

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

"На правах рукопису"
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“ ” _____ 2018р.

Магістерська дисертація

зі спеціальності 121 Інженерія програмного забезпечення
за спеціалізацією Програмне забезпечення розподілених систем
на тему Програмне забезпечення для реалізації ODM для мови JavaScript
об'єктів СУБД Intersystems Caché

Виконав: студент 6 курсу, групи ТВ-371мп
_____ Ковальчук Костянтин Олександрович
(прізвище, ім'я, по батькові)

_____ (підпис)

Науковий керівник к.т.н., доц.. Михайлова І. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

Рецензент д. т. н., проф., проф. Кабанячий В. В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних
посилань.

Студент _____
(підпис)

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет	теплоенергетичний (повна назва)
Кафедра	автоматизації проектування енергетичних процесів і систем (повна назва)
Рівень вищої освіти	другий (магістерський)
Спеціальність	121 – Інженерія програмного забезпечення (код і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____	_____
(підпис)	О.В. Коваль (ініціали, прізвище)

« ____ » _____ 20__ р.

**ЗАВДАННЯ
на магістерську дисертацію студенту**

_____ Ковальчуку Костянтину Олександровичу
(прізвище, ім'я, по батькові)

1. Тема дисертації: Програмне забезпечення для реалізації ODM для мови JavaScript об'єктів СУБД Intersystems Caché
науковий керівник Михайлова І. Ю., к.т.н. доцент
дисертації (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від « ____ » _____ 2018 р. № _____

2. Термін подання студентом дисертації: 10 грудня 2018 року

3. Об'єкт дослідження: Засоби забезпечення для реалізації ODM

4. Предмет дослідження: Засоби реалізації ODM для мови JavaScript

5. Перелік питань, які потрібно розробити:

5.1. Аналіз сучасних методів реалізації ODM для мови JavaScript.

5.2. Аналіз модулів системи, що придатні для реалізації ODM.

5.3. Аналіз шляхів оптимізації реалізації ODM.

5.4. Розробка оптимізованого реалізації ODM.

5.5. Тестування розробленого модуля на результативність.

5.6. Розробка засобів прискорення реалізації ODM.

5.7. Розробка стартап-проекту.

6. Орієнтовний перелік ілюстративного матеріалу:

6.1. Презентація PowerPoint відповідно до теми дисертації.

7. Дата видачі завдання « 11 » вересня 2018р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
	Затвердження теми роботи	17.05.2018	
	Вивчення та аналіз задачі. Проведення дослідження по вибраній темі	01.06.2018- 03.09.2018	
	Розробка архітектури та загальної структури системи	03.09.2018- 28.09.2018	
	Програмна реалізація системи	01.10.2018- 26.10.2018	
	Захист програмного продукту	22.10.2018	
	Оформлення пояснювальної записки	02.09.2018- 10.12.2018	
	Передзахист	26.11.2018- 30.11.2018	
	Захист	19.12.2018	

Студент

(підпис)

Ковальчук К.О.

(прізвище та ініціали)

Науковий керівник

(підпис)

Михайлова І. Ю

(прізвище та ініціали)

РЕФЕРАТ

Структура й обсяг дипломної роботи

Магістерська дисертація складається зі вступу, 6 розділів, висновку, переліку посилань з 21 найменування, 2 додатків і містить 34 рисунки, 34 таблиці. Повний обсяг магістерської дисертації складає 90 сторінок, з яких перелік посилань займає 2 сторінки, додатки – 9 сторінок.

Актуальність теми. Програми, що використовують бази даних є невід'ємною частиною сучасної інформаційної інфраструктури. Дані програмні системи використовують мови програмування загального призначення та бази даних для забезпечення одночасного доступу до даних, можливості пошуку у великих масивах даних та надійного і безпечного оновлення даних. Мова JavaScript – найпоширеніша на сьогодні мова програмування загального призначення, а Intersystems Caché – надійна та перевірена СУБД, що дозволяє реалізовувати мультипарадигменний доступ до бази даних.

Таким чином актуальною є проблема розробки прикладного програмного інтерфейсу для доступу до СУБД Intersystems Caché з мови програмування JavaScript.

Мета дослідження полягає в розробці прикладного програмного інтерфейсу для доступу до СУБД Intersystems Caché з мови програмування JavaScript.

Для досягнення поставленої мети були сформульовані наступні **завдання дослідження**, що визначили логіку дослідження та його структуру:

- проаналізувати існуючі методи доступу до СУБД Intersystems Caché для різних мов програмування та мови JavaScript зокрема;
- обрати засоби для реалізації прикладного програмного інтерфейсу мовою JavaScript;
- розробити архітектуру та структуру програмного рішення;
- розробити прикладний програмний інтерфейс для доступу до СУБД.

Об'єктом дослідження є ODM для об'єктів, що зберігаються в СУБД Intersystems Caché.

Предметом дослідження є програмне забезпечення для реалізації об'єктно-документного відображення об'єктів, що зберігаються в СУБД Intersystems Caché.

Практичне значення одержаних результатів роботи полягає в розробці зручного високорівневого прикладного програмного інтерфейсу для доступу до СУБД Intersystems Caché, що дозволить прискорити розробку прикладних програм та знизити витрати на розробку програмного забезпечення.

Апробація результатів дисертації

1. Міжнародна науково-практична конференція “Підсумки розвитку наукової думки: 2018” (Івано-Франківськ, 5 грудня 2018 р.).

Публікації. Наукові положення дипломної роботи опубліковані у 1 роботі.

ОСНОВНІ ПУБЛІКАЦІЇ ПО ТЕМІ ДИСЕРТАЦІЇ

1. Ковальчук К. О. / Організація доступу до мультипарадигмної СУБД Intersystems Caché на платформі node.js / Ковальчук К. О., Михайлова І.Ю. // Матеріали міжнародної науково-практичної конференції “Підсумки розвитку наукової думки” (Івано-Франківськ, 5 грудня 2018 р.): тези доп. /Вінниця, 2018. – С. 103–105.

Ключові слова. INTERSYSTEMS CACHE, NODE.JS, JAVASCRIPT, ОБ'ЄКТНО-ДОКУМЕНТНЕ ВІДОБРАЖЕННЯ.

ABSTRACT

Structure and amount of work

Master's thesis consists of introduction, 6 chapters, conclusion, bibliography with 21 items, 2 supplements and includes 34 figures, 34 tables. Total amount of master's thesis is 90 pages, including 2 pages of bibliography and 9 pages of supplements.

Topicality. Programs that use databases are an inevitable part of a modern informational infrastructure. These programs use general purpose programming languages and databases to provide a concurrent access to data, search through large data arrays and to perform secure and safe update of the data. JavaScript is the most popular general-purpose programming language nowadays, whereas InterSystems Caché is a reliable and proven DBMS that allows for a multiparadigm database access.

Consequently, development of application programming interface for InterSystems Caché is a relevant problem.

Purpose of work is to develop application programming interface for InterSystems Caché DBMS access from JavaScript programming language.

To achieve the purpose of work, the **following objectives of study**, which defined the logical structure of the research, were set:

- analyze existing methods for InterSystems Caché DBMS access for different programming languages and JavaScript in particular;
- choose means for implementation of application programming interface in JavaScript programming language;
- design architecture and structure of the software solution;
- develop application programming interface for DBMS access.

Object of research is ODM for objects stored in InterSystems Caché DBMS.

Subject of research is software for implementation of object-document mapping for objects stored InterSystems Caché DBMS.

Practical value of research is development of a convenient and high-level application programming interface for InterSystems Caché DBMS that will speed-up the development of application software and reduce maintenance costs.

Research results approbation

1. International scientific-practical conference «Results of development of scientific thought: 2018» (Ivano-Frankivsk, December 5, 2018).

Publications. Scientific aspects of the thesis were published in 1 work.

PUBLICATIONS ON THESIS' TOPIC

1. Kovalchuk K. / Organization of access to multiparadigm DBMS Intersystems Caché on node.js platform / Kovalchuk K., Myhaylova I. // Materials of international scientific-practical conference “Results of development of scientific thought: 2018” (Ivano-Frankivsk, December 5, 2018): theses / Vinnytsia, 2018. – pp.103–105.

Keywords. INTERSYSTEMS CACHÉ, NODE.JS, JAVASCRIPT, OBJECT-DOCUMENT MAPPING.

ЗМІСТ

Перелік умовних позначень	11
Вступ.....	12
1. Задача доступу до СУБД Intersystems Caché з мови програмування javascript	14
2. Огляд існуючих програмних рішень доступу до СУБД Intersystems Caché	16
2.1 Опис предметної області доступу до СУБД з високорівневих мов програмування	16
2.2 Огляд існуючих шляхів доступу до СУБД Intersystems Caché.....	18
2.2.1 Використання JavaScript API для node.js.....	18
2.2.2 Використання C++ API.....	19
2.2.3 Використання ODBC	19
2.3 Використання Java API.....	20
3. Методи та засоби реалізації ODM для об'єктів СУБД Intersystems Caché мовою javascript.....	23
3.1 СУБД Intersystems Caché	23
3.1.1 Вбудовані мови програмування Intersystems Caché	25
3.1.2 Об'єктна модель Intersystems Caché.....	26
3.1.3 Реляційний доступ до бази даних.....	28
3.2 Середовище виконання JavaScript node.js	29
3.2.1 C++ модулі для node.js.....	30
3.2.2 Альтернативні середовища виконання	31
3.3 Бібліотека NAN (Native Abstractions for Node) для написання модулів C++ для node.js.....	32
3.4 Бібліотека Bluebird як реалізація специфікації Promise/A+.....	33
3.4.1 Promise API	33
3.4.2 Огляд бібліотеки Bluebird	34
3.5 Бібліотека nanopdbс для з'єднання з БД.....	34
3.6 Порівняльна характеристика методів доступу до БД.....	35

3.7 Інші засоби розробки	36
4. Опис програмної реалізації ODM для об'єктів СУБД Intersystems Caché мовою javascript.....	38
4.1 Засоби реалізації об'єктно-реляційного відображення	38
4.1.1 Шаблон Active Record.....	38
4.1.2 Шаблон Data Mapper.....	39
4.1.3 Вибір шаблону реалізації	39
4.2 Архітектура та структура програмного рішення	39
4.2.1 Модуль Cachéodbc.....	41
4.2.2 Модуль Cache-ODM.....	46
5. Методика роботи користувача з ODM для об'єктів СУБД Intersystems Caché для мови javascript.....	50
5.1 Інсталяція та системні вимоги для роботи з програмним продуктом	50
5.1.1 Технічні характеристики	50
5.1.2 Операційна система	50
5.1.3 Драйвер ODBC	51
5.2 Сценарії роботи користувача	52
5.2.1 Конфігурація з'єднання з СУБД	52
5.2.2 Робота з об'єктами СУБД Caché	53
5.2.3 Виконання SQL-запитів.....	54
5.2.4 Інтеграція з Web-застосунками	55
5.3 Приклад використання розробленого API.....	56
6. Стартап проект.....	60
6.1 Ідея проекту	60
6.2 Маркетинговий аналіз стартап проекту.....	60
6.3 Технологічний аудит ідеї проекту.....	64
6.4 Аналіз ринкових можливостей	65
6.5 Розробка ринкової стратегії проекту.....	73

	10
6.6 Розробка маркетингової програми	76
Висновки	79
Список використаних джерел	82
Додаток А.....	84
Додаток Б.....	86

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	Application Programming Interface, прикладний програмний інтерфейс
COS	Caché ObjectScript, мова програмування
DNS	Domain Name Service, доменна служба імен
HTTP	HyperText Transfer Protocol, протокол передачі гіпертексту
JDBC	Java DataBase Connectivity, API для доступу до БД мовою Java
LRU	Least Recently Used, витіснення давно використовуваних елементів кешу
LTS	Long Term Support, довгострокова підтримка
ODBC	Open DataBase Connectivity, API для доступу до БД
ODM	Object-Document Mapping, об'єктно-документне відображення
ORM	Object-Relational Mapping, об'єктно-реляційне відображення
SQL	Structured Query Language, мова структурованих запитів
SSL	Secure Socker Layer, рівень захищених сокетів
STL	Standard Template Library, стандартна бібліотека шаблонів
TMDM	Technology and a Transactional Multidimensional Data Model, транзакційна багатовимірна модель даних
TTD	Test-Driven Development, розробка на основі тестів
БД	База даних
ООБД	Об'єктно-орієнтована база даних
ООП	Об'єктно-орієнтоване програмування
СУБД	Система управління базами даних

ВСТУП

Сучасне програмне забезпечення часто потребує доступу до баз даних. Операції з даними повинні проводитися в безпечний спосіб, без пошкодження чи втрати даних. В той же час, дані повинні бути зручними для користування в програмному забезпеченні.

На сьогодні доступ до баз даних є як ніколи актуальним, адже як об'єм, так і різноманіття даних, що зберігаються, постійно зростає.

База даних Caché надає мультипарадигменний доступ до даних (доступ може здійснюватися як за допомогою мови SQL, так і за допомогою прямого доступу до багатовимірних структур даних – глобалів). Глобали дозволяють використовувати Caché як базу даних типу "ключ-значення", стовпчикову, графову або документну СУБД [1].

СУБД Intersystems Caché надає багато API для доступу з різних мов програмування, проте мові JavaScript не приділено достатньої уваги. Для вирішення даної проблеми було запропоновано написати програмну систему для ODM для об'єктів СУБД Intersystems Caché.

Написання ПЗ, що надає прикладному програмісту високорівневий прикладний програмний інтерфейс для доступу до СУБД мовою JavaScript дозволить мінімізувати витрати часу на написання Web-застосунків, скоротити витрати на підтримку та час на вивчення нових технологій та в той же час забезпечити прийнятну швидкодію.

Пояснювальна записка представлена шістьма розділами. В першому розділі розглянута задача доступу до СУБД Caché з мови програмування javascript. В другому розділі наведено огляд існуючих програмних рішень доступу до СУБД, а також наведено огляд існуючих API для СУБД Intersystems Caché. В третьому розділі описуються методи та засоби програмної реалізації ODM для доступу до об'єктів СУБД Intersystems Caché для мови JavaScript. В четвертому розділі описуються

програмна реалізація ODM для об'єктів СУБД Intersystems Caché мовою JavaScript. В п'ятому розділі наведено методику роботи користувача-прикладного програміста по роботі із системою, а також результати тестування та випробування програмного продукту. В шостому розділі проведено розробку стартап-проекту та економічний аналіз програмного продукту.

1. ЗАДАЧА ДОСТУПУ ДО СУБД INTERSYSTEMS CACHE З MOBI ПРОГРАМУВАННЯ JAVASCRIPT

Метою дипломної роботи є розробка прикладного програмного інтерфейсу для доступу до СУБД Intersystems Caché з мови програмування JavaScript.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати існуючі методи доступу до СУБД Intersystems Caché для різних мов програмування та мови JavaScript зокрема;
- обрати засоби реалізації програмної системи для реалізації прикладного програмного інтерфейсу мовою JavaScript;
- розробити архітектуру та структуру програмного рішення;
- розробити прикладний програмний інтерфейс для доступу до СУБД.

Програмний інтерфейс повинен надавати доступ платформі node.js до СУБД Intersystems Caché та підтримувати доступ до об'єктів (глобалів). Крім того API має надавати наступний базовий функціонал для роботи з об'єктами даної СУБД.

1. Підтримка транзакцій.
2. Підтримка доступу до окремих об'єктів (документів, таблиць).
3. Підтримка SQL запитів.

Продукт повинен мати наступні властивості.

1. Бути кросплатформним (підтримка Windows, Linux та MacOS).
2. Надавати доступ до об'єктів СУБД Caché.
3. Підтримувати основні підтримувані на сьогодні LTS версії node.js (6, 8, 10).
4. Підтримувати основні версії Intersystems Caché.

Вхідними даними для даної системи є послідовність команд і запитів для роботи з БД.

Вихідними даними для даної системи є дані, що записуються в СУБД в результаті виконання команд або об'єкти мови JavaScript, що формуються в результаті виконання запитів.

Даний програмний інтерфейс призначений для наступних користувачів.

1. Прикладних програмістів мовою JavaScript для використання при розробці веб-сайтів.

2. Непрофесійних розробників для використання при автоматизації задач або аналізі даних.

Завдяки даному прикладному програмному інтерфейсу користувач може використовувати СУБД Intersystems Caché для автоматизації задач та розробки Web-сайтів мовою JavaScript.

2. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ДОСТУПУ ДО СУБД INTERSYSTEMS CACHE

2.1 Опис предметної області доступу до СУБД з високорівневих мов програмування

Програми, що використовують бази даних, є невід'ємною частиною інформаційної інфраструктури. Дані програмні системи використовують мови програмування загального призначення та бази даних для забезпечення одночасного доступу до даних, можливості пошуку у великих масивах даних та надійного і безпечного оновлення даних.

Типовим набором задач, що необхідно вирішити при інтеграції мови програмування та СУБД, є наступні.

1. Невідповідність примітивних типів БД та цільової мови програмування. Це може виражатися невідповідністю діапазонів значень, а також інтерпретацією невизначених значень.

2. Оптимізація пошуку та вибірки даних.

3. Оптимізація навігації по даним.

4. Перевикористання запитів.

Програмні інтерфейси для доступу до БД лежать між двома екстремумами:

- ортогональна перзистентність – чистий підхід до перзистентності, при якому механізми перзистентності та наявність БД приховані від програміста;

- явне виконання запитів – підхід за якого програміст явно викликає операції, пов'язані з БД [2].

Ортогональна перзистентність (рисунок 2.1) є розширенням поняття часу життя змінної. Вона дозволяє об'єктам чи значенням існувати після завершення роботи певної програми.


```

void printInfo(String prefix) {
    for (Employee e in db.allEmployees() )
        if ( e.name.startsWith(prefix) && e.salary > e.manager.salary ) {
            print( e.name );
            print( e.salary );
            print( e.department.name );
        }
    }
}

```

Рисунок 2.1 - Псевдокод для системи з ортогональною перзистентністю

Даний підхід характерний (в деякій мірі) для ООБД та систем об'єктно-реляційного відображення (паприклад, TopLink, JDO, EJB, Hibernate).

Альтернативою ортогональній перзистентності є явне виконання запитів на спеціалізованій мові запитів (рисунок 2.2).

```

string empQuery = "SELECT e.name, e.salary, d.name as deptName"
+ "FROM (Employee e INNER JOIN Department d ON d.ID = e.department)"
+ "INNER JOIN Employee m ON m.ID = e.manager"
+ "WHERE e.name LIKE ? AND e.salary > m.salary"

```

```

Connection conn = DriverManager.getConnection(...);
PreparedStatement stmt = con.prepareStatement(empQuery);
stmt.setString(1, prefix + "%");
ResultSet rs = stmt.executeQuery(empQuery);
while ( rs.next() ) {
    print( rs.getString("name") );
    print( rs.getDecimal("salary") );
    print( rs.getString("deptName") );
}

```

Рисунок 2.2 - Псевдокод для явного виконання запитів

Реалізація мови запитів може бути:

- вбудованою, при якій запити до БД вбудовуються в мову програмування (рисунок 2.3);

```

&sql(SELECT DOB INTO :a FROM Sample.Person)
WRITE "1st date of birth is ",a,!

