Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №3

із дисципліни «Технології Розробки Програмного Забезпечення»

Тема: «ДІАГРАМА РОЗГОРТАННЯ. ДІАГРАМА КОМПОНЕНТІВ. ДІАГРАМА ВЗАЄМОДІЙ ТА ПОСЛІДОВНОСТЕЙ»

Варіант-11

Виконав:
Студент групи ІА-24
Коханчук Михайло Миколайович

Перевірив:
Мягкий Михайло Юрійович

Київ-2024

ЗМІСТ

Лабораторна робота №3 1
МЕТА 3
Теоритичні відомості: 4
СТРУКТУРНІ ДІАГРАМИ 4
ПОВЕДІНКОВІ ДІАГРАМИ 18
ДІАГРАМИ ВЗАЄМОДІЇ 25
Завдання: 33
Хід роботи: 33
Рис. 1. Діаграма розгортання для JSON Tool 33
Рис. 2. Діаграма компонентів реалізованої частини системи 34
Рис. 3. Діаграма компонентів послідовностей частини системи 37
ВИСНОВОК 39

МЕТА

Метою роботи є проєктування системи шляхом розробки діаграми розгортання, діаграми компонентів та діаграми послідовностей, а також підготовка звіту про виконану роботу відповідно варіанту(JSON Tool).

Теоритичні відомості:

UML (Unified Modeling Language) — це стандартизована мова для візуального моделювання, опису та документування програмного забезпечення, бізнес-процесів чи будь-яких інших систем. UML допомагає розробникам, аналітикам, тестувальникам та іншим учасникам проекту спілкуватися на єдиній мові.

Основні елементи UML:

Актори – зовнішні системи чи користувачі, які взаємодіють із системою.
Класи та об'єкти – описують статичну структуру системи.
Ассоціації, залежності та зв'язки – вказують на взаємодії між елементами.
Активності та стани – описують динамічну поведінку.
Компоненти та вузли – вказують на фізичні аспекти системи.

UML включає 14 типів діаграм, які діляться на *структурні* та *поведінкові*.

СТРУКТУРНІ ДІАГРАМИ

Діаграма компонентів (Component Diagram)

Один із типів структурних діаграм у Unified Modeling Language (UML), який показує, як компоненти системи організовані та взаємодіють один з одним, називається діаграмою на основі компонентів або просто компонентною діаграмою.
Компоненти системи — це модульні одиниці, які надають набір інтерфейсів і інкапсулюють реалізацію. Ці діаграми ілюструють, як компоненти з'єднуються для формування більших систем, детально описуючи їх залежності та взаємодії.
Компонентні діаграми широко використовуються у проєктуванні систем для сприяння модульності та покращення розуміння архітектури системи.
Компонентні діаграми в UML складаються з кількох ключових елементів, кожен із яких відіграє особливу роль у відображенні архітектури системи. Ось основні компоненти та їхні ролі:

1. Компонент
Представляє модульні частини системи, які інкапсулюють функціональність. Компонентами можуть бути програмні класи, колекції класів або підсистеми.
Символ: Прямокутники з позначенням стереотипу («component»).
Функція: Визначають і інкапсулюють функціональність, забезпечуючи модульність і повторне використання.

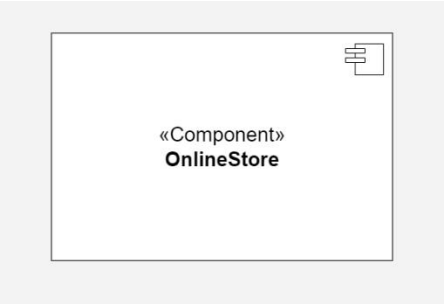


Рис.1.1.1 Компонент

2. Інтерфейси
Визначають набір операцій, які компонент пропонує або потребує, слугуючи контрактом між компонентом і його середовищем.

Символ: Коло (лінія з "льодяником") для запропонованих інтерфейсів і півколо (гніздо) для потрібних інтерфейсів.
Функція: Визначають, як компоненти взаємодіють між собою, забезпечуючи можливість їх незалежної розробки та підтримки.

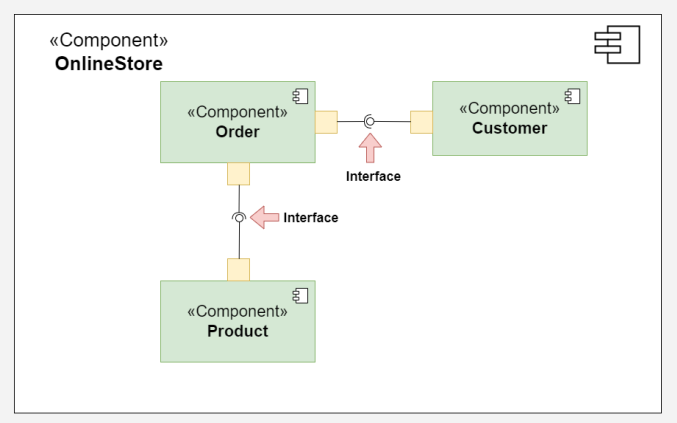


Рис.1.1.2 Інтерфейси

3. Зв'язки
Відображають з'єднання та залежності між компонентами та інтерфейсами.
Символи: Лінії та стрілки.

- Залежність (пунктирна стрілка): Вказує, що один компонент залежить від іншого.
- Асоціація (суцільна лінія): Показує більш постійний зв'язок між компонентами.
- З'єднувач складання: З'єднує потрібний інтерфейс одного компонента із запропонованим інтерфейсом іншого.
- Функція: Візуалізує, як компоненти взаємодіють і залежать один від одного, підкреслюючи шляхи комунікації та потенційні точки відмови.

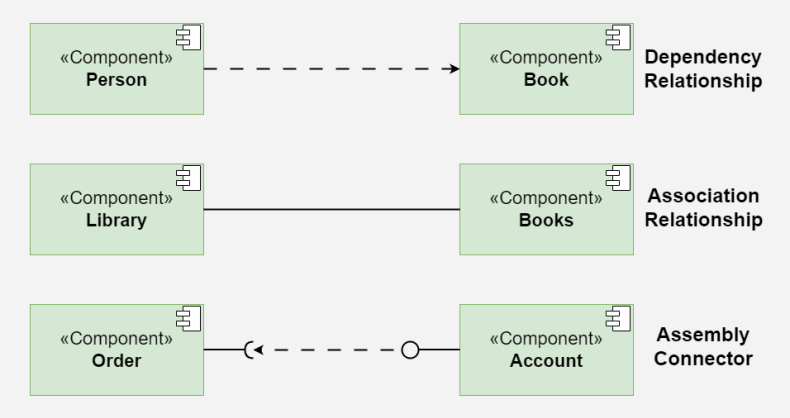


Рис.1.1.3 Типи зв'язків діаграми компонентів

4. Ports
Роль: Представляють конкретні точки взаємодії на межі компонента, де надаються або потребуються інтерфейси.
Символ: Маленькі квадрати на межі компонента.
Функція: Дозволяють більш точно визначати точки взаємодії, сприяючи детальному проєктуванню та реалізації.

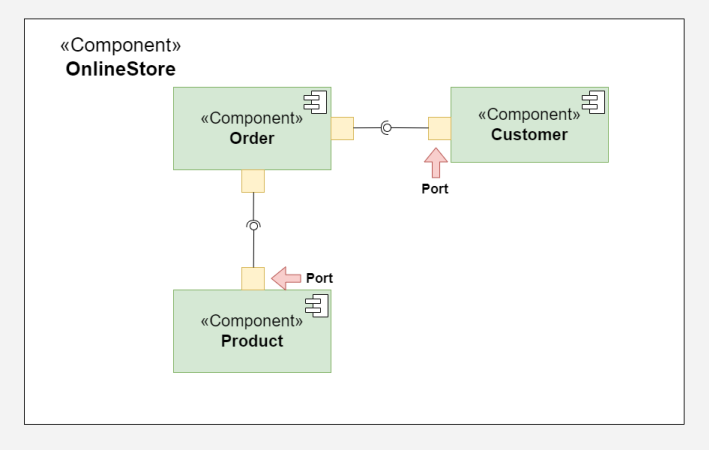


Рис.1.1.4 Порти

5. Artifacts
Представляють фізичні файли або дані, що розгортаються на вузлах.
Символ: Прямокутники з позначенням стереотипу («artifact»).
Функція: Показують, як програмні артефакти, такі як виконувані файли або файли даних, пов'язані з компонентами.

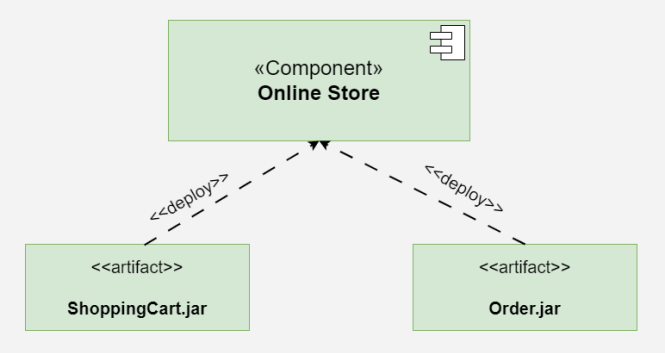


Рис.1.1.5 Порти

6. Nodes
Представляють фізичні або віртуальні середовища виконання, де розгортаються компоненти.
Символ: 3D коробки.
Функція: Забезпечують контекст для розгортання, показуючи, де компоненти знаходяться та виконуються в інфраструктурі системи.

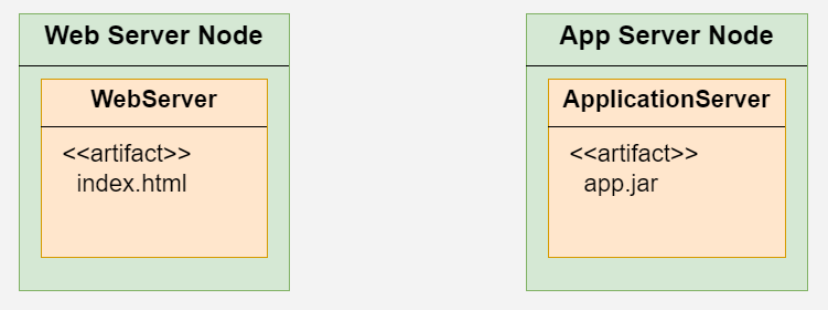


Рис.1.1.6 Порти

Діаграма класів (Class Diagram)

Діаграма забезпечує базове позначення для інших структурних діаграм, передбачених UML.
Корисна для розробників та інших членів команди.
Бізнес-аналітики можуть використовувати діаграми класів для моделювання систем з бізнесової точки зору.

Діаграма класів UML складається з:

- Набору класів
- Набору зв’язків між класами

Клас - опис групи об’єктів, усі з подібними ролями в системі, що складається з:

- **Структурних характеристик (атрибутів):**
  Визначають, що об’єкти класу "знають".
  Представляють стан об’єкта класу.
  Є описами структурних або статичних характеристик класу.
- **Характеристик поведінки (операцій):**
  Визначають, що об’єкти класу "можуть робити".
  Описують спосіб взаємодії об’єктів.
  Операції є описами поведінкових або динамічних характеристик класу.

Позначення класу складається з трьох частин:

- **Назва класу**
  Ім'я класу відображається у першій частині.
- **Атрибути класу**
  Атрибути показуються у другій частині.
  Тип атрибуту відображається після двокрапки.
  Атрибути відповідають змінним-членам (даним) у коді.
- **Операції класу (методи)**
  Операції показуються у третій частині. Вони є сервісами, які надає клас.
  Тип повернення методу вказується після двокрапки в кінці підпису методу.
  Тип повернення параметрів методу вказується після двокрапки після імені параметра.
  Операції відповідають методам класу в коді.

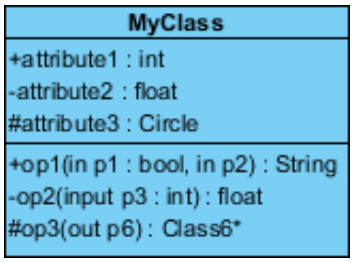


Рис.1.2.1 Клас з атрибутами та методами

Клас може бути залучений в одне або кілька зв'язків з іншими класами. Зв'язок може бути одним із наступних типів:

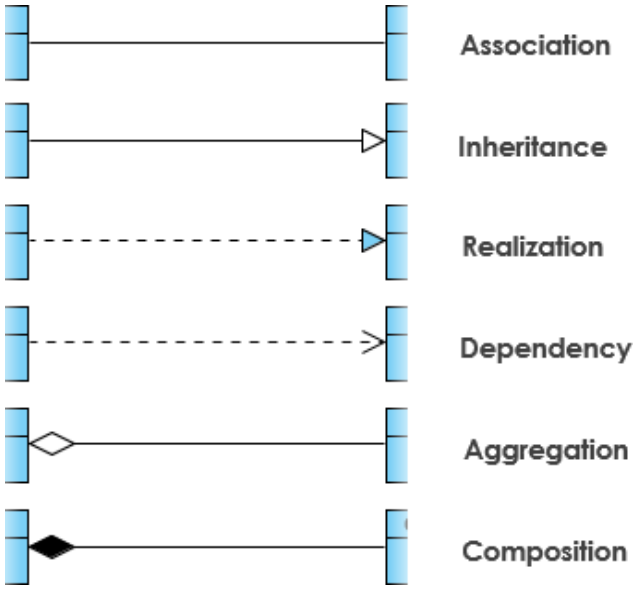


Рис.1.2.2 Типи зв’язків діаграми класів

Асоціації — це зв’язки між класами в UML-діаграмі класів. Вони зображаються у вигляді суцільної лінії між класами. Асоціації зазвичай називаються за допомогою дієслова або дієслівної фрази, яка відображає реальну предметну область.
Проста асоціація – це структурний зв’язок між двома рівноправними класами.
Наприклад, існує асоціація між Class1 і Class2.

