Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Розробка Web-застосунку на мові Crystal

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки та інформаційні технології» - 122

Керівник курсової роботи
ст. викладач
Захоженко П.О.

_____

(Підпис)
"___" _____ 2020 року
Виконала студентка КНІТ-4
Юр'єва О.К.
"___" _____ 2020 року

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри мультимедійних систем,
старший викладач _____ Захоженко П.О.
(підпис)
„____”_____2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студентці Юр’євій Ользі Костянтинівні
факультету інформатики 4-го курсу бакалаврської програми
ТЕМА: Розробка Web-застосунку на мові Crystal

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план
Анотація
Вступ
1. Дослідження та аналіз предметної області
2. Інструменти для розробки веб-застосунку
3. Опис веб-застосунку

Висновки
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2020 р. Керівник_____
(підпис)
Завдання отримала ___
(підпис)

Тема: Розробка Web-застосунку на мові Crystal

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	10.10.2019	
2.	Пошук відповідної літератури	19.10.2019	
3.	Ознайомлення з літературою	25.10.2019	
4.	Вивчення предметної області	03.11.2019	
5.	Огляд технологій для побудови веб-застосунку	15.11.2019	
6.	Вивчення технологій	20.11.2019	
7.	Створення бази даних та CRUD операцій	02.02.2020	
8.	Створення сервісів та реалізація логіки	15.02.2020	
9.	Написання першої частини курсової роботи	02.03.2020	
10.	Написання другої частини курсової роботи	13.03.2020	
11.	Написання висновків курсової роботи	02.04.2020	
14.	Створення презентації	18.04.2020	
15.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	18.04.2020	
16.	Захист роботи	19.04.2020	

Студент Юр’єва О.К.

Керівник Захоженко П.О.

“____” _____

Зміст

Анотація

Метою даної роботи є дослідження мови програмування Crystal, її особливостей, переваг та недоліків. Як результат буде створено веб-застосунок, який дозволить оцінити використання даної мови програмування на практиці.

Як предметну область майбутнього веб-застосунку обрано створення каналу зв’язку між студентами та адміністрацією університету. Проведено опис предметної області, а власне процесів взаємодії студентів з університетськими відділами та запропоновано альтернативу у вигляді спільноти університетських зв’язків.

Робота включає аналіз предметної області, огляд використаних технологій з їх характеристикою та опис основних моментів процесу розробки.

Вступ

Програмування можна представити як набір послідовних команд для певного виконавця. Їх виконання забезпечує досягнення певної мети. Перші програми писалися на машинній мові програмування і це була єдина мова для взаємодії з апаратним забезпеченням комп'ютера. Дані і команди записувалися у цифровому вигляді, що тільки сприяло ускладненню програм. До того ж кожна обчислювальна машина розуміє лише свою машинну мову.

Щоб мати можливість оперувати словами, а не числами, з'явилися асемблери. У них замість чисельного позначення команд та областей пам'яті використовуються словесно-буквені. Але постала проблема: машини не розуміють набір букв. Тому необхідно було створити якийсь перекладач на машинну мову. З цією метою були створені транслятори — спеціальні програми, що перетворюють програмний код з мови програмування на машинний код.

Після асемблерів з'явилися мови програмування високого рівня. Вони є більш зручними для людини і можуть бути виконані на будь-якій машині.

Наступним кроком в еволюції мов програмування була поява об'єктно-орієнтованих мов. Вони дозволяють оперувати віртуальними об'єктами, які є аналогами об'єктів з реального світу.

На сьогодні існує велика кількість мов програмування. Це пов'язано з тим, що збільшується кількість та різноманітність задач, а мови є лише інструментом для їх вирішення [1].

Crystal — об'єктно-орієнтована мова програмування. Її можна охарактеризувати однією фразою: виразна як Ruby, швидка як C. Тобто вона поєднує в собі динамічність разом з статичною типізацією та наявністю компілятора. Вона є проста для писання та читання і націлена на швидкодію. Більш детально Crystal буде описана нижче у цій роботі.

Crystal також підтримує можливість створення веб-застосунків, який і буде створено в рамках цієї роботи.

Веб-застосунок — застосунок, в якому клієнтом виступає браузер, а сервером — веб-сервер. Браузер здатний відображати веб-сторінки, а вся логіка застосунку зосереджена на сервері. Однією з переваг такого підходу є те, що клієнти не залежать від конкретної операційної системи. Внаслідок цієї універсальності і розвитку інтернету веб-застосунки стали широко популярними.

Як предметну область було обрано створення «Системи спілкування для університетської спільноти». Ця область є досить актуальною та має багато реалізацій, які і будуть проаналізовані. На основі зібраної інформації буде запропоновано створення альтернативної системи.

Роботу поділено на три розділи:

У першому розділі розглянуто предметну область, опис функціональності, можливості користувачів та проаналізовано схожі підходи до вирішення розглянутої проблеми.

Другий розділ присвячено аналізу мови програмування, її можливостей та доступних інструментів розробки.

У третьому розділі описується власне створений застосунок та його функціональні можливості.

Постановка задачі:

1) Дослідити предметну область та визначити основні функціональні потреби.
2) Розробити дизайн архітектури та визначитися з засобами розробки.
3) Розробка системи та аналіз результату.

Розділ 1. Дослідження та аналіз предметної області

1.1 Опис предметної області

З моменту зарахування студента до навчального закладу на нього покладаються певні обов’язки та необхідність бути в курсі останніх новин з деканату, кафедри та інших університетських відділів. Також виникає безліч питань стосовно певних процесів, а саме: уточнення процедури поселення в гуртожиток, сплати за навчання або можливості навчання закордоном тощо.

Всі ці питання вирішуються шляхом звернення безпосередньо до необхідного відділу та людей, які там працюють. Тобто необхідно бути присутнім в університеті.

Також наразі існує група студентів, які називають себе «бадді» (від англ. “buddy” — приятель). Їх головною метою є допомога студентам у адаптації до університетського життя. Вони надають основну інформацію про правила університету та відповідають на популярні питання першокурсників. З ними можна підтримувати зв’язок через соціальні мережі. Створюються групи з більш ніж 30-ти людей, в яких студенти можуть задавати питання, а більш обізнані студенти діляться інформацією. Цей шлях комунікації є досить ефективним, але досить багато корисних питань та відповідей губляться під загальними повідомленнями, які не несуть змістовного навантаження.

1.2 Опис груп користувачів

Головними користувачами системи є студенти, особливо першокурсники.

Адміністрація надає права студентам та слідкує за порядком.

Деяким студентам або співробітникам надаються повноваження власника групи. Це означає, що вони вважаються експертами в певній області, тому за ними закріплюється група за даним напрямком.

1.3 Функціональні вимоги до системи

Студент:

1) реєстрація в системі;

2) ознайомлення з існуючими групами;
3) запит на додання до групи;
4) можливість залишати повідомлення та продивлятися відповіді.

Адміністратор:

1) реєстрація в системі;
2) надання студентам прав власника групи;
3) створення груп.

1.4 Дослідження ринку на наявність систем-аналогів

Facebook — соціальна мережа, яка, крім того, що надає людям можливість спілкуватись, дозволяє створювати та долучатися до груп і спільнот за інтересами. Тим самим вона згуртовує людей між собою. Як відомо, ця мережа спочатку працювала виключно в межах Гарвардського університету і призначалася для його студентів. Але кількість користувачів збільшувалась і вона вийшла за рамки університету.

Telegram — месенджер, програмне забезпечення для пристроїв, яке дозволяє обмінюватися текстовими повідомленнями, графічними та відеофайлами, а також безкоштовно телефонувати іншим користувачам програми. Головна особливість — обліковий запис користувача прив’язується до номеру мобільного телефону.

1.5 Висновки

Щороку університети відкривають свої двері для нових студентів. Це двері у нове життя, на порозі яких з’являється безліч питань та необхідність у підтримці з боку більш досвідчених студентів. Навіть знайомство зі своїми одногрупниками чи старостою може сприйматися як виклик, адже все, що ти про них знаєш — це ім’я та прізвище.

Існує безліч каналів зв’язку (Facebook, Telegram тощо), які полегшують пошук людей та спілкування з ними. Це, без сумніву, зручний спосіб спілкування, якому майже неможливо скласти конкуренцію, але ці системи

дозволяють користувачам самим обирати свої імена в системі, давати собі прізвиська або ставити фотографії, які не ідентифікують особистість. Також існує досить багато людей з однаковими прізвищами та співпадінням імен.

Наведені вище моменти показують певні труднощі, на вирішення яких націлена дана робота. Буде створено «Систему спілкування…», яка допоможе у вирішені питань, пов’язаних з:

1. Централізованим пошуком одногрупників.

2. Централізованим викладенням інформації від студентських організацій та новин університету (з метою відокремитись від соціальних мереж і тримати у курсі подій студентів, які не мають аккаунтів у них). Іноді студенти навіть не уявляють про існування певних каналів новин деяких студентських організацій.

3. Відсутністю необхідності бути присутнім в університеті з метою отримання інформації про певні процеси чи терміни (або просити когось спитати за тебе).

4. Наданням якісної та однозначної інформації від людей, які дійсно розбираються в поставленому студентом питанні.

5. Відсутністю спаму та інформації, яка не стосується поставленого питання чи теми.

Розділ 2. Інструменти для розробки веб-застосунків

Веб-застосунок — це сервіс, доступ до якого здійснюється через браузер. Його перевагою є те, що користувачі не залежать від конкретної операційної системи та мають вільний доступ до функціональності застосунку без додаткових налаштувань. Використання веб-застосунку для даної предметної області має низку переваг, зокрема:

1) масштабованість веб-системи;
2) зручне розгортання та обслуговування системи;
3) інтеграція з іншими системами;
4) розмежування прав доступу до різної функціональності.