```

Рисунок 2.3 - Приклад вбудованого запиту на мові ObjectScript

- інтерфейси рівня виклику, що забезпечують доступ до БД на основі стандартизованого API. Ключовою особливістю інтерфейсу рівня виклику є виконання запитів та команд БД, що представлені у вигляді структур даних часу виконання.

При цьому інтерфейс доступу до СУБД повинен забезпечувати:

- а) зручну модель програмування;
- б) високу швидкодію [2].

2.2 Огляд існуючих шляхів доступу до СУБД Intersystems Caché

При аналізі існуючих рішень було виявлено, що подібні рішення вже існують, проте вони мають певні недоліки та особливості, що ускладнюють доступ з мови javascript платформи node.js або не надають повного функціоналу роботи з БД.

2.2.1 Використання JavaScript API для node.js

JavaScript API для node.js містить методи для роботи з глобалами [3]:

- set – задати значення глобалу;
- get – отримати значення глобалу;
- kill – видалити глобал;
- next, prev – отримати наступний та попередній глобал;
- invoke_classmethod – викликати метод класу;
- create_instance – створити об'єкт класу;
- set_property – встановити властивість об'єкту;
- get_property – отримати значення об'єкту;
- save_instance – зберегти об'єкт.

Основним недоліками даного API є:

- відсутність підтримки останніх версій node.js (10+);
- закритість доступу до вихідного коду;
- відсутність відкритого доступу до пакету, що не дає можливості залучити нових користувачів, що лише експериментують з СУБД Caché.

2.2.2 Використання C++ API

СУБД Caché підтримує динамічний C++ API, що базується на використанні класів `Dyn_obj` (динамічний об'єкт) та `dyn_ref` (динамічне посилання на об'єкт). Архітектура C++ прив'язки зображена на рисунку 2.4.

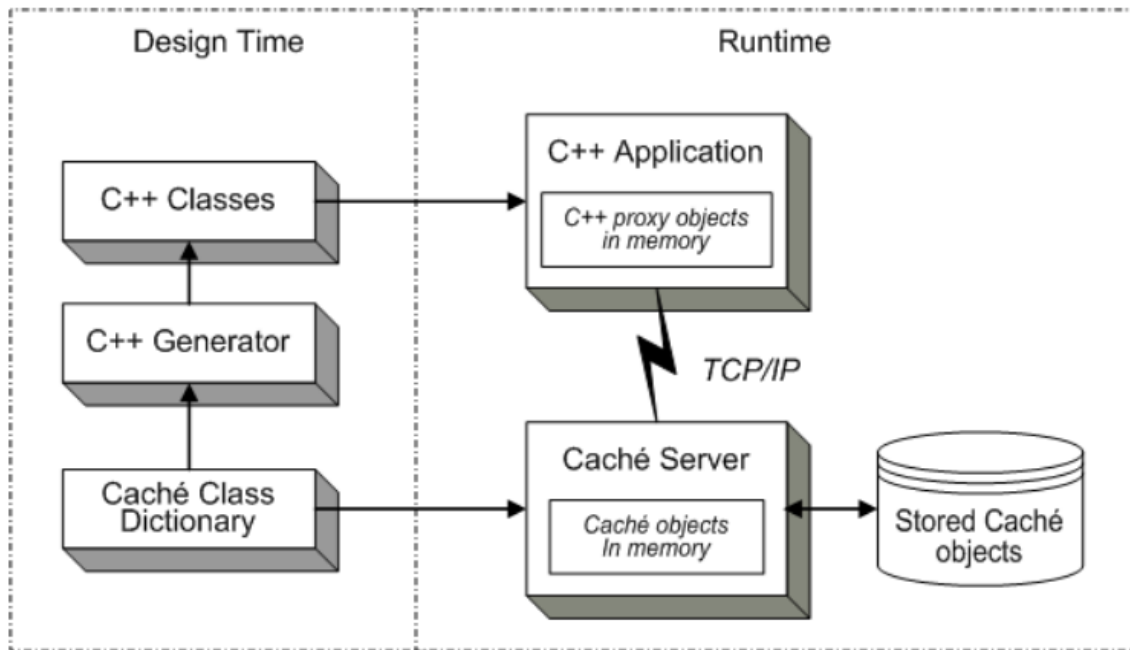


Рисунок 2.4 - Архітектура прив'язки мовою C++

Перевагами даного підходу є наступні.

1. Наявність високорівневого API для доступу до об'єктів.
2. Підтримка об'єктного доступу.

Недоліками є наступні.

1. Залежність від версії СУБД.
2. Відсутність інструментарію для крос-платформенної компіляції.

2.2.3 Використання ODBC

ODBC – стандарт API для доступу до БД для мови С, орієтований на реляційні бази даних, стандартизований ISO/IEC [4].

ODBC має наступні характеристики:

- інтерфейс рівня викликів;

- визначає стандартну граматику SQL;
- надає менеджер драйверів, щоб забезпечити роботу з одним або декількома драйверами та СУБД.

Архітектура ODBC має наступні компоненти.

1. Застосунок – виконує обробку даних та виклики функцій для обробки SQL запитів та отримання результатів.
2. Менеджер драйверів – завантажує та вивантажує драйвери за вимогою застосунків.
3. Драйвер – обробляє ODBC виклики та відсилає SQL запити до вказаного джерела даних.
4. Джерело даних – дані, до яких потребує доступу застосунок та відповідна операційна система, БД та, можливо, мережа для доступу до БД.
5. СУБД Caché реалізує стандарт ODBC 3.5, що дозволяє використовувати її в якості джерела даних ODBC.

Типова організація роботи із з'єднанням ODBC зображена на рисунку 2.5.

Для початку роботи з ODBC API необхідно викликати функцію `SQLAllocEnv`, отримавши посилання (`handle`) на середовище, а після цього викликати `SQLConnect` для встановлення з'єднання.

Потім для виконання запиту необхідно виконати дії, показані на рисунку 2.6. Спочатку необхідно виділити запит, викликавши `SQLAllocHandle`, отримавши посилання на запит, здійснити прив'язку одного або декількох параметрів.

Після завершення роботи потрібно вивільнити пам'ять, передавши відповідні посилання за допомогою функцій `SQLFreeHandle`.

2.3 Використання Java API

Прив'язки для мови Java дозволяють застосункам на мові Java отримувати доступ до класів СУБД Intersystems Caché. Існує два способи доступу до БД:

- прив'язки для мови Java дозволяють працювати з об'єктами на сервері Caché

напрямую. Для кожного класу Caché створюється відповідний клас-замісник для мови Java;

- JDBC драйвер для Caché – надає реляційний доступ до СУБД Caché. Драйвер JDBC є драйвером 4-го типу [5], що забезпечує крос-платформенність.

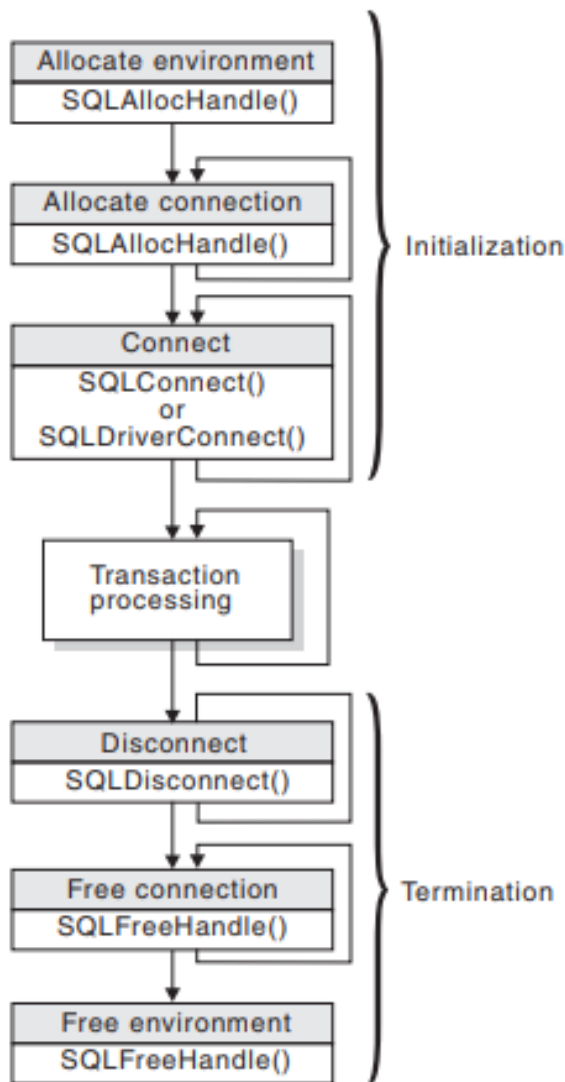


Рисунок 2.5 - Робота з середовищем та з'єднанням ODBC

Архітектура API для мови Java зображена на рисунку Рисунок 2.7 - . Це архітектура типу "клієнт-сервер", в якій Java-застосунок відіграє роль клієнта, а БД - сервера. Комунікація між БД та сервером здійснюється за допомогою TCP/IP сокета.

Перевагами використання Java разом із СУБД Caché є крос-платформенність та наявність як об'єктного, так і реляційного видів доступу.

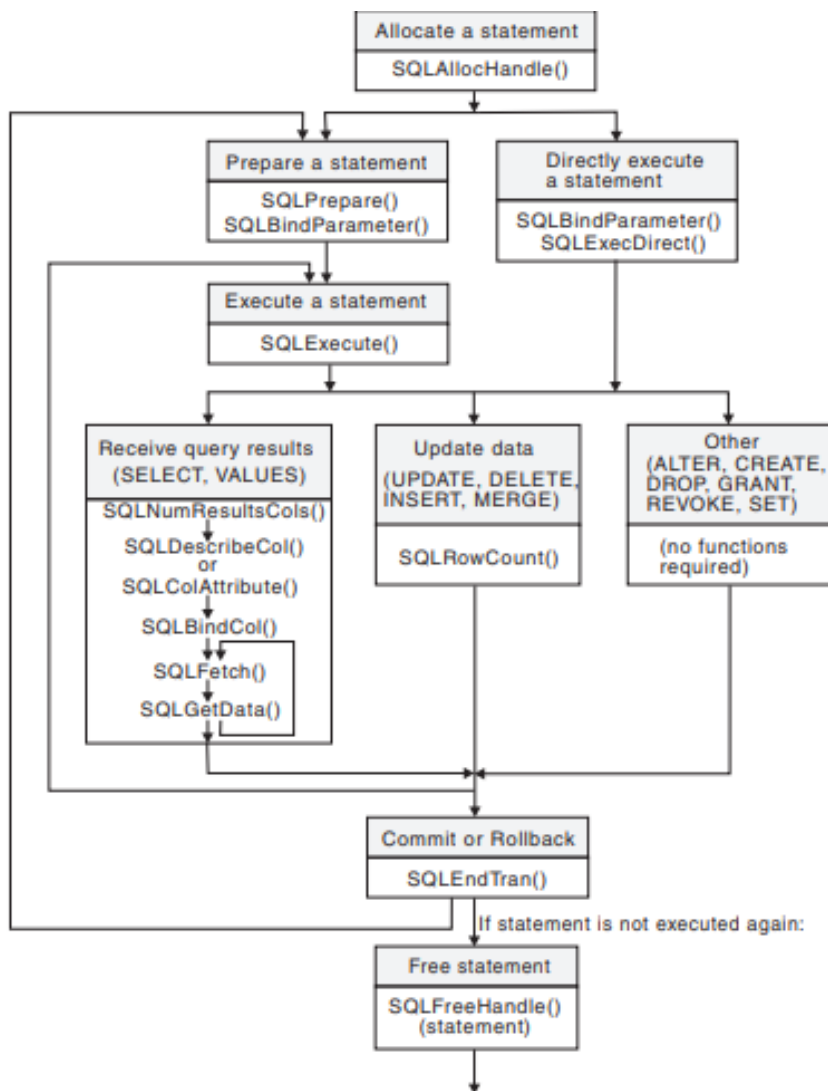


Рисунок 2.6 - Робота з даними за допомогою ODBC

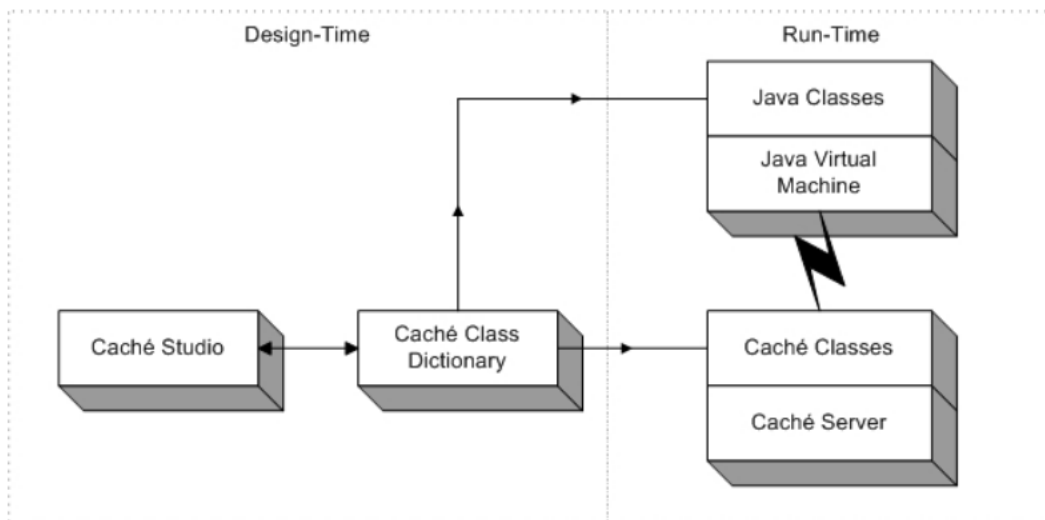


Рисунок 2.7 - Архітектура Java API

3. МЕТОДИ ТА ЗАСОБИ РЕАЛІЗАЦІЇ ODM ДЛЯ ОБ'ЄКТІВ СУБД INTERSYSTEMS CACHE МОВОЮ JAVASCRIPT

3.1 СУБД Intersystems Caché

Intersystems Caché — це сучасна система управління базами даних і середовище для швидкої розробки застосунків. Дана СУБД дозволяє істотно вдосконалити обробку і аналіз складних «великих даних», а також розробку мобільних і Web-застосунків. При цьому вимоги до апаратного забезпечення та адміністрування системи залишаються мінімальними [6].

Вбудованою мовою програмування для Caché є мова ObjectScript (COS), яка, в свою чергу, є надмножиною мови програмування MUMPS. Крім COS, Caché надає розробникам API для використання об'єктного і SQL-доступу до одних і тих самих даних.

Caché зберігає дані в багатовимірних масивах (рисунок 3.1), здатних містити ієрархічно структуровані дані. Ці ж "глобальні" структури застосовуються в мові програмування MUMPS; вони так само подібні до структур, що використовуються в MultiValue-системах.

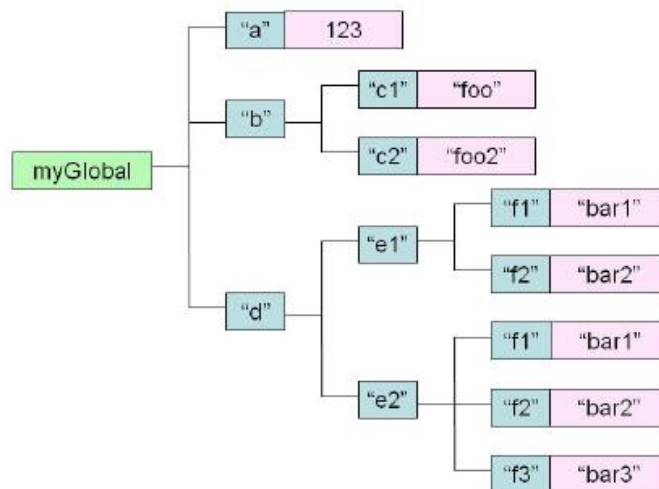


Рисунок 3.1 - Глобали СУБД Intersystems Caché

Зовнішні інтерфейси включають пряме зв'язування об'єктів C++, Java, EJB, ActiveX і .NET. Caché підтримує JDBC і ODBC для реляційного доступу. Також підтримуються XML і web-сервіси [7].

Основними компонентами СУБД Caché є наступні (рисунок 3.2).

1. TMDM – багатовимірне ядро системи що орієнтоване на роботу з транзакціями.

2. Сервер Caché Objects – призначений для подання багатовимірних структур даних ядра системи у вигляді об'єктів, що інкапсулюють як дані, так і методи їх обробки.

3. Сервер Caché SQL – подає багатовимірні структури даних у вигляді реляційних таблиць.

4. Сервер прямого доступу – надає прямий доступ до багатовимірних структур даних ядра системи.

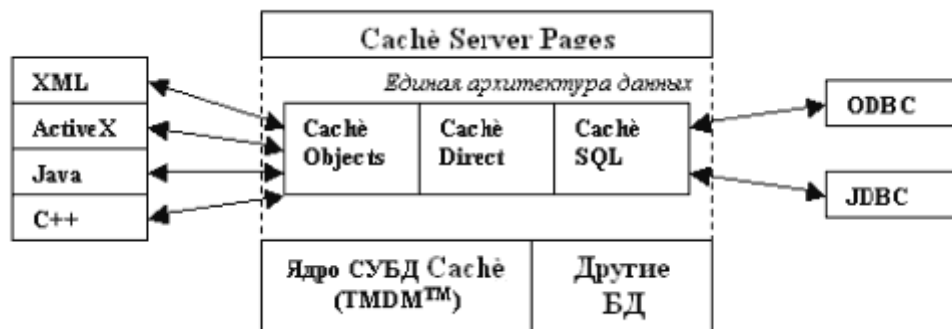


Рисунок 3.2 - Архітектура СУБД Intersystems Caché

Розглянемо докладніше призначення і функціональні можливості основних компонентів системи.

Дані в Caché зберігаються у вигляді розріджених масивів, що носять назву глобалів. Кількість індексів масиву може бути довільна. Це дозволяє описувати та зберігати структури даних довільного рівня складності. Індеси глобалів не мають типів, тобто можуть мати довільний тип даних.

Застосування розріджених масивів дозволяє оптимізувати використання об'єму жорсткого диска і скоротити час, необхідний для виконання операцій введення-виведення і вилучення даних.

В СУБД Caché реалізована розвинена технологія обробки транзакцій і вирішення конфліктів. Блокування даних проводиться на логічному рівні. Це дозволяє враховувати особливість багатьох транзакцій, які змінюють невеликий обсяг інформації. Крім цього, в Caché реалізовані атомарні операції додавання і видалення без блокування. Це знаходить застосування зокрема для лічильника ID об'єктів в базі даних.

3.1.1 Вбудовані мови програмування Intersystems Caché

Caché ObjectScript (COS) – це потужна об'єктно-орієнтована мова програмування, що вбудована в СУБД Caché. Синтаксис мови близький до синтаксису широко відомих мов програмування.