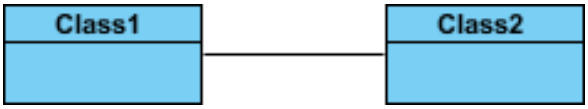


Рис.1.2.3 Проста асоціація

Кардинальність у UML визначає кількість об’єктів, які можуть бути пов’язані між собою через асоціацію. Вона виражається у вигляді:

- "Один до одного" (one to one)
- "Один до багатьох" (one to many)
- "Багато до багатьох" (many to many)

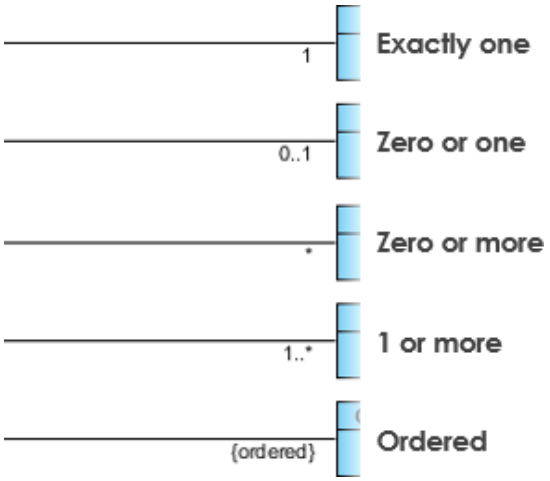


Рис.1.2.4 Кардинальність

Наслідування (Узагальнення) — це таксономічний зв’язок між більш загальним класифікатором і більш специфічним класифікатором. Кожен екземпляр специфічного класифікатора також є непрямим екземпляром загального класифікатора. Таким чином, специфічний класифікатор успадковує характеристики більш загального класифікатора.
Представляє відношення "is-a". Назва абстрактного класу позначається курсивом.
На рисунку нижче показано приклад ієрархії наслідування. SubClass1 і SubClass2 успадковуються від SuperClass. Зв’язок зображено як суцільну лінію з порожньою стрілкою, яка вказує від дочірнього елемента до батьківського.

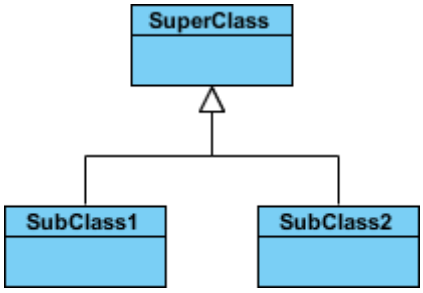


Рис.1.2.5 Приклад наслідування

Реалізація — це відношення між класом- планом і об'єктом, що містить відповідні деталі реалізації. Цей об'єкт вважається таким, що реалізує клас-план. Іншими словами, це відношення між інтерфейсом і класом, що його реалізує.
Наприклад, інтерфейс Owner може визначати методи для придбання та розпорядження власністю. Класи Person і Corporation повинні реалізувати ці методи, можливо, дуже різними способами.

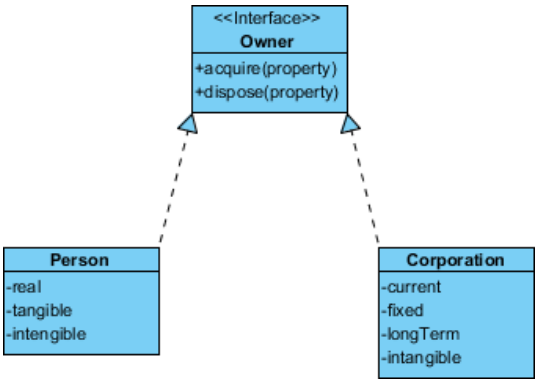


Рис.1.2.6 Приклад реалізації

Залежність
Існує між двома класами, якщо зміни у визначенні одного класу можуть призвести до змін у іншому (але не навпаки).
Class1 залежить від Class2.
Пунктирна лінія з відкритою стрілкою.

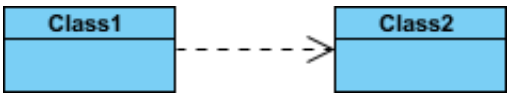
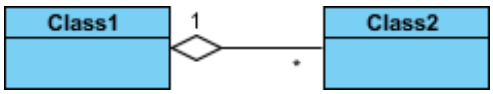


Рис.1.2.7 Приклад залежності

Агрегація - особливий тип асоціації. Вона представляє відношення "частина від".
Class2 є частиною Class1.
Об’єкт може існувати незалежно від іншого об’єкта, який його включає
Об’єкти Class1 і Class2 мають окремі життєві цикли.



Суцільна лінія з порожнім ромбом на кінці асоціації, з’єднаному з класом-композитом.

Композиція - особливий тип агрегації, де частини знищуються разом із цілим.
Об’єкти Class2 існують і припиняють існування разом із Class1.
Class2 не може існувати самостійно.

Суцільна лінія із заповненим ромбом на кінці асоціації, з’єднаному з класом-композитом.

Діаграма об'єктів (Object Diagram)

Об’єктна діаграма показує зв’язок між екземплярами класів і визначеними класами, а також зв’язок між цими об’єктами в системі. Вона може бути корисною для пояснення менших частин вашої системи, коли діаграма класів системи є дуже складною, а також іноді для моделювання рекурсивних зв’язків.

Основні символи та позначення об’єктної діаграми:

Назви об’єктів:

Рис.1.3.1 Кожен об’єкт зображається у вигляді прямокутника, який містить ім’я об’єкта та його клас, підкреслені й розділені двокрапкою.

Атрибути об’єкта:

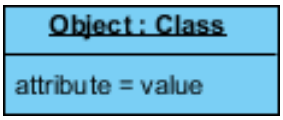


Рис.1.3.2 Подібно до класів, ви можете перерахувати атрибути об’єкта в окремому розділі. Однак, на відміну від класів, атрибути об’єкта повинні мати присвоєні значення.

Зв’язки:

Зв’язки зазвичай є екземплярами асоціацій. Ви можете зображати зв’язки, використовуючи лінії, які застосовуються в діаграмах класів.

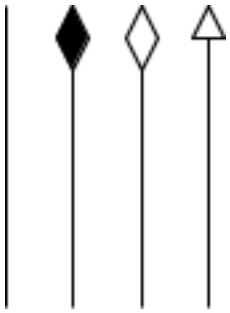


Рис.1.3.3 Типи зв’язків діаграми обєктів еквіваленті зв’язкам діаграми класів

Діаграма розгортання (Deployment Diagram)

Діаграма розгортання показує, як програмний дизайн перетворюється на фактичну фізичну систему, на якій буде виконуватися програмне забезпечення. Вона демонструє, де програмні компоненти розміщуються на апаратних пристроях і як вони взаємодіють між собою. Ця діаграма допомагає візуалізувати, як програмне забезпечення працюватиме на різних пристроях.

Нижче наведено компоненти та їх позначення в діаграмі розгортання:

1. *Component*

   Компонент представляє модульну та багаторазово використовувану частину системи, зазвичай реалізовану як програмний модуль, клас або пакет. Він інкапсулює свою поведінку та дані й може розгортатися незалежно.

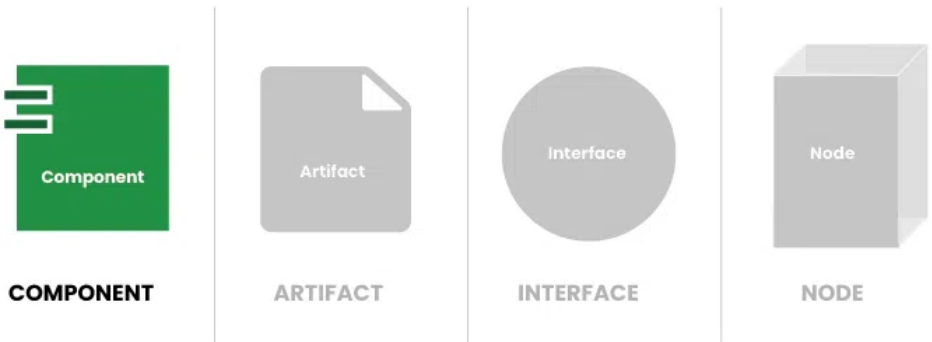


Рис.1.4.1 Зазвичай зображується у вигляді прямокутника з двома меншими прямокутниками, що виступають з його боків, що позначають порти для з'єднань. Назва компонента пишеться всередині прямокутника.

2. *Artifact*

Артефакт представляє фізичний елемент інформації або даних, який використовується чи створюється в процесі розробки програмного забезпечення. Він може включати файли вихідного коду, виконувані файли, документи, бібліотеки, конфігураційні файли або будь-які інші елементи.

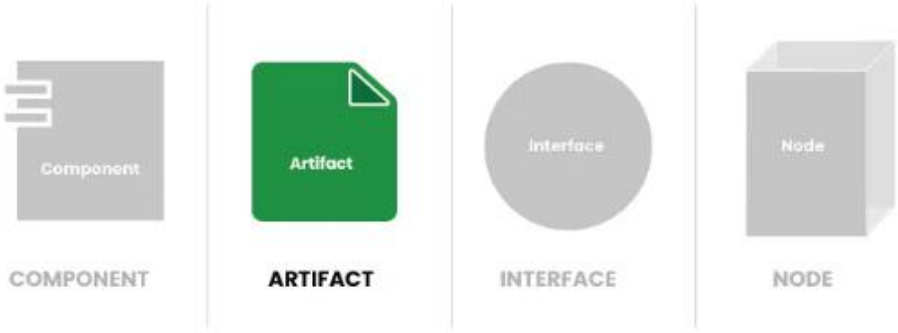


Рис.1.4.2 Зазвичай зображується у вигляді прямокутника із загнутим кутом, позначеного назвою артефакта. Артефакти також можуть включати додаткову інформацію, таку як розширення файлів або версії.

3. *Interface*

Інтерфейс визначає контракт, який задає методи або операції, що компонент повинен реалізувати. Він представляє точку взаємодії між різними компонентами або підсистемами.

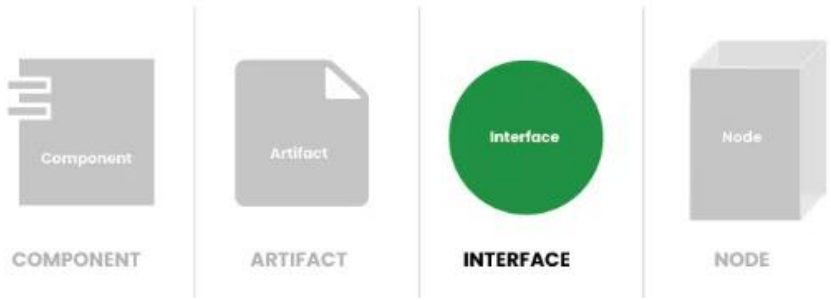


Рис.1.4.3 Зображується у вигляді кола або еліпса, позначеного назвою інтерфейсу. Інтерфейси також можуть включати надані й необхідні інтерфейси, позначені відповідно символами "+" і "-"

4. *Node*

Вузол представляє фізичний або обчислювальний ресурс, наприклад апаратний пристрій, сервер, робочу станцію або обчислювальний ресурс, на якому можуть розгортатися або виконуватися програмні компоненти.

