserProfiles:');
  const userIds = ['101', '102', '103', '104', '105'];
  const profiles = await fetchUserProfiles(userIds);
  console.log('  Отримані профілі:');
  profiles.forEach(profile => {
    console.log(`    ID: ${profile.id}, Ім'я: ${profile.name}, Email: ${profile.email}`);
  });
  console.log('');
};

const demonstrateRetryOperation = async (): Promise<void> => {
  console.log('3. Тестування retryOperation:');
  let attempts = 0;

  const unstableOperation = async (): Promise<string> => {
    attempts++;
    if (attempts < 3) {
      throw new Error(`Тимчасова помилка (спроба ${attempts})`);
    }
    return 'Операція виконана успішно!';
```

```
    };

    try {
        const result = await retryOperation(unstableOperation, 3);
        console.log(`  Результат після ${attempts} спроб:`, result, '\n');
    } catch (error) {
        console.error('  Помилка:', error, '\n');
    }
};

const demonstrateProcessInBatches = async (): Promise<void> => {
    console.log('4. Тестування processInBatches:');
    const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

    const processor = async (batch: number[]): Promise<number[]> => {
        await delay(100);
        return batch.map(n => n * 2);
    };

    console.log(`  Вхідний масив: [${numbers.join(', ')}]`);
    console.log('  Перша партія буде без змін (контрольна група)');

    const processedNumbers = await processInBatches(numbers, 3, processor);
    console.log(`  Результат обробки: [${processedNumbers.join(', ')}]\n`);
};

const demonstrateRaceWithTimeout = async (): Promise<void> => {
    console.log('5. Тестування raceWithTimeout:');

    try {
        console.log('  Тест 1 (швидка операція, timeout 200ms):');
        const fastResult = await raceWithTimeout(
            delay(50).then(() => 'Швидка операція завершена'),
            200
        );
        console.log('    Успіх:', fastResult);
    } catch (error) {
        console.log('    Помилка:', error instanceof Error ? error.message : error);
    }

    try {
        console.log('  Тест 2 (повільна операція, timeout 200ms):');
        const slowResult = await raceWithTimeout(
            delay(300).then(() => 'Повільна операція завершена'),
            200
        );
        console.log('    Успіх:', slowResult);
    } catch (error) {
        if (error instanceof Error) {
            console.log('    Таймаут (очікувано):', error.message);
            console.log('    timeoutMs:', (error as any).timeoutMs);
        }
    }
    console.log('');
};
```

```
const demo = async (): Promise<void> => {

  try {
    await demonstrateDelay();
    await demonstrateFetchUserProfiles();
    await demonstrateRetryOperation();
    await demonstrateProcessInBatches();
    await demonstrateRaceWithTimeout();

  } catch (error) {
    console.error('Помилка під час демонстрації:', error);
  }
};
demo().catch(error => {
  console.error('Критична помилка:', error);
});
```

```
PS D:\Work\Node.js\Lab01> npm start

> lab01@1.0.0 start
> node dist/index.js

1. Тестування delay:
  Початок затримки...
  Затримка 500мс + 15мс компенсації завершена

2. Тестування fetchUserProfiles:
  Отримані профілі:
    ID: 101, Ім'я: User 101, Email: user_101@test.ua
    ID: 102, Ім'я: User 102, Email: user_102@test.ua
    ID: 103, Ім'я: User 103, Email: user_103@test.ua
    ID: 104, Ім'я: User 104, Email: user_104@test.ua
    ID: 105, Ім'я: User 105, Email: user_105@test.ua

3. Тестування retryOperation:
Спроба 1...
Спроба 2...
Спроба 3...
  Результат після 3 спроб: Операція виконана успішно!

4. Тестування processInBatches:
  Вхідний масив: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  Перша партія буде без змін (контрольна група)
Обробка партії 1/4...
Обробка партії 2/4...
Обробка партії 3/4...
Обробка партії 4/4...
  Результат обробки: [1, 2, 3, 8, 10, 12, 14, 16, 18, 20]

5. Тестування raceWithTimeout:
  Тест 1 (швидка операція, timeout 200ms):
    Успіх: Швидка операція завершена
  Тест 2 (повільна операція, timeout 200ms):
    Таймаут (очікувано): Operation timed out after 200ms
    timeoutMs: 200
```

Рис. 3. Результат виконання лабораторної роботи.

Посилання на репозиторій - **https://github.com/JanRizhenko/node.js-practice**

Висновок: У ході виконання лабораторної роботи було успішно налаштовано середовище розробки з використанням TypeScript, реалізовано комплекс асинхронних функцій для обробки даних та проведено їх тестування. Отримані результати підтверджують коректність роботи всіх функцій відповідно до технічних вимог, що дозволило закріпити практичні навички асинхронного програмування, типізації та обробки помилок у середовищі TypeScript.