В мові розрізняють локальні та глобальні змінні. Глобальні змінні (глобали) зберігаються в базі даних, їх імена починаються з символу «^». В інших аспектах, робота з локальними та глобальними змінними однакова. В COS відсутнє оголошення змінних; єдиний тип даних – це рядок символів змінної довжини, хоча внутрішніми засобами системи результат арифметичної або логічної операції інтерпретується як число чи логічне значення. Приклад коду мовою COS наведено на рисунку 3.3 [8].

```
ClassMethod AllPersonsWithA()
{
    set rs = ##class(%ResultSet).%New()
    do rs.Prepare("select ID from Sample.Person where substr(name,1,1) = 'A'")
    do rs.Execute()
    while rs.Next() {
        set p = ##class(Sample.Person).%OpenId(rs.Get("ID"))
        set p.Office = "NY"
        write p.Name, " ", p.SSN, !
        kill p
    }
    kill rs
}
```

Рисунок 3.3 - Приклад коду мовою ObjectScript

Intersystems Caché також підтримує й інші мови, наприклад Caché Basic. Це досягається за рахунок використання віртуальної машини Caché (рисунок 3.4).



Рисунок 3.4 - Віртуальна машина Caché

3.1.2 Об'єктна модель Intersystems Caché

Об'єктна модель Caché відповідає об'єктній моделі стандарту ODMG (Object Data Management Group). Відповідно до ODMG кожен об'єкт Caché має єдиний визначений тип. Поведінка об'єкта визначається операціями (методами), а стан об'єкта — значеннями його властивостей. Властивості і операції визначають характеристики типу.

Відповідно до стандарту в Caché реалізовано два типи класів.

1. Класи типів даних (літерали).
2. Класи об'єктів (об'єкти).

Класи типів даних визначають допустимі значення констант (літералів) і дозволяють їх контролювати. Літерал не може існувати незалежно від свого значення, в той час як об'єкти мають унікальну ідентифікацію.

Класи типів даних поділяються на наступні два підкласи.

1. Атомарні.
2. Структуровані.

Атомарними літеральними типами в Caché є наступні типи даних: %String, %Integer, %Float, %Date тощо.

В Caché реалізовані два структурованих типи даних, а саме: список і масив. Кожен літерал унікально ідентифікується індексом в масиві і порядковим номером в списку.

Розрізняють два підтипи класів об'єктів — зареєстровані та незареєстровані. Зареєстровані класи мають визначену поведінку, тобто набір методів, успадкованих від системного класу %RegisteredObject і відповідають за створення нових об'єктів, а також за управління розміщенням об'єктів в пам'яті. Незареєстровані класи не мають заздалегідь визначеної поведінки, тобто розробка функцій методів класу цілком покладається на розробника.

Зареєстровані класи можуть бути двох типів – вбудовані і збережені. Вбудовані класи успадковують свою поведінку від системного класу %SerialObject. Основною особливістю зберігання екземпляра вбудованого класу є те, що об'єкти вбудованих класів існують в пам'яті як незалежні екземпляри, однак можуть бути збережені в БД тільки у разі збереження в інший об'єкт.

Основною перевагою використання вбудованих класів є мінімізація витрат, пов'язаних з можливою в майбутньому зміною набору однакових властивостей класів, представлених у вигляді вбудованого об'єкта.

Збережені класи успадковують свою поведінку від системного класу %Persistent. %Persistent надає великий набір функцій своїм нащадкам, що включає: створення об'єкта, підкачування об'єкта з БД в пам'ять, видалення об'єкта і т.п. Кожен екземпляр збереженого класу має 2 унікальних ідентифікатора — OID і OREF. OID (object ID) характеризує об'єкт, записаний в БД, тобто на фізичному носії, а OREF (object reference) характеризує об'єкт, що був завантажений з БД і знаходиться в пам'яті.

Об'єктна модель Cache повністю підтримує всі основні концепції ООП [8].

1. Успадкування — об'єктна модель Cache підтримує множинне спадкування.

2. Поліморфізм — об'єктна модель Cache дозволяє створювати застосунки цілком і повністю незалежними від внутрішньої реалізації методів об'єкта.

3. Інкапсуляція — об'єктна модель Cache забезпечує приховування окремих деталей внутрішньої будови класів від зовнішніх по відношенню до нього об'єктів або користувачів. Розділяють інтерфейс класу і конкретну реалізацію. Інтерфейс потрібен для взаємодії з іншими об'єктами. Реалізація ж приховує внутрішні особливості класу, тобто все, що не входить в інтерфейс.

3.1.3 Реляційний доступ до бази даних

Система Cache підтримує наступні види зберігання об'єктів:

- автоматичне зберігання в багатовимірній БД;
- зберігання в користувацьких структурах даних;
- зберігання в таблицях зовнішніх реляційних баз даних, доступних через шлюз

Cache SQL Gateway [9].

Відповідність між об'єктною та реляційною моделлю наведена в таблиці 3.1 - .

Таблиця 3.1 - Відповідність між об'єктними та реляційними поняттями

Об'єктне поняття	Реляційне поняття
Клас	Таблиця
Екземпляр	Рядок
Ідентифікатор (OID)	ID-стовпчик у вигляді первинного ключа
Властивість-константа	Стовпчик
Посилання на збережений об'єкт	Зовнішній ключ
Вбудований об'єкт	Окремі стовпчики
Колекція-список	Стовпчик з полем-списком
Колекція-масив	Підтаблиця
Потік даних	BLOB
Індекс	Індекс
Запис	Збережена процедура або представлення
Метод класу	Збережена процедура

3.2 Середовище виконання JavaScript node.js

Node.js – подійно-орієнтоване середовище виконання JavaScript для рушія v8. Його основним призначенням є написання масштабованих мережеских програм. Програми, написані на JavaScript з використанням асинхронного введення-виведення, зменшують накладні витрати та збільшують масштабованість застосунків.

Node.js використовує подійно-орієнтовану модель для обробки веб-запитів. Схематично дана модель зображена на рисунку 3.5.

За цієї моделі один потік виконання обробляє події (наприклад, новий HTTP запит), що надходять в чергу, одна за одною, виконуючи відповідні обробники подій [10].

Середовище виконання node.js складається з наступних основних компонентів:

- рушія для мови JavaScript v8;
- бібліотеки для крос-платформенного введення-виведення та багатопотоковості libuv;
- бібліотеки для DNS (c-ares);
- бібліотеки для криптографії OpenSSL;
- бібліотеки для компресії даних zlib.

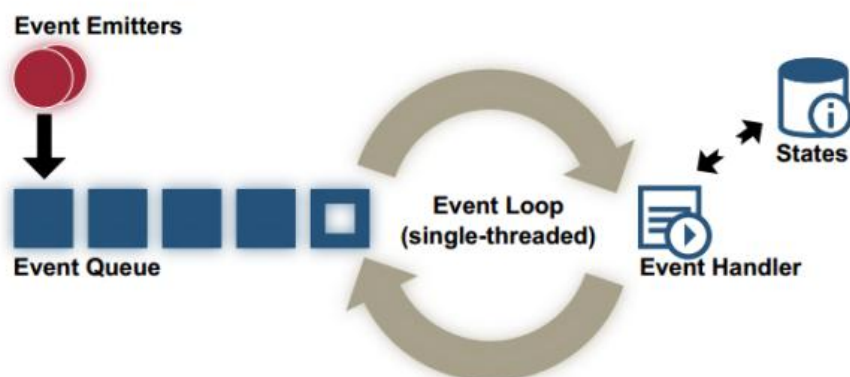


Рисунок 3.5 - Схематичне зображення подійно-орієнтованої архітектури

Архітектура середовища виконання node.js зображена на рисунку 3.6.

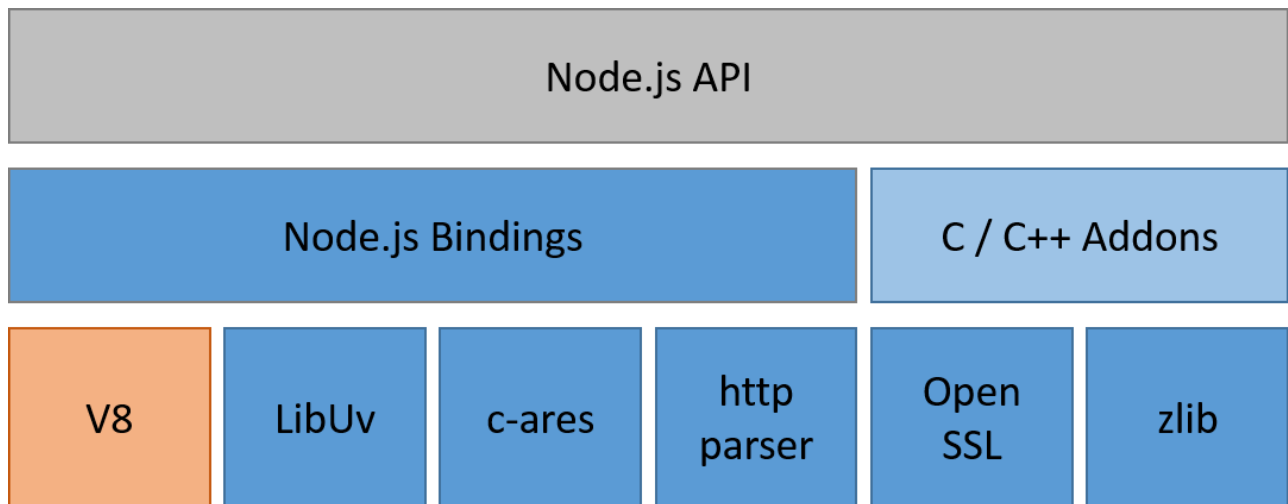


Рисунок 3.6 - Архітектура node.js

3.2.1 C++ модулі для node.js

Node.js підтримує написання застосунків мовою C++ з використанням API v8 та libuv. Дані застосунки є динамічно лінкованими бібліотеками, що дозволяють отримувати доступ до C++ коду з мови JavaScript [11].

Для того, щоб не блокувати основний потік, модулі, як правило, реалізують неблокуючий виклик за допомогою пулу потоків, що надає libuv. Типовий алгоритм роботи такого модуля зображено на рисунку 3.7.

При виклику асинхронної функції, написаної на мові C++, спочатку відбувається конвертація вхідних даних в C++ об'єкт. Потім задача на виконання функції кладеться в чергу задач, що надається libuv. Після цього libuv здійснює розподіл задач між потоками на одній з ітерацій циклу подій. Закінчивши роботу, потік викликає функцію `uv_async_send`, що посилає результат на обробку циклу подій. На наступній ітерації циклу викликається функція зворотного виклику разом із зв'язаним JavaScript кодом [12].

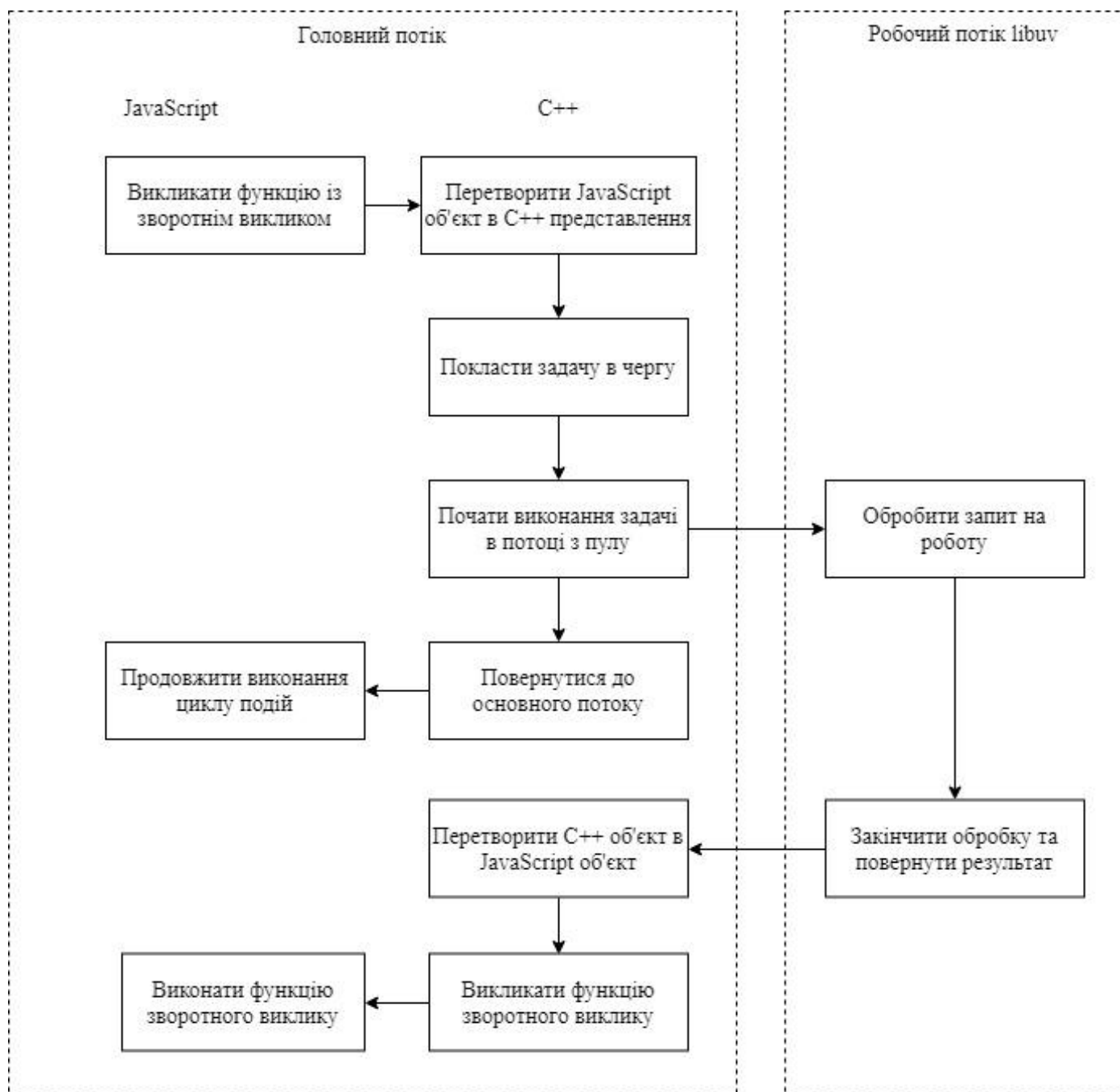


Рисунок 3.7 - Організація асинхронного виклику в модулі мовою C++

3.2.2 Альтернативні середовища виконання

Існують і інші середовища виконання JavaScript, окрім node.js, що використовують v8. До найпопулярніших слід віднести Electron.js, NW.js. Вони сумісні з C++ API node.js і призначені для створення застосунків для платформ MacOS, Linux

та Windows. Зазначимо, що модуль мовою C++ можна зробити сумісним і з даним типом середовищ виконання.

3.3 Бібліотека NAN (Native Abstractions for Node) для написання модулів C++ для node.js

При написанні модулів мовою C++ виникають наступні проблеми.

1. Несумісність між версіями node.js.
2. Необхідність написання однакового коду для асинхронного виклику (рисунок 3.7).
3. Необхідність написання однакового коду для виділення об'єктів мови JavaScript та керування їх циклом життя.

Для вирішення цих проблем була написана бібліотека з відкритим вихідним програмним кодом NAN. Дана бібліотека надає наступний набір примітивів.

1. Типи для аргументів методів.
2. Типи для оголошення методів.
3. Типи для керування життєвим циклом об'єктів (вказівники та області видимості).
4. Методи для створення примітивних типів.
5. Методи для конвертації типів.
6. Шаблони для виконання асинхронних викликів.
7. Методи для виконання коду мовою JavaScript.
8. Методи для обробки помилок.
9. Допоміжні методи для роботи з рушієм v8.

Перевагою використання даної бібліотеки у порівнянні із прямим використанням API v8 є незалежність від версії середовища виконання та менший об'єм продубльованого коду.

3.4 Бібліотека Bluebird як реалізація специфікації Promise/A+

3.4.1 Promise API

Promise – альтернативний до функцій зворотного виклику спосіб представлення асинхронних обчислень.

Даний інтерфейс є контейнером для значення, що не є відомим на момент створення об'єкту [12].

Об'єкт Promise може знаходитися у наступних станах:

- очікування (обчислення не виконано та незавершене);
- виконано (операція завершена успішно);
- відхилено (операцію завершено з помилкою).

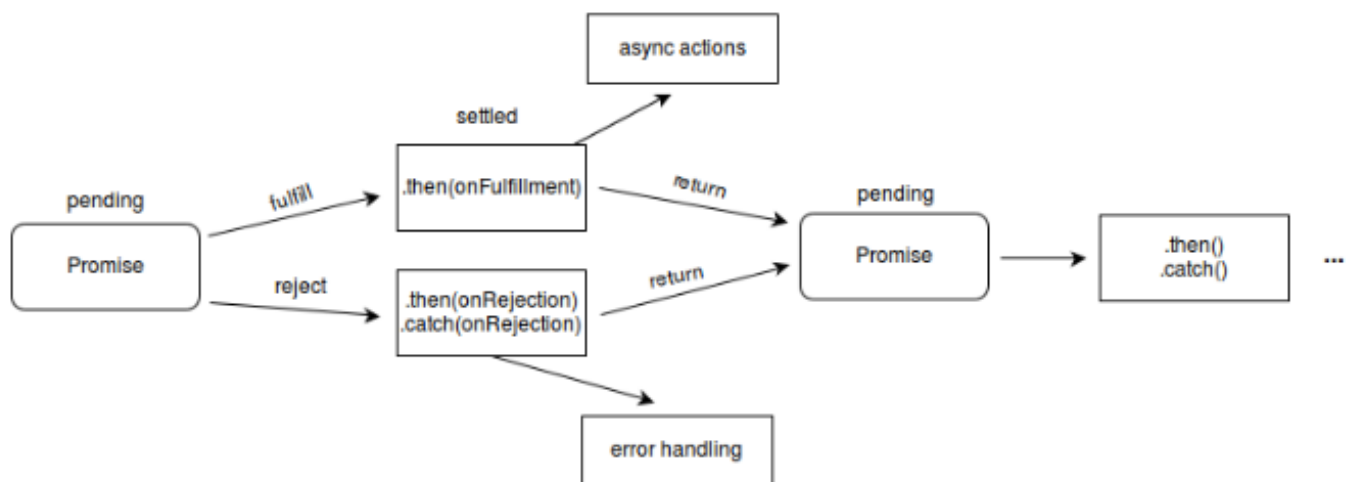


Рисунок 3.8 - Діаграма станів об'єкту Promise

Кожен метод об'єкту Promise повертає інший об'єкт Promise, дозволяючи утворювати ланцюг асинхронних обчислень, що подібний до синхронного коду.

Основними методами інтерфейсу Promise є:

- then – обробити значення, коли воно стане доступним;
- catch – обробити помилку.

3.4.2 Огляд бібліотеки Bluebird

Бібліотека Bluebird реалізує інтерфейс Promise та надає набір методів для перетворення методів, що використовують функції зворотного виклику:

- `promisify` – перетворити метод, що приймає функцію зворотного виклику, на Promise.
- `promisifyAll` – застосувати `promisify` до всіх методів об'єкта.

У порівнянні з вбудованими в платформу `node.js`, набір методів бібліотеки має більший функціонал для роботи з Promise, що дозволяє обробляти масиви об'єктів Promise:

- `map` – відображує колекцію аргументів на колекцію результатів;
- `filter` – фільтрує елементи за заданим предикатом;
- `reduce` – виконує згортку об'єктів за допомогою бінарного оператора;
- `mapSeries` – виконує операцію `map` послідовно.

