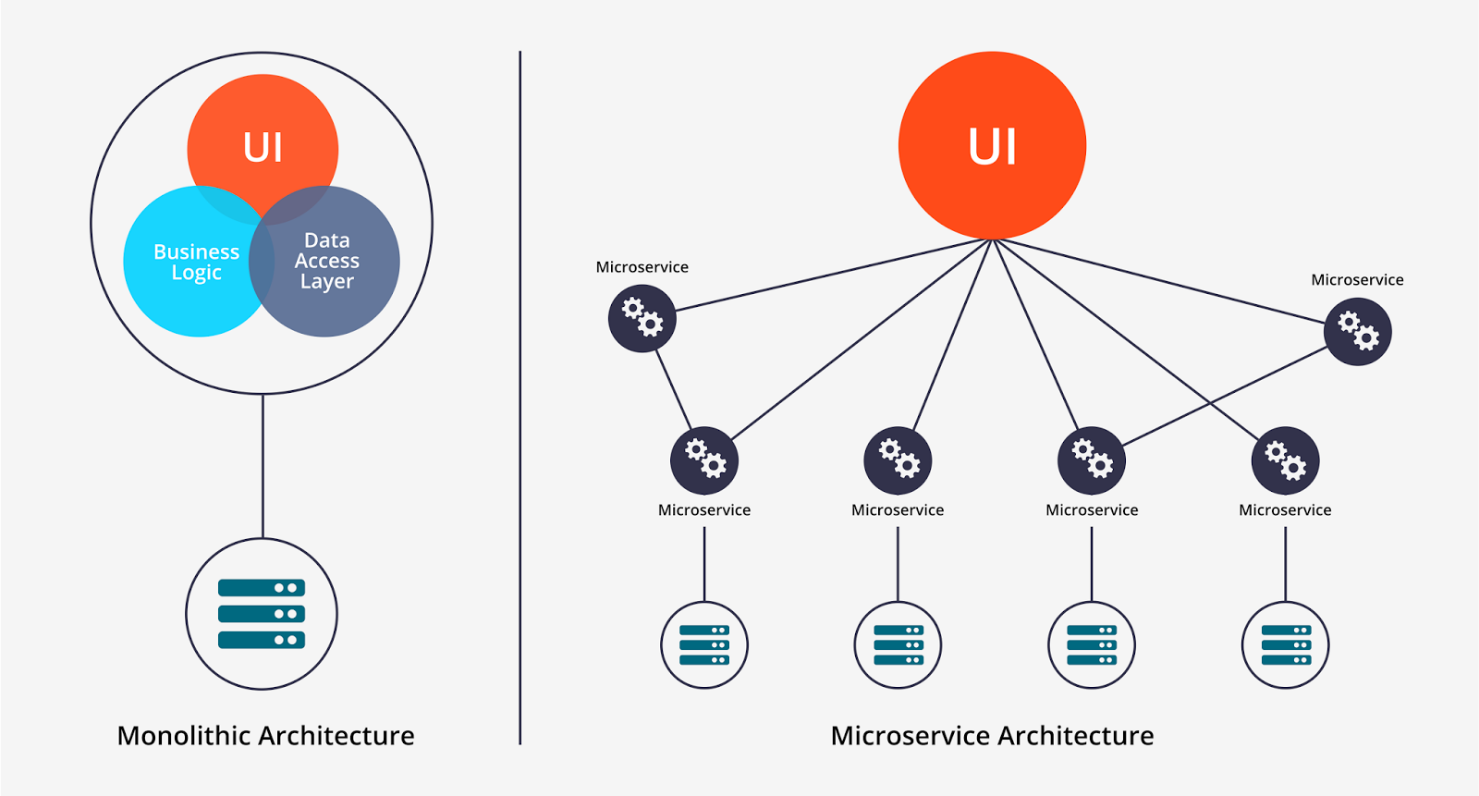
**З приводу архітектури:**

У нас user-to-server додаток

Пропоную використовувати мікросервісную архітектуру, де кожна функція буде реалізововаться як окремий модуль, картинка додається нижче (ч.2):

Нам потрібно організувати ПП в якому будуть функції управління кафедрою, сповіщень, можливістю внесення змін до кожен з модулів і т.п

Нам потрібно реалізувати такі концептуальні особливості:

**Ефективність системи**. В першу чергу програма, звичайно ж, повинна вирішувати поставлені завдання і добре виконувати свої функції, причому в різних умовах. Сюди можна віднести такі характеристики, як надійність, безпеку, продуктивність, здатність справлятися зі збільшенням навантаження (масштабованість) і т.п.