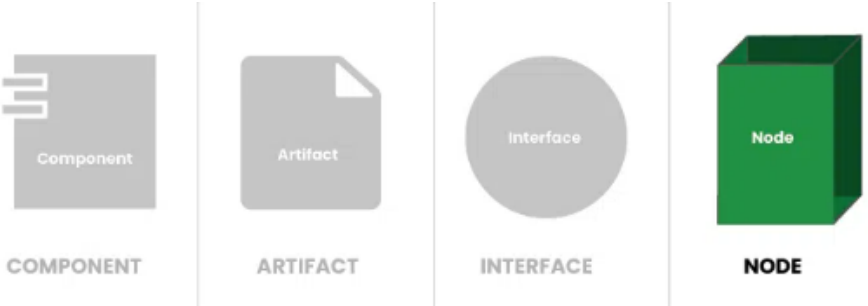


Рис.1.4.4 Зображується у вигляді прямокутника із закругленими кутами, зазвичай позначеного назвою вузла. Вузли також можуть містити вкладені вузли для представлення ієрархічних структур.

5. *Communication path*

Пряма лінія представляє зв’язок між двома вузлами пристроїв. Пунктирні лінії на діаграмах розгортання представляють зв’язки або залежності між елементами, що вказують на те, що один елемент пов’язаний з іншим або залежить від нього.

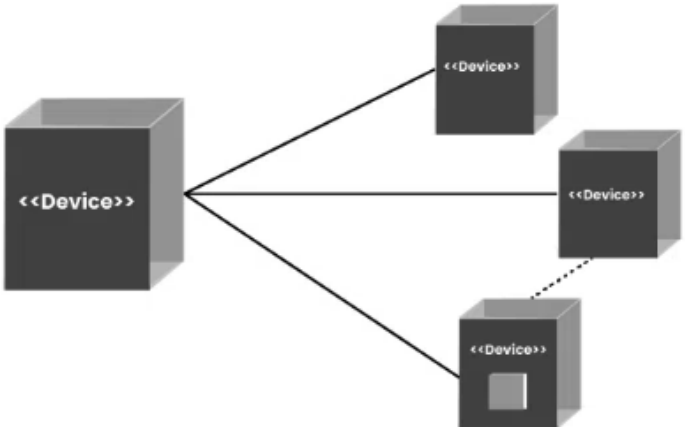


Рис.1.4.5 Communication path

Діаграма пакетів (Package Diagram)

Діаграми пакетів використовуються для структурування елементів високого рівня системи. Пакети застосовуються для організації великих систем, які містять діаграми, документи та інші ключові артефакти.

Діаграма пакетів — це тип структурної діаграми в UML, що організовує і групує пов'язані класи та компоненти в пакети. Вона візуально представляє залежності та взаємозв'язки між цими пакетами, допомагаючи ілюструвати, як різні частини системи взаємодіють між собою.

1. *Package*

Рис.1.5.1 Наведеним вище є позначення простого пакету

2. *Subsystem*

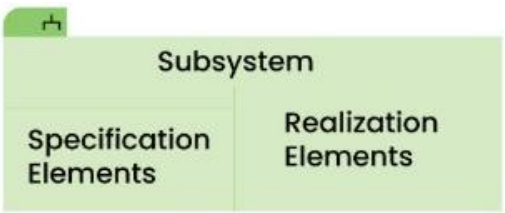


Рис.1.5.2 Наведеним вище є позначення підсистеми

3. *Dependency*

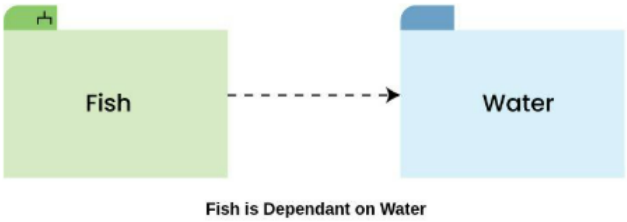


Рис.1.5.3 Пунктирна стрілка використовується для показу залежності між двома елементами або двома пакетами.

4. *Import*

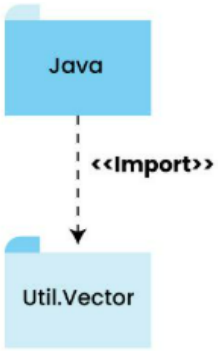


Рис.1.5.4 Наведене вище позначення є позначенням імпорту, де також використовується пунктирна стрілка, але відмінність полягає в тому, що слово <<import>> написано для позначення того, що нижчевказаний пакет, функція або елемент були імпортовані з вищезгаданого пакету.

5. *Merge*

Рис.1.5.5 Наведене позначення вказує на те, що Пакет 1 може бути об'єднаний з Пакетом 2.

Відношення у діаграмі пакетів

1. Відношення злиття
Це відношення використовується для позначення того, що вміст одного пакету можна об'єднати з вмістом іншого пакету. Це означає, що джерело та цільовий пакет мають деякі спільні елементи, що дозволяє їх об'єднати.

Приклад:

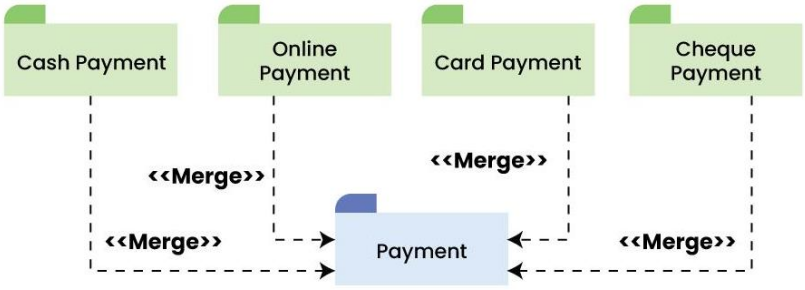


Рис.1.5.6 Наведена діаграма показує, що пакети представляють різні типи платежів, але всі вони є механізмами оплати, тому їх можна об'єднати під загальною назвою "платежі".

2. Відношення залежності
Пакет може залежати від інших різних пакетів, що означає, що джерельний пакет якось залежить від цільового пакету.

Приклад:

Рис.1.5.7 Наведена діаграма показує, що пакет онлайн-платежів залежить від пакету Інтернету і використовує залежність "need".

3. Відношення імпорту
Це відношення використовується для позначення того, що один пакет імпортує інший для використання. Це означає, що пакет-імпортер може отримувати доступ до публічних елементів імпортованого пакету.

Приклад:

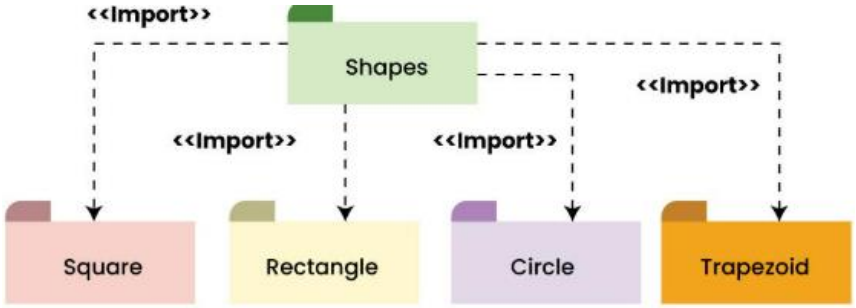


Рис.1.5.8 Наведена діаграма пакетів показує відношення імпорту між основним пакетом Shapes та його підпакетами Square, Rectangle тощо. Всі вони імпортують реальний пакет Shapes, щоб мати змогу використовувати публічні елементи цього пакету.

4. Відношення доступу
Цей тип відношення вказує на те, що між двома або більше пакетами існує відношення доступу, що означає, що один пакет може отримати доступ до вмісту іншого пакету без його імпорту.

Приклад:

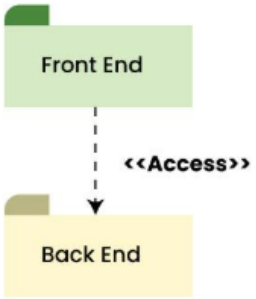


Рис.1.5.9 Наведена діаграма показує відношення доступу між сервісами Front End та Back End. Дуже важливо, щоб сервіс Front End міг безперешкодно отримувати доступ до важливих сервісів Back End для виконання будь-яких операцій.

Діаграма профілю (Profile Diagram)

Діаграма профілю є механізмом розширення, який дозволяє розширювати та налаштовувати UML, додаючи нові будівельні блоки, створюючи нові властивості та визначаючи нову семантику, щоб зробити мову підходящою для вашої конкретної предметної області.
Діаграма профілю має три типи механізмів розширення:

1. Стереотипи
2. Мітки (Tagged Values)
3. Обмеження (Constraints)

Стереотипи

Стереотипи дозволяють розширити словник UML. Ви можете додавати, створювати нові елементи моделі, похідні від існуючих, але з конкретними властивостями, що підходять для вашої предметної області. Стереотипи використовуються для введення нових будівельних блоків, які говорять на мові вашої області та виглядають як примітиви. Вони дозволяють вводити нові графічні символи. Наприклад: при моделюванні мережі вам можуть знадобитися символи для <<маршрутизатор>>, <<комутатор>>, <<хаб>> тощо. Стереотип дозволяє зробити ці об'єкти виглядати як примітиви.

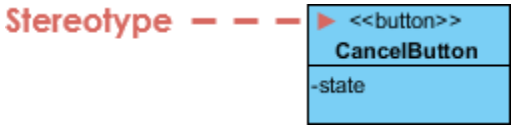


Рис.1.6.1 Приклад стереотипу

Мітки (Tagged Values)

Мітки використовуються для розширення властивостей UML, щоб ви могли додавати додаткову інформацію в специфікацію елемента моделі. Вони дозволяють визначити пари ключ-значення для моделі, де ключі є атрибутами. Мітки графічно зображаються як рядки, оточені дужками.

Наприклад: Розглянемо команду, відповідальну за складання, тестування та розгортання системи. У такому випадку необхідно відстежувати версію та результати тестування основного підсистеми. Мітки використовуються для додавання такої інформації.

Мітки можуть бути корисними для додавання властивостей до моделі для різних корисних цілей:

- Генерація коду
- Керування версіями
- Управління конфігурацією
- Авторство

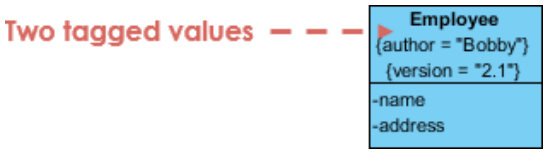


Рис.1.6.2 Приклад мітки

Обмеження (Constraints)

Обмеження — це властивості, які використовуються для визначення семантики або умов, які повинні бути дійсними в будь-який момент часу. Вони дозволяють розширити семантику будівельних блоків UML шляхом додавання нових протоколів. Графічно обмеження зображуються як рядок, оточений дужками, розташований поруч з відповідним елементом.
Наприклад: У розробці реальної системи в реальному часі необхідно додати до моделі деяку необхідну інформацію, таку як час відповіді. Обмеження визначає зв'язок між елементами моделі, який повинен використовувати {subset} або {xor}. Обмеження можуть застосовуватись до атрибутів, похідних атрибутів і асоціацій. Вони також можуть бути прикріплені до одного або кількох елементів моделі, що показуються як примітка.

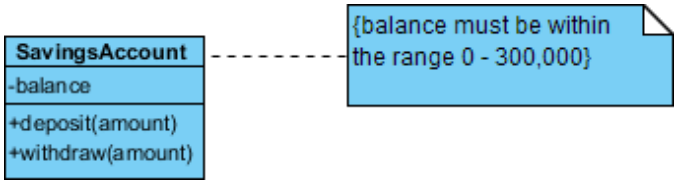


Рис.1.6.3 Приклад обмеження

Діаграма композитної структури (Composite Structure Diagram)

Діаграми композитної структури дозволяють користувачам "зазирнути всередину" об'єкта, щоб побачити, з чого він складається.
Внутрішні дії класу, включаючи зв'язки між вкладеними класами, можуть бути детально представлені. Об'єкти показуються як складені з інших класифікованих об'єктів.

Діаграми складної структури показують внутрішні частини класу.Частини мають імена: partName:partType. Агреговані класи є частинами класу, але частини не обов'язково є класами, частина — це будь-який елемент, який використовується для створення вмістового класу.

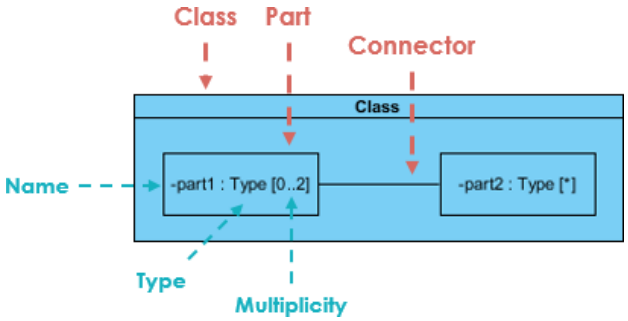


Рис.1.7.1 Приклад діаграми композитної структури

Основні концепції діаграми складної структури
Ключовими сутностями складної структури, визначеними в специфікації UML 2.0, є структуровані класифікатори, частини, порти, з'єднувачі та співпраці.

