**СЛАЙД 1**

Тема: Система віддаленої взаємодії із віртуальними машинами

Виконав: студент 4 курсу Борсук Андрій Юрійович

Науковий керівник: Валь Олександр Олександрович

**СЛАЙД 2**

Актуальність

Багато девопсів стикаються з тим, що проекти, над якими вони працюють, потребують прямого втручання, проте люди не можуть постійно перебувати на робочому місці. Багатьом із них стало б пригоді рішення, яке надавало б можливість взаємодіяти зі своїми машинами віддалено.

**СЛАЙД 3**

Мета

Метою розроблюваного застосунку є надання можливості девопсам віддалено, з будь-якої точки світу, виправляти проблеми, що виникають або просто слідкувати за станом їх віртуальних машин.

**СЛАЙД 4**

Завдання проекту

* Розробити можливість авторизації та реєстрації
* Розробити клієнт SSH-підключення
* Розробити клієнт SFTP-підключення
* Розробити клієнт збору метрик
* Реалізувати можливість додання віртуальних машин
* Розробити захищений API
* Локалізувати систему
* Інтегрувати в проект Telegram Bot Api

**СЛАЙД 5**

Об’єкт дослідження

Реалізація системи віддаленої взаємодії із віртуальними машинами.

**СЛАЙД 6**

Основною мовою додатку є мова програмування C#. Головними її перевагами, які стали визначальними для проекту є об’єктно-орієнтованість, високий рівень безпеки, LINQ, широкі можливості бібліотек та фреймворк ASP.NET Core, який дозволяє легко будувати веб застосунки.

Мовою, якою буде написаний клієнт, що збиратиме метрики, був обраний Python.

Основною базою даних проєкту буде Microsoft SQL Server, в ній буде зберігатися інформація про користувачів та віртуальні машини.

Для кешування сесійних даних користувача буде використана база даних Redis.

Entity Framework Core – ORM-фреймворк для взаємодії з базою даних. Він дозволяє легко та зручно під’єднуватися до БД та взаємодіяти з її сутностями.

SSH.NET – бібліотека, яка надає функціонал, для створення SSH- та SFTP-з’єднань з віртуальними машинами.

**СЛАЙД 7**

Архітектура

Архітектурою додатка була обрана N-tier архітектура.

Для Api ж була використана архітектура REST Api.

**СЛАЙД 8**

Реалізація.

Кінцевий застосунок було розроблено у вигляді 2 окремих частин.

**СЛАЙД 9**

Перша частина проекту відповідає за облік даних у базі даних та створення каналу взаємодії між користувачем та машиною.

**СЛАЙД 10**

Всі дані, що зберігаються в базі даних захищені, всі паролі зашифровані. Це означає, що мережею підчас http-запитів не передаються відкрито критичні дані. Навіть в разі перехоплення зловмисником цих запитів не дає йому доступ до даних, бо ключі для розшифрування знає тільки сам застосунок, ці ключі нікуди не передаються і знаходяться тільки в ізольованому середовищі застосунку.

**СЛАЙД 11**

Система вміє створювати ізольований та захищений шлюз взаємодії між клієнтом та віртуальною машиною. Це дозволяє безпечно та віддалено виконувати на стороні машини команди та отримувати й інтерпретувати її відповідь на стороні користувача. Що надає змогу деповсам постаратися виправити неполадку, що виникла, не перебуваючи на робочому місці, якщо це сталося, наприклад, не в робочий час.

**СЛАЙД 12**

Також за допомогою цього шлюзу можна не тільки виконувати команди, а й маніпулювати файловою системою машини. Це означає, що користувач отримує майже повний, дуже гнучкий і при цьому віддалений доступ до віртуальної машини. І за потреби він може завантажити якийсь скрипт на машину, після чого одразу його виконати. Або навпаки швидко завантажити файл із машини та виконати над ним будь-які дії, наприклад, поділитися із колегою. Користувач перебуваючи вдома забезпечений системою дуже широким спектром функціоналу, що дуже полегшує його роботу та дозволяє вчасно реагувати на проблеми, що виникають.

**СЛАЙД 13**

Система дозволяє користувачам отримувати графіки, які описують картину стану машини. В системи є свій клієнт для збору метрик, який вона завантажує на віртуальну машину. Клієнт є python-скриптом, який збирає метрики та повертає їх системі, яка переводить дані у графіки та надсилає їх користувачу. Також в користувача є можливість отримати метрики в «сирому» вигляді json-об’єкта.