3.5 Бібліотека `nanodbc` для з'єднання з БД

Бібліотека `nanodbc` є бібліотекою з відкритим вихідним кодом мовою C++ для роботи з базами даних, що підтримують ODBC. Вона надає об'єкти для роботи з БД, що базуються на використанні ODBC API, а також реалізує обробку помилок за допомогою механізму виключень. Основними об'єктами, що надаються є:

- `connection` (з'єднання з БД) – надає методи для отримання інформації до БД та виконання запитів;
- `transaction` (транзакція) – основними операціями є `commit` (підтвердити транзакцію) та `rollback` (відкотити транзакцію);
- `statement` (запит до БД) – дозволяє виконувати параметризовані збережені запити;
- `result` (результат запиту до БД) – дозволяє обробляти результати запиту;
- `database_error` – виключення, що відображає помилку в БД.

3.6 Порівняльна характеристика методів доступу до БД

Для вибору найкращого способу доступу до БД складемо їх порівняльну характеристику, визначивши їх переваги та недоліки для вирішення даної задачі. Порівняльна характеристика наведена у таблиці 3.2.

Таблиця 3.2 - Порівняльна характеристика методів доступу до СУБД Caché

Шлях доступу	Переваги	Недоліки
C++	підтримка багатьох ОС, швидкодія, об'єктний доступ	SQL доступ обмежений, залежність від версії СУБД, відсутність кросплатформеної збірки
SOAP, REST, Web services	кросплатформеність, незалежність від мови програмування	великі витрати на виклик, необхідність написання сервісів в БД складність підтримання SQL
JDBC, Java APIs	кросплатформеність	необхідність міжпроцесної взаємодії з JVM
JavaScript API	кросплатформеність	відсутня підтримка останніх версій node.js, закритий доступ

Для рішення поставленої задачі було вирішено використовувати реляційний варіант доступу до відношень (об'єктів) за допомогою ODBC. Даний метод має наступні переваги.

1. Надає стандартний інтерфейс доступу до бази даних, що спрощує міграцію з інших баз даних.
2. Дозволяє використовувати SQL, наприклад, для застосунків, що потребують складної вибірки та аналізу даних, при цьому менше вибагливих до затримок (аналітика).
3. Крос-платформеність. ODBC підтримується на Windows, Linux та MacOS.

4. Потокобезпечність. Потокобезпечність є важливою характеристикою, адже написання застосунків для node.js вимагає використання пулу потоків для того, щоб виконувати потенційно тривалі блокуючі операції (якими є запити до бази даних).

5. Можливість використання існуючих бібліотек для роботи з SQL в node.js.

6. Сумісність між версіям Intersystems Caché.

7. Економічна доцільність.

8. Можливість виконувати складні запити.

Недоліки використання ODBC наступні.

1. Менша швидкодія порівняно з об'єктним чи прямим доступом до глобалів.

2. Об'єктний доступ необхідно емулювати за допомогою ORM (об'єктно-реляційного відображення).

Альтернативними варіантами до ODBC є:

- об'єктний JavaScript API;
- C++ API для Caché.

Основним недоліком JavaScript API є закритість вихідного коду.

Недоліком C++ API для Caché є те, що він залежить від версії Caché. Це значно збільшить затрати на підтримку програмного продукту.

Враховуючи наведені вище фактори для початкової реалізації продукту було вибрано ODBC API, адже це зменшить витрати на підтримку, при цьому інший заявлений функціонал можна реалізувати на прийнятному рівні.

3.7 Інші засоби розробки

Сторонніми залежностями та інструментами, що необхідні для розробки є:

- система контролю версій Git;
- менеджер пакетів nodejs npm;
- тестові бібліотеки для мови JavaScript: chai, mocha;
- компілятор мови C++ (Visual Studio 2017 для Windows, gcc 6.3.0+ для Linux,

MacOS);

- хмарне середовище для збірки travis-ci;
- середовище для контейнеризації Docker;
- система збірки cmake (Linux, MacOS, Windows);
- утиліта make (Linux, MacOS);
- статичний аналізатор коду C++ cppcheck;
- лінтер для JavaScript jslint.

Продукт публікується до репозиторію npm.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ODM ДЛЯ ОБ'ЄКТІВ СУБД INTERSYSTEMS CACHE МООВОЮ JAVASCRIPT

4.1 Засоби реалізації об'єктно-реляційного відображення

Основними шаблонами проектування, що застосовуються для об'єктно-реляційного відображення є:

- Active Record;
- Data Mapper.

4.1.1 Шаблон Active Record

При використанні даного шаблону (рисунок 4.1) кожному відношенню (таблиці) ставиться у відповідність єдиний рядок у таблиці. Після створення об'єкта новий рядок додається до таблиці. Кожен об'єкт, що завантажується, отримує інформацію з БД. Об'єкт має властивості для кожного рядка в БД.

Перевагами данного методу є простота реалізації та використання.

Недоліками є складність у тестуванні без БД, а також низька ефективність (кожна модифікація в об'єкт одразу потрапляє до БД, що призводить до більшої кількості дрібних модифікацій).

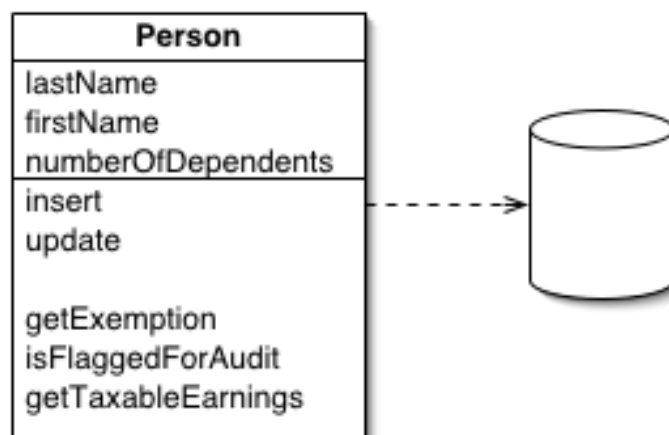


Рисунок 4.1 - Приклад шаблону Active Record

4.1.2 Шаблон Data Mapper

При використанні даного шаблону проектування створюється рівень доступу до даних (Data Access Layer), що виконує двостороннє відображення між БД та представленням об'єкта в пам'яті. Метою шаблону є відокремлення представлення даних від їх зберігання. Рівень складається з одного або більше об'єктів доступу до даних (Data Access Object), що виконують перенесення даних (рисунок 4.2).

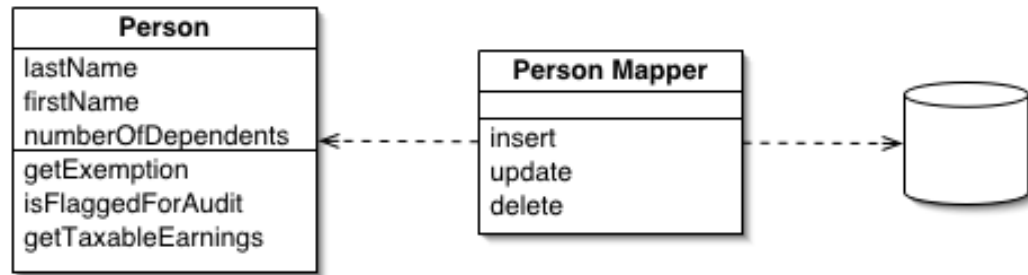


Рисунок 4.2 - Приклад шаблону Data Mapper

Перевагою даного методу є простота у тестуванні (об'єкт повністю незалежний від БД), а також можливість оптимізації доступу до БД.

До недоліків слід віднести складність у реалізації [13].

4.1.3 Вибір шаблону реалізації

Для реалізацій об'єктно-реляційного відображення в даній роботі було обрано шаблон Active Record. Причини вибору наступні.

1. Даний підхід загальноприйнятий в СУБД Caché (клас %Persistent).
2. СУБД Caché не є реляційною по суті, тому зберігання неатомарних атрибутів (наприклад, списків) є нормальним. Це означає, що в основному випадку одній таблиці відповідатиме один клас (таблиця 3.1 -).
3. Простий в реалізації та використанні.

4.2 Архітектура та структура програмного рішення

Загальна структура Caché ODM зображена на рисунку 4.3.

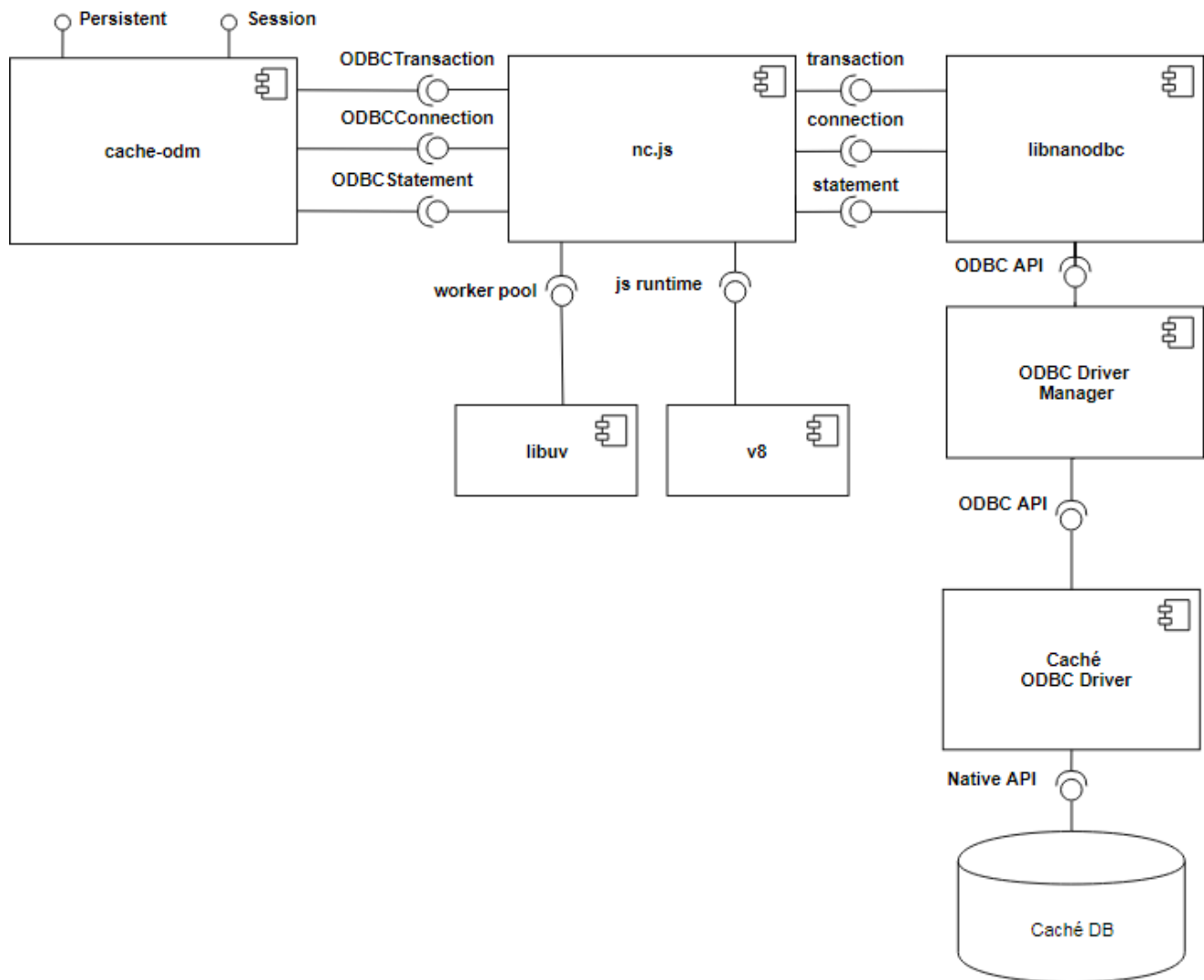


Рисунок 4.3 - Діаграма компонентів Caché-odm

Програмний продукт складається з наступних модулів.

1. Cacheodbc – реалізує примітиви доступу до бази даних Intersystems Caché, а саме:

- запит (об'єкт ODBCStatement);
- з'єднання (об'єкт ODBCConnection);
- транзакція (об'єкт ODBCTransaction).

2. Caché-odm – виконує об'єктно-документне (реляційне) відображення. Надає прикладному програмісту можливість високорівневої маніпуляції об'єктами. Модуль надає два інтерфейси: Persistent – збережений об'єкт (подібно до мови ObjectScript)

та Session – об’єкт сесії, що призначений для виконання сукупності операцій над збереженими об’єктами в одній транзакції (за потреби).

3. libnanodbc – бібліотека для мови C++, що спрощує роботу з ODBC, надаючи об’єктні абстракції (connection – з’єднання, transaction – транзакція, statement – запит, операція) та обробку помилок за допомогою виключень.

Програма Caché-ODM використовує пул потоків та примітиви синхронізації (м’ютекси, умовні змінні, семафори), що надаються libuv, а також маніпулює об’єктами JavaScript, використовуючи прикладний програмний інтерфейс рушія v8 (за допомогою набору утиліт NAN, на схемі не показаний).

Також для функціонування програми необхідно мати встановленим та налаштованим ODBC драйвер для СУБД Intersystems Caché.

В якості реалізації Promise API для JavaScript використовувалась бібліотека Bluebird.

4.2.1 Модуль Cachéodbc

Для реалізації доступу до СУБД Intersystems Caché в мові JavaScript було написано розширення мовою C++, що реалізує основні операції для доступу до бази даних у вигляді об’єктів JavaScript, а саме:

- ODBCConnection (з’єднання) реалізує методи:
 - connect – з’єднатися з базою даних;
 - disconnect – закрити з’єднання;
 - query – виконати запит;
 - execute – виконати команду, що не повертає жодного результату;
- ODBCStatement (запит) реалізує методи:
 - prepare – підготувати запит;
 - query – виконати запит;
 - execute – виконати команду;
- ODBCTransaction (транзакція) реалізує методи:
 - begin – почати транзакцію;

- `commit` – підтвердити транзакцію;
- `rollback` – скасувати транзакцію.

Всі методи виконано неблокуючими: кожна операція виконується в окремому потоці із пулу потоків, що надає `libuv`, а основний потік `v8` звільняється для обробки наступних запитів, зв'язок з основним потоком здійснюється за допомогою зворотного виклику (рисунок 3.7).

Модуль використовує сторонню бібліотеку `nanodbc` для виконання більшості запитів та операцій з драйвером ODBC, проте деякі операції виконуються з драйвером напряму. Основною функціональністю даного модуля є конвертація типів даних мови JavaScript у типи даних ODBC, а також представлення результатів запитів у вигляді масиву об'єктів JavaScript. Для забезпечення сумісності із різними версіями `node.js` використовується бібліотека `NAN` (native abstractions for node).

В фрагменті коду на рисунку 4.4 створюється з'єднання, а потім виконується метод класу `PersonSets` з пакету `Sample`, а результат виводиться на екран.

```
const connection = require('nc').createConnection();
const dsn = ...;
connection.connect(dsn, () => {
  console.log('Створено з'єднання');
  connection.query("CALL Sample.PersonSets('D', 'NY')", (err, res) => {
    console.log('Результат');
    console.log(JSON.stringify(res));
    connection.disconnect(() => {
      console.log(`З'єднання закрито`);
    });
  });
});
```

Рисунок 4.4 - Приклад використання модуля `Cachéodbc`

Розглянемо детальніше будову даного модуля. На рисунку 4.5 зображено діаграму класів, що реалізують інтерфейс, доступний з мови JavaScript.

Набір класів `ODBCStatement`, `ODBCConnection` та `ODBCTransaction` покладаються на допоміжний клас `SafeUnwrap` для конвертації із об'єкта мови JavaScript у вказівник на клас, який вони зберігають. Наприклад, у випадку класу

ODBCConnection це UVMonitor<ConnectionAwareStatement>, де UVMonitor – клас, що реалізує монітор для організації взаємного виключення при доступі до об'єкта з'єднання.

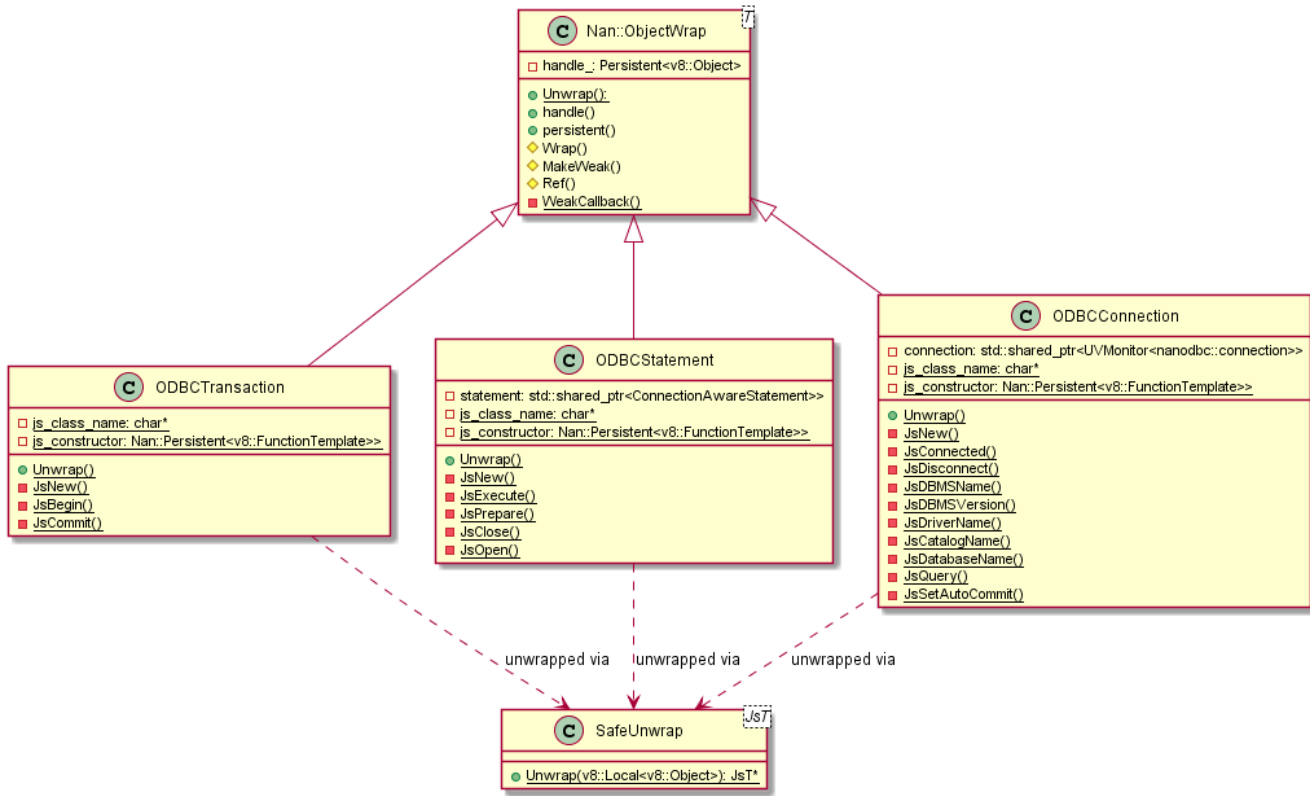


Рисунок 4.5 - Діаграма класів C++, що реалізують JavaScript-класи

Методи класів ODBCConnection, ODBCStatement та ODBCTransaction один до одного відповідають методам, що доступні в мові JavaScript (без приставки Js). Опис даних методів наведено вище.