Колаборація(Collaboration)
Колаборація описує структуру співпрацюючих частин (ролей). Колаборація приєднується до операції або класифікатора через Collaboration Use. Використовуєте колаборацію, коли хочете визначити лише ролі та з'єднання, необхідні для досягнення конкретної мети співпраці.
Наприклад, метою колаборації може бути визначення ролей або компонентів класифікатора. Визначивши основні ролі, колаборація спрощує структуру та уточнює поведінку в моделі.

У цьому прикладі Колеса та Двигун є частинами колаборації, а Передня вісь та Задня вісь є з'єднувачами. Автомобіль є складною структурою, яка показує частини та з'єднання між частинами:

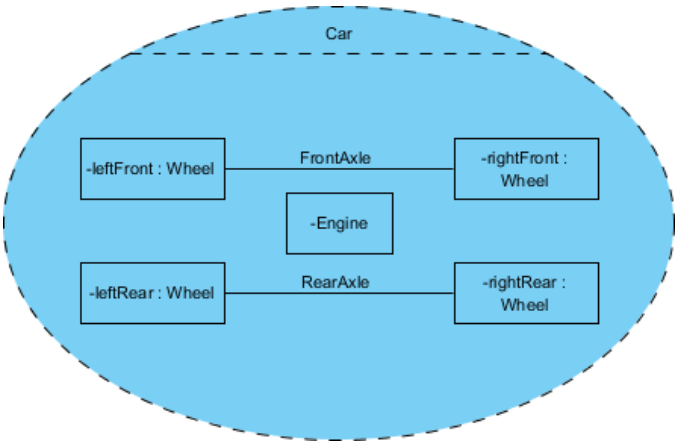


Рис.1.7.2 Приклад свівпраці

Частини(Parts)
Частина — це елемент діаграми, який представляє набір з однієї або більше інстанцій, що належать структурованому класифікатору. Частина описує роль інстанції в класифікаторі. Ви можете створювати частини в структурному відсіку класифікатора, а також у різних діаграмах UML, таких як діаграми складної структури, класів, об'єктів, компонентів, розгортання та пакунків.

Порт
Порт визначає точку взаємодії між інстанцією класифікатора та її середовищем або між поведінкою класифікатора та його внутрішніми частинами.

Інтерфейс
Діаграма складної структури підтримує позначення м'ячика та розетки для наданих і вимогливих інтерфейсів. Інтерфейси можуть бути показані або приховані в діаграмі за потребою.

З'єднувач(Connector)
Лінія, яка представляє відносини в моделі. Коли ви моделюєте внутрішню структуру класифікатора, можна використовувати з'єднувач для позначення зв'язку між двома або більше інстанціями частини або порту. З'єднувач визначає відношення між об'єктами або інстанціями, які прив'язані до ролей у тому ж структурованому класифікаторі, і він ідентифікує комунікацію між цими ролями. Програмне забезпечення автоматично визначає, який тип з'єднувача потрібно створити.

ПОВЕДІНКОВІ ДІАГРАМИ

Діаграма випадків використання (Use Case Diagram)

Діаграми випадків використання зазвичай розробляються на ранній стадії розробки, і їх часто використовують для наступних цілей:

- Визначення контексту системи
- Збір вимог до системи
- Перевірка архітектури системи
- Управління реалізацією та створення тестових випадків
- Розробка аналітиками разом з експертами в предметній області

Основні елементи нотації діаграми випадків використання:

Актор

- Особа, яка взаємодіє з випадком використання (функцією системи).
- Називається іменником.
- Актор відіграє роль у бізнесі.
- Актор ініціює випадок використання.
- Актор має відповідальність перед системою (вхідні дані), а також має очікування від системи (вихідні дані).

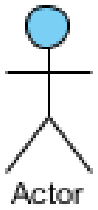


Рис.2.1.1 Актор

Випадок використання(прецедент)

- Функція системи (процес — автоматизований або ручний).
- Називається дієсловом + іменник (або іменникова фраза).
- Кожен актор повинен бути пов'язаний з випадком використання, хоча деякі випадки використання можуть не бути пов'язані з акторами.

Рис.2.1.2 Прецедент

Зв'язок(Communication Link)

- Участь актора у випадку використання показується шляхом з'єднання актора з випадком використання за допомогою суцільного зв'язку.
- Актори можуть бути з'єднані з випадками використання через асоціації, що вказують на те, що актор та випадок використання взаємодіють між собою за допомогою повідомлень.

Рис.2.1.3 Зв'язок(відношення)

Межі системи

- Межа системи потенційно охоплює всю систему, як це визначено в документі вимог.
- Для великих і складних систем кожен модуль може бути межами системи.
- Наприклад, для ERP-системи організації кожен з модулів, таких як персонал, заробітна плата, бухгалтерія тощо, може формувати межу системи для випадків використання, що стосуються кожної з цих бізнес-функцій.
- Вся система може охоплювати всі ці модулі, зображуючи загальні межі системи.

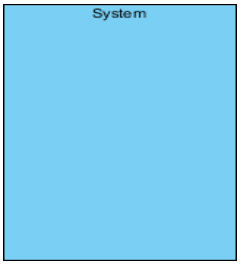


Рис.2.1.4 Межа системи

Взаємозв'язки між випадками використання

Випадки використання мають різні типи взаємозв'язків. Визначення взаємозв'язку між двома випадками використання є завданням аналітиків програмного забезпечення для діаграми випадків використання. Взаємозв'язок між двома випадками використання за суттю моделює залежність між ними. Повторне використання існуючого випадку використання за допомогою різних типів взаємозв'язків зменшує загальні витрати на розробку системи. Взаємозв'язки між випадками використання можна класифікувати наступним чином:

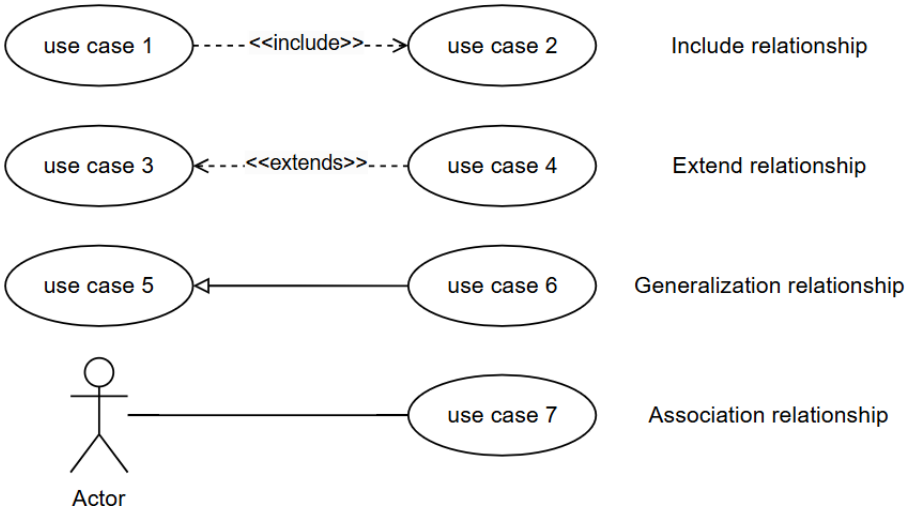


Рис.2.1.5 Типи зв'язків діаграми випадків використання

Зв'язок типу «***include***» вказує на те, що один прецедент включає інший. Це означає, що включений прецедент є частиною основного прецеденту і є необхідним для його виконання. Зв'язок «include» часто використовується для відображення спільного або повторно використовуваного функціоналу. Наприклад, прецедент «Здійснити платіж» може включати прецедент «Автентифікація користувача».

Зв'язок типу «***extend***» представляє необов'язкову або умовну поведінку, яка може розширювати функціонал базового прецеденту за певних умов. Він вказує на те, що розширюючий прецедент може додати додаткову поведінку до базового прецеденту, якщо виконуються певні умови. Наприклад, прецедент «Обробка замовлення» може бути розширений прецедентом «Застосування знижки», якщо користувач має право на знижку.

У UML зв'язок «***generalization***» представляє наслідування. Коли один прецедент узагальнює інший, це означає, що узагальнений прецедент слугує суперкласом, а прецедент, який узагальнює, є підкласом, що успадковує його поведінку. Цей зв'язок часто використовується для відображення того, як більш конкретний прецедент успадковує характеристики від більш загального.

Зв'язок «***association***» використовується для позначення того, що два або більше прецеденти пов'язані або асоційовані між собою певним чином. Цей зв'язок не вказує напрямок взаємодії, а лише позначає загальну асоціацію. Наприклад, якщо два прецеденти часто виконуються разом або мають спільні елементи, це можна відобразити за допомогою зв'язку «association».

Діаграма діяльності (Activity Diagram)

Діаграми діяльності описують, як координуються активності для надання сервісу, що може бути на різних рівнях абстракції. Зазвичай, подія має бути досягнута через деякі операції, особливо якщо операція спрямована на виконання кількох різних завдань, що потребують координації, або якщо потрібно показати, як події в одному варіанті використання пов'язані між собою.
Ці діаграми особливо підходять для моделювання того, як група варіантів використання координується для представлення бізнес-процесів.

Нотація діаграми діяльності:
Активність
Використовується для представлення набору дій.

Рис.2.2.1 Активність

Дія
Завдання, яке потрібно виконати.

Рис.2.2.2 Дія

Потік керування
Показує послідовність виконання.

Потік об'єкта
Показує потік об'єкта від однієї активності (або дії) до іншої активності (або дії).

Рис.2.2.3 Потік керування/об'єктів

Початковий вузол
Зображує початок набору дій або активностей.

Рис.2.2.4 Початковий вузол

Кінцевий вузол активності
Завершує всі потоки керування та потоки об'єктів в активності (або дії).

Рис.2.2.5 Кінцевий вузол активності

Об'єктний вузол
Представляє об'єкт, який з'єднаний з набором потоків об'єктів.

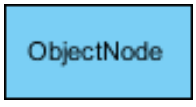


Рис.2.2.6 Об'єктний вузол

Вузол прийняття рішення
Представляє умову тестування, щоб забезпечити, що потік керування або потік об'єктів йде лише одним шляхом.

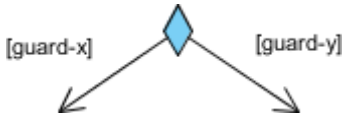


Рис.2.2.7 Вузол прийняття рішення

Вузол злиття

Об'єднує різні шляхи прийняття рішення, які були створені за допомогою вузла прийняття рішення.

Рис.2.2.8 Вузол злиття

Вузол розгалуження

Розділяє поведінку на набір паралельних або одночасних потоків активностей (або дій).

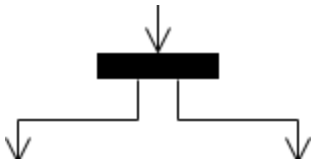


Рис.2.2.9 Вузол розгалуження

Вузол з'єднання

Об'єднує набір паралельних або одночасних потоків активностей (або дій).

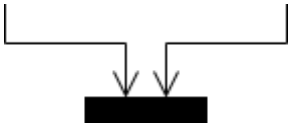


Рис.2.2.10 Вузол з'єднання

Swimlane та розділ

Спосіб групування активностей, виконуваних одним і тим самим актором на діаграмі активностей, або групування активностей в одному потоці.

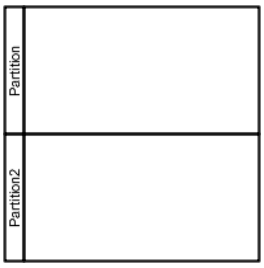


Рис.2.2.11 Swimlane та розділ

Діаграма станів (State Machine Diagram)

Діаграма станів — це діаграма UML, яка використовується для представлення стану системи або її частини у визначені моменти часу. Це поведінкова діаграма, яка описує поведінку системи через переходи між кінцевими станами.

Основні компоненти та позначення діаграми станів

Нижче наведено основні компоненти та їх позначення для діаграми станів:

1. Початковий стан
   Чорний заповнений круг використовується для представлення початкового стану системи або класу.

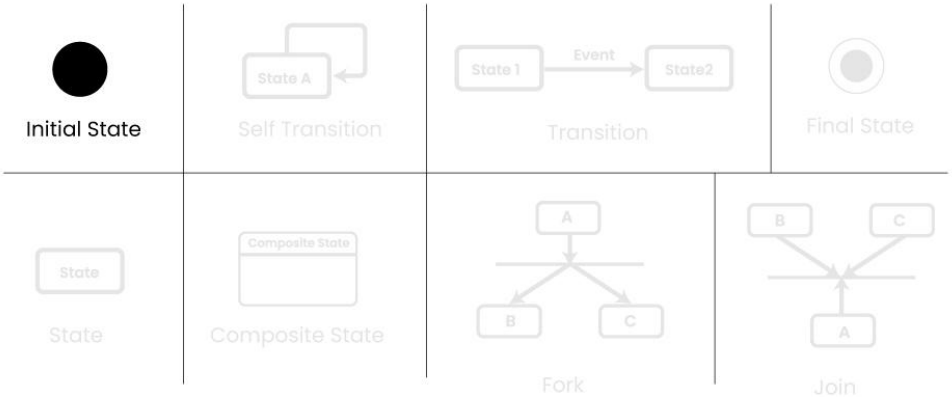


Рис.2.3.1 Початковий стан

2. Перехід
   Ми використовуємо суцільну стрілку для представлення переходу або зміни управління з одного стану в інший. Стрілка позначається подією, яка викликає зміну стану.