2.1 Crystal

Crystal — об’єктно-орієнтована мова програмування, яка має схожий синтаксис з мовою Ruby, але є більш ефективною на стадії виконання. Ця ефективність досягається за рахунок того, що Crystal є компільованою мовою програмування.

Crystal була започаткована у 2011 році і досить швидко вся команда з Аргентинської компанії Manas Technology Solutions прийняла участь у її розробці. Зараз це є повністю відкритий проект на GitHub [2], що налічує більш ніж 250 контрибюторів. На даний момент Crystal знаходиться на 0.33.0 версії і впевнено прямує до 1.0 релізної версії.

Crystal характеризується такими особливостями:

1) статичною системою типів, де типи більшу частину часу визначаються автоматично;
2) автоматичним збирачем сміття, що дозволяє безпечно контролювати використання пам’яті;
3) компіляцією у машинний код з використанням LLVM (Low Level Virtual Machine — проект програмної інфраструктури, який полегшує написання компіляторів для різних мов програмування та платформ);
4) повністю об’єктно-орієнтованим стилем — в Crystal все є об’єктами;

5) підтримка generics так само як і перевантаження методів та операторів;

6) масштабованість і конкурентність.

2.1.1 Основні характеристики

Розглянемо основні характеристики, які виділяють дану мову програмування.

2.1.1.1 Наявність компілятора

Кожна програма являє собою набір інструкцій на виконання певних дій, написаних на мові, зрозумілій для людського сприйняття. З метою зробити ці інструкції зрозумілими для машин/комп’ютерів використовуються компілятори та інтерпретатори. Завдання компіляторів одразу перекласти програму в машинний код, а трансляторів — зчитувати інструкцію одна за одною та виконувати її.

Розглянемо переваги та недоліки компільованих мов програмування [3].

Переваги:

1. Швидкість та ефективність.

За рахунок того, що компільована мова програмування перетворюється одразу в машинний код, який процесор здатен виконати, досягається більша швидкість та ефективність виконання порівняно з інтерпретованими мовами.

2. Більше можливостей контролю.

Компіляція дає розробникам можливість контролю низькорівневих аспектів програмування, таких як керування пам’яттю або використання CPU (Central Processing Unit — функціональна частина комп'ютера, що призначена для інтерпретації команд).

3. Зручність експлуатації.

Компільовану мову програмування зазвичай легше поширювати, тому що залежності компілюються у один файл і мають швидший час запуску.

Недоліки:

1) при внесенні змін необхідно перекомпілювати всю програму;

2) згенерований машинний код залежить від платформи.

Тепер розглянемо переваги та недоліки інтерпретованих мов програмування.

Переваги:

1) незалежність від платформи;
2) можливість для використання динамічної типізації.

Недоліки:

1) швидкість виконання порівняно з компільованими мовами.

Для вдосконалення ефективності виконання інтерпретованих програм використовують JIT (Just-In-Time)-компіляцію [4]. Вона націлена на використання переваг наведених вище підходів: під час роботи інтерпретованої програми JIT-компілятор визначає найбільш використовуваний код (в залежності від компілятора: методи або секції коду) та компілює його в машинний код. Важливо розуміти, що JIT-компілятор компілює байткод в машинний код. Байткод — це початковий код, згенерований у код, який може бути виконаним віртуальною машиною. В той же час, машинний код напряму виконується CPU [5].

Crystal не має ні інтерпретатора, ні віртуальної машини і є компільованою мовою програмування.

Щоб скомпілювати програму у виконуваний код, компілятору необхідно знати типи всіх виразів у коді. Компілятор у Crystal використовує inference algorithm, який дозволяє йому розпізнавати більшу частину типів, тому розробнику не часто доводиться їх явно вказувати.

2.1.1.2 Статична типізація

Дана характеристика означає, що кожна змінна має один або декілька потенційних типів і вони повинні бути визначені під час компіляції. Це запобігає багатьом помилкам на моменті виконання.

Crystal вимагає явно визначати тип змінної в програмі лише тоді, коли синтаксично незрозуміло якого типу вона повинна бути. Це означає, що зазвичай можна не вказувати тип до якого відноситься змінна.

Ще одною важливою характеристикою є неможливість отримати Nil/Null помилок в режимі роботи програми оскільки компілятор перевіряє на відсутність об’єкту перед тим як взаємодіяти з ним [6].

2.1.1.3 Багатопоточність

В наш час з багатоядерними процесорами та розподіленими обчисленнями розробники потребують від мов програмування відмінну підтримку конкурентності (concurrency) та паралелізму.

Конкурентність означає, що декілька задач починаються, працюють та закінчуються незалежно одна від одної та в довільному порядку. Вона має місце тоді, коли ми говоримо про як мінімум дві задачі чи більше. Ми можемо назвати застосунок конкурентно спроможним тоді, коли він може виконувати дві задачі в один проміжок часу. Здається, що задачі виконуються паралельно, але насправді це не так. Вони користуються функцію розподілу часу процесора в операційній системі, де кожна задача виконує певну свою частину і переходить в статус очікування. Коли перша задача в статусі очікування, то процесор дозволяє другій задачі виконати частину своєї роботи. Операційна система, базуючись на пріоритетності задачі, виділяє процесорний час та надає необхідні обчислювальні можливості.

Паралелізм — це коли декілька задач або декілька частин однієї задачі працюють в один проміжок часу, наприклад на багатоядерному процесорі. Він не вимагає існування двох завдань. Він буквально виконує частини завдань або декількох завдань одночасно, використовуючи багатоядерну інфраструктуру процесора, присвоюючи одне ядро кожному завданню або під задачі. Паралелізм вимагає наявність декількох ядер у процесора. Наявність одного ядра здатна забезпечити роботу конкурентності, але не паралелізму [7, 8].

Crystal націлена з самого початку на підтримку конкурентності та паралельних розрахунків.

Модель одночасності в Crystal базується на двох концепціях:

1. Волокна програмування (fibers) — це легкі потоки, контроль за якими належить Crystal.

2. Канали — через них волокна програмування взаємодіють один з одним.

Головний потік також є волокном програмування, але інші волокна програмування будуть працювати на задньому фоні без блокування головного потоку. Каналам необхідно знати, який саме тип даних проходить через них, тому вони типізовані.

Приклад створення та роботи волокон програмування можна розглянути в додатку Б.

До 0.26.1 версії Crystal працювала в одному потоці. Це означає, що вона підтримувала конкурентність, але вона все ще не могла виконувати паралельні обчислення. Починаючи з вересня 2019 року вийшли нові оновлення, які дозволяють використовувати паралелізм.

2.1.2 Огляд механізму виконання програм

Коли запускається програма, головний потік розпочинає роботу з метою виконання коду верхнього рівня. В цьому коді можуть бути викликані інші волокна програмування, які будують чергу. На задньому фоні Crystal виконує набір мінімальних задач за допомогою певних компонентів, а саме:

1. Runtime Fiber Scheduler. Його головною задачею є виконання всіх волокон програмування у черзі в конкретний момент часу. Якщо з якихось причин волокно програмування не може бути виконано, то воно посилає сигнал до Scheduler, щоб той переключився на виконання іншого волокна програмування або відновив роботу попереднього.

2. Non-blocking Event Loop — це волокно програмування, яке відповідає за все, що відноситься до I\O (input\output) — асинхронних задач, відправки пакетів по мережі, таймерів тощо.

3. Garbage Collector — очищає пам’ять, яку програма більше не використовує.

Базовий розмір кожного волокна програмування становить 4KB і може рости до 8MB. Можна створити дуже багато волокон програмування, але є певні обмеження для старих систем. Наприклад на 64-бітній машині можна створити мільйони волокон програмування, але на 32-бітній — лише 512 [6].

Зі схемою роботи програми на Crystal можна ознайомитись в додатку В.

2.1.3 Огляд базових можливостей

Crystal надає досить багато утилітних команд, які дозволяють швидко розпочати проект. Крім того, дивлячись на те, що компілятор здатен відловлювати досить багато помилок самостійно, Crystal фактично вже написала багато вхідних тестів, які необхідні при кодуванні на динамічних мовах. Для тестування також існує стандартна бібліотека, яка складається з модулю Spec та підтримує unit-тестування та BDD (Behavior-Driven Development).

Короткий перелік того, що доступно від Crystal:

1) консольна команда для створення проекту з готовими шаблонами коду та класами тестування;
2) контроль за залежностями, який дозволяє легко та автоматично додавати сторонні бібліотеки (shards);
3) форматувальник коду;
4) генератор документації;
5) веб-середовище для роботи з кодом, яка має назву Crystal playground.

Перелік застосунків для яких Crystal підходить краще за все:

1) швидкі веб-додатки — аналоги тим, які написані на Ruby;
2) швидкі розширення для Ruby;
3) веб-сервіси та мікросервіси;
4) консольні застосунки;
5) програми, які відмальовують ігри та графіки;
6) маленькі утилітні програми;
7) Iot застосунки.

Crystal гарно підходить для оптимізації вже написаних на Ruby застосунків з метою покращення їх продуктивності. Близько 15 компаній, таких як ProTel, Bulutfon, DuoDesign, Appmonit, RainForest QA і Manas, вже використовують дану мову програмування на реальних проектах [6].

2.1.4 Огляд базових сервісів

Якщо звернути увагу на вже створені проекти на Crystal, то всі вони мають однаково просту структуру. Це досягається за рахунок наявності утилітної команди, яка генерує цю базову структуру проекту та робить проекти більш зручними для дослідження.