**СЛАЙД 14**

У користувача є можливість створювати декілька акаунтів та прив’язувати до них різні віртуальні машини. Це дозволить йому розділити віртуальні машини за скопами роботи, наприклад, на робочі та власні.

**СЛАЙД 15**

Друга ж частина додатку відповідає за графічний інтерфейс та отримання команд від користувача. Оскільки додаток має бути доступним майже з кожного девайсу та з будь-якого місця, то було вирішено в якості графічного інтерфейсу використати Telegram бота.

**СЛАЙД 16**

Спілкування між користувачем та додатком відбувається за допомогою повідомлень. Система вміє розпізнавати дію, яку користувач хоче виконати за ключовими словами в його повідомленнях. Користувач використовуючи додаток просто віддає команди, які система розпізнає та виконує.

**СЛАЙД 17**

Система на повну використовує такий функціонал Telegram Bot Api, як клавіатури. На відміну від клавіатури девайсу, з якого відбувається взаємодія, у клавіатурах ботів наперед визначені команди, на які повинен реагувати бот. Клавіатури дозволяють надавати користувачу тільки той функціонал, який йому зараз потрібен та моделювати необхідні сценарії взаємодії між ним та системою. Вони знаходяться в нижній частині інтерфейсу.

**СЛАЙД 18**

Вся взаємодія в додатку захищена за допомогою JWT токенів. Кожного разу, як користувач виконує команду або операцію над файловою системою, то бот виконує http-запит на захищений Api системи. Для доступу використовуються JWT токени, які генеруються при вході користувача в систему. Якщо користувач спробує виконати команду не увійшовши до системи, то вона попросить його спочатку авторизуватися.

**СЛАЙД 19**

Користувач може користуватися системою тією мовою, якою зручно.

**СЛАЙД 20**

Зручний вивід відповіді від віртуальної машини. Для виконання команди на віртуальній машині користувачу достатньо ввести її боту. Далі система розпізнає команду, створить шлюз спілкування за допомогою вищеописаного алгоритму, виконає її та верне користувачу результат виконання.

**СЛАЙД 21**

Користувач завжди може звернутися за порадою до ChatGPT. Якщо йому необхідно дізнатися, що означає та чи інша команда або дізнатися про її параметри, то він може це запитати в мовної моделі OpenAI.

**СЛАЙД 22**

Висновки

1. Було сформульовано проблему та визначено необхідний функціонал
2. Використані технології: C#, Python, MS SQL Server, Redis, Entity Framework Core та SSH.NET
3. Було спроектовано додаток
4. Розроблено алгоритми функціонування системи, визначено порядок взаємодії класів під час виконання програмного коду та реалізовано додаток
5. Проведено тестування додатку

Реалізований проект, складається з 2 частин. В поєднанні вони створюють зручний кишеньковий додаток, який вирішує поставлену проблему

**СЛАЙД 23**

Q&A

1. **БІЛЬШЕ ПРО АРХІТЕКТУРУ: ТЕКСТ ЗІ СЛАЙДІВ СКАЗАТИ, А ЗАМІСТЬ НЬОГО ПОСТАВИТИ ГРАФІКИ**
2. **ПОДУМАТИ ЯК СТИСНУТИ ДЕКІЛЬКА СЛАЙДІВ З ОДИНИМ РЕЧЕННЯМ В ОДИН БІЛЬШ ЗАГАЛЬНИЙ**
3. **СЛАЙДИ З UI РОБИТИ ПО 2 СКРІНИ НА СЛАЙД І ТЕКСТ, ЯКИЙ ЇХ ОПИСУЄ, ГОВОРИТИ ПРО ОБИДВА СЛАЙДИ**
4. **ДОДАТИ ПРИКЛАДИ ГРАФІКІВ МЕТРИК**
5. **ПЕРЕРОБИТИ ВИСНОВИК, ЇХ МОЖНА РОЗДІЛИТИ НА ДЕКІЛЬКА СЛАЙДІВ І 22 КЕГЛЬ**
6. **ЧАС, ТРІШКИ ЗАДОВГО ГОВОРИВ**
7. **СЛАЙДІВ МАКСИМУМ 18**