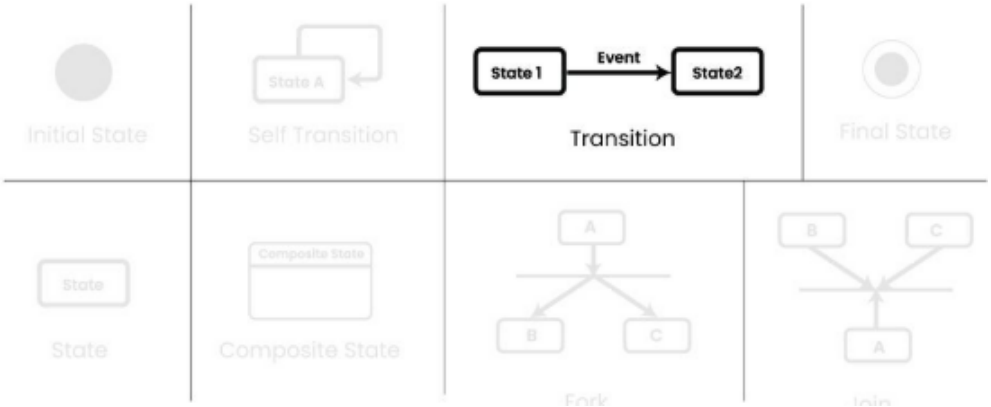


Рис.2.3.2 Перехід

3. Стан
   Ми використовуємо прямокутник із заокругленими кутами для позначення стану. Стан представляє умови або обставини об'єкта класу в певний момент часу.

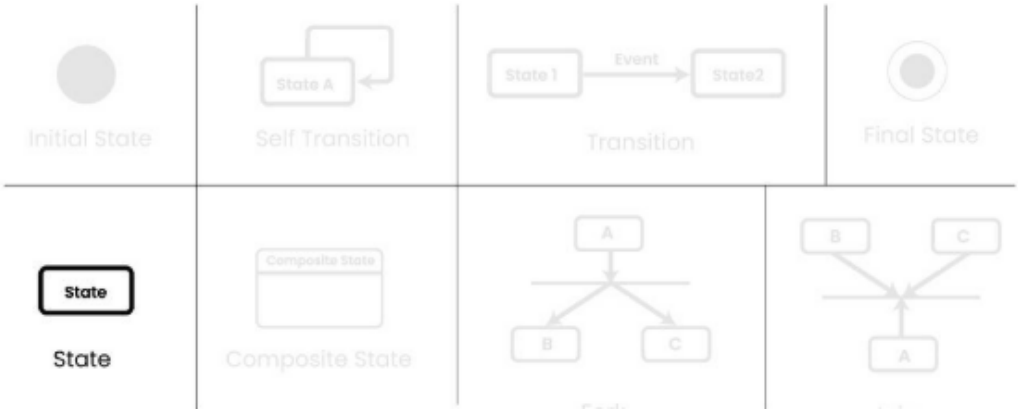


Рис.2.3.3 Стан

4. Розгалуження
   Ми використовуємо заокруглену суцільну прямокутну смужку для позначення розгалуження з вхідною стрілкою від початкового стану та вихідними стрілками до новостворених станів. Розгалуження використовується для позначення поділу стану на два або більше паралельних стани.

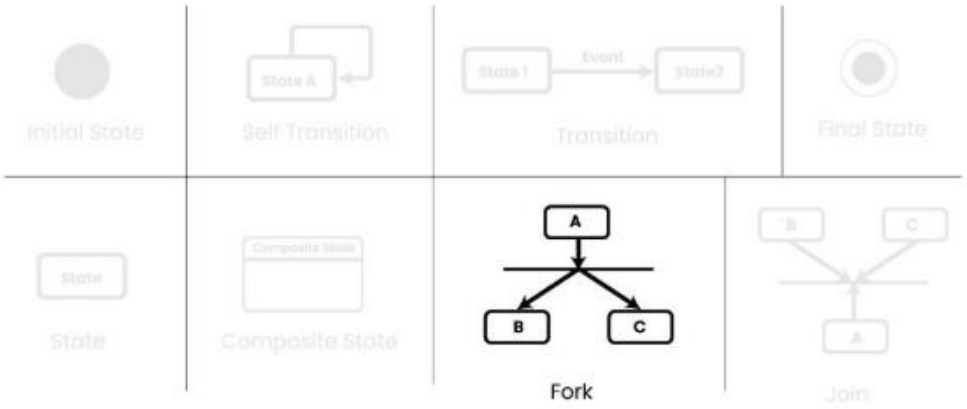


Рис.2.3.4 Розгалуження

5. Злиття
   Ми використовуємо заокруглену суцільну прямокутну смужку для позначення злиття з вхідними стрілками від об'єднуваних станів та вихідною стрілкою до спільного цільового стану. Злиття використовується, коли два або більше стани одночасно сходяться в один при виникненні події або подій.

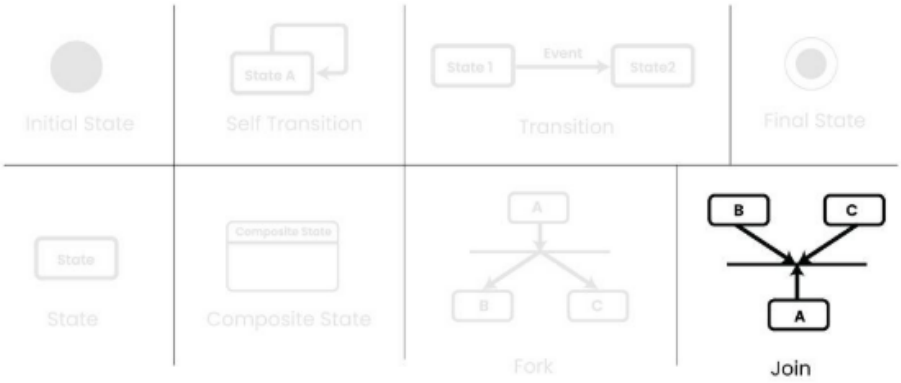


Рис.2.3.5 Злиття

6. Самоперехід
   Ми використовуємо суцільну стрілку, що вказує назад на той самий стан, для позначення самопереходу. Можуть бути ситуації, коли стан об'єкта не змінюється при виникненні події. Самопереходи використовуються для представлення таких випадків.

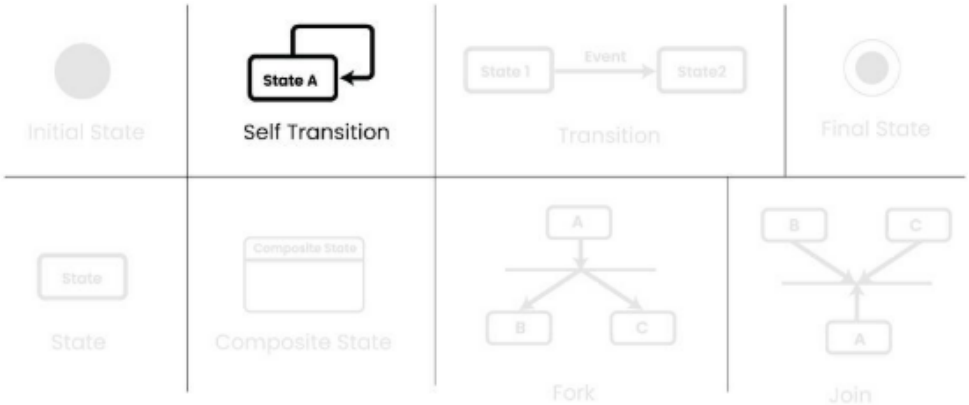


Рис.2.3.6 Самоперехід

7. Складений стан
   Ми використовуємо заокруглений прямокутник для позначення складеного стану. Складений стан використовується для представлення стану з внутрішніми діями.

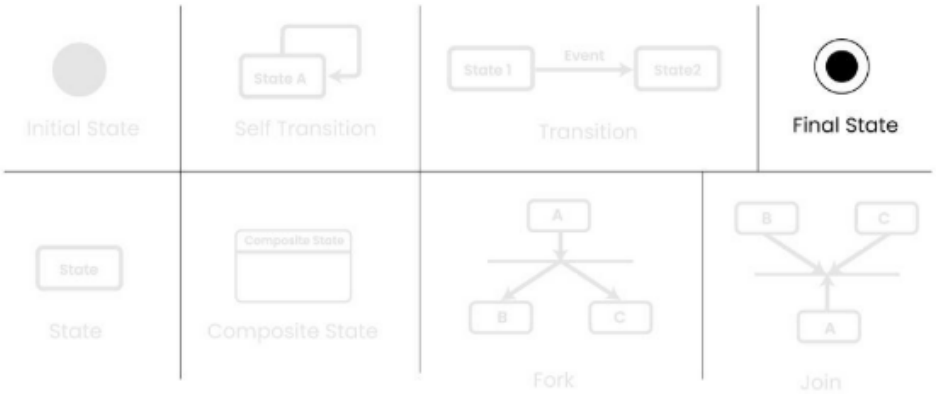


Рис.2.3.7 Складений стан

8. Кінцевий стан
   Ми використовуємо позначення заповненого кола всередині іншого кола для представлення кінцевого стану в діаграмі станів.

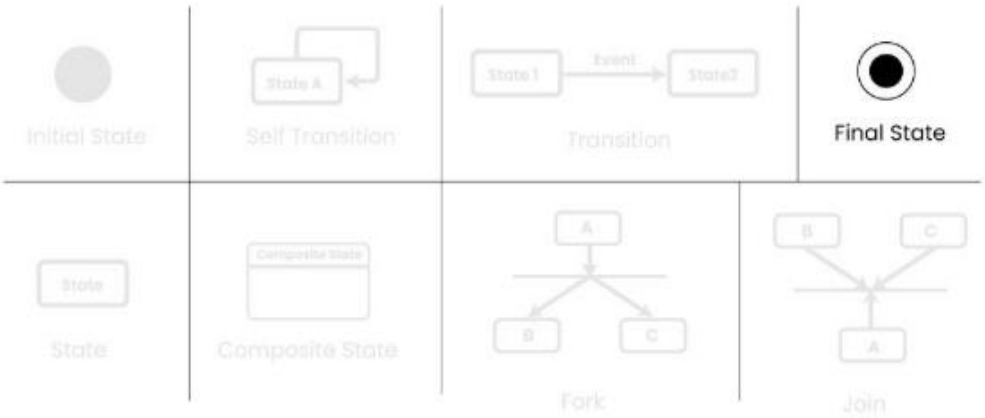


Рис.2.3.8 Кінцевий стан

ДІАГРАМИ ВЗАЄМОДІЇ

Діаграма послідовностей (Sequence Diagram)

Діаграми послідовності показують елементи, які взаємодіють з часом, і вони організовані відповідно до об'єкта (горизонтально) та часу (вертикально):
Горизонтальна вісь показує елементи, які беруть участь у взаємодії.
Зазвичай об'єкти, які беруть участь в операції, перераховуються зліва направо відповідно до того, коли вони беруть участь у послідовності повідомлень. Однак елементи на горизонтальній осі можуть бути розташовані в будь-якому порядку.
Вертикальна вісь представляє процеси часу (або їх прогресування) вниз по сторінці.
Час на діаграмі послідовності — це питання порядку, а не тривалості. Вертикальний простір на діаграмі взаємодії не має значення для тривалості взаємодії.

Нотація діаграми діяльності:

Актор
Це роль, яку виконує сутність, що взаємодіє з об'єктом (наприклад, обмінюючись сигналами та даними) і знаходиться зовні цього об'єкта (тобто екземпляр актора не є частиною екземпляра його відповідного об'єкта).
Актори можуть представляти ролі, що виконуються людьми, зовнішніми пристроями або іншими системами.

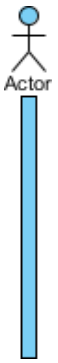


Рис.3.1.1 Актор

Лінія життя
Лінія життя представляє окремого учасника взаємодії.

Рис.3.1.2 Лінія життя

Активації
Тонкий прямокутник на лінії життя представляє період, протягом якого елемент виконує операцію.
Верхня та нижня частини прямокутника вирівняні з часом ініціації та завершення відповідно.