Команда для створення нового проекту:

$ crystal init TYPE NAME [DIR]

TYPE — може набувати значень app чи lib

NAME — назва проекту

[DIR] — необов'язковий параметр для зазначення папки, в якій буде створено проект

Розглянемо більш детально згенеровані файли (рисунок 2.1).

```
root@LAPTOP-89CH9ORI:/usr# /usr/bin/crystal init app structure_app
    create  structure_app/.gitignore
    create  structure_app/.editorconfig
    create  structure_app/LICENSE
    create  structure_app/README.md
    create  structure_app/.travis.yml
    create  structure_app/shard.yml
    create  structure_app/src/structure_app.cr
    create  structure_app/spec/spec_helper.cr
    create  structure_app/spec/structure_app_spec.cr
Initialized empty Git repository in /usr/structure_app/.git/
```

Рисунок 2.1. Згенеровані файли після створення проекту

1) .gitignore — ініціалізований та порожній гіт-репозиторій;
2) .editorconfig — дозволяє керувати форматуванням файлів;
3) .travis.yml — дозволяє інтеграцію з Travis для неперервної інтеграції;
4) shard.yml — файл для керування залежностями;
5) src — папка, в якій знаходиться вихідний код;
6) spec — папка, в якій знаходяться тести.

В цьому розділі буде розглянуто сервіси, які стають доступними після генерації проекту.

2.1.4.1 Shards

Shards — менеджер залежностей [9]. Застосунок та бібліотеки на Crystal очікують наявність shard.yml для контролю проектних залежностей.

При виростанні команди для встановлення залежностей, з’явиться файл shard.lock, в якому фіксуються встановлені залежності. При додаванні нових залежностей необхідно оновлювати shard.lock файл командними утилітами.

Всі зовнішні бібліотеки зберігають в папці lib.

2.1.4.2 Travis

Travis CI — розподілений веб-сервіс для тестування програмного забезпечення, який інтегрується з GitHub [10, 11]. Програмна частина сервісу теж розміщена на GitHub. Використовується для запуску збірок і тестів на кожному коміті та підтримує більше 20 різних мов програмування.

Неперервна інтеграція (Continuous Integration/CI) — це частина процесу розробки, в якій проект збирається та тестується в різних середовищах виконання автоматично та неперервно.

Мета даного підходу — виявлення помилок інтеграції на ранній стадії.

Налаштування:

Travis CI налаштовується шляхом додавання файлу з ім'ям .travis.yml. В цьому файлі визначається мова проекту, бажане середовище побудови та тестування.

Принцип роботи:

1) розробники роблять локальні зміни та відправляють їх на сервер в загальний репозиторій;
2) репозиторій відправляє запит (webhook) CI-системі;
3) CI-система запускає завдання (інспекція коду, тести);
4) у разі успішного виконання або виникнення помилок, CI-система сповіщає розробників (як варіант — надсилання електронного листа).

Особливості роботи:

1) Коли файл .travis.yml потрапляє в корінь github-репозиторію, виконання завдань розпочинається тільки тоді, коли звільниться обробник для заданої мови програмування.

2) Кожна нова збірка проекту починається з самого початку. Тобто зміни, які були отримані в рамках даної збірки, наприклад при тестуванні, будуть недоступні в наступних збірках.

3) Є валідатор файлу — travis-lint.

2.1.4.3 Sentry

Sentry — система відслідковування помилок для виявлення та моніторингу проблем [12, 13]. Являє собою дошку зі списком помилок і можливістю виконувати над ними різні дії.

Основні можливості:

1) Список помилок, який оновлюється в режимі реального часу.

2) Помилки групуються та відображаються за частотою виникнення.

3) Є можливість фільтрації помилок за різними критеріями: статус, рівень логування, ім’я серверу тощо.

2.1.5 Обмеження мови програмування

1. Не існує версії компілятора, яка б працювала на Windows.

2. За Crystal не закріпилось поняття «мова програмування» тому важко знаходити якусь інформацію.

3. Маленька спільнота людей, які мають змогу допомогти при вирішенні проблеми, з якою можна стикнутися на етапі розробки.

4. Через статичну типізацію дуже важко створити REPL (Read-eval-print loop — цикл читання-обчислення-друку або інтерактивне середовище програмування), еквівалент irb у Ruby.

5. Не дуже великий вибір допоміжних засобів розробки, а деякі з них не достатньо оптимізовані та вирішують не всі задачі.

2.2 Веб-фреймворк

Веб-фреймворк — це інструмент, який полегшує процес створення та запуску веб-застосунків. Він бере на себе свого роду інкапсуляцію певного функціоналу з метою скоротити кількість написаного коду програмістом та полегшити програму. Також веб-фреймворки задають загальний підхід до розробки веб-застосунків.

Загалом архітектура майже всіх фреймворків заснована на відділені декількох рівнів, а саме модель (model), рівень представлення даних (view) та контролер (controller). Це три складові MVC (Model-View-Controller) — архітектурний шаблон, який поділяє систему на три взаємопов'язані частини. Перевагою цього підходу є модульність — можливість замінять якийсь з рівнів на інший без суттєвих труднощів.

Також веб-фреймворки є багатофункціональними. Це означає, що вони можуть підтримувати такі допоміжні можливості як керування безпекою, веб-кешування, систему веб-шаблонів та інші.

2.2.1 Огляд веб можливостей Crystal

Стандартна бібліотека Crystal має модуль HTTP, який забезпечує базовими веб-техніками, а саме: відловлювання помилок, логування, веб-сокети, використання форм тощо. Також сюди входить веб сервер, який можна підняти лише в декілька рядків коду. Нижче наведена частину коду та її аналіз (рисунок 2.2).

```
require "http/server"

① server = HTTP::Server.new do |ctx|
②   ctx.response.content_type = "text/plain"
③   ctx.response.print "Crystal web server: got #{ctx.request.path}"
end

server.bind_tcp 8080
④ server.listen
puts "Crystal web server listening on http://localhost:8080"

# => in browser with the URL http://localhost:8080/
# "Crystal web server: got /"
```

Рисунок 2.2. Створення веб-серверу

Щоб розпочати, необхідно оголосити HTTP модуль за допомогою require.

1 крок: створюється об’єкт класу Server. Створюється волокно програмування, яке реагує на кожен запит, тому Crystal веб-сервер повністю багатопоточний. Отриманий запит переходить для обробки у блок do-end. У ньому ми можемо отримати доступ до ctx параметру (об’єкт класу Context), який вміщує в собі інформацію з запиту. Волокно програмування встановлює відповідь від серверу:

2 крок: вказуємо content-type. На цьому кроці можна додати статус або заголовки.

3 крок: записуємо рядок у відповідь. Також показано як ми можемо діставати параметри з запиту. Сервер відправляє цю інформацію веб-клієнту (зазвичай браузеру), який обробить запит та виведе його. Також варто прив’язати сервер до IP-адреси і порту використовуючи bind_tcp метод. Якщо даному методу передати параметр true, то веб-сервер дозволить повторне використовувати вказаний порт. Це означає, що декілька сокетів можуть під’єднатися до одного й того ж порту на одному хості, симулюючи багатопоточну поведінку.

4 крок: стартуємо сервер.

Наведений вище приклад показує створення веб-сервера використовуючи наявний в Crystal модуль з коробки. Веб-сервер має гарну продуктивність, але в той же час вимагає досить багато коду тому розглянемо використання фреймворків [6].

2.2.2 Kemal

Kemal — швидкий та є визнаним стандартом серед веб-фреймворків для Crystal [14]. Він має модульну архітектуру, а однією з його головних цілей є простота. Його CPU та вимоги до пам’яті досить маленькі: один сервер здатен обробляти десятки тисяч клієнтів.

Щоб використовувати Kemal у своєму застосунку його необхідно додати до залежностей shard.yml. Окрім, власне, самого фреймворку також будуть додані залежності на:

1. Radix — використовується для зв’язування запитів з відповідними обробниками.

2. Kilt — використовується для відображення шаблонів.

Нижче буде розглянуто створення вже готового контролеру (рисунок 2.3).

```
require "./simple_kemal_app/*"
① require "kemal"

② get "/" do
    "My first Kemal app is alive!"
  end

③ Kemal.run
```

Рисунок 2.3. Створення контролеру

1 крок: додати залежність на Kemal.

2 крок: відловлювання GET запитів за шляхом: "http://localhost:3000/" і повернення стрічки клієнту.

3 крок: власне запуск веб-сервера.

Як параметр до блока можна використовувати змінну середовища env. Вона дозволяє отримати доступ до параметрів та формувати відповідь.

Kemal може відловлювати всі HTTP методи, а саме GET, POST, PUT, PATCH і DELETE. Це означає, що ваш застосунок забезпечений RESTful архітектурою. Зауваження: шляхи відловлюються за порядком їх визначення — буде активовано перший шлях з яким співпаде запит.

Для відображення динамічних сторінок використовується ECR (Embedded Crystal) модуль. Це мова шаблонів зі стандартної бібліотеки, яка використовується для вставлення частин коду на Crystal у текст як HTML. Всі шаблони компілюються в один рядок, що досить ефективно.

Стисло про інші можливості Kemal:

1. Додавання фільтрів для створення власної логіки обробки запитів та відповідей.
2. Модульність Kemal досягається за рахунок хендлерів (handler) — класів, які викликаються в залежності від HTTP методу. Їх використовують з метою логування, обробки помилок, аутентифікації, csrf захисту тощо.
3. Kemal може зберігати інформацію в сесії. Також в пам’яті, файлах, базах даних.
4. Kemal підтримує веб-сокети, які працюють значно швидше порівняно з Node.js.
5. Застосунок на Kemal можна викласти на Heroku.

