Tiers

Announcements

May 5: Special lecture, 15:00-17:00. Combined with other class.

Title: How to Think like a Founder; Entrepreneurship for Computer Science Grads

Description: A Step by Step guide to building a startup, from formation to fundraising

Lecture: Tiers

Different tiers in architecture help us have a bird’s eye view of the design.

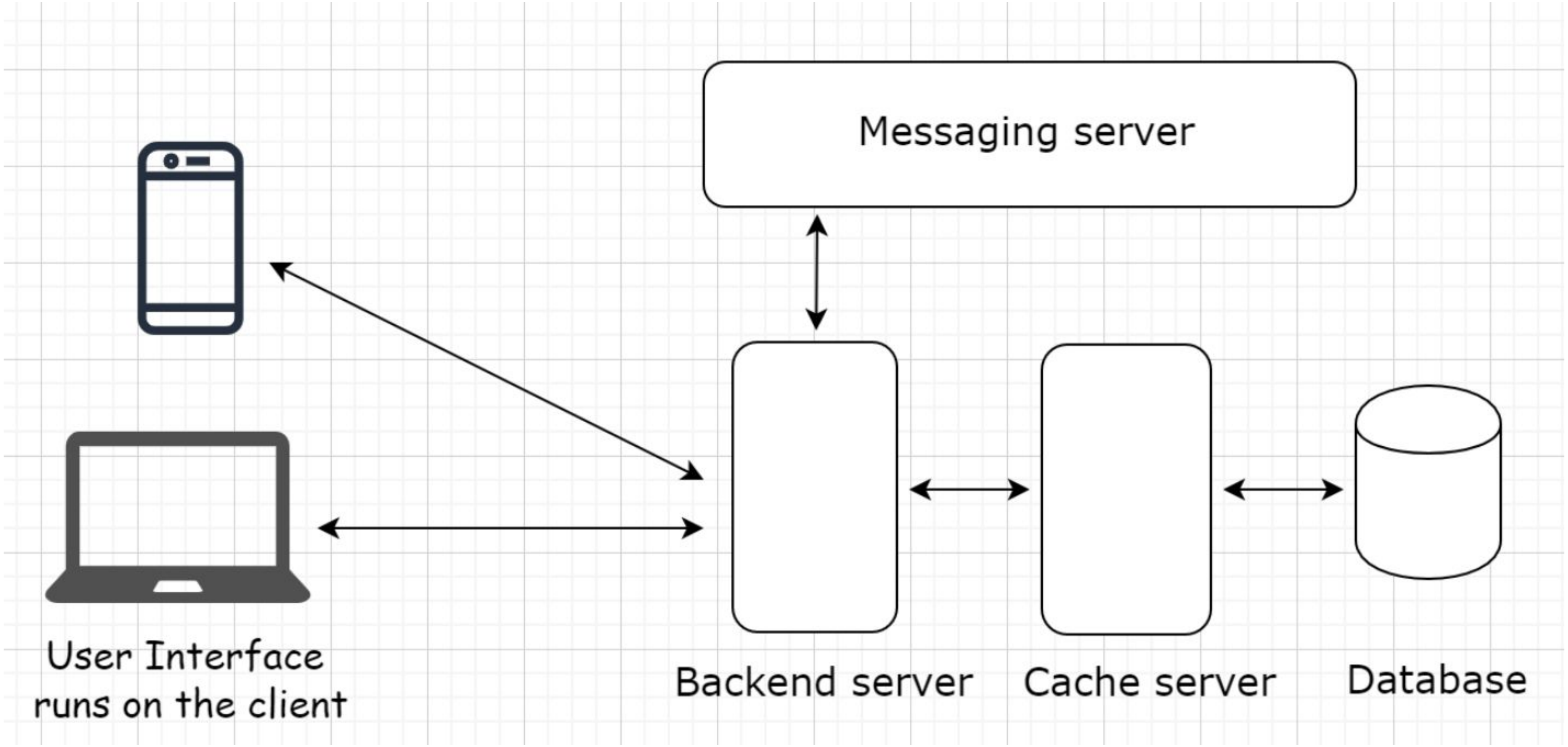
This lecture will help us understand:

- What is a tier?
- Why do software applications have different tiers?
- How do I decide how many tiers my application should have?

What is a tier?

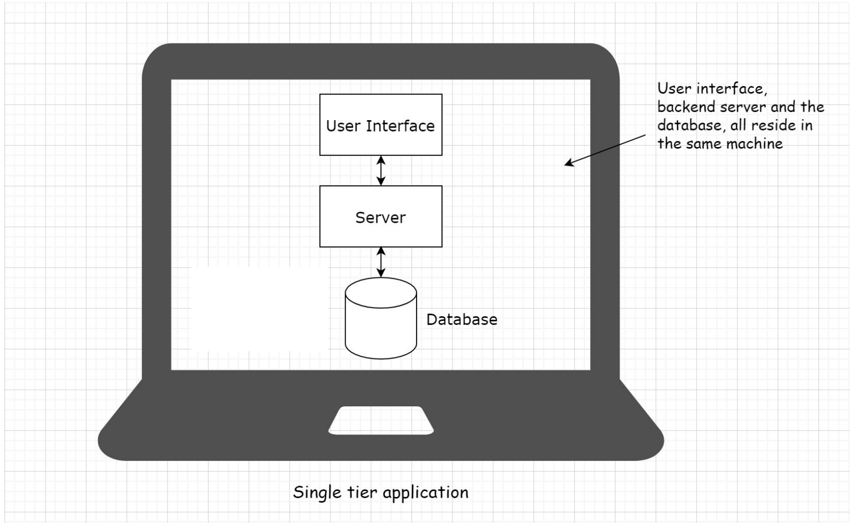
- Think of a tier as both logical and physical separation of components in an application or a service.
- This separation is at a component level, not the code level.
- Examples of Components:
    - Database
    - Backend application server
    - User interface
    - Messaging
    - Caching etc.
- These are the components that make up a web service
    - Other components for other things

What is a tier?



Single Tier Applications

The user interface, backend business logic, and the database reside in the same machine.


User Interface
Server
Database
User interface, backend server and the database, all reside in the same machine
Single tier application

Typical examples of single-tier applications are desktop applications like:

- MS Office, downloadable PC Games, local image editing software, etc.

Pros of Single Tier Applications

- No network latency
  - Every component is located on the same machine.
  - No data requests sent to other tiers
  - More predictable performance
  - Data is readily available since all the components are located in the same machine.
  - CAVEAT: actual performance of a single-tier app largely depends on the application’s hardware requirements and how powerful the machine it runs on is.
- Data privacy and safety
  - User’s data always stays in their machine and doesn’t need to be transmitted over a network for persistence.

Cons of Single Tier Applications

- Application’s publisher has no control over the application.
  - Once the software is shipped, no code or feature updates can be made until the customer manually updates it
  - Example: ship a buggy game? wait for customer to update :(

Cons of Single Tier Applications

- Application's publisher has no control over the application.
    - Once the software is shipped, no code or feature updates can be made until the customer manually updates it
    - Example: ship a buggy game? wait for customer to update :(
- Code vulnerable to being tweaked and reversed engineered.
    - Product security for the app publisher is minimal.
    - A bad actor (with some effort) can get access to the application's source code
        - Modifying or copying it for profit.
        - This is unlikely in an architecture where the company controls the application server and implements security to fend off the hackers.