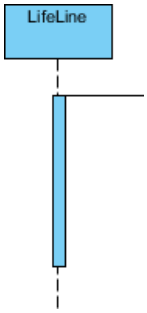


Рис.3.1.3 Активації

Повідомлення виклику

Повідомлення визначає певну комунікацію між лініями життя в межах взаємодії. Повідомлення виклику є видом повідомлення, що представляє виклик операції цільової лінії життя.

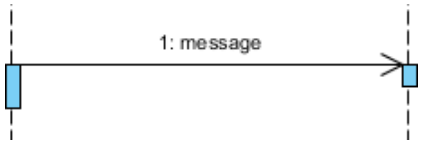


Рис.3.1.4 Повідомлення виклику

Повідомлення повернення

Повідомлення визначає певну комунікацію між лініями життя в межах взаємодії. Повідомлення повернення є видом повідомлення, що представляє передачу інформації назад до викликача відповідного попереднього повідомлення.

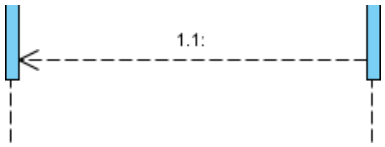


Рис.3.1.5 Повідомлення повернення

Повідомлення самозвернення

Повідомлення визначає певну комунікацію між лініями життя в межах взаємодії. Повідомлення самозвернення є видом повідомлення, що представляє виклик повідомлення тією ж самою лінією життя.

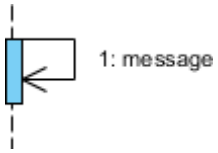


Рис.3.1.6 Повідомлення самозвернення

Рекурсивне повідомлення

Повідомлення визначає певну комунікацію між лініями життя в межах взаємодії. Рекурсивне повідомлення є видом повідомлення, що представляє виклик повідомлення тією ж самою лінією життя. Його цільова точка вказує на активацію, що знаходиться поверх активації, з якої було викликано це повідомлення.

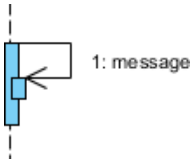


Рис.3.1.7 Рекурсивне повідомлення

Повідомлення створення

Повідомлення визначає певну комунікацію між лініями життя в межах взаємодії. Повідомлення створення є видом повідомлення, що представляє інстанціювання (цільової) лінії життя.

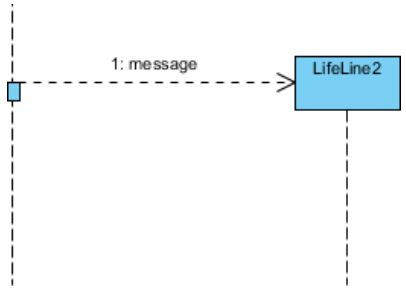


Рис.3.1.8 Повідомлення створення

Повідомлення знищення

Повідомлення визначає певну комунікацію між лініями життя в межах взаємодії. Повідомлення знищення є видом повідомлення, що представляє запит на знищення життєвого циклу цільової лінії життя.

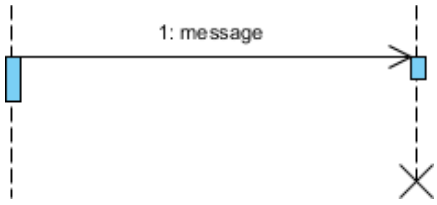


Рис.3.1.9 Повідомлення знищення

Повідомлення тривалості

Повідомлення визначає певну комунікацію між лініями життя в межах взаємодії. Повідомлення тривалості показує відстань між двома часовими моментами для виклику повідомлення.

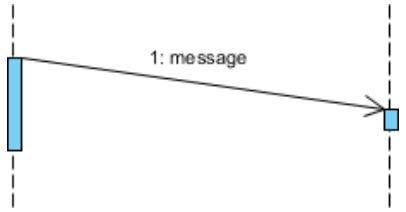


Рис.3.1.10 Повідомлення тривалості

Коментар

Примітка (коментар) дає можливість додавати різні зауваження до елементів. Коментар не має семантичного значення, але може містити інформацію, яка буде корисною для моделювальника.

Рис.3.1.11 Повідомлення тривалості

Діаграма взаємодії (Communication Diagram)

Діаграми комунікації UML, як і діаграми послідовності, є типом діаграм взаємодії, що показують, як об'єкти взаємодіють. Діаграма комунікації є розширенням діаграми об'єктів, яка показує об'єкти разом із повідомленнями, що передаються між ними. Окрім асоціацій між об'єктами, діаграма комунікації відображає повідомлення, які об'єкти надсилають одне одному.

Мета діаграми комунікації:

- Моделювання передачі повідомлень між об'єктами або ролями, що реалізують функціональність випадків використання та операцій.
- Моделювання механізмів у межах архітектурного дизайну системи.
- Захоплення взаємодій, які показують передані повідомлення між об'єктами та ролями в рамках сценарію співпраці.

- Моделювання альтернативних сценаріїв у випадках використання або операціях, які включають співпрацю різних об'єктів та взаємодії.
- Підтримка ідентифікації об'єктів (а отже, класів), їх атрибутів (параметрів повідомлень) і операцій (повідомлень), які беруть участь у випадках використання.

Діаграма комунікації в загальних рисах

У прикладі позначення для діаграми комунікації об'єкти (актори в випадках використання) представлені прямокутниками. У прикладі (загальна діаграма комунікації):

- Об'єкти — це Object1, Object2, Object..., ObjectN-1 ..., та ObjectN.
- Повідомлення, які передаються між об'єктами, представлені мітками на стрілках, які починаються від відправника (актора) і закінчуються отримувачем.
- Повідомлення між об'єктами позначаються 1: message1, 2: message2, 3: message3 тощо, де числовий префікс до назви повідомлення вказує на його порядок у послідовності.
- Спочатку Object1 надсилає Object2 повідомлення message1, Object2 в свою чергу відправляє ObjectN-1 повідомлення message2 і так далі.
- Повідомлення, які об'єкти надсилають самі собі, позначаються як цикли (наприклад, повідомлення message5).

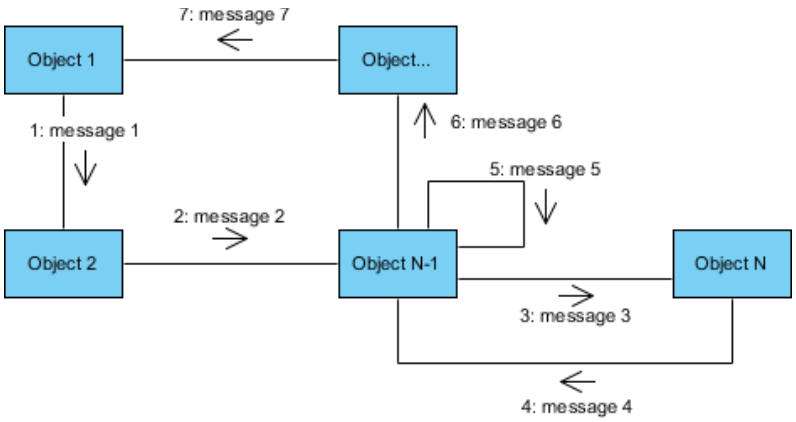


Рис.3.2.1 Приклад діаграми взаємодії

Діаграма таймінгу (Timing Diagram)

Діаграми часу зосереджуються на змінах умов в межах життєвих циклів (lifelines) та між ними вздовж лінійної осі часу. Діаграми часу описують поведінку як окремих класів, так і взаємодій між класами, зосереджуючи увагу на часі виникнення подій, які спричиняють зміни в умовах, що моделюються для життєвих циклів.

Представлення хронології станів

Зміни від одного стану до іншого відображаються зміною рівня життєвого циклу (lifeline). Протягом часу, коли об'єкт перебуває в певному стані, хронологія буде паралельна цьому стану. Зміна стану відображається як вертикальна зміна рівня з одного на інший. Причиною зміни, як і в діаграмі станів чи послідовності, є отримання повідомлення, подія, яка викликає зміну, умова в системі або навіть просто плин часу.

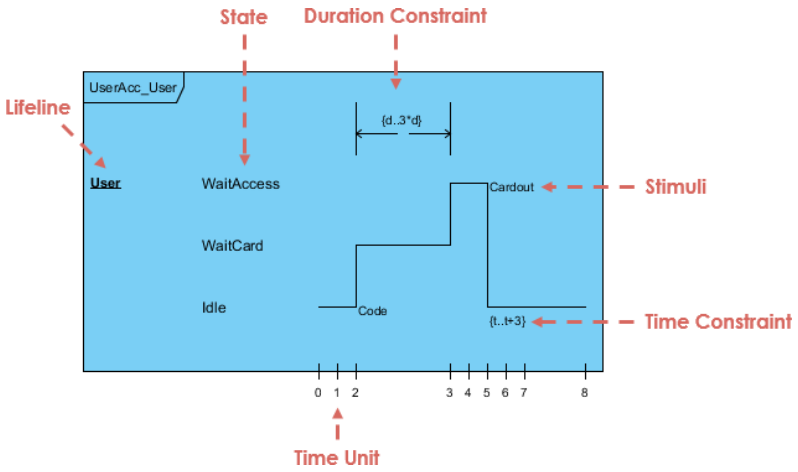


Рис.3.3.1 Представлення хронології станів

Представлення значень життєвого циклу (Value Lifeline)
Нижче наведена альтернатива нотації UML для діаграми часу. Вона показує стан об'єкта між двома горизонтальними лініями, які перетинаються кожного разу, коли стан змінюється.

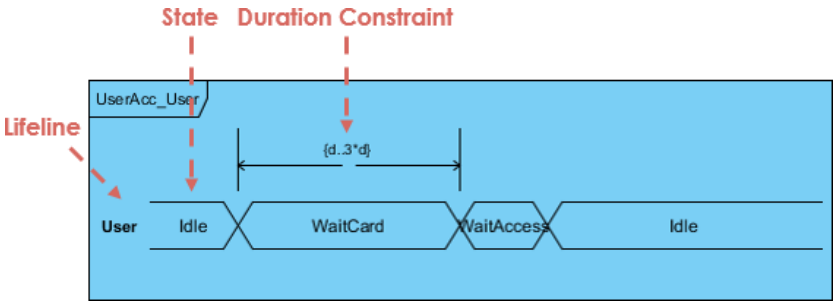


Рис.3.3.2 Представлення значень життєвого циклу

Основні концепції діаграм часу:

Лінія життєвого циклу (Lifeline)
Лінія життєвого циклу на діаграмі часу формує прямокутний простір у межах області вмісту рамки. Лінія життєвого циклу є іменованим елементом, що представляє індивідуального учасника взаємодії. Вона зазвичай розташована горизонтально для зручного читання зліва направо.

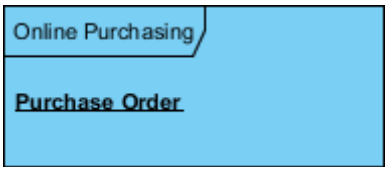


Рис.3.3.3 Лінія життєвого циклу

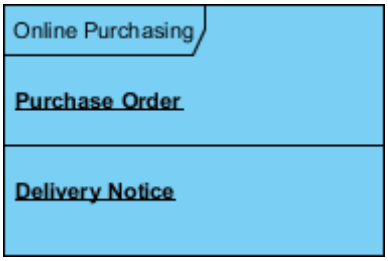


Рис.3.3.4 Кілька ліній життєвого циклу можуть бути розміщені в одній рамці, щоб моделювати взаємодію між ними.

State Timeline

Таймлайн стану в діаграмі часу представляє набір дійсних станів і часу. Стані розташовані на лівому краю лінії життєвого циклу згори вниз. Причиною зміни, як у діаграмі станів або послідовності, може бути отримання повідомлення, подія, що викликає зміну, умова в системі або навіть просто плин часу.

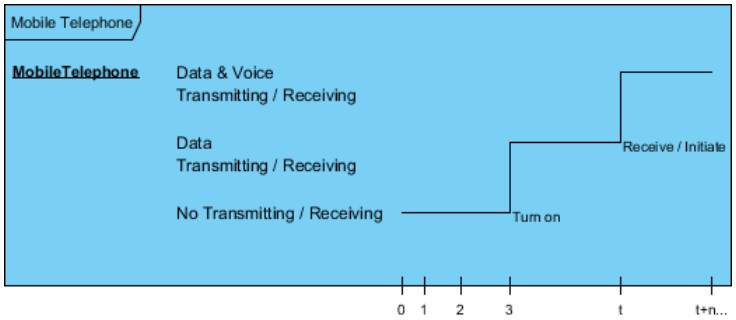


Рис.3.3.5 State Timeline

Множинні відсіки(Multiple Compartments)

Можливо розмістити кілька ліній життєвого циклу різних об'єктів в одній діаграмі часу, одну над іншою. Повідомлення, що надсилаються від одного об'єкта до іншого, можна зображати за допомогою простих стрілок. Початкова та кінцева точки кожної стрілки вказують, коли було надіслано та отримано кожне повідомлення.