Даний фреймворк і буде використовуватись в рамках цієї роботи так як вміщує в собі всі необхідні функції [6].

2.3 ORM

ORM — технологія програмування, яка поєднує бази даних з концепціями об’єктно-орієнтованих мов програмування.

В об’єктно-орієнтованому програмуванні об’єкти в програмі представляють об’єкти з реального світу і при розробці важливо мати можливість оперувати цими об’єктами. Тобто необхідно перетворити їх у певну форму, в якій вони можуть бути збережені у файлах чи в базах даних і якими можна буде легко маніпулювати в подальшому, зі збереженням властивостей та відношень між цими об’єктами.

Одним з варіантів вирішення проблеми зберігання даних є реляційні бази даних. Модель реляційної бази даних характеризується простою структурою даних у вигляді таблиць і можливістю використання алгебри відношень і реляційного обчислення для обробки даних.

В розробці програм виникає необхідність постійного перетворення між двома різними формами даних — об’єктною та реляційною. Це створює труднощі так як кожна форма представлення даних накладає певні обмеження одна на одну.

З метою вирішення цієї проблеми було розроблено безліч пакетів, які автоматично виконують наведені вище перетворення, маючи список таблиць бази даних та об'єктів в програмі. З програмної точки зору для додавання нового об'єкту у реляційну базу даних варто лише створити цей самий об'єкт в програмі. На практиці, кожна ORM має певні особливості та все таки не дає забути про використання бази даних.

ORM здатна як вберегти розробників від помилок при розробці запитів, оптимізувати певні процесі та позбавити від дублювання коду, так і накласти залежності на архітектуру та уповільнити застосунок.

При використанні ORM є два базові підходи до розробки проекту:

1. Repository pattern (рисунок 2.4).

```
Entity e = Entity.create();
e.setSomeField("abc");
repository.save(e);
```

Рисунок 2.4. Приклад коду з використанням Repository pattern

У даній моделі є набір класів-репозиторіїв, які відповідають за маніпулювання даними та класи DTO (Data Transfer Object), які просто зберігають в собі дані і не мають ніякої логіки. Класи-репозиторії беруть на себе управління доступом до інформаційних ресурсів. Це можуть бути як бази даних так і сторонні веб-API. Репозиторії також керують транзакціями та кешуванням даних.

2. Active Record Pattern (рисунок 2.5).

```
Entity e = Entity.create();
e.setSomeField("abc");
e.save();
```

Рисунок 2.5. Приклад коду з використанням Active Record Pattern

Даний підхід є протилежним до Repository pattern так як класи-об’єкти інкапсулюють в собі логіку виконання операцій і самі вирішують як зберігати дані та як маніпулювати своїми полями. Тобто вся бізнес логіка переноситься в об’єкти.

Головна мета — отримувати доступ до бази даних всередині бази даних, а так як класи в програмі виступають у ролі обгорток таблиць бази даних, то в них і мають бути методи роботи з базою даних [15].

2.3.1 Jennifer

Jennifer — це ORM розроблена для Crystal, яка базується на Active Record Pattern та підтримує DSL-запити разом з механізмом міграцій.

2.3.1.1 DSL-запити

DSL (Domain-Specific Language) — мова програмування, призначена для вирішення задач конкретної предметної галузі. Вона є протилежністю до мов програмування загального призначення.

DSL-запити це інструмент для роботи з базою даних, який дозволяє працювати з нею навіть не знаючи SQL і використовуючи звичайні програмні об’єкти. DSL працює з елементами бази даних як зі звичайними полями класу.

Розглянемо приклади основних запитів. Ілюстрації коду взяті з документації [16, 23].

1. Where

Щоб дістати всі поля моделі використовується #all.

```
Contact.all
```

Представити запит where можна різними шляхами:

```
Contact.where { c("id") == 1 }
Contact.where { _id == 1 }
Contact.all.join(Address) { Contact._id == _contact_id }
Contact.all.relation(:addresses).where { __addresses { _id > 1 } }
Contact.all.where { _contacts__id == 1 }
```

Метод приймає блок, який і є умовою запиту where.

До поля таблиці можна звернутися за допомогою методу #c, передавши рядкове представлення імені поля або використовуючи конструкцію _field_name.

2. Select

Для використання базового запиту sql SELECT використовується метод #select.

Він має найвищій пріоритет при побудові запиту.

```
Contact
  .all
  .select("COUNT(id) as count, contacts.name")
  .group("name")
  .having { sql("COUNT(id)") > 1 }
  .pluck(:name)
```

3. From

Також можна використати метод #from для визначення запиту from.

```
Contact.all.from("select * from contacts where id > 2")
Contacts.all.from(Contact.where { _id > 2 })
```

4. Join

Щоб поєднати одну таблицю з іншою використовується метод #join. В нього передається модель класу або ім’я таблиці та тип запиту join.

```
Contact.all.left_join(Address) { _contacts__id == _contact_id }
Contact.all.right_join("addresses") { _contacts__id == c("contact_id") }
```

Існують такі типи з’єднання [17, 18]:

1. Внутрішнє — INNER

Повертає записи, які спільні для обох табличок. За замовчуванням JOIN використовує саме з’єднання INNER (рисунок 2.6).

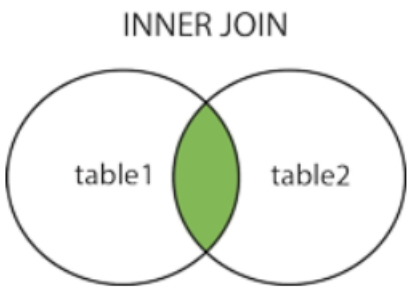


Рисунок 2.6. Ілюстрація запиту INNER JOIN

2. Зовнішнє — OUTER

- Ліве — LEFT OUTER

Даний тип повертає всі рядки таблиці з лівої сторони з'єднання та рядки, які співпали з правої таблиці. Рядки з лівої сторони, які не мають співпадінь заповнюються null (рисунок 2.7).

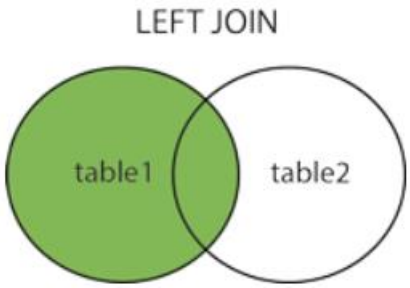


Рисунок 2.7. Ілюстрація запиту LEFT OUTER

- Праве — RIGHT OUTER

Даний тип з'єднання схожий на попередній. Він повертає всі рядки таблиці, яка знаходиться з правої сторони з'єднання та рядки, які співпали з лівої таблиці. Рядки, для яких немає співпадінь з лівої таблиці заповнюються null (рисунок 2.8).

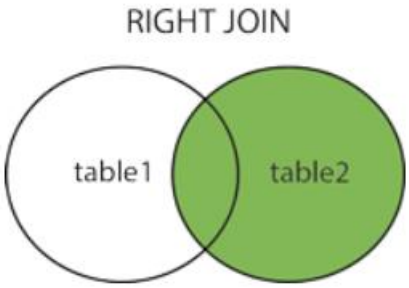


Рисунок 2.8. Ілюстрація запиту RIGHT OUTER

- Повне — FULL OUTER

Створюється проміжна таблиця, яка поєднує два типи LEFT JOIN та RIGHT JOIN. В ній будуть наявні рядки з двох таблиць. Рядки, для яких не буде значення заповнюються null (рисунок 2.9).

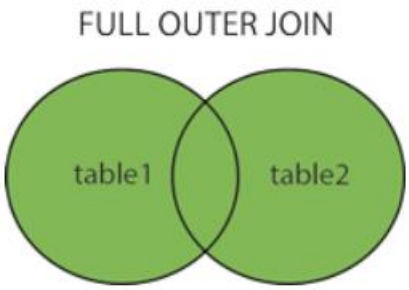


Рисунок 2.9. Ілюстрація запиту FULL OUTER

3. Перехресне — CROSS

При такому з’єднані кожен рядок першої таблиці поєднується з рядками іншої таблиці. Якщо перша таблиця містить x рядків, а друга таблиця — y, тоді результатом буде таблиця з x*y кількістю рядків (рисунок 2.10).

Рисунок 2.10. Ілюстрація запиту CROSS

4. Самоз’єднання — SELF-JOIN

При такому типі таблиця з’єднується сама з собою.

2.3.1.2 Функція зворотного виклику

Jennifer дозволяє керувати всім необхідним для моделей програми — починаючи з міграцій до бази даних і відношенням полей до стовпчиків

таблиць, закінчуючи функцією зворотного виклику (callback) та побудовою запитів [19].

Функція зворотного виклику — частина виконуваного коду, що передається як аргумент до іншого коду, який має викликати цей код у відповідь, тобто виконати аргумент у певний момент часу [20].

Під час роботи програмного застосунку об’єкти створюються, оновлюються та видаляються. Jennifer дозволяє контролювати та маніпулювати об’єктами в різні цикли його життя.

Розглянемо приклад створення зворотного виклику (рисунок 2.11).

```
class User < Jennifer::Base::Model
  mapping(
    id: Primary32,
    email: String
  )

  before_validation :clean_up_email

  private def clean_up_email
    self.email = email.gsub('+', "")
  end
end
```

Рисунок 2.11. Приклад коду створення зворотного виклику

Спочатку необхідно визначити та створити функцію зворотного виклику як метод, а потім використати макрос (macro) для реєстрації.

Макроси — це методи класу, які створюють нові методи сутностей. Загалом, макроси — це фрагменти коду, які генерують інший код.