**Гнучкість системи.** Будь-який додаток доводиться міняти з часом - змінюються вимоги, додаються нові. Чим швидше і зручніше можна внести зміни в існуючий функціонал, чим менше проблем і помилок це викличе - тим гнучкіше і конкурентоздатною система. Тому в процесі розробки намагайтеся оцінювати те, що виходить, на предмет того, як вам це потім, можливо, доведеться міняти. Запитайте у себе: «А що буде, якщо поточний архітектурне рішення виявиться невірним?», «Яка кількість коду піддасться при цьому змін?». Зміна одного фрагмента системи не повинно впливати на її інші фрагменти. По можливості, архітектурні рішення не повинні «вирубувати в камені», і наслідки архітектурних помилок повинні бути в розумній мірі обмежені. "Хороша архітектура дозволяє ВІДКЛАДАТИ прийняття ключових рішень" (Боб Мартін) і мінімізує «ціну» помилок.

**Можливість розширення системи**. Можливість додавати в систему нові сутності та функції, не порушуючи її основної структури. На початковому етапі в систему має сенс закладати лише основний і самий необхідний функціонал (принцип YAGNI - you is not gonna need it, «Вам це не знадобиться») Але при цьому архітектура повинна дозволяти легко нарощувати додатковий функціонал в міру необхідності. Причому так, щоб внесення найбільш ймовірних змін вимагало найменших зусиллі.

**Вимога, щоб архітектура системи володіла гнучкістю і розширюваністю** (тобто була здатна до змін і еволюції) є настільки важливим, що воно навіть сформульовано у вигляді окремого принципу - «Принципу відкритості / закритості» (Open-Closed Principle - другий з п'яти принципів SOLID) : Програмні суті (класи, модулі, функції і т.п.) повинні бути відкритими для розширення, але закритими для модифікації.

Іншими словами: Повинна бути можливість розширити / змінити поведінку системи без зміни / переписування вже існуючих частин системи.

Це означає, що додаток слід проектувати так, щоб зміна його поведінки і додавання нових функцій був би досягнутий за допомогою написання нового коду (розширення), і при цьому не доводилося б змінювати вже існуючий код. В такому випадку поява нових вимог не спричинить за собою модифікацію існуючої логіки, а зможе бути реалізовано перш за все за рахунок її розширення. Саме цей принцип є основою «плагін архітектури» (Plugin Architecture). Про те, за рахунок яких технік це може бути досягнуто, буде розказано далі.

**Масштабованість процесу розробки**. Можливість скоротити термін розробки за рахунок додавання до проекту нових людей. Архітектура повинна дозволяти распараллелить процес розробки, так щоб безліч людей могли працювати над програмою одночасно.

**Здатність до тестування. Код, який легше тестувати, буде містити менше помилок і надійніше працювати. Але тести не тільки покращують якість коду. Багато розробники приходять до висновку, що вимога «хорошою тестованості» є також спрямовуючою силою, автоматично веде до хорошого дизайну, і одночасно одним з найважливіших критеріїв, що дозволяють оцінити його якість: "Використовуйте принцип« тестованості »класу в якості« лакмусового папірця »хорошого дизайну класу. Навіть якщо ви не напишіть жодного рядка тестового коду, відповідь на це питання в 90% випадків допоможе зрозуміти, наскільки все «добре» або «погано» з його дизайном "(Ідеальна архітектура).**

**Можливість повторного використання. Систему бажано проектувати так, щоб її фрагменти можна було повторно використовувати в інших системах.**

**Супроводжуваність. Над програмою, як правило, працює безліч людей - одні йдуть, приходять нові. Після написання супроводжувати програму теж, як правило, доводиться людям, які не брав участь в її розробці. Тому хороша архітектура повинна давати можливість відносно легко і швидко розібратися в системі нових людей. Проект повинен бути добре структурований, не містити дублювання, мати добре оформлений код і бажано документацію. І по можливості в системі краще застосовувати стандартні, загальноприйняті рішення звичні для програмістів. Чим екзотичніше система, тим складніше її зрозуміти іншим (Принцип найменшого подиву - Principle of least astonishment. Зазвичай, він використовується відносно призначеного для користувача інтерфейсу, але застосуємо і до написання коду).**