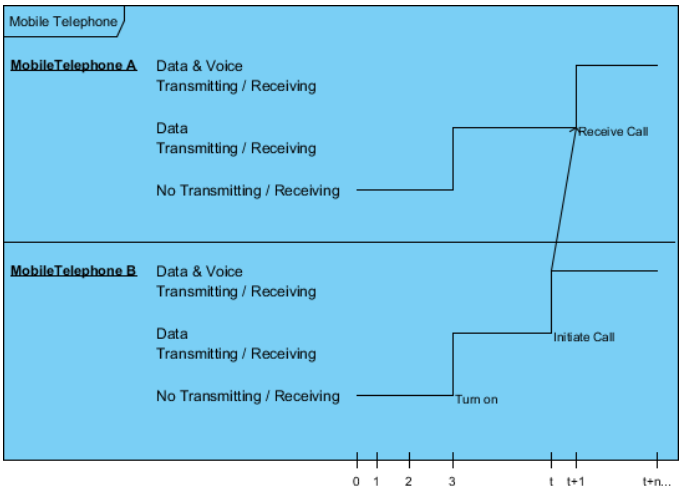


Рис.3.3.6 Multiple Compartments

Лінія життєвого циклу стану(State Lifeline)

Лінія життєвого циклу стану показує зміну стану елемента з часом. Осьова вісь (X) відображає пройдений час у вибраних одиницях, а вісь Y підписується зі списком доступних станів. Нижче наведено приклад лінії життєвого циклу стану*:*

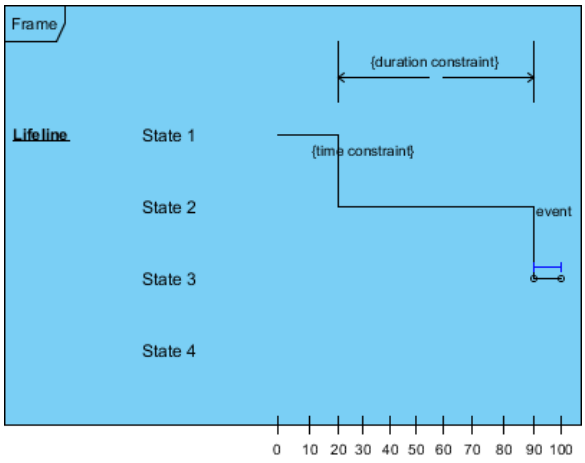


Рис.3.3.7 State Lifeline

Лінія значень життєвого циклу(Value Lifeline)

Лінія значень життєвого циклу показує зміну значення елемента з часом. Осьова вісь (X) відображає пройдений час у вибраних одиницях, так само як і для лінії життєвого циклу стану. Значення відображаються між парою горизонтальних ліній, які перетинаються при кожній зміні значення.

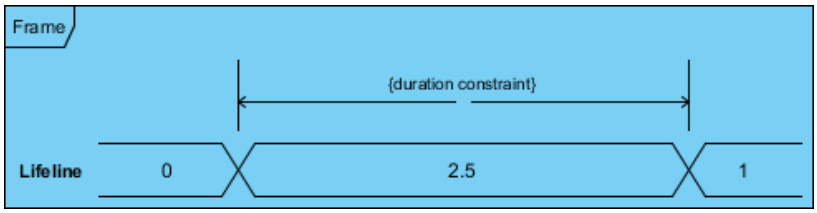


Рис.3.3.8 Value Lifeline

Часова лінія та обмеження(Timeline and Constraints)

Ми можемо використовувати довжину часової лінії для вказівки того, як довго об'єкт перебуває в певному стані, читаючи її зліва направо. Для асоціації вимірів часу на нижній частині рамки показують мітки часу.

Приклад нижче показує, що подія входу (Login) отримана через три одиниці часу після початку послідовності. Для показу відносних часів можна позначити конкретний момент часу, використовуючи змінну назву. На малюнку вказано час, коли подія sendMail була отримана як час.

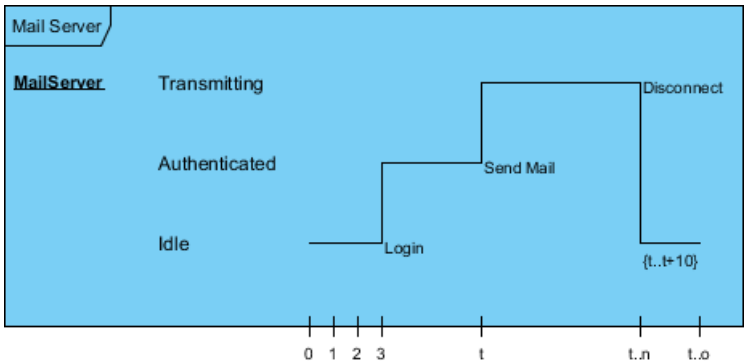


Рис.3.3.9 Ви можете використовувати відносні мітки часу в обмеженнях, щоб вказати, що повідомлення повинно бути отримано протягом визначеного проміжку часу.

Лінії значень та станів життєвого циклу

Лінії значень та станів життєвого циклу можуть бути розміщені одна за одною в будь-якому поєднанні. Повідомлення можуть передаватися від однієї лінії життєвого циклу до іншої. Кожен перехід стану або значення може мати визначену подію, обмеження часу, яке вказує, коли подія повинна відбутися, і обмеження тривалості, яке вказує, як довго стан або значення повинні залишатися в дії.

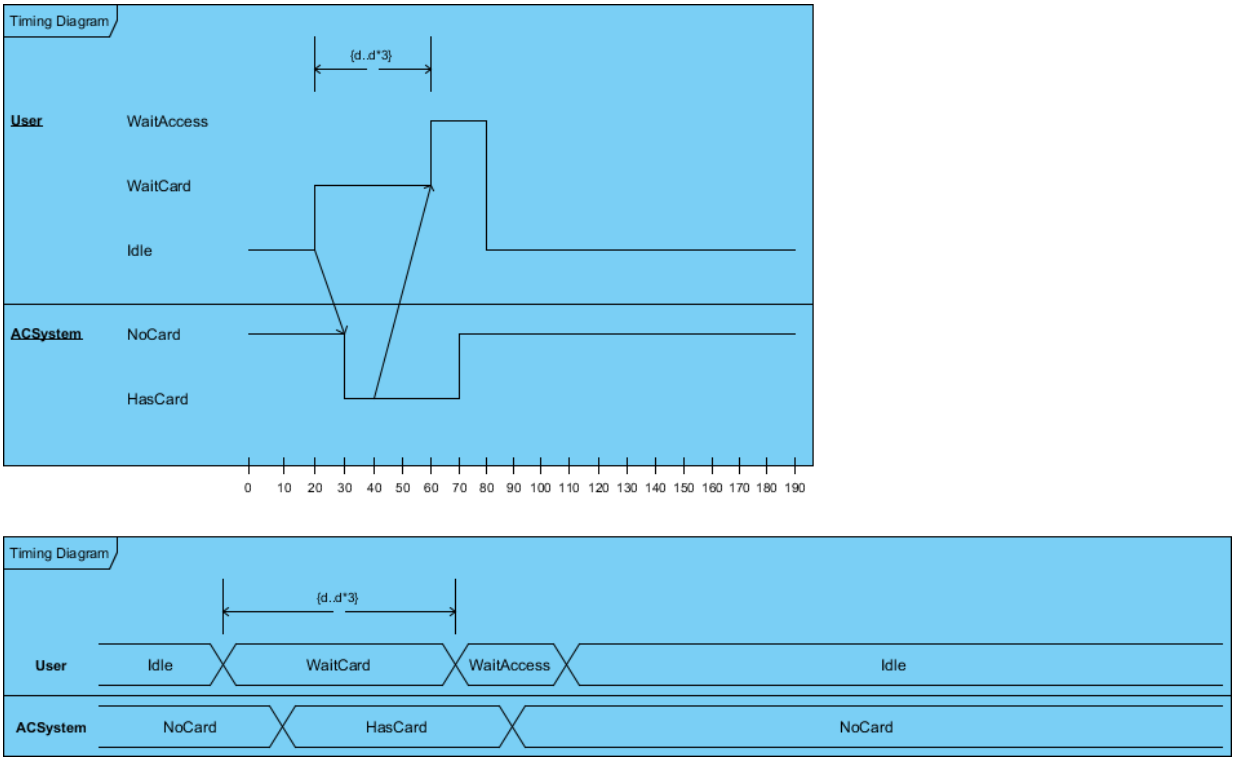


Рис.3.3.10 Лінії значень та станів життєвого циклу разом

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Розробити діаграму розгортання для проектованої системи.
3. Розробити діаграму компонентів для проектованої системи.
4. Розробити діаграму послідовностей для проектованої системи.
5. Скласти звіт про виконану роботу

Хід роботи:

Для початку побудуємо діаграму розгортання для обраної теми роботи – JSON Tool.

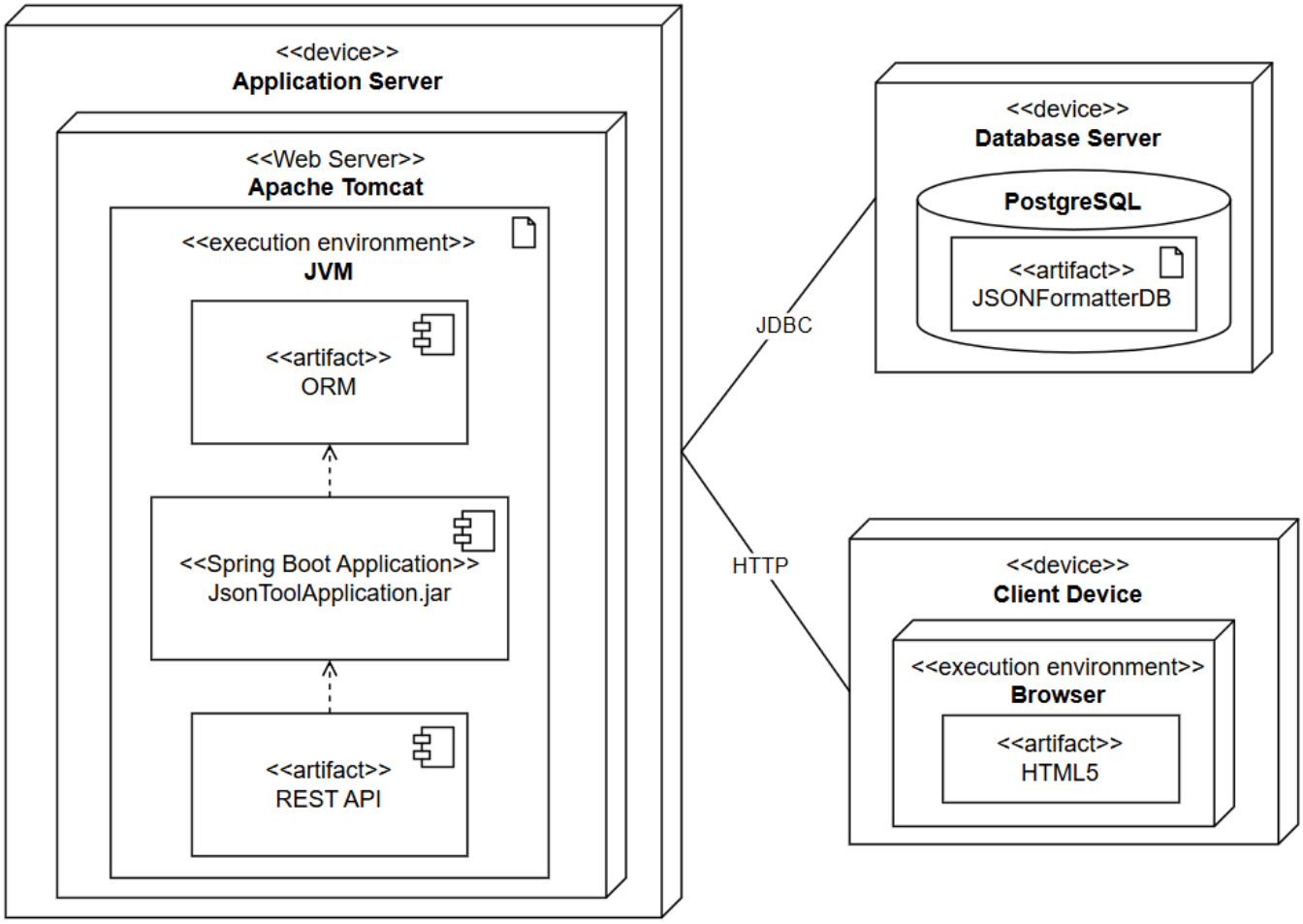


Рис. 1. Діаграма розгортання для JSON Tool

Застосунок розгортається на пристрої Application Server, який обробляє запити через REST API і взаємодіє з Database Server за допомогою JDBC. Клієнт отримує доступ до сервера через браузер по HTTP.