Класи ODBCConnection, ODBCStatement та ODBCTransaction делегують роботу об'єктам класів:

- nanodbc::connection;
- ConnectionAwareStatement;
- ConnectionAwareTransaction.

Делегація відбувається за допомогою класу SingleResultWorker (рисунок 4.6), що успадковується від класу AsyncWorker бібліотеки Nan. В класі перевизначені методи Execute та HandleOKCallback. Таким чином реалізується паттерн

проектування «шаблонний метод» [14]: в даному випадку основна логіка реалізована в класі `AsyncWorker`, а в дочірньому класі лише уточнено, що повинно виконуватися. В даному випадку в методі `Execute` описується алгоритм виклику об'єкта-команди з типом `CommandT`, що виконує над об'єктом з типом `OwnerT` дію, задану командою, приймаючи список аргументів `Args...`, та повертає результат типу `ResultT`. В методі `HandleOKCallback` відбувається виклик функції зворотного виклику із значенням, що було обчислене в методі `Execute`. У разі помилки у методі `Execute` викликається метод `SetErrorMessage`, що генерує помилку. При цьому обробка помилки вже реалізована класом `AsyncWorker`.

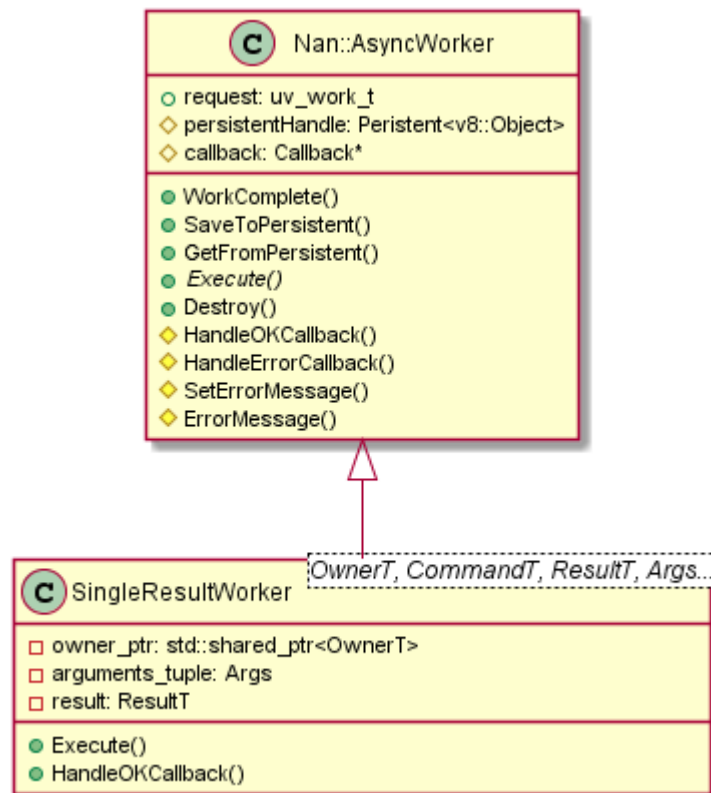


Рисунок 4.6 - Клас `SingleResultWorker`

Виклик методів об'єкту `ConnectionAwareTransaction` відбувається за рахунок виконання відповідної команди `TransactionCommand` із об'єкту `ODBCTransaction` за допомогою використання об'єкту `SingleResultWorker`. Для вибору команди

використовуються поліморфізм часу компіляції [15]. Тип команди TransactionCommand параметризований типом TransactionCommands – перерахуванням можливих команд (рисунок 4.7).

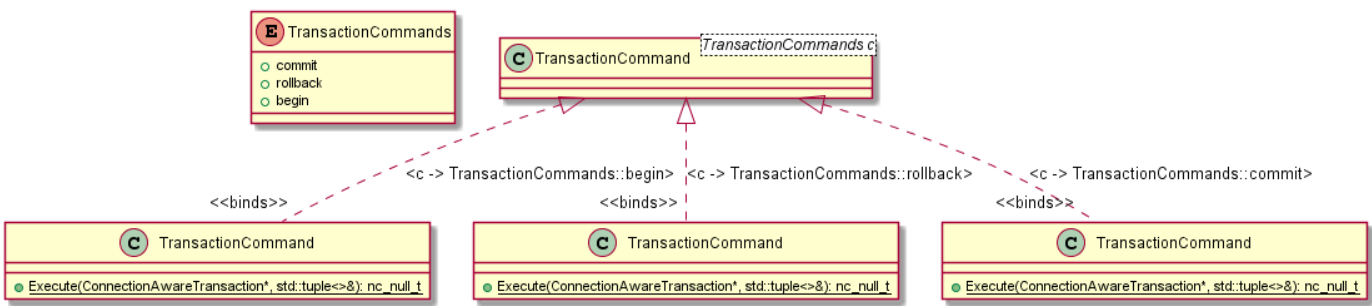


Рисунок 4.7 - Ієрархія TransactinCommand

Ієрархія класів ConnectionCommand та StatementCommand аналогічна, за винятком заміни перерахування TransactionCommands на перерахування ConnectionCommands та StatementCommands відповідно (рисунок 4.8).

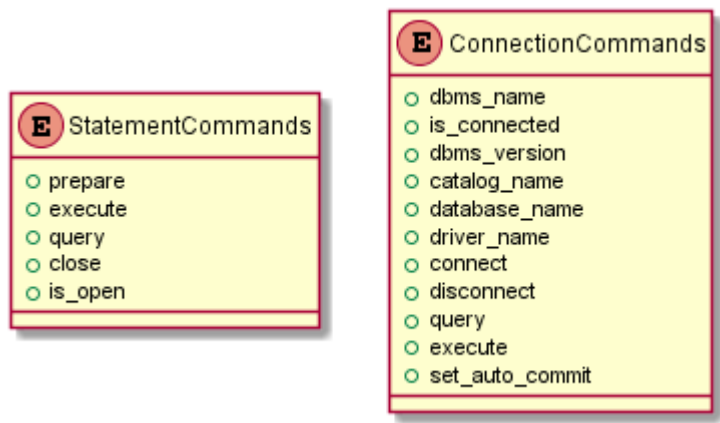


Рисунок 4.8 - Перерахування ConnectionCommands та StatementCommands

Самі класи ConnectionAwareStatement та ConnectionAwareTransaction реалізують послідовність викликів ODBC API, що синхронізовані по поточному з'єднанню, а клас panodbc::connection є класом бібліотеки panodbc. Методи та властивості даних класів наведені на рисунку 4.9.

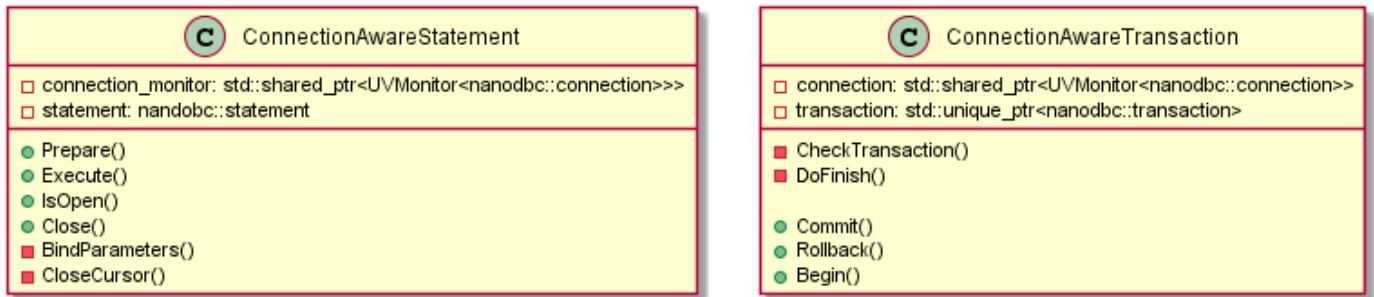


Рисунок 4.9 - Класи ConnectionAwareStatement та ConnectionAwareTransaction

Методи класу ConnectionAwareStatement наведені в таблиці 4.1.

Таблиця 4.1 - Методи класу ConnectionAwareStatement

Назва методу	Призначення
Prepare(query, tout)	Підготувати запит до БД із заданим в рядку <i>query</i> запитом та таймаутом <i>tout</i> (за замовчуванням тайм-аут відсутній)
Execute(bindings)	Виконати запит до БД із заданими в масиві <i>bindings</i> значеннями
IsOpen	Повертає значення «істина» тоді і лише тоді, коли з'єднання відкрите
BindParameters	Здійснити прив'язку параметрів до фактичних значень

Методи класу ConnectionAwareTransaction наведені в таблиці 4.2.

Таблиця 4.2 - Методи класу ConnectionAwareTransaction

Назва методу	Призначення
CheckTransaction	Перевірити чи користувач виконав метод Begin(), інакше згенерувати виключення
Begin	Почати транзакцію
Commit	Підтвердити транзакцію
Rollback	Скасувати транзакцію

4.2.2 Модуль Cache-ODM

В даному модулі реалізується об'єктно-документне відображення за шаблоном Active Record (розділ 4.1). Основним класом є Persistent. В ньому реалізуються методи доступу, наведені в таблицях 4.3 та 4.4, що дозволяють маніпулювати

об'єктами в базі даних. Програмний інтерфейс максимально наближений до того, що використовується в мові JavaScript.

Таблиця 4.3 - Методи класу Persistent

Назва	Опис
save()	зберегти об'єкт
attach()	приєднати об'єкт до поточної сесії (підтягнути дані з бази)
delete()	видалати об'єкт з бази даних

Таблиця 4.4 - Статичні методи класу Persistent

Назва	Опис
existsId(id)	повертає значення «істина», якщо об'єкт з вказаним ключем <i>id</i> існує в БД
openId(id, projection)	відкриває об'єкт з атрибутами, вказаними в масиві <i>projection</i> в базі даних, або повертає <i>null</i> , якщо такий об'єкт відсутній
findBy(map, projection)	відкриває об'єкт з атрибутами, вказаними в масиві <i>projection</i> , або повертає <i>null</i> , якщо такий об'єкт відсутній
findAll(projection)	знайти всі об'єкти в таблиці з атрибутами, вказаними в масиві <i>projection</i>

Іншим основним методом для роботи з БД є клас *Session* (сесія), що реалізує наступні шаблонні методи доступу до бази даних, що описані в таблиці 4.45. Всі методи повертають об'єкт інтерфейсу *Promise*. Даний клас дозволяє зручно виконувати типові операції з об'єктами БД та компонувати їх.

У реалізації інтерфейсу беруть участь наступні допоміжні класи:

1. *PersistentProху* – об'єкт-замісник [14], що відслідковує зміни властивостей та використовується з метою оптимізації доступу до бази даних.

2. *ConnectionPool* – пул з'єднань з базою даних, що зберігає підготовлені з'єднання та використовується для прискорення виконання запитів (усуває

необхідність встановлювати з'єднання щоразу).

3. Connection – об'єкт-з'єднання з базою даних.

Таблиця 4.5 - Статичні методи класу Session

Назва	Опис
transact(doInTransaction)	виконує дії, вказані у функції <i>doInTransaction</i> в транзакції
exec(action)	виконує дії, вказані у функції <i>action</i> в, транзакційність при цьому не гарантується
destroy()	вивільнити ресурси, пов'язані із сесією

Дані класи не є частиною публічного інтерфейсу, а використовуються для оптимізації доступу до СУБД. Повна діаграма взаємодії між програмними компонентами наведена на рисунку 4.10.

Для виконання транзакції користувач викликає статичний метод `transact` об'єкта `Session`, передаючи функцію зворотного виклику. Після цього `Session` дістає об'єкт з пулу за допомогою метода `acquire`. Викликає метод `beginTransaction`, виконує дії, вказані користувачем (в даному прикладі відкриття за ключем). Клас `Persistent` дістає об'єкт з бази та повертає об'єкт-замісник (`Proxy`), що відслідковує зміну властивостей об'єкта в ході транзакції. Після виклику методу `save` відправиться SQL запит лише на змінені властивості. По завершенні операцій зворотного виклику, `Session` завершує транзакцію та повертає результат користувачу. У разі неуспішної операції, користувачеві повертається помилка, а транзакція скасовується. У разі виклику методу `exec` послідовність аналогічна, проте транзакція не створюється.

Це забезпечує швидкодію, проте жертвує консистентністю даних.

Також для оптимізації швидкодії з кожним об'єктом з'єднання (`Connection`) асоційовано LRU-кеш, що зберігає останні виконані запити. Дана стратегія кешування дозволяє не повторювати декілька разів створення однакових запитів, що зустрічаються найчастіше, та зменшити навантаження на БД.

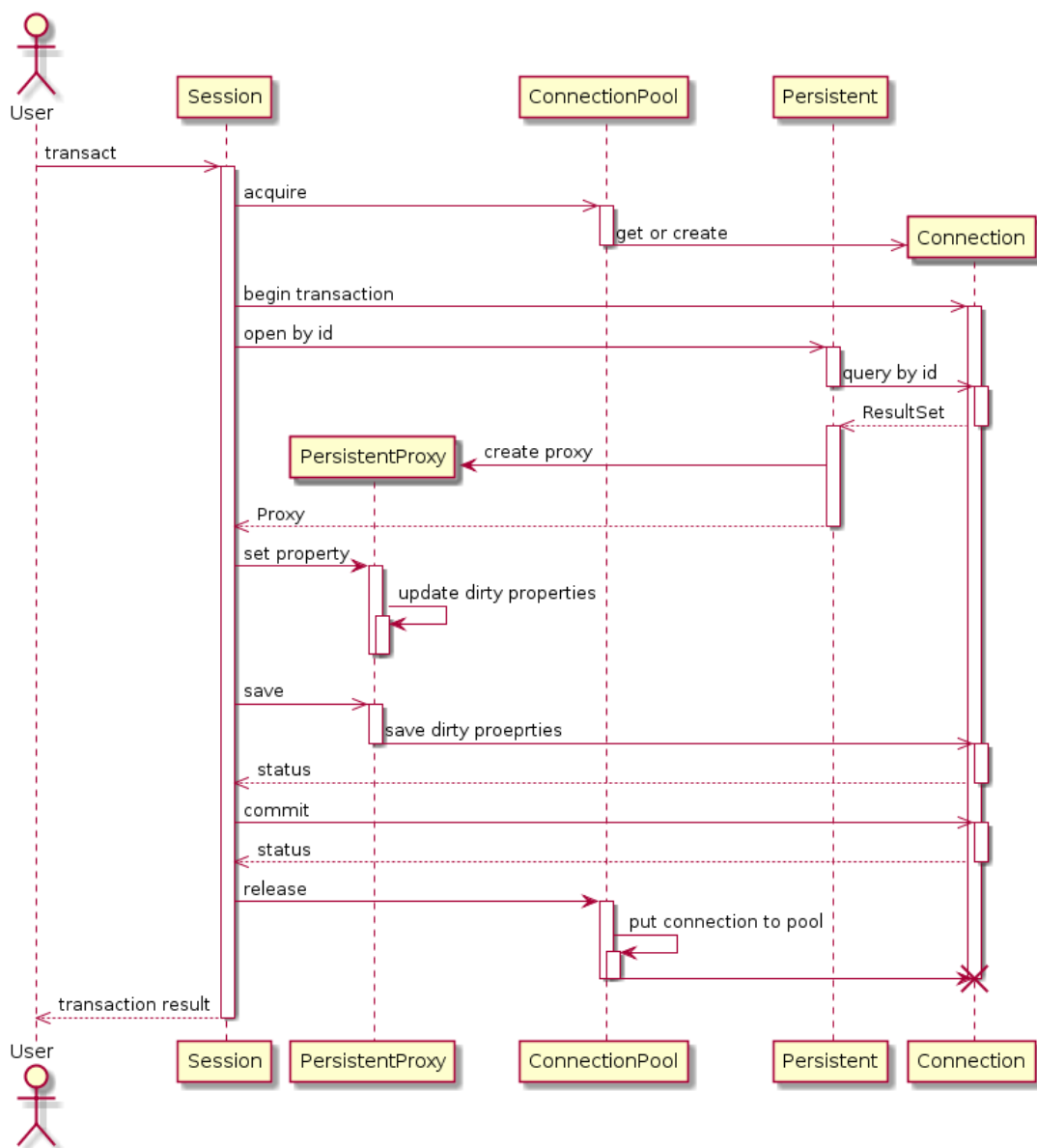


Рисунок 4.10 - Діаграма взаємодій між програмними компонентами для однієї транзакції

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ODM ДЛЯ ОБ'ЄКТІВ СУБД INTERSYSTEMS CACHE ДЛЯ МОБИ JAVASCRIPT

Для роботи з прикладним програмним інтерфейсом необхідно дотримуватися вимог з інсталяції та використання програмного продукту.

5.1 Інсталяція та системні вимоги для роботи з програмним продуктом

5.1.1 Технічні характеристики

Розроблений прикладний програмний інтерфейс призначається для середовища виконання JavaScript node.js. Для коректної роботи з системою повинен бути встановлений node.js та менеджер пакетів npm (node package manager). Підтримуються версії node.js 6, 8 та 10.

Мінімальні технічні характеристики наведені в таблиці 5.1.

Таблиця 5.1 - Мінімальні технічні вимоги

Характеристика	Значення
Оперативна пам'ять	2 Gb
Процесор	x86, Intel / AMD (2 Cores, 1M Caché, 1.66 Ghz)
Жорсткий диск	20 Gb

5.1.2 Операційна система

Основною операційною системою, на яку орієнтований програмний продукт, є Linux, проте програма також функціонує на Windows та MacOS. Також для зручності розробника надається Dockerfile, що дозволяє запускати ізольоване Linux-середовище (контейнер) на різних операційних системах.

Підтримуються версії Windows, починаючи з 7-ї версії. Для доступу до СУБД необхідно налаштувати відповідні джерела даних ODBC для встановлення з'єднання з базою даних.

Для UNIX-подібних операційних систем необхідно мати менеджер драйверів `unixodbc` та стандартну бібліотеку `libstdc++.so.6`, а також задати налаштування `odbc.ini` та `odbcinst.ini` в папці налаштувань `/etc` або `/usr/local/etc`.

Доступна також можливість збірки під інші операційні системи (Solaris, FreeBSD) та апаратні платформи (PowerPC). Для цього необхідно самостійно будувати вихідні файли програми за допомогою системи збірки CMake.

5.1.3 Драйвер ODBC

Для забезпечення SQL доступу до БД необхідно встановити менеджер драйверів ODBC (для Linux та MacOS) та драйвер ODBC Intersystems (для всіх операційних систем) та налаштувати джерело даних ODBC (рисунок 5.1).