Доступні функції зворотного виклику:

1) під час створення нового об’єкту:
   - before_validation
   - after_validation

- before_save
- before_create
- after_create
- after_save
- after_commit / after_rollback

2) під час оновлення існуючого об’єкту:
   - before_validation
   - after_validation
   - before_save
   - before_update
   - after_update
   - after_save
   - after_commit / after_rollback
3) під час видалення об’єкту:
   - before_destroy
   - after_destroy
   - after_commit / after_rollback

2.3.1.3 Асоціації та міграції

Асоціації це зв’язки між моделями. Вони необхідні для простішого виконання базових операцій в коді. Зв’язки реалізуються з використанням макро-викликів. Наприклад використовуючи зв’язок belongs_to, Jennifer зберігає інформацію про зовнішні ключі між сутностями двох моделей та отримує декілька корисних методів, доданих до моделі [21, 24].

Jenifer підтримує чотири типи зв’язків:

1) belongs_to

Дана асоціація встановлює зв’язок один-до-одного з іншою моделлю так, що кожна сутність, яка проголошує зв’язок belongs_to, є сутність іншої моделі. Наприклад є таблиці автори та книги і кожна книга може бути пов’язана тільки з одним автором (рисунок 2.12).

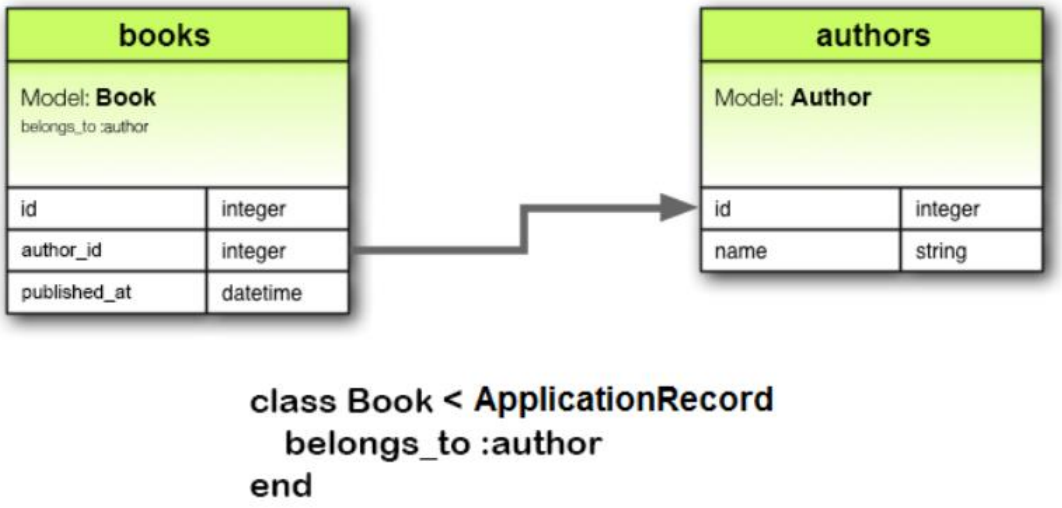


Рисунок 2.12. ER-модель запиту belongs_to

Міграція до бази даних матиме вигляд як на рисунку 2.13.

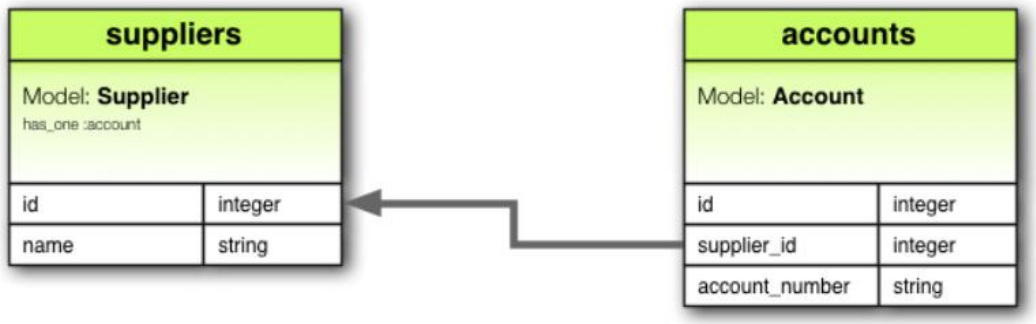
```
class CreateOrders < ActiveRecord::Migration[5.0]
  def change
    create_table :authors do |t|
      t.string :name
      t.timestamps
    end

    create_table :books do |t|
      t.belongs_to :author
      t.datetime :published_at
      t.timestamps
    end
  end
end
```

Рисунок 2.13. Приклад міграції для запиту belongs_to

2) has_one

Також визначає зв’язок один-до-одного з іншою моделлю. Цей зв’язок показує, що кожна сутність моделі містить одну сутність іншої моделі. Наприклад кожен постачальник має лише один обліковий запис (рисунок 2.14).




```
class Supplier < ApplicationRecord
  has_one :account
end
```

Рисунок 2.14. ER-модель запиту has_one

Міграція матиме вигляд як на рисунку 2.15.

```
class CreateSuppliers < ActiveRecord::Migration[5.0]
  def change
    create_table :suppliers do |t|
      t.string :name
      t.timestamps
    end

    create_table :accounts do |t|
      t.belongs_to :supplier
      t.string :account_number
      t.timestamps
    end
  end
end
```

Рисунок 2.15. Приклад міграції для запиту has_one

3) has_many

Визначає зв’язок один-до-багатьох с іншою моделлю. Він вказує на те, що кожна сутність моделі може мати нуль або більше сутностей іншої моделі. Наприклад автор може мати багато книжок (рисунок 2.16).

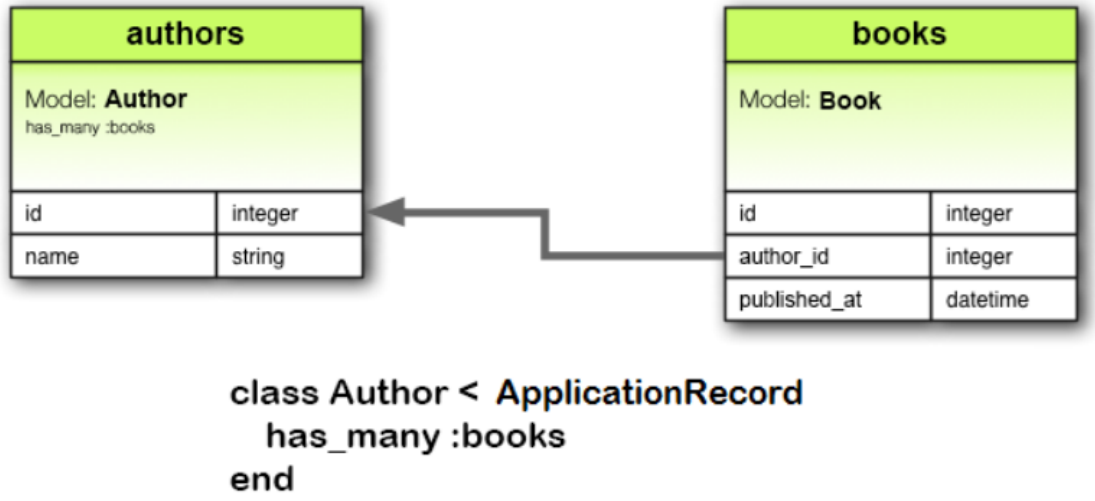


Рисунок 2.16. ER-модель запиту has_many

Міграція матиме вигляд як на рисунку 2.17.

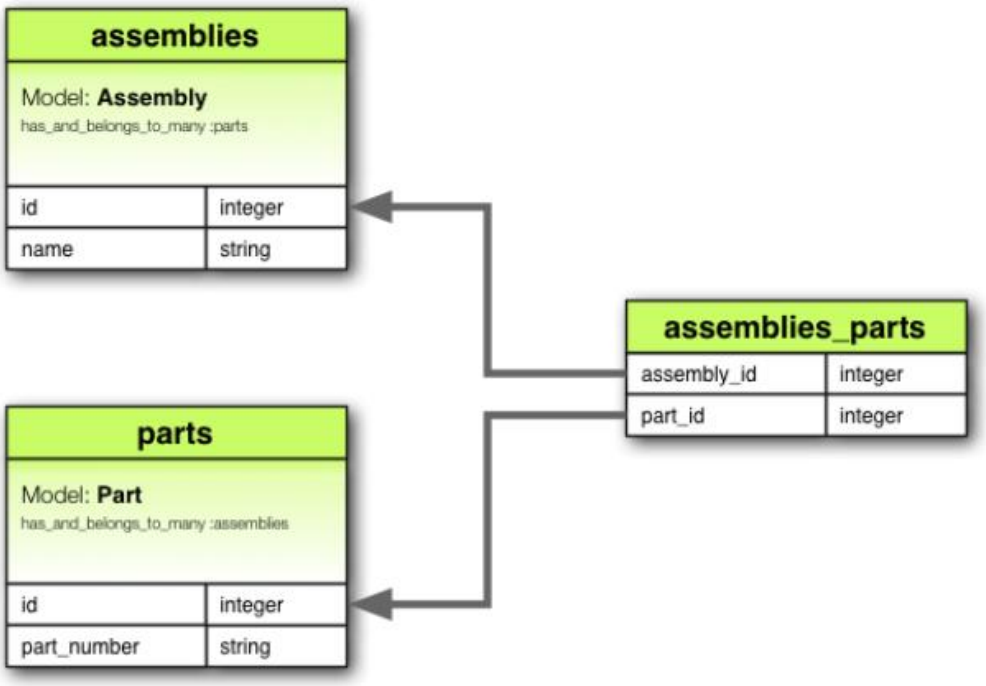
```
class CreateAuthors < ActiveRecord::Migration[5.0]
  def change
    create_table :authors do |t|
      t.string :name
      t.timestamps
    end

    create_table :books do |t|
      t.belongs_to :author
      t.datetime :published_at
      t.timestamps
    end
  end
end
```

Рисунок 2.17. Приклад міграції для запиту has_many

4) has_and_belongs_to_many

Тип описує прямий зв’язок багато-до-багатьох з іншою моделлю, без проміжної моделі. Наприклад застосунок складається зі збірки та деталей, де кожна збірка складається з багатьох деталей та кожна деталь входить до багатьох збірок (рисунок 2.18).




```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

Рисунок 2.18. ER-модель запиту has_and_belongs_to_many

Міграція матиме вигляд (рисунок 2.19).