Компоненти діаграми

1. ***Application Server***
   - Web Server: Apache Tomcat виконує роль веб-сервера для обслуговування запитів.
   - Execution Environment: JVM (Java Virtual Machine), на якому працює застосунок.
   - Spring Boot Application:
     - Артефакт JsonToolApplication.jar є основним застосунком.
     - ORM (Object-Relational Mapping) відповідає за взаємодію із базою даних.
     - REST API: Забезпечує HTTP-комунікацію з клієнтами.
2. ***Database Server***
   - PostgreSQL: Використовується як СУБД.
   - Артефакт: JSONFormatterDB, який зберігає JSON-схеми та інші дані.
   - JDBC: Протокол для комунікації між застосунком та базою даних.
3. ***Client Device***
   - Execution Environment: Browser, що працює на клієнтському пристрої.
   - HTML5: Використовується для відображення інтерфейсу користувача.

Зв'язки

- **HTTP**: Комунікація між REST API на сервері та браузером на клієнтському пристрої.
- **JDBC**: Зв'язок між ORM і PostgreSQL для доступу до бази даних.

Далі побудуємо діаграму компонентів розробленої частини застосунку:

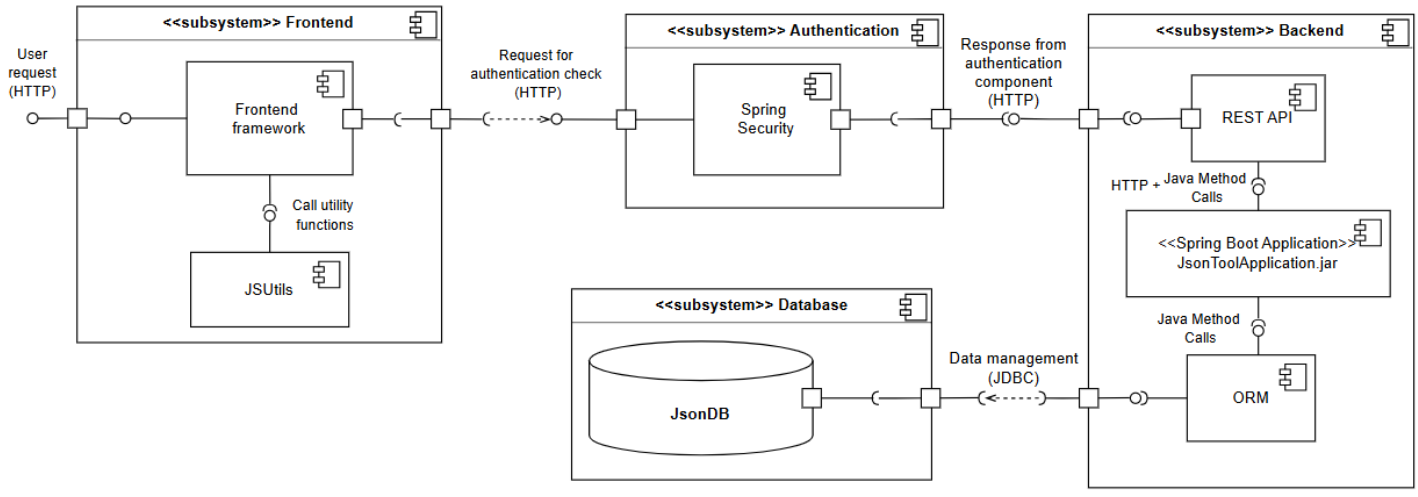


Рис. 2. Діаграма компонентів реалізованої частини системи

Загальний опис діаграми

Діаграма показує, як HTTP-запити користувача проходять через різні компоненти застосунку. Застосунок складається з чотирьох ключових підсистем:

1. Frontend
2. Authentication
3. Backend
4. Database

1. Підсистема: Frontend

Обробляє запити користувача (HTTP) та взаємодіє з підсистемою Backend для виконання функцій форматування JSON.

Компоненти:

- Frontend framework: Основний інтерфейс для обробки запитів користувача.
- JSUtils: JavaScript утиліти для виконання додаткових функцій на фронтенді (наприклад, валідація даних чи маніпуляції з JSON).

Потік даних:

- Користувач надсилає HTTP-запит через інтерфейс (Frontend framework).
- Викликаються допоміжні функції JSUtils для попередньої обробки запиту.
- Далі запит передається до Authentication для перевірки.

2. Підсистема: Authentication

Аутентифікація користувача за допомогою Spring Security.

Компоненти:

- Spring Security: Компонент для обробки HTTP-запитів на аутентифікацію користувача.

Потік даних:

- HTTP-запит передається в перевірку на аутентифікацію.
- Після успішної аутентифікації відповідь повертається назад до Frontend.

3. Підсистема: Backend

Обробляє бізнес-логіку та форматує JSON.

Компоненти:

- REST API: Обробник HTTP-запитів, що надає API для взаємодії з Frontend.
- JsonToolApplication.jar: Основний Spring Boot застосунок, який реалізує функціонал форматування JSON.
- ORM: Служить для взаємодії з базою даних через JDBC.

Потік даних:

- Після успішної аутентифікації, Frontend надсилає запит на REST API.
- REST API викликає відповідні методи в JsonToolApplication.jar для обробки JSON.
- Застосунок взаємодіє з ORM, якщо потрібна робота з даними.

4. Підсистема: Database

Зберігає та управляє даними для застосунку.

Компоненти:

- JsonDB: База даних, яка використовується для збереження JSON-об'єктів або логів.

Взаємодія:

- Використовується JDBC для обробки даних між Backend та базою даних.

Загальний потік даних:

1. Користувач надсилає HTTP-запит через Frontend.
2. Запит передається на Spring Security для аутентифікації.
3. Після успішної аутентифікації запит переходить до Backend.
4. REST API обробляє запит і викликає методи з JsonToolApplication.jar для форматування JSON.
5. Якщо необхідно, ORM виконує операції з JsonDB за допомогою JDBC.
6. Результат повертається назад до користувача через Frontend.

Далі розробимо діаграму послідовностей:

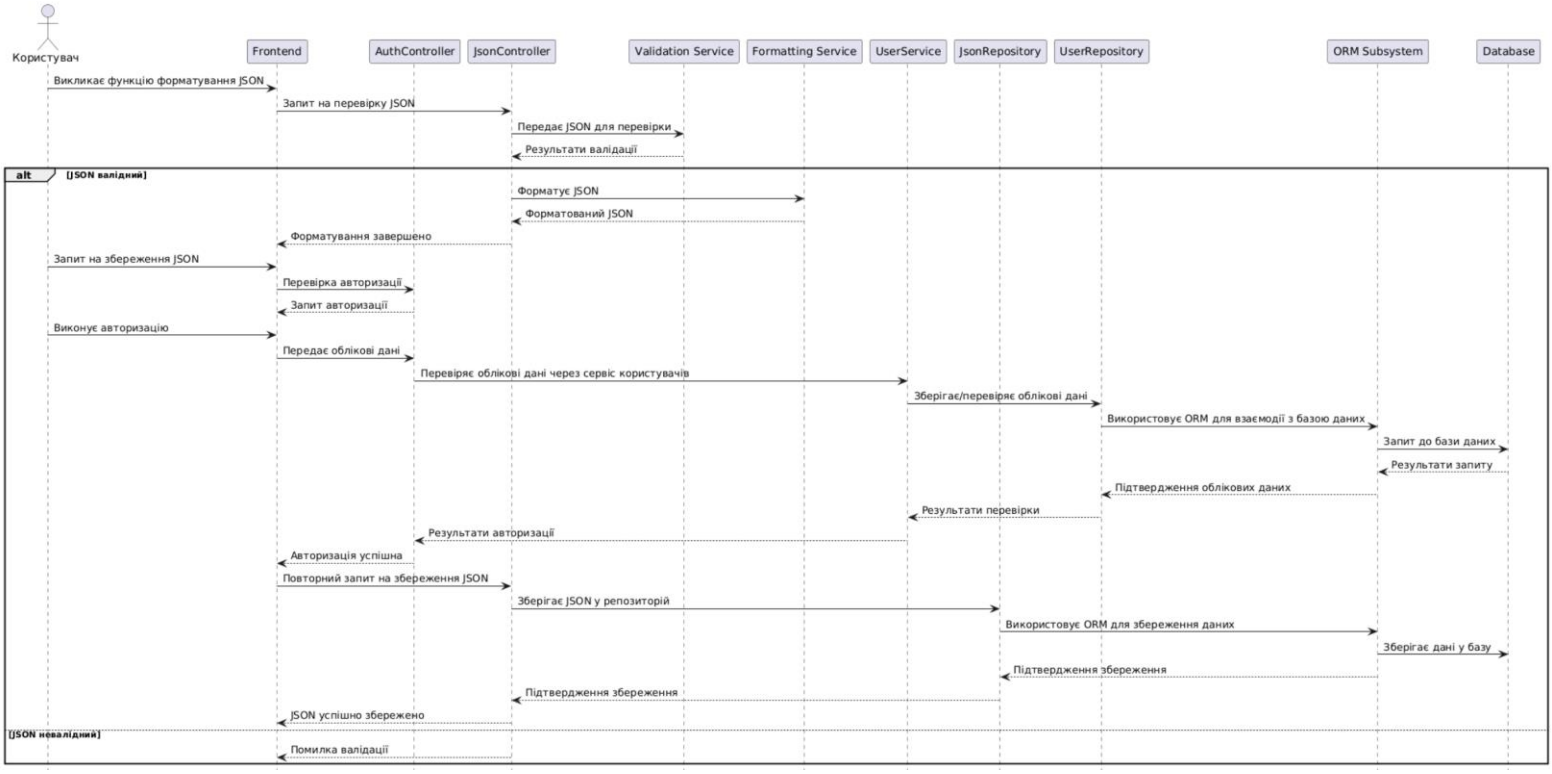


Рис. 3. Діаграма компонентів послідовностей частини системи

Ця діаграма показує, як відбувається обробка JSON, включно з форматуванням, авторизацією користувача та збереженням у базу даних. Вона демонструє взаємодію між усіма основними сервісами, контролерами та сховищами даних.

Основний процес включає два варіанти сценаріїв: успішна валідація JSON та помилка валідації.

Актор «Користувач» ініціює запит на форматування та збереження JSON.

Основні модулі:

1. Frontend – інтерфейс користувача, який надсилає запити до контролерів.
2. AuthController – відповідає за авторизацію користувача.
3. JsonController – основний контролер для обробки запитів щодо JSON.
4. Validation Service – сервіс, який перевіряє валідність JSON.
5. Formatting Service – відповідає за форматування JSON.
6. UserService – сервіс для перевірки облікових даних користувача.
7. JsonRepository – відповідає за збереження JSON у сховище.
8. UserRepository – взаємодіє з базою даних для перевірки облікових даних.
9. ORM Subsystem – для взаємодії з базою даних.
10. Database – кінцеве сховище даних.

Основний сценарій: Успішна валідація JSON

1. Користувач викликає функцію форматування JSON (запит передається через Frontend до JsonController).

2. JsonController:
   - Передає JSON у Validation Service для перевірки валідності.
   - Отримує результат валідації.

3. Якщо JSON валідний:
   - Formatting Service форматує JSON.
   - Форматування завершується, і надходить запит на збереження JSON.

4. Проводиться перевірка авторизації:
   - AuthController надсилає запит до UserService.
   - UserService взаємодіє з UserRepository та ORM Subsystem для перевірки облікових даних у Database.

5. У разі успішної авторизації:
   - Запит на збереження JSON повторно надходить у JsonRepository.
   - JsonRepository використовує ORM для збереження даних у базу.
   - Дані успішно зберігаються, і відправляється підтвердження збереження користувачу.

Альтернативний сценарій: Помилка валідації JSON

1. Якщо JSON не проходить перевірку у Validation Service, користувач отримує помилку валідації.
2. Збереження JSON не виконується.

Основні комунікації:

- Frontend ↔ JsonController: Вхідні запити та результати обробки.
- JsonController ↔ Validation Service: Перевірка JSON.
- JsonController ↔ Formatting Service: Форматування JSON.
- AuthController ↔ UserService: Перевірка авторизації користувача.
- UserService ↔ UserRepository ↔ ORM Subsystem ↔ Database: Доступ до бази даних для перевірки облікових даних.
- JsonRepository ↔ ORM Subsystem ↔ Database: Збереження JSON у базі даних.

ВИСНОВОК

У ході роботи було спроєктовано систему JSON Tool шляхом розробки діаграми послідовностей яка показує взаємодію користувача з компонентами системи під час форматування валідації та збереження JSON, діаграми розгортання яка представлена архітектурою системи з серверами середовищем виконання та комунікацією через HTTP і JDBC та діаграми компонентів яка відображає структуру основних модулів системи та їх функціональність. Розроблені діаграми забезпечують чітке розуміння архітектури та взаємодії системи для подальшої реалізації