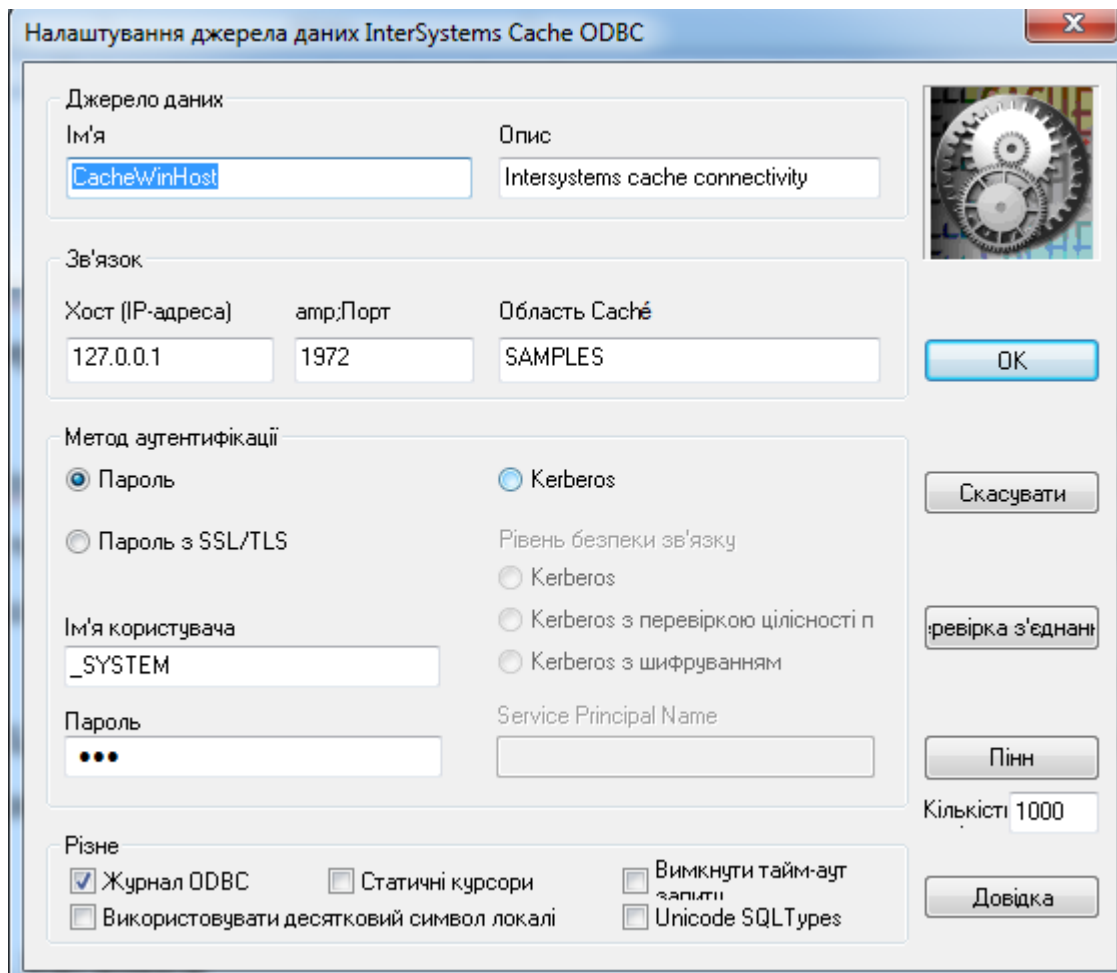


Рисунок 5.1 - Налаштування джерела ODBC для ОС Windows

Установити менеджер драйверів можна за допомогою менеджера пакетів.

Наприклад:

`apt-get install unixodbc` (для дистрибутивів `debian`, `ubuntu` та похідних)

`yum install unixodbc` (для дистрибутивів `CentOS`, `RHEL` та похідних)

`brew install unixodbc` (для `MacOS` та `homebrew`)

Для `Linux` та `MacOS` необхідно відредагувати файл `/usr/local/etc/odbc.ini` та налаштувати джерело даних у форматі "ключ-значення".

5.2 Сценарії роботи користувача

5.2.1 Конфігурація з'єднання з СУБД

Для конфігурації з'єднання з базою даних необхідно виконати наступні дії.

1. Включити модуль `Cache-odm` в проект.
2. Створити об'єкт бази даних.

Можна сконфігурувати наступні параметри (рисунок 5.2):

- `dsn` – ім'я джерела даних ODBC;
- `logLevel` (`debug`, `info`) – рівень повідомлень у журналі. Рівень `debug` додатково відображає інформацію на екран;
- `defaultNamespace` – простір імен за замовчуванням;
- `poolSizeMin` – мінімальний розмір пулу з'єднань до БД;
- `poolSizeMax` – максимальний розмір пулу з'єднань до БД.

```
const odm = require('cache-odm');

const db = orm({
  'dsn': 'DSN=CacheHost',
  'logLevel': 'info',
  'defaultNamespace': 'Sample',
  "poolSizeMin": 3,
  "poolSizeMax": 10
});
```

Рисунок 5.2 - Створення об'єкта бази даних

5.2.2 Робота з об'єктами СУБД Caché

Алгоритм використання прикладного програмного інтерфейсу наступний.

1. Успадкуватися від класу Persistent.
2. За необхідності присвоїти властивість класу description, вказавши наступні властивості: namespace – простір імен та name – назва класу.
3. Викликати метод transact або exec, вказавши у методі transact або exec послідовність дій, які потрібно виконати.

Приклад використання прикладного програмного інтерфейсу наведено на рисунку 5.3. В прикладі визначається об'єкт класу Employee (співробітник); перевизначаються методи для пошуку всіх співробітників (findAll) та методи отримання і установки значень (get та set) для роботи зі списком улюблених кольорів, що зберігається у вигляді рядка, розділеного комами.

```
const employeeFindAllProjection = [ 'ID', 'Name', 'Title', 'Office_City', 'Salary' ];
class Employee extends Persistent {
  static findBy(keyValue) {
    return super.findBy(keyValue, employeeFindAllProjection);
  }

  static findAll() {
    return super.findAll(employeeFindAllProjection);
  }

  set FavoriteColorsList(value) {
    if (Array.isArray(value)) {
      this.FavoriteColors = value.join(',');
    } else if (typeof value === 'string') {
      this.FavoriteColors = value;
    } else {
      throw new TypeError('Illegal value type for FavouriteColorsList');
    }
  }

  get FavoriteColorsList() {
    return this.FavoriteColors && this.FavoriteColors.split(',');
  }
}
```

Рисунок 5.3 - Приклад використання програмного продукту

Для виконання операцій з результатом слід використовувати функції `map` та `tap` для з'єднання асинхронних операцій як показано на рисунку 5.4. Результат виконання запиту зображений на рисунку 5.5.

```
const Session = db.Session;
Session.exec(() => Employee.findBy({
  Name: 'Edison,Frances X.'
}))
  .map(result => {
    if (result.length === 1) {
      return result[0];
    } else {
      throw Error('Expecting unique result');
    }
  })
  .tap(employee => {
    console.log(`Employee has title: ${employee.Title}`);
    console.log(`Employee has salary: ${employee.Salary}`);
  })
  .finally(() => Session.destroy());
```

Рисунок 5.4 - Приклад використання API для пошуку співробітників

```
Employee has title: Assistant Accountant
Employee has salary: 11378
```

Рисунок 5.5 - Результат виконання та обробки запитів

5.2.3 Виконання SQL-запитів

Програмний продукт також надає підтримку для SQL доступу до об'єктів (включаючи розширення `Caché`). Для отримання SQL-доступу до об'єктів необхідно скористатися об'єктом `Reader`, що надає доступ до поточного з'єднання. Приклад використання SQL-доступу наведено на рисунку 5.6. В ньому виконується SQL-запит для визначення середньої зарплатні співробітників у компанії. Використання методу здійснюється аналогічно до інших методів, визначених у інтерфейсі `Persistent`.

Завдяки використанню даного прикладного програмного забезпечення прикладний програміст мовою JavaScript може використовувати об'єктний та

реляційний доступ до СУБД Caché у Web-застосунках на платформі node.js.

```

const r = db.Reader;
class Company extends Persistent {

  averageEmployeeSalary() {
    return r(connection =>
      connection.prepareStatementPromise(`
        SELECT AVG(e.Salary) as value
        FROM Sample.Company AS c JOIN Sample.Employee AS e
        ON c.ID = e.Company
        WHERE c.ID = ?`)
      .then(statement => statement.queryPromise([this.ID]))
      .then(result => {
        if (!result || result.length !== 1) {
          throw {
            'message': 'Query returned invalid result'
          }
        }
        var result = result[0];
        result.value = Number(result.value);
        return result;
      }));
  }
}

```

Рисунок 5.6 - Приклад визначення SQL запиту в об'єкті та обробки помилок

5.2.4 Інтеграція з Web-застосунками

Для роботи з Web-застосунками непотрібно ніяких додаткових зусиль. Можна використовувати об'єкти класу Persistent як і звичайні JavaScript об'єкти. Для коректної роботи необхідно викликати статичний метод destroy класу Session при закритті веб-сервера для того, щоб вивільнити пул з'єднань до БД.

Приклад такого зворотного виклику наведений на рисунку 5.7.

```

process.on('exit', function() {
  Session.destroy();
});

```

Рисунок 5.7 - Закриття сесії при виході

5.3 Приклад використання розробленого API

Для тестування програмного забезпечення було написано набір інтеграційних тестів за допомогою тестового фреймворку mocha та бібліотеки chai (для мови JavaScript), що дозволило використовувати підхід до розробки на основі тестів (TTD) [15].

Для апробації отриманого результату було розроблено Web-застосунок, з використанням фреймворку express.js.

Для компіляції під різні системи та публікації продукту до репозиторію було розроблено конвеєр (рисунок 5.8) із застосування хмарної системи неперервного тестування та інтеграції.



Рисунок 5.8 - Конвеєр для збірки програмного забезпечення

Web-застосунок реалізовував доступ до каталогу компаній та їх робітників.

Приклад роботи розробленого Web-застосунку зображено на рисунках 5.9, 5.10, 5.11, 5.12. На них зображено процедуру додавання нового співробітника до демонстраційного застосунку.

#	Name	Revenue	Mission
1	Biogy Partners	322848020	Building shareholder value by delivering interactive optical forecasting middle-ware for the pharmaceutical industry.
2	GlobaSys LLC.	297189977	Enabling individuals and businesses to manage open HTML5 devices and middle-ware for industry and government.
3	HyperMatix Partners	571961884	Experts in open ISO 9003-ready devices and gaming for industry and academia.
4	InterPlex Holdings Inc.	233434464	On-line distributors of cloud-based distributed forecasting marketing services for the Entertainment industry.
5	KwalLateral Group Ltd.	617938737	Providers of sustainable mission-critical data warehouse instruments for the Fortune 50.
6	Kwalmo LLC.	587182861	The industry leader in interactive seven-sigma Internet apps for mobile devices.
7	KwalTron Associates	996766769	Developers of just-in-time secure forecasting media for our long-term clients.
8	OctoWare Associates	922301944	Specializing in the development and manufacturing of synergistic hyper-database services for consumers.
9	PicoDynamics Holdings Inc.	386925738	Resellers of premise-based ISO 9003-ready platforms for instruments for the Entertainment industry.
10	QuantaSoft Associates	777433808	Leaders in compliant distributed devices and productivity tools for the desktop.
11	QuantaTech Media Inc.	65557226	Specializing in the development and manufacturing of open hyper-media for emerging markets.

Рисунок 5.9 - Web-застосунок, розроблений з використанням Caché ODM

Home Companies

SSN
 SSN in format: XXX-XX-XXXX

Name

Title

Salary

Short bio

Location

Date of birth

Рисунок 5.10 - Форма додавання робітника

#	Name	Position	Location	Salary
1	Anderson,Rob M.	Assistant Engineer	Fargo	93410
2	Burroughs,Sophia K.	Laboratory Research Asst.	Newton	19470
3	Eastman,Amanda E.	Global Accounts Rep.	Zanesville	59744
4	Kostiantyn Kovalchuk	Student	Kiev	500
5	Kovalev,Jules A.	Global Research Asst.	Reston	96753
6	Nelson,Stuart A.	Assistant Accounts Rep.	Hialeah	29126
Average salary				49833.83

[Add new employee](#)

Рисунок 5.11 - Результат додавання робітника до компанії

Employee details for 455-22-0000	
Name	Kostiantyn Kovalchuk
Title	Student
Salary	500
Company	HyperMatix Partners
Short bio	Kostiantyn is a student.
Location	Kiev
Age	7

Рисунок 5.12 - Деталі доданого співробітника

Архітектура тестового Web-застосунку із застосування node.js, фреймворку express.js [17] та Vue.js зображена на рисунку 5.13.

Архітектура розробленого тестового Web-застосунку відповідає типовій триланковій архітектурі Web-застосунку [15], де в якості клієнта виступає односторінковий застосунок, що написаний з використанням фреймворка Vue.js, серверу застосунків – node.js, а в якості бази даних – СУБД Caché.

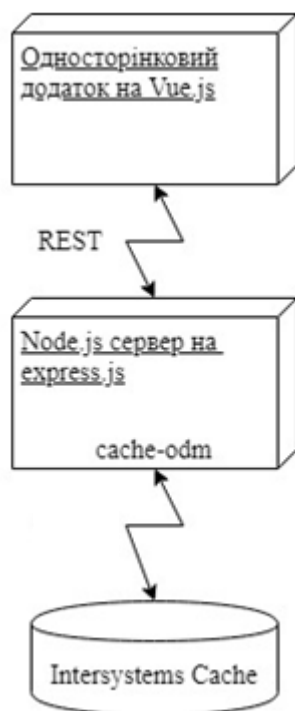


Рисунок 5.13 - Архітектура Web-застосунку з використанням Cache-odm та node.js

6. СТАРТАП ПРОЕКТ

6.1 Ідея проекту

В даному розділі наведено економічне обґрунтування стартап проекту "ODM для об'єктів СУБД Intersystems Caché для мови JavaScript". Метою розділу є опис економічних та технічних характеристик майбутнього стартапу, а також економічних аспектів його реалізації та впровадження [17].

6.2 Маркетинговий аналіз стартап проекту

Основні ідеї для стартап-проекту наведено в таблиці 6.1.

Таблиця 6.1 - Опис ідей стартап проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Реалізувати зв'язок Intersystems Caché з платформи node.js	1. Розробка веб-сайтів	- незалежність від ОС за рахунок використання мови JavaScript; - JavaScript – найпопулярніша мова програмування. Це зменшує поріг входу для розробника; - дозволяє використовувати одну мову програмування для клієнтської та серверної частини
	2. Розробка застосунків для ПК	- незалежність від ОС за рахунок використання мови javascript; - JavaScript – найпопулярніша мова програмування. Це зменшує поріг входу для розробника; - можливість застосування Web-технологій для розробки застосунків та перенесення існуючого функціоналу з Web-версії застосунку

Продовження таблиці 6.1

	3. Розробка застосунків для IoT	- можливість передачі великих масивів (наприклад від датчиків) даних у СУБД; - мінімальні вимоги до апаратних ресурсів
Реалізувати об'єктно-документне відображення для мови JavaScript	1. Розробка веб-сайтів	- високорівневий прикладний програмний інтерфейс, що пришвидшує розробку; - усуває необхідність знання детальної структури СУБД
	2. Розробка застосунків для ПК	- високорівневий прикладний програмний інтерфейс, що пришвидшує розробку; - усуває необхідність знання детальної структури СУБД
Реалізувати підтримку SQL-запитів для мови JavaScript	1. Розробка веб-сайтів	- дозволяє програмісту реалізовувати складний аналіз даних у веб-застосунку - дозволяє повторно використовувати функціонал вже реалізований для реляційних баз даних.
	2. Генерація звітів	- дозволяє користувачу, що не є професійним програмістом (аналітику, менеджеру, бухгалтеру), генерувати звіти мовою SQL

Intersystems Caché – нереляційна СУБД, що є одним з лідерів у сегменті високопродуктивних БД.

Для даної СУБД надається широкий набір бібліотек для доступу до БД на різних мовах програмування, проте для мови JavaScript надається лише обмежений низькорівневий прикладний програмний інтерфейс. Це уповільнює розробку мовою JavaScript та робить використання часу розробника малоефективним.

Особливо критична наявність високорівневого доступу, що дозволить швидко розробку для Web-застосунків, адже даний вид застосунків має швидкі темпи розробки та короткий цикл життя.

Програмування мовою JavaScript на сервері на базі платформи node.js є новим та перспективним напрямом у Web-розробці, проте він, на жаль, не отримав належної уваги з боку розробників бібліотек для СУБД Intersystems Caché.

Для Intersystems Caché існує багато платних бібліотек для різноманітних мов програмування (Java, C++, C#), проте для мови JavaScript існує лише одна, проте вона має наступні недоліки.

1. Низькорівневий прикладний програмний інтерфейс, що вимагає від програміста значних зусиль при розробці та докладного знання структури представлення в БД.

2. Не підтримується останні версії середовища node.js.

3. Закритий вихідний код унеможлиблює внесення удосконалень та виправлення помилок з боку спільноти.

Така ситуація призводить до того, що нові покупці:

1) обирають іншу мову програмування, що у випадку Web-проектів, як правило, призводить до додаткових витрат на підтримку (адже необхідна підтримка двох мов програмування: JavaScript та серверної мови програмування);

2) обирають іншу базу даних, що підтримує кращий функціонал;

3) обирають написання бізнес-логіки мовою ObjectScript, що значно сповільнює розробку та зменшує гнучкість проекту.

Даний програмний продукт покликаний зайняти нову нішу на ринку та орієнтований на нову цільову аудиторію, а також може залучити покупців, які до цього використовували існуючі програмні продукти.

Для оцінки кокурентноздатності та можливостей виходу на ринок, у порівнянні з іншими продуктами було використано наступні характеристики для порівняння (таблиця 6.2):

- доступність з мови JavaScript;
- відкритий програмний код;
- підтримка SQL для СУБД Intersystems Caché;

- кросплатформеність;
- підтримка об'єктно-документного відображення;
- швидкодія.

Таблиця 6.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики	(Потенційні) товари / концепції конкурентів				W(слабка сторона)	N(нейтральна сторона)	S(сильна сторона)
		Мій проект	Інші API	Object Script	Javascript API			
1.	Доступність з мови JavaScript	Так	Ускладнена	Ускладнена	Так			+
2.	Відкритий програмний код	Так	Ні	Ні	Ні			+
3.	Підтримка SQL для СУБД Intersystems Caché	Так	Так	Так	Ні			+
4.	Кросплатформеність	Так	Так	Так	Так		+	
5.	Підтримка об'єктно-документного відображення	Так	Ні	Ні	Ні			+
6.	Швидкодія	Середня	Середня	Висока	Висока	+		

Проект має наступні сильні сторони:

- підтримка об'єктно-документного відображення;
- підтримка мови SQL;
- підтримка об'єктно-документного відображення;
- відкритий вихідний код програми.

Слабкою стороною є порівняно низька швидкодія, що зумовлена наступними чинниками:

- динамічним характером мови JavaScript;
- необхідністю підтримки одразу декількох платформ.

6.3 Технологічний аудит ідеї проекту

Проведемо аудит ідей для реалізації проекту з метою перевірки здійсненності та економічної обґрунтованості ідей проекту (таблиця 6.3).