```
class CreateAssembliesAndParts < ActiveRecord::Migration[5.0]
  def change
    create_table :assemblies do |t|
      t.string :name
      t.timestamps
    end

    create_table :parts do |t|
      t.string :part_number
      t.timestamps
    end

    create_table :assemblies_parts, id: false do |t|
      t.belongs_to :assembly
      t.belongs_to :part
    end
  end
end
```

Рисунок 2.19. Приклад міграції для запиту has_and_belongs_to_many

2.3.1.3.1 Вибір між belongs_to та has_one

При налаштуванні зв’язку один-до-одного необхідно використовувати ці два відношення та обрати, до якої моделі додати belongs_to, а до якої has_one.

Belongs_to використовується в моделі, в якій повинен бути зовнішній ключ. Також варто дивитись на предметне значення відношень. Так, в наведеному нижче прикладі, постачальник володіє обліковим записом, а не обліковий запис — постачальником (рисунок 2.20).

```
class Supplier < ApplicationRecord
  has_one :account
end

class Account < ApplicationRecord
  belongs_to :supplier
end
```

Рисунок 2.20. Приклад моделі з використанням зв’язку один-до-одного

2.4 Heroku

Для того аби викласти свій застосунок в інтернет та мати можливість доступатися до нього за URL-адресою буде використано сервіс Heroku.

Heroku — хмарка PaaS(Platform as a service)-платформа, що підтримує різні мови програмування.

Платформа як послуга (англ. Platform as a service, PaaS) — модель надання хмарних обчислень, при якій споживач отримує доступ до використання інформаційно-технологічних платформ: операційних систем, систем управління базами даних, зв'язного програмного забезпечення, засобів розробки і тестування розміщених у хмарних провайдерах [22].

Початкові кроки використання платформи:

1. Аби розпочати користування платформою необхідно створити обліковий запис.

2. Після цього варто створити git-репозиторій. Heroku підтримує лише Git як систему контролю версій.

3. Далі визначаємо назву проекту. Ця назва буде відображатися в URL на якому буде розміщено застосунок.

4. Коли будуть створені перші коміти (commit) тоді можна розгорнути застосунок за допомогою git-команди — git push heroku master

5. В своєму особистому кабінеті на сайті Heroku можна подивитися URL, за якою розміщено застосунок, а також вікно з логуванням.

База даних MySQL також можна викласти у хмару для віддаленого доступу. Для цього використовується утиліта ClearDB MySQL. Її також можна підключити в особистому кабінеті Heroku.

2.5 Висновки

Crystal — об’єктно-орієнтована мова програмування, яка має такі особливості як наявність компілятора, статична типізація та багатопочність. Вона підтримує написання макросів, які дозволяють розробникам писати функції, котрі будуть виконані на етапі виконання. Також варто зазначити про наявність системи керування залежностями Shards, яка вміщує в собі багато корисних можливостей. В цьому розділі були описані основні моменти функціонування програм, написаних на мові Crystal та сервіси, доступ до яких пропонується за замовчуванням. Так як дана мова ще на етапі становлення, то не обійтись від обмежень, які накладаються мовою на програміста, але вони не є критичними та точно не можуть затьмарити переваги Crystal.

Немалу увагу було приділено веб-фреймворку Kemal. Він є легким для ознайомлення, але це не накладає обмежень по функціональності. Kemal — надзвичайно швидкий веб-фрейворк, який забезпечує гарну продуктивність. До того ж він гарно взаємодіє з Heroku тому у розробника не виникне ніяких труднощів при розгортанні проекту в хмарі.

На мою думку ORM є також важливою складовою при розробці проекту. Crystal має досить багато доступних ORM, але, на жаль, не всі з них

підтримують гарну функціональність та легку взаємодію. Jennifer була другою ORM-системою, яку я спробувала, та яка задовольнила мої потреби при розробці. Вона гарно взаємодіє з веб-фрейморков, надає механізм підтримання міграцій, створення асоціацій і найголовніше — DSL.

Розділ 3. Опис веб-застосунку

3.1 Загальна характеристика

«Система спілкування…» передбачає можливість спілкуватися починаючи від двох користувачів та більше. З цією метою було використано технологію веб-сокетів.

Веб-сокети — це дві кінцеві точки двонаправленого каналу зв'язку. Через цей канал клієнт (браузер) буде спілкуватися з сервером (веб-застосунком) в режимі реального часу [25, 26].

На відміну від HTTP-протоколу, при якому браузер постійно відправляє запити до серверу і перевіряє на наявність оновлення та обробляє їх лише тоді, коли отримує відповідь з даними, веб-сокету не потрібно постійно відправляти запити на сервер. Достатньо один раз зробити запит і чекати поки сервер відповість.