Таблиця 6.3 - Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
1.	Реалізувати зв'язок Intersystems Caché з платформою node.js	Низькорівнивий Javascript API для Intersystems Caché	Наявні, проте для обмеженого числа версій node.js	Недоступні для розширення через закритий вихідний код
		C++ API	Наявні, проте змінюються від версії до версії СУБД	Доступні, проте потребують суттєвого часу на налагодження крос-платформеності
		ODBC	Наявні, забезпечують незалежність від версії СУБД	Доступні. Є широкий вибір бібліотек, які можна використати
2.	Реалізувати об'єктно-документне відображення для мови JavaScript	JavaScript (без використання бібліотек)	Наявні	Доступні. Дозволяють реалізувати знайомий розробникам Intersystems Caché інтерфейс

Продовження таблиці 6.3

		Sequelize ORM	Наявні, проте можуть бути не знайомі розробнику для Intesystems Caché	Доступні, проте потребують від прикладного програміста додаткових зусиль на вивчення
3.	Реалізувати підтримку SQL-запитів для мови JavaScript	ODBC	Необхідно дороблювати прив'язки для мови JavaScript	Доступні. Є широкий вибір бібліотек, які можна використати

Для реалізації проекту на початковому етапі доцільно застосувати технологію ODBC, адже вона задовольняє вимогам кросплатформеності, незалежності від версії СУБД, дозволяє реалізовувати SQL-доступ та емулювати об'єктний (за допомогою об'єктно-реляційного відображення).

Це дозволить вийти на ринок в якнайкоротший термін та визначити зацікавленість цільової аудиторії у продукті.

6.4 Аналіз ринкових можливостей

З метою отримання попередньої характеристики ринку та визначення обмежень для входу та потенційних конкурентів. Результати аналізу з попередніми даними про динаміку ринку, а також специфічні вимоги до товару наведені у таблиці 6.4.

Таблиця 6.4 - Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних гравців, од.	2
2.	Загальний обсяг продаж, грн. / ум. од.	28000
3.	Динаміка ринку	Зростає
4.	Наявність обмеження для входу	Відсутні. Ринкова ніша майже незайнята конкурентами

Продовження таблиці 6.4

5.	Специфічні вимоги до стандартизації та сертифікації	Мінімальні. Лише ліцензування продукту (за необхідності)
6.	Середня норма рентабельності в галузі, %	50%

У порівнянні з банківським відсотком на вкладення (бл. 20%), проект має набагато вищу рентабельність. Ринок є привабливим для вкладення. Враховуючи високу динаміку ринку, подальші вкладення залежатимуть від динаміки споживачів.

Проведемо характеристику потенційних клієнтів та їх потреб у таблиці 6.5.

Таблиця 6.5 - Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1.	Доступ до СУБД Intersystems Caché з мови JavaScript	Прикладні програмісти, компанії з розробки ПЗ	Індивідуальні розробники, як правило, мають менший бюджет на розробку ПЗ, що потребує забезпечення для даної категорії покупців особливої цінової політики.	Вимоги до продукції: - зручність у використанні прикладного програмного інтерфейсу; - прийнятна швидкодія; - можливість безкоштовного використання (для індивідуальних розробників); - наявність технічної підтримки програмного продукту

Продовження таблиці 6.5

2.	Підтримка SQL запитів для СУБД Intersystems Caché	Прикладні програмісти, аналітики, компанії з розробки ПЗ	Прикладний програмний інтерфейс має бути простим та інтуїтивно зрозумілим для того, щоб аналітики змогли використовувати його для генерації запитів	Вимоги до продукції - наявність документації; - підтримка спрощеного API для генерації запитів; - наявність технічної підтримки
3	Об'єктно-документне відображення для СУБД Intersystems Caché	Прикладні програмісти, компанії з розробки ПЗ, розробники для СУБД Intersystems Caché	Прикладний програмний інтерфейс має бути знайомим для розробників мовою ObjectScript, щоб вони були зацікавлені у користуванні програмним пакетом	Вимоги до продукції: - подібність у використанні до API мови ObjectScript; - можливість відключення даної функціональності для розробників, яким не потрібен даний функціонал

З характеристики робимо висновок, що основною цільовою аудиторією є розробники мовою JavaScript та непрофесійні розробники, що використовують JavaScript для візуалізації даних або звітів. Розробники іншими мовами є меншою за чисельністю цільовою аудиторією.

Проаналізуємо загрози для виходу на ринок у таблиці 6.6.

Таблиця 6.6 - Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1.	Низька обізнаність покупців	Користувачі не будуть використовувати продукт через низьку його відомість або страх змін	- забезпечити доступну документацію; - забезпечити технічну підтримку та механізм зворотного зв'язку

Продовження таблиці 6.6

2.	Відповідь конкурентів	Користувачі не будуть використовувати програмний продукт через реалізацію відповідного функціоналу у конкурентів	- постійно моніторити дії конкурентних продуктів; - постійна інтеграція нового функціоналу на основі відгуків користувачів
----	-----------------------	--	---

Проаналізуємо фактори можливостей у таблиці 6.7.

За результатами аналізу факторів ризику та можливостей, можна зробити висновок, що в даному випадку можливо прийняти наявні ризики задля зайняття нової ринкової ніші та отримання прибутку.

Таблиця 6.7 - Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1.	Використання сучасних практик програмування	Можливість побудови програмного продукту, що буде кращим за якістю через використання розробки	- будувати розробку ПЗ на основі вимог користувачів, виражаючи їх у вигляді тестів
2.	Залучення незалежних розробників із спільноти розробки	Можливість отримувати допомогу та зворотній зв'язок від спільноти програмістів	- викладати вихідні коди програми у відкритий доступ
3.	Зайняти нову нішу на ринку	Розробка мовою JavaScript із застосуванням СУБД Intersystems Caché лише набуває популярності	- забезпечити якнайшвидший вихід продукції на ринок за рахунок сучасних відкритих засобів неперервної інтеграції

Проведемо ступеневий аналіз конкуренції та визначимо основну тактику дій відповідно до особливостей конкурентного середовища (таблиця 6.8).

Таблиця 6.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентноспроможною)
Тип конкуренції – чиста	Ринок програмного забезпечення – динамічне ринкове середовище, де на вибір покупців впливає виключно співвідношення ціна / якість. Користувачі легко змінюють програмний продукт	<ul style="list-style-type: none"> - постійне покращення програмного продукту - забезпечення непевної доставки оновлень для користувачів - постійна технічна підтримка
Тип конкурентної боротьби – локальний	Боротьба за користувача відбувається у вузькому сегменті ринку користувачів даної СУБД	- формувати спільноти користувачів, забезпечити можливість обміну досвідом між користувачами
Конкуренція за видами товарів – видова	Конкуренти надають доступ до СУБД іншими мовами програмування, що орієнтовані на різні кола розробників	- популяризувати розробку за допомогою даного програмного пакету як найбільш продуктивну
За характером конкурентних переваг – нецінова	Користувачі в першу чергу орієнтуються на функціонал та продуктивність розробки, а не на ціну	<ul style="list-style-type: none"> - постійне вдосконалення проекту і моніторинг функціоналу конкурентів; - забезпечити непевну доставку оновлень для клієнтів
Інтенсивність – не марочна	Марка не має значення для кінцевого користувача – прикладного програміста. Основними критеріями покупців є надійність, швидкодія, наявність документації та підтримки	<ul style="list-style-type: none"> - орієнтуватися на функціонал, а не бренд; - забезпечити ліберальне ліцензування програмного продукту

За результатами аналізу бачимо, що для того, щоб бути конкурентноспроможними, необхідно орієнтуватися на функціонал та користувача.

Проаналізуємо детальніше конкуренцію за М. Портером в таблиці 6.9.

Таблиця 6.9 - Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Intersystems	Нові гравці на ринку	Intersystems	Прикладні програмісти, аналітики, компанії з розробки ПЗ	Програмні пакети Intersystems, продукти інших розробників
Висновки	Конкурентна боротьба з боку прямих конкурентів інтенсивна – кожен прагне до монополізації ринку	- бар'єр для входження на ринок мінімальний; - потенційні конкурентом може стати компанія Intersystems, що прагне до монополізації ринку доступу до БД	Постачальник є одночасно розробником СУБД, тому він диктує умови роботи на ринку	Клієнти в повній диктують умови роботи на ринку. Якщо програмний пакет не задовольняє вимогам користувачів, вони переходять на інший	Обмеження є орієнтованість програмного пакету на одну платформу для розробки. Це обмежує його використання користувачами інших мов

Обґрунтуємо на основі конкуренції та вимог користувачів до товару фактори конкурентоспроможності у таблиці 6.10.

Таблиця 6.10 - Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1.	Зручність розробки платформі node.js, наявність високорівневого	Дана ринкова ніша не зайнята. Це забезпечить відсутність конкуренції на початковому етапі розвитку продукції.

	прикладного програмного інтерфейсу	
2.	Відкритий вихідний код та прямий зв'язок з користувачами	Відкритий вихідний код та прямий зв'язок дозволяє постійно покращувати якість продукту та забезпечувати його конкурентоспроможність
3.	Кросплатформеність	Кросплатформеність дуже важлива для розробки сучасних Web-застосунків, адже вони часто розгортаються на різних апаратних платформах (наприклад, у хмарі)

На основі отриманих результатів, проведемо порівняльний аналіз сильних та слабких сторін проекту у таблиці 6.11.

Таблиця 6.11 - Порівняльний аналіз сильних та сторін проетку

№	Фактор конкурентноспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з ФОП "Ковальчук"							
			-3	-2	-1	0	+1	+2	+3	
1	Доступність мови JavaScript	20		+						
2	Відкритий програмний код	19	+							
3	Підтримка SQL для СУБД Intersystems Caché	19			+					
4	Кросплатформеність	20		+						
5	Підтримка об'єктно-документного відображення	17	+							
6	Швидкодія	13								+

За результатами порівняльного аналізу можна зробити висновок, що найбільшими перевагами стартап-проекту є його доступність та зручність у використанні. Недоліком є порівняно низька швидкодія, проте його можна нейтралізувати, оптимізуючи код за необхідності.

За результатами порівняння, можна та аналізу вимог користувачів, можна зробити наступні висновки (у формі SWOT) у таблиці 6.12.

Таблиця 6.12 - SWOT-аналіз стартап-проекту

Сильні сторони (Strength): перший проект на ринку в своєму роді; орієнтованість на користувача-розробника; дозволяє використовувати розробку під сучасні програмні платформи	Слабкі сторони (Weaknesses): орієнтованість на одну платформу та мову програмування; існує початковий бар'єр входження на ринок;
Можливості (Opportunities): зайняти ринок першими; заключати дострокові договори про підтримку;	Загрози (Threats): відповідь з боку конкурентів-розробників пакетів для інших мов програмування консервативність розробників (небажання змінювати технології)

Згідно з результатами SWOT-аналізу розроблено наступні альтернативи просування товару на ринках. Результати узагальнено в таблиці 6.13.

Таблиця 6.13 - Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтований комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Реалізувати продукт за допомогою низькорівневих програмних інтерфейсів	50%	12 місяців
2.	Реалізувати продукт за допомогою ODBC API	80%	8 місяців

Відповідно до таблиці 6.13, більш привабливим з економічної точки зору є використання доступу за допомогою ODBC API, адже це забезпечить менші терміни реалізації та більшу ймовірність отримання ресурсів.

6.5 Розробка ринкової стратегії проекту

Визначимо цільові групи потенційних користувачів серед прикладних розробників мовами COS та JavaScript у таблиці 6.14.

За результатами аналізу було обрано розробників мовою JavaScript, а також непрофесійних програмістів (аналітиків та адміністраторів), адже цим нішам мало приділена увага на ринку. Також прийняте рішення частково задовольнити потреби і розробників на ObjectScript, надавши їм знайомий API.

Таблиця 6.14 - Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи клієнтів	Готовність споживачів сприйняти продукт	Орієнтований попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Прикладні розробники мовою JavaScript	Висока, адже відповідних альтернатив мало	Високий	Низька	Сегмент ринку на даний момент є вільним
2	Прикладні розробники мовою ObjectScript	Середня, розробники, скоріше за все, прив'язані до функціоналу старих проектів на ObjectScript	Низький	Висока	Сегмент ринку є монополізованим
3	Прикладні розробники іншими мовами програмування	Середня, розробники можуть роботи вибір лише для нових проектів	Середній	Висока	Сегмент ринку монополізований, наявна залежність від одного постачальника

Продовження таблиці 6.14

4	Аналітики, непрофесійні програмісти, адміністратори	Висока, JavaScript – проста та популярна скрипкова мова	Високий	Низька	Сегмент ринку є вільним
---	---	---	---------	--------	-------------------------

Обрано наступну цільову групу:

- прикладні розробники мовою JavaScript;
- аналітики, непрофесійні програмісти.

Визначимо на основі проведеного вище аналізу, визначимо базові стратегії розвитку у таблиці 6.15.

На початковому етапі впровадження слід вибрати стратегію орієнтації на JavaScript розробників та непрофесійних програмістів, а згодом надати програмні пакети і для інших категорій користувачів.

Таблиця 6.15 - Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Розвиток програмної платформи для зручності у користуванні для розробників мовою node.js та непрофесійних програмістів	Ексклюзивний розподіл	Підтримка платформи node.js, використання API бібліотек, знайомих javascript-розробникам	Стратегія спеціалізації
2	Надання кількох версій програмного пакету для різних розробників	Ексклюзивний розподіл	Підтримка декількох API для різних категорій розробників	Стратегія диференціації

Визначимо базову стратегію конкурентної поведінки у таблиці 6.16 за результатами вибору базової стратегії розвитку.

Таблиця 6.16 - Визначення стратегії конкурентної поведінки

Чи є проект "першопрохідцем" на ринку?	Чи буде компанія шукати нових споживачів або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Частково. Продукт є першим на ринку для мови JavaScript	Компанія як буде залучати нових споживачів, так і залучати	Так, прикладний програмний інтерфейс	Стратегія зайняття конкурентної ніші

Визначимо стратегію позиціонування продукту на основі обраної стратегії поведінки та потреб користувачів та наведемо результат у таблиці 6.17.

Продукт буде позиціонуватися як зручний та простий у налаштуванні продукт для кінцевого користувача, що інтегрується з node.js "безшовно".

Таблиця 6.17 - Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Простота використання	Стратегія диференціації	зручність API, підтримка мови SQL, надійність	- простота; - елегантність; - доступність; - надійність
2	Підтримка node.js	Стратегія диференціації	інтеграція node.js	- простота налаштування; - знайомий API

6.6 Розробка маркетингової програми

Визначимо ключові переваги концепції потенційного товару. Підсумуємо результати у таблиці 6.18.

Таблиця 6.18 - Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Програмний інтерфейс взаємодії з СУБД для мови JavaScript	Доступ до СУБД на високорівневій мові програмування забезпечує швидку розробку якісного продукту та пришвидшує його вихід на ринок	Надання високо рівневого, зручного в користуванні прикладного програмного інтерфесу.
2	Можливості об'єктно-документного відображення	Пришвидшення розробки, зменшення порогу входу у розробку	Надання функціоналу, що за замовчуванням доступний лише в ObjectScript
3	Можливість генерації звітів	Можливість використання продукту непрограмістами	Простота для використання програмістами-непрофесіоналами

Опишемо три рівні моделі товару у таблиці 6.19.

Таблиця 6.19 - Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
i. Товар за задумом	Надання зручного та швидкого доступу до СУБД Intersystems Caché мовою javascript
ii. Товар у реальному виконанні	Пакет двікових та мініфікованих вихідних файлів, завірених цифровим підписом. Розповсюдження здійснюється через Інтернет
iii. Товар із підкріпленням	До продажу: відсутня технічна підтримка
	Після продажу: надається технічна підтримка за номером ліцензії або підпискою

Продовження таблиці 6.19

Товар не буде захищено від копіювання. Можна буде вільно скористатися його копією у будь-яких цілях. Проте у випадку використання непідписаної версії програми, технічна підтримка та гарантії втрачаються.

Визначимо межі встановлення цін у таблиці 6.20 з урахуванням цін на товари-аналоги.

Таблиця 6.20 - Визначення меж встановлення ціни

№	Рівень цін на товари замітники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на послугу
1	29000	30000	260000	25000-40000

Визначимо оптимальну систему збуту в таблиці 6.21.

Таблиця 6.21 - Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система
1	Придбання річної підписки на технічну підтримку	Продаж	0 (наряму) 1 (через посередника)	0 (напряму)
2	Додаткова реалізація нового функціоналу	Продаж Постановка ТЗ	0 (напряму)	0 (напряму)

Продаж підписки на технічну та додаткового функціоналу здійснюватимемо напряму через мережу Інтернет. Посередники непотрібні, вигідніше здійснювати напряму. Це також сприятиме налагодженню бізнес-зв'язків та залученню незалежних розробників до проекту.

Визначимо основні концепції маркетингових комунікацій із використанням специфіки поведінки клієнтів, наведеної в у таблиці 6.21. Результати підсумуємо у таблиці 6.22.

Маркетингову комунікацію доцільно здійснювати через мережу інтернет при цьому орієнтуючись на пряму комунікацію між клієнтами. Також варто організувати рекламу в тематичних виданнях та виступи на профільних конференція з метою збільшення обізнаності цільової аудиторії про продукт.

Таблиця 6.22 - Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Незалежні розробники прагнуть мінімізувати свої витрати та обирати мінімальний пакет підтримки	Інтернет, тематичні конференції, професійна періодика	Донести можливість пробного безкоштовного використання	Новий зручний спосіб доступу до СУБД
2	Компанії з розробки ПО прагнуть отримувати гарантії та технічну підтримку	Інтернет, тематичні конференції, зв'язок з бізнес партнерами	Донести можливість надання технічної підтримки та гарантій роботи	Надійний та зручний спосіб доступу до СУБД, що зменшує витрати на підтримку та штат програмістів

ВИСНОВКИ

В ході даної магістерської дисертації було досліджено основні підходи до організації доступу до баз даних загалом та до СУБД Intersystems Caché зокрема.

Було проведено огляд предметної області магістерської дисертації, а розглянуто основні принципи побудови прикладних програмних інтерфейсів для доступу до БД, що були використані в подальшому.

Відповідно до наявних засобів доступу та аналізу їх переваг, недоліків, а також техніко-економічних характеристик було обрано інструменти для реалізації програмного продукту.

В ході реалізації було створено програмний продукт, що призначений для доступу до СУБД Intersystems Caché. При реалізації враховувалась специфіка мови JavaScript та виконувались неблокуючі (по відношенню до основного потоку виконання) виклики за рахунок використання пулу потоків libuv.

Продукт розділено на два основних компоненти, а саме модуль для доступу до даних (Cacheodbc) та модуль для об'єктно-реляційного відображення, що емулює об'єктний доступ (Cache-ODM).

Програмний продукт розроблено з використаннями TTD. Також було налаштовано конвеєр для збірки продукту під різні операційні системи. Дані методології дозволили виявити недоліки реалізації на ранній стадії та визначити подальші шляхи покращення продукту.

В ході аналізу стартап-проекту було визначено, що проект є можливим для комерційної реалізації та рентабельним для роботи на ринку. На програмні продукти високої якості, що зменшують кошти та швидкість розробки програмних продуктів є постійний попит на ринку.

Програмний продукт має усі перспективи бути впровадженим для широкого загалу користувачів, адже ринок має низький бар'єр для входження через відсутність

конкуренції в своєму ринковому сегменті та легко доступний для користувача через мережу Інтернет.

Для початкової реалізації та входження на ринок доцільно обрати стратегію спеціалізації на прикладних розробниках на платформі node.js.

Подальший розвиток продукту визначатиметься динамікою ринку. У разі успіху розроблятимуться подальші вдосконалення до продукту, а також розроблятиметься нова маркетингова стратегія для залучення нових клієнтів.

Також було проведено тестування та випробування продукту для створення Web-застосунку мовою JavaScript із застосуванням сучасних JavaScript бібліотек (express.js). Результати випробувань підтвердили можливість використання даного програмного продукту Web-розробниками.

Перевагою розробленого рішення у порівнянні з аналогами є:

1. Забезпечення SQL-доступу.
2. Емуляція об'єктного доступу за допомогою об'єктно-реляційного доступу за допомогою об'єктно-реляційного відображення за шаблоном Active Record.
3. Можливість доступу з платформи node.js без написання коду мовою ObjectScript.
4. Можливість використання існуючих інструментів JavaScript для реалізації об'єктно реляційного відображення.
5. Підтримка останніх версій node.js (8, 10).

При тестуванні програмного забезпечення було виявлено наступні недоліки:

1. Недостатня підтримка Unicode на Linux та MacOS.
2. Некоректна конвертація дат на ОС Windows (необхідна попереднє приведення до рядка).
3. Відсутність підтримки підготовлених запитів для DML-запитів.

Дані недоліки було виявлено при проведенні автоматизованого тестування та планується виправити в наступних версіях програми. Вони накладають певні

обмеження на роботу з програмним продуктом, проте не є перешкодою для написання коректних програм.

Подальший розвиток продукту полягатиме у підвищенні його стабільності (виправленні наявних неточностей у роботі), заміні емуляції об'єктного доступу на прямий доступ до об'єктів для оптимізації швидкодії, а також покращенню підтримки платформ для розробки настільних застосунків таких як Electron.

Програмне забезпечення було реалізоване в рамках грантової програми компанії Intersystems та може використовуватися для розробки додаків, що працюють з СУБД Intersystems Caché мовою JavaScript.

Список використаних джерел

1. Rob Tweed, George James. A Universal NoSQL Engine, Using a Tried and Tested Technology // <http://www.mgateway.com>. 2010. URL: <http://www.mgateway.com/docs/universalNoSQL.pdf> (дата звернення: 09.09.2018).
2. William R. Cook, Ali H. Ibrahim. Integrating Programming Languages & Databases: What's the Problem? // Operational Database Management Systems. 2005. URL: http://www.odbms.org/wp-content/uploads/2006/10/010.03-Cook-Integrating-Programming-Languages-and-Databases_What-is-the-Problem-September-2006.pdf
3. Using Node.js with Caché 2018. URL: <https://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=BXJS>
4. Sanders R.E. ODBC 3.5 Developers Guide. New York: McGraw-Hill, 1998.
5. Maydene Fisher, Jon Ellis, Jonathan Bruce. JDBC API Tutorial and Reference, Third Edition. Addison Wesley, 2003.
6. Caché Overview [Електроний ресурс] // Intersystems: [сайт]. [2018]. URL: <http://www.intersystems.com/ru/our-products/cache/cache-overview/> (дата звернення: 22.10.2018).
7. InterSystems Caché Technology Guide URL: <https://www.intersystems.com/wp-content/uploads/sites/8/CacheTechnologyGuide.pdf>
8. Иванчева Н. А., Иваньчева Т. А. Постреляционная СУБД Caché (методическое пособие). Новосибирск: Новосибирский государственный университет, 2004.
9. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений. Москва: Вильямс, 2010. 720 pp.
10. Сиротюк О. Отличительные особенности СУБД Caché // CIT Forum. 2002. URL: <http://citforum.ck.ua/database/articles/subdcache.shtml> (дата звернення: 22.10.2018).
11. Iyer GR, "Node.js: Event-driven Concurrency for Web Applications," 2013.

12. Sandro Pasquali, Kevin Faaborg. *Mastering Node.js*. Birminham: Packt, 2016.
13. Ковальчук К.О. Підсумки розвитку наукової думки: 2018 // Організація доступу до мультипарадигмної СУБД Intersystems Caché на платформі node.js. Івано-Франківськ. 2018. Vol. 4.
14. Parker D. *JavaScript with Promises*. Boston: O'Reilly, 2015.
15. Fowler M. *Patterns of enterprise application architecture*. Addison-Wesley, 2011.
16. Эрих Гамма Р.Д.Р.Х.Д.В. Приемы объектно-ориентированного проектирования. Паттерны проектирования. Санкт-Петербург: Питер, 2012. 366 pp.
17. Stroustrup B. *The C++ programming language*. Ann Arbor: Pearson Education, 2013.
18. Beck K. *Test Driven Development*. Pearson Education, 2004.
19. Браун И. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. Санкт-Петербург: Питер, 2017.
20. П.В. Бураков, В.Ю. Петров. Введение в системы баз данных. Санкт-Петербург: ИТМО, 2010.
21. В. Г.О. Розроблення стартап-проекту. Київ: НТУУ «КПІ ім. Ігоря Сікорського», 2016.

ДОДАТОК А

Програмне забезпечення для реалізації ODM для об'єктів СУБД
Intersystems Caché для мови JavaScript

Акт впровадження

УКР.НТУУ "КПІ". ТВ7128_18

Аркушів 2

2018

" 03 " грудня 2018 р.

м. Київ

АКТ ВПРОВАДЖЕННЯ

результатів дипломної роботи Ковальчука К.О. за темою "Програмне забезпечення для реалізації ODM для мови javascript для об'єктів СУБД Intersystems Caché", що була виконана в Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського»:

Нами, представником кафедри автоматизації проектування енергетичних процесів і систем НТУУ "КПІ" та директором ТОВ "НВЦ "Плазер", складено даний акт про те, що при розробці спеціалізованого програмного забезпечення в ТОВ "НВЦ "Плазер" використані результати магістерської дипломної роботи Ковальчука К.О., а саме програмне забезпечення для реалізації ODM для мови javascript для об'єктів СУБД Intersystems Caché, а також документація програмного супроводу.

Представник кафедри
АПЕПС НТУУ "КПІ",
керівник дипломної роботи

Директор ТОВ "НВЦ "Плазер"

І. Ю. Михайлова

М.Ф. Короб



ДОДАТОК Б

Програмне забезпечення для реалізації ODM для об'єктів СУБД
Intersystems Caché для мови JavaScript

Апробація

УКР.НТУУ "КПІ". ТВ7128_18

Аркушів 7

2018

ГРОМАДСЬКА ОРГАНІЗАЦІЯ
«ЄВРОПЕЙСЬКА НАУКОВА ПЛАТФОРМА»
ОО «ЕВРОПЕЙСКАЯ НАУЧНАЯ ПЛАТФОРМА» • NGO «EUROPEAN SCIENTIFIC PLATFORM»

ЛОГОΣ

ЗБІРНИК НАУКОВИХ ПРАЦЬ

ЗА МАТЕРІАЛАМИ МІЖНАРОДНОЇ
НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ

**«ПІДСУМКИ РОЗВИТКУ
НАУКОВОЇ ДУМКИ: 2018»**

5 ГРУДНЯ 2018 РІК

ТОМ 4

Івано-Франківськ • Україна

УДК 001(08)
П 62

П 62 **Підсумки розвитку наукової думки: 2018:** зб. наук. праць «ΛΟΓΟΣ» з матеріалами міжнар. наук.-практ. конф., м. Івано-Франківськ, 5 грудня, 2018 р. Вінниця : ГО «Європейська наукова платформа», 2018. Т.4. с. 128.

ISBN 978-617-7171-80-4

Викладено тези доповідей та статті учасників міжнародної науково-практичної конференції «Підсумки розвитку наукової думки: 2018», яка відбулася у місті Івано-Франківськ, 5 грудня 2018 року.

Збірник присвячено для студентів, аспірантів, докторантів, здобувачів, молодих фахівців, викладачів, науковців та інших зацікавлених осіб, а також для широкого кола читачів.

Бібліографічний опис матеріалів конференції зареєстровано в міжнародній наукометричній базі «Google Scholar».



УДК 001 (08)

ISBN 978-617-7171-80-4

© Колектив авторів конференції, 2018
© Збірник наукових праць «ΛΟΓΟΣ», 2018
© ГО «Європейська наукова платформа», 2018

МЕТОДИ ЗМЕНШЕННЯ ОКСИДІВ АЗОТУ Задорожня А.О.	98
МОБІЛЬНИЙ ДОДАТОК ДЛЯ ПІДВИЩЕННЯ РОЗУМОВОЇ ДІЯЛЬНОСТІ ШЛЯХОМ ФІЗИЧНИХ НАВАНТАЖЕНЬ Зубарчук О.Д.	99
ОРГАНІЗАЦІЯ ДОСТУПУ ДО МУЛЬТИПАРАДИГМЕННОЇ СУБД INTERSYSTEMS CASHE НА ПЛАТФОРМІ NODE.JS Ковальчук К.О.	103
ПЕРСПЕКТИВИ ВИКОРИСТАННЯ СИСТЕМ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ В ТУРИСТИЧНІЙ ГАЛУЗІ Науково-дослідна група: Штокало В.Я., Штокало Л.Я., Слободян Р.О.	105
ПРИМЕНЕНИЕ УГЛЕПЛАСТИКОВ В КАЧЕСТВЕ КОНСТРУКЦИОННЫХ МАТЕРИАЛОВ ПРИ ИЗГОТОВЛЕНИИ ОПТИКО-МЕХАНИЧЕСКОГО БЛОКА КА ДЗЗ Легенкова Л.Д.	109
ПРОБЛЕМАТИКА ДОСЛІДЖЕНЬ ХАОТИЧНИХ РУХІВ У СУЦІЛЬНИХ СЕРЕДОВИЩАХ Кошлатий М.Л.	111
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МОДЕЛЮВАННЯ РОЗСІЮВАННЯ АНТРОПОГЕННОГО ЗАБРУДНИКА В АТМОСФЕРІ Завалко У.В.	113
СИСТЕМА ПРОСТЕЖУВАНOSTІ ДЛЯ ВИРОБНИЦТВА ЖИТНЬО-ПШЕНИЧНОГО ХЛІБА Петриченко В.І.	120
ФІЛЬТРУВАЛЬНИЙ МАТЕРІАЛ ДЛЯ БАРОМЕМБРАННОГО ОЧИЩЕННЯ ВОДИ Вовк Д.М., Кріт М.А.	122
ХАРЧОВІ ДОБАВКИ ДЛЯ ВИРОБІВ СПЕЦІАЛЬНОГО ПРИЗНАЧЕННЯ Зразюк М.Д.	124

3. Воробьева Е.В. Психогенетическое исследование взаимосвязи темперамента и мотивации достижения. *Известия высших учебных заведений. Северо-Кавказский регион. Общественные науки.* 2006. № 3. С. 110-111.
4. Дейтел П., Дейтел Х., Уолд А. *Android для разработчиков.* 3-е видання. 2016.

ОРГАНІЗАЦІЯ ДОСТУПУ ДО МУЛЬТИПАРАДИГМЕННОЇ СУБД INTERSYSTEMS CACHE НА ПЛАТФОРМІ NODE.JS

Ковальчук Костянтин Олександрович

Науковий керівник: канд. т. наук, доцент Михайлова І.Ю.
*Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»,
Україна*

Сучасне програмне забезпечення часто потребує доступу до баз даних. Операції з даними повинні проводитися в безпечний спосіб, без пошкодження чи втрати даних. В той же час, дані повинні бути зручними для користування в програмному забезпеченні.

На сьогодні доступ до баз даних є як ніколи актуальним, адже як об'єм, так і різноманіття даних, що зберігається постійно зростає.

Використання реляційних баз даних часто призводить до проблеми об'єктно-реляційного розриву, адже реляційна модель та об'єктно-орієнтоване програмування, що найчастіше використовується, не є сумісними. Реляційна модель призначена для зберігання та маніпуляції великими масивами інформації, в той час як інша призначена для призначена для обробки даних.

Для вирішення цієї проблеми часто застосовуються бібліотеки, що здійснюють об'єктно-реляційне відображення (ORM). При цьому часто погіршується ефективність запитів до бази даних, адже програміст практично не має доступу до генерації запитів.

Особливістю СУБД InterSystems Caché є мультипарадигмений доступ на основі є використання єдиної моделі даних, до яких наявний як реляційний так, і об'єктний доступ до інформації, що зберігається [1].

Для об'єктного доступу до СУБД InterSystems Caché із платформи node.js вендор надає прикладний програмний інтерфейс. Організація ж реляційного доступу вимагає від програміста додаткових зусиль для зв'язку з наявними інтерфейсами (наприклад, ODBC для мови С).

Основною проблемою, що виникає при використанні API сторонніх мов є організація виклику, що не блокує цикл подій node.js [2]. Для цього можна використовувати пул потоків, що надається бібліотекою libuv. Організація такого виклику показана на рисунку 1 [3].

Використання комбінованого (реляційного та об'єктного одночасно) підходу до доступу дозволяє уникнути недоліків об'єктно-реляційного відображення (велика кількість запитів), в і в той же час не втрачати можливість агрегації даних за допомогою мови SQL.

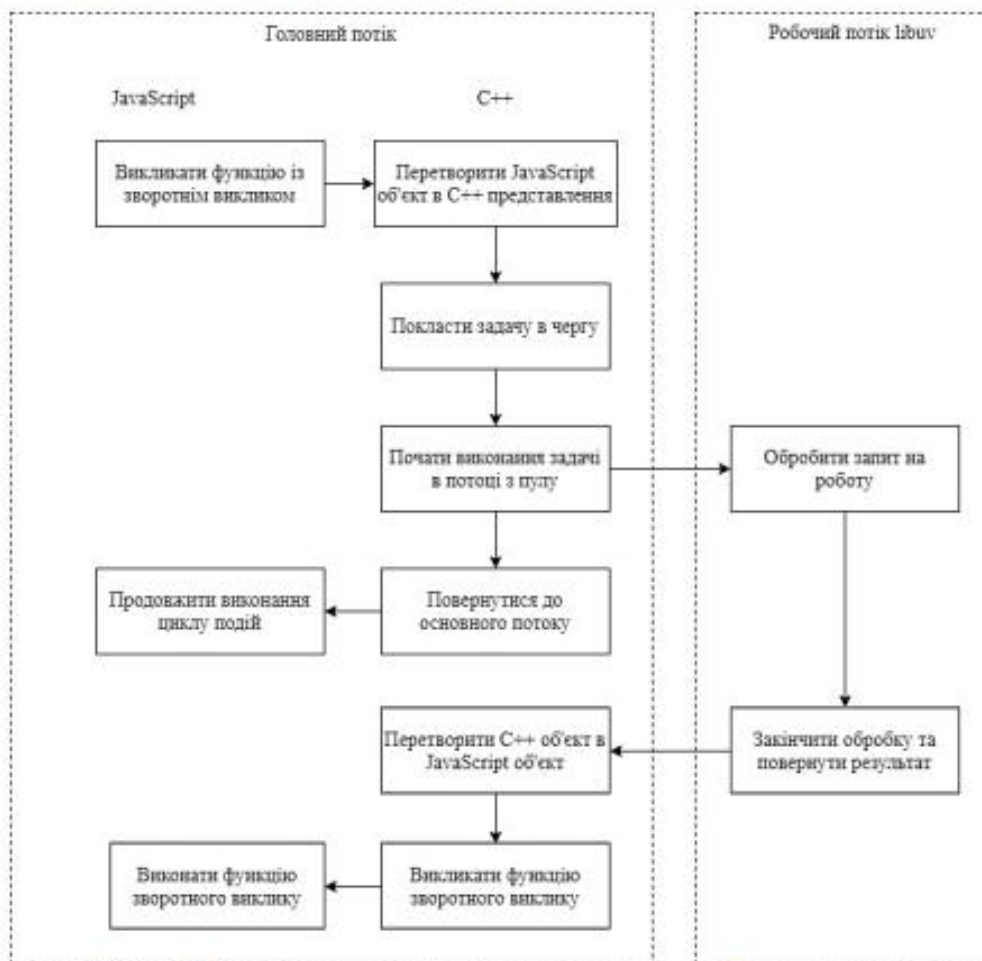


Рис. 1. Організація асинхронного виклику до БД на платформі node.js

Використання високорівневої мови JavaScript дозволяє надати прикладному програмісту зручний інтерфейс для доступу до даних, що потребує мінімальних зусиль від програміста. При цьому доцільно використовувати вже існуючий API мови ObjectScript як прототип. Приклад встановлення такої відповідності наведений в таблиці 1.

Таблиця. 1

Емуляція API мови ObjectScript мовою JavaScript

Метод об'єкта мови JavaScript	Метод об'єкта мови ObjectScript
let p = new Person()	SET p = Person.%New()
p.save()	DO p.%Save()
p.update()	DO p.%Save()

5 грудня 2018 рік • Івано-Франківськ, Україна • 105

<code>let p = Person.openId(1)</code>	<code>SET p = ##class(S.Person).%OpenId(1)</code>
<code>let f = Person.existsId(1)</code>	<code>SET p = ##class(S.Person).%ExistsId(1)</code>

Список використаних джерел:

1. Кирстен В., Ирингер М., Кюн М., Рериг Б. Постреляционная СУБД Cache 5. Объектно-ориентированная разработка приложений. Санкт-Петербург : Питер, 2001. 416 с.
2. Хэррон Д. Node.js Разработка серверных веб-приложений на JavaScript. Москва : ДМК Пресс, 2014. 144 с.
3. Kasiuk A. On problems with threads in node. Future-processing.pl. URL: <https://www.future-processing.pl/blog/on-problems-with-threads-in-node-js/>.

ПЕРСПЕКТИВИ ВИКОРИСТАННЯ СИСТЕМ АВТОМАТИЗАЦІЇ УПРАВЛІННЯ В ТУРИСТИЧНІЙ ГАЛУЗІ

Науково-дослідна група:
викладач 1-ї категорії Штокало Володимир Ярославович,
викладач 1-ї категорії Штокало Леся Ярославівна,
викладач 2-ї категорії Слободян Руслан Олесьйович
*Технічний коледж Тернопільського національного
технічного університету ім. Івана Пулюя
Україна*

Постановка проблеми. Сьогодні в світовому співтоваристві відбуваються глобальні зміни, викликані проникненням у всі сфери життя інформаційних технологій. Не залишився осторонь від цієї тенденції і туризм як сфера економіки, діяльності та зайнятості людей. Інформаційні технології, проникаючи в бізнес, змінюють технології управління, допомагають отримувати відповіді на будь-які питання про стан справ і приймати оперативні рішення в лічені секунди [1].

Аналіз досліджень та публікацій. Питання автоматизації діяльності як туристичного, так і підприємства в цілому було висвітлено в працях багатьох зарубіжних і вітчизняних авторів: Хамера, Чампі, Джонсона, Брандона та інших. В українській науці проблемою автоматизації туристичних підприємств займалися С. Мельниченко, А. Татаринцева, О. Олійник. Разом із тим, надалі актуальними залишаються питання сучасного стану інформаційних технологій та систем автоматизації управління в індустрії туризму з метою підвищення її конкурентоспроможності [2].

Метою статті є огляд основних переваг практичного використання систем автоматизації управління в туристичній сфері.

Виклад основного матеріалу. Автоматизація та широке застосування електронної техніки стають одним з найактуальніших завдань в галузі туризму. Створення потужних комп'ютерних систем бронювання засобів і транспорту, екскурсійного й культурно-оздоровчого обслуговування,