Тобто дана технологія чудово підходить для розробки системи, в якій необхідна взаємодія користувачів в режимі реального часу.

3.2 Опис функціоналу

При переході за Url-адресою (http://kma-community.herokuapp.com), користувач потрапляє на головну сторінку веб-застосунку (рисунок 3.1).

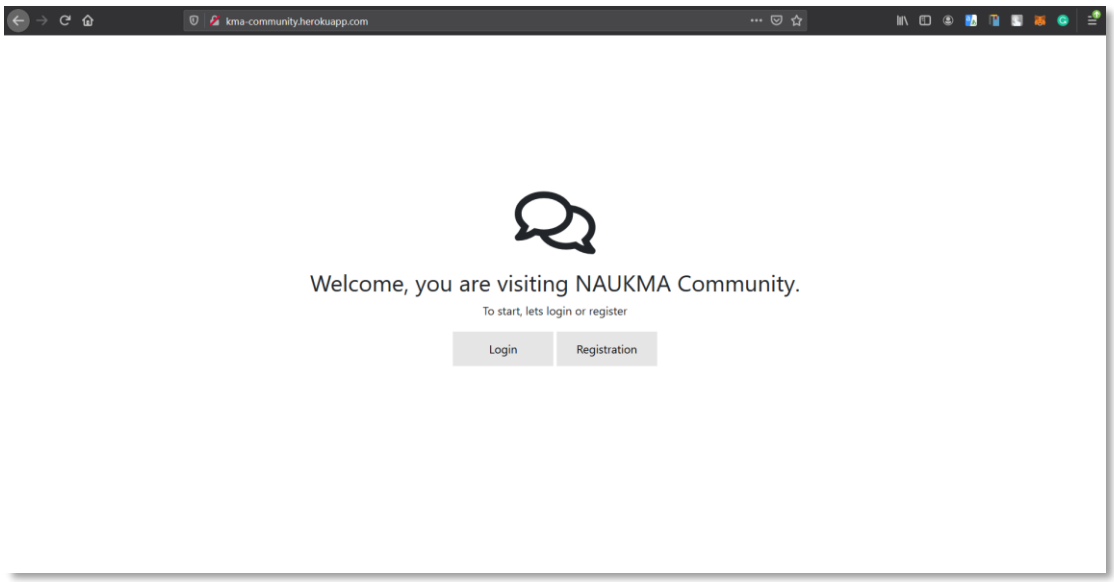


Рисунок 3.1. Головна сторінка

«Система спілкування…» доступна лише для зареєстрованих користувачів тому гостю надається можливість створити обліковий запис (рисунок 3.2) або використати вже існуючий (рисунок 3.3).

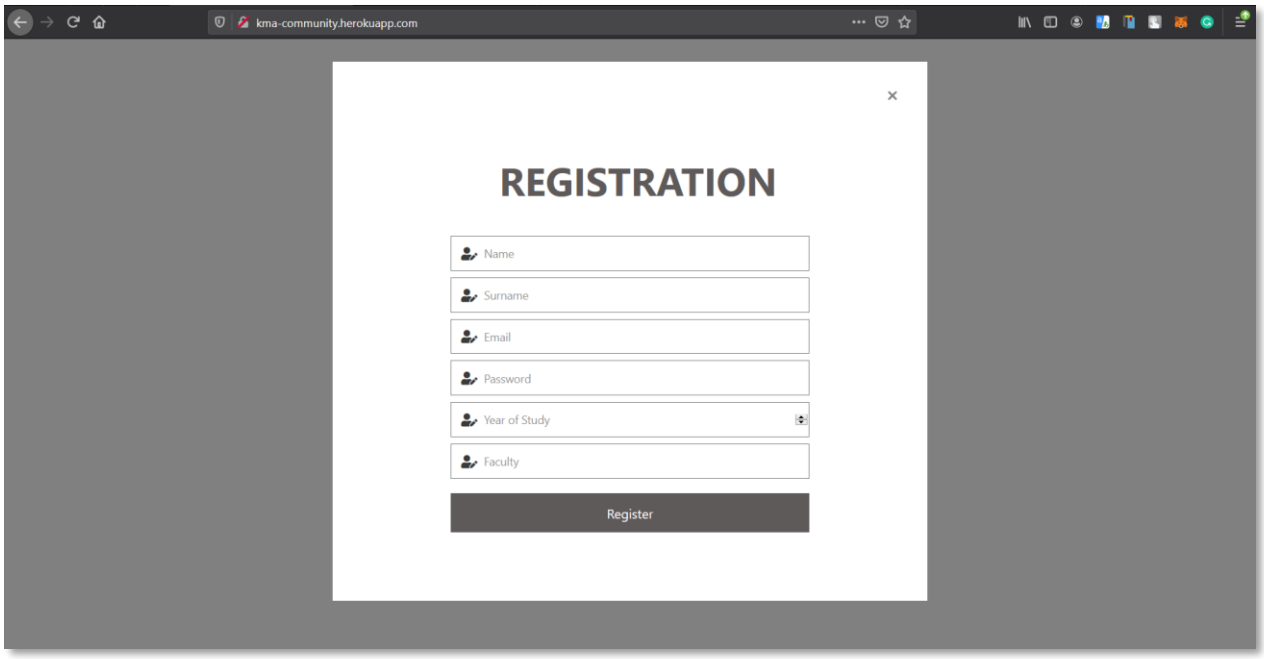


Рисунок 3.2. Сторінка реєстрації

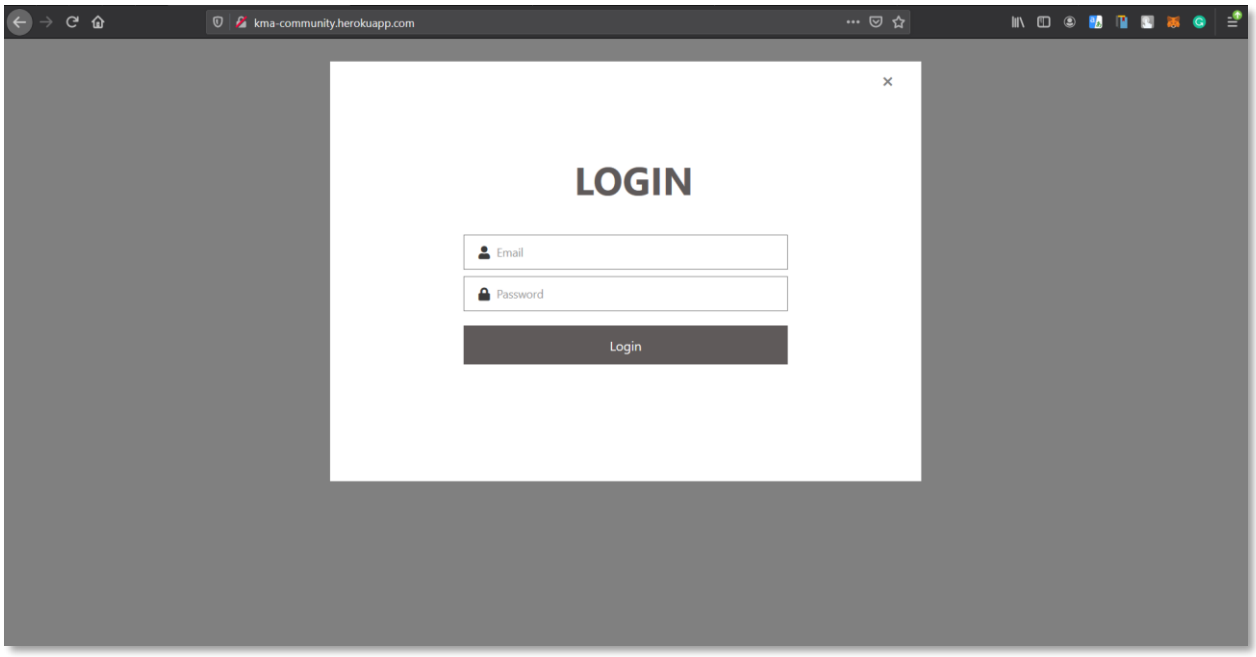


Рисунок 3.3. Сторінка логіну

Після успішного входу до системи, користувач бачить домашню сторінку системи спілкування.

У верхньому лівому куті можна побачити інформацію про залогіненого користувача. Нижче зліва розміщено блок з усіма групами, в яких зареєстрований користувач. Трохи нижче є вікно з усіма доступними групами, в яких користувачі можуть зареєструватися.

Посередині знаходиться власне чат — вікно з повідомленнями та полем для вводу повідомлень (рисунок 3.4).

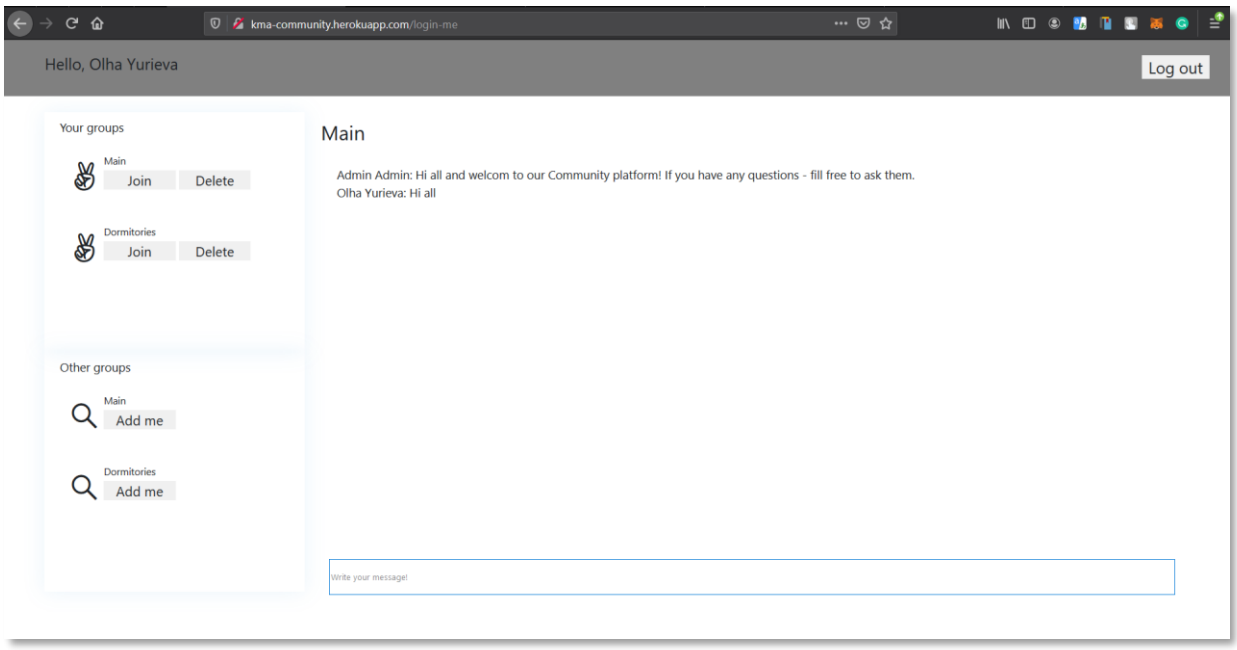


Рисунок 3.4. Сторінка чату

3.3 Висновки

В цьому розділі було описано основну технологію, яка надає можливість створення системи спілкування в режимі реального часу — веб-сокети. Також було розглянуто функціональні можливості створеного веб-застосунку та показано інтерфейс, за допомогою якого користувач може взаємодіяти з системою.

Висновки

Метою даної роботи було дослідження мови програмування Crystal та створення веб-застосунку — «Системи спілкування для університетської спільноти», який дозволить оцінити використання цієї мови програмування на практиці.

Актуальність вибраної предметної області обумовлена тим, що існуючі аналоги характеризуються багатофункціональністю, яка у випадку поставлених нами цілей є й головним їх недоліком. Вона не дозволяє сконцентруватися на головному, а саме — вирішенні нагальних питань без додаткових затрат часу. «Система спілкування…» надає гарантію кваліфікованої допомоги, сконцентрованості на вирішенні питання та інформування студентів незалежно від їх можливого упередженого ставлення до існуючих месенджерів і соціальних мереж.

Проаналізовано основні переваги та недоліки мови програмування Crystal, яку було використано для розробки веб-застосунку. Серед її переваг слід відзначити зручність та швидкість у роботі. Разом з тим, оскільки це мова програмування, яка лише формується, вона має певні недоліки. Серед них: відсутність компілятора для Windows та маленька спільнота розробників, що поряд з перевагами може ускладнювати написання тієї чи іншої функціональності. Документація є досить вичерпною, але потрібно знати, що шукаєш.

Дуже важливим для створення веб-застосунку був вибір правильних інструментів для розробки. Найважливішими з них стали: веб-фреймворк Kemal та ORM Jennifer. Практика показала, що вони гарно взаємодіють один з одним. Це важливо, адже, наприклад конфлікт версій є нагальною проблемою. З випуском нової версії мови програмування Crystal не всі залежності одразу її підтримують або ж зупиняються на якійсь версії й далі не оновлюються. Тому важливо враховувати статус проекту перед його використанням для своєї задачі.

Наведено функціональні можливості створеного веб-застосування та описано інтерфейс, за допомогою якого користувач може взаємодіяти з «Система спілкування…».

Список використаної літератури

1. Програмування на мові Python (3.x). Початковий курс [Електронний ресурс] // Google Sites. — Режим доступу: https://sites.google.com/site/pythonukr/urok-1-istoria-mov-programuvanna-kompilacia-ta-interpretacia. — Accessed 17 квітня 2020 р.

2. The Crystal Programming Language [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/crystal-lang/crystal. — Accessed 17 квітня 2020 р.

3. Compiled Versus Interpreted Languages [Електронний ресурс] // freeCodeCamp. — Режим доступу: https://guide.freecodecamp.org/computer-science/compiled-versus-interpreted-languages/. — Accessed 17 квітня 2020 р.

4. Just in Time Compilation [Електронний ресурс] // freeCodeCamp. — Режим доступу: https://guide.freecodecamp.org/computer-science/just-in-time-compilation/. — Accessed 17 квітня 2020 р.

5. What is the Difference Between Machine Code and Bytecode [Електронний ресурс] / Lithmee // Pediaa.Com. — 17 жовтня, 2018 р. — Режим доступу: https://pediaa.com/what-is-the-difference-between-machine-code-and-bytecode/.

6. Balbaert I. Programming Crystal [Електронний ресурс] / I. Balbaert, S. St. Laurent // O'Reilly. — Режим доступу: https://learning.oreilly.com/library/view/Programming+Crystal/9781680506631/f_0032.xhtml#fibers. — Accessed 17 квітня 2020 р.

7. Concurrency vs. Parallelism [Електронний ресурс] / Gupta L // HowToDoInjava. — Режим доступу: https://howtodoinjava.com/java/multi-threading/concurrency-vs-parallelism/. — Accessed 17 квітня 2020 р.

8. Parallelism in Crystal [Електронний ресурс] / Juan Wajnerman, Brian J. Cardiff // CRYSTAL. — 6 вересня, 2019 р. — Режим доступу: https://crystal-lang.org/2019/09/06/parallelism-in-crystal.html.

9. Shards [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/crystal-lang/shards. — Accessed 17 квітня 2020 р.

10. Travis CI [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/travis-ci. — Accessed 17 квітня 2020 р.

11. Резванов С. Что такое travis-ci.org и с чем его едят? [Електронний ресурс] // ХАБР. — Режим доступу: https://habr.com/ru/post/140344/. — Accessed 17 квітня 2020 р.

12. Sentry [Електронний ресурс] // GitHub. — Режим доступу: https://github.com/getsentry/. — Accessed 17 квітня 2020 р.

13. Муллагалиев В. Sentry — мониторинг ошибок в Django [Електронний ресурс] // ХАБР. — Режим доступу: https://habr.com/ru/post/111283/. — Accessed 17 квітня 2020 р.

14. Kemal [Електронний ресурс] / Serdar Dogruyol // GitHub. — Режим доступу: https://github.com/sdogruyol/kemal-blog/blob/master/src/kemal-blog.cr. — Accessed 17 квітня 2020 р.

15. Павлов Е. Active Record Pattern [Електронний ресурс] // ХАБР. — Режим доступу: https://habr.com/ru/post/155545/. — Accessed 17 квітня 2020 р.

16. Query DSL [Електронний ресурс] / Roman Kalnytskyi // Imdrasil Homebrew Stuff. — Режим доступу: https://imdrasil.github.io/jennifer.cr/docs/query_dsl. — Accessed 17 квітня 2020 р.

17. SQL Joins [Електронний ресурс] // w3schools.com. — Режим доступу: https://www.w3schools.com/sql/sql_join.asp. — Accessed 17 квітня 2020 р.

18. SQL Cross Join [Електронний ресурс] // w3resource. — Режим доступу: https://www.w3resource.com/sql/joins/cross-join.php. — Accessed 17 квітня 2020 р.

19. Callbacks [Електронний ресурс] / Roman Kalnytskyi // Imdrasil Homebrew Stuff. — Режим доступу: https://imdrasil.github.io/jennifer.cr/docs/callbacks. — Accessed 17 квітня 2020 р.

20. Callback [Електронний ресурс] // Stack Overflow. — Режим доступу: https://stackoverflow.com/questions/824234/what-is-a-callback-function. — Accessed 17 квітня 2020 р.

21. Relations [Електронний ресурс] / Roman Kalnytskyi // Imdrasil Homebrew Stuff. — Режим доступу: https://imdrasil.github.io/jennifer.cr/docs/relations. — Accessed 17 квітня 2020 р.

22. Butler B. PaaS Primer: What is platform as a service and why does it matter? [Електронний ресурс] // Network World. — 2013.— Режим доступу до журн.: https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html.

23 .Mernik M. When and How to Develop Domain-Specific Languages / M. Mernik, J. Heering, A.M. Sloane // ACM Computing Surveys. — 2005. — № 37(4). — С. 316–344. — Режим доступу до журн.: https://www.researchgate.net/publication/200040446_When_and_How_to_Develop_Domain-Specific_Languages.

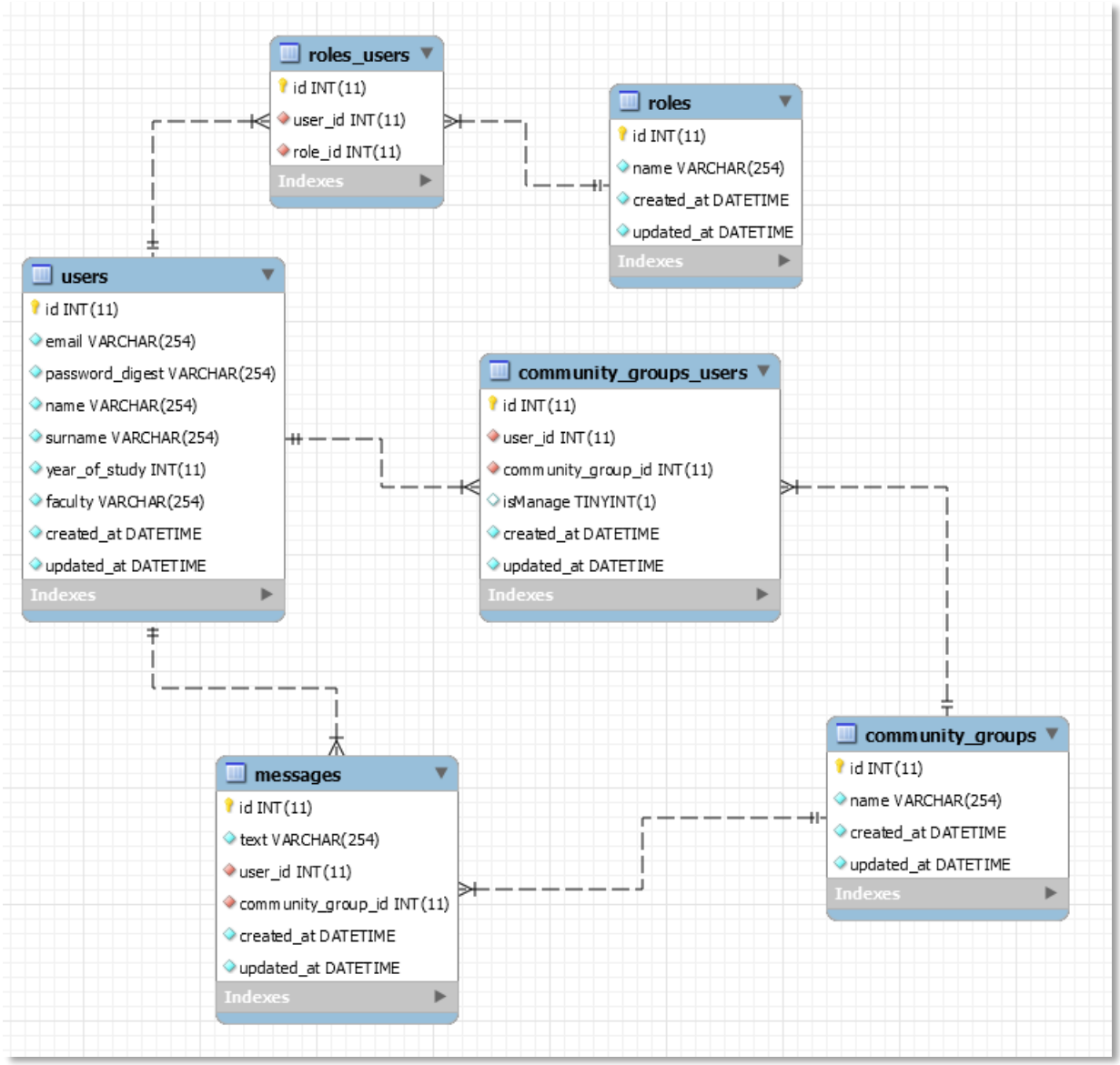
24. Active Record Associations [Електронний ресурс] // Rails Guides. — Режим доступу: https://guides.rubyonrails.org/association_basics.html. — Accessed 17 квітня 2020 р.

25. What are Web Sockets? [Електронний ресурс] / Tarnowski D // Medium. — 15 лютого, 2017 р. — Режим доступу: https://medium.com/@dominik.t/what-are-web-sockets-what-about-rest-apis-b9c15fd72aac.

26. Simonov S. Классы Socket и ServerSocket, или «Алло, сервер? Ты меня слышишь?» [Електронний ресурс] // JAVARUSH. — Режим доступу: https://javarush.ru/groups/posts/654-klassih-socket-i-serversocket-ili-allo-server-tih-menja-slihshishjh. — Accessed 17 квітня 2020 р.

Додатки

Додаток А. ER-модель бази даних



Додаток Б. Приклад створення та роботи волокон програмування

```
chan = Channel(String).new
num = 10000
num.times do |i|
  spawn do
    chan.send "fiber #{i}: I like crystals!"
  end
  puts chan.receive
end

# =>
# fiber 0: I like crystals!
# fiber 1: I like crystals!
# fiber 2: I like crystals!
# ...
# fiber 9999: I like crystals!
```

Додаток В. Багатопоточна схема роботи програми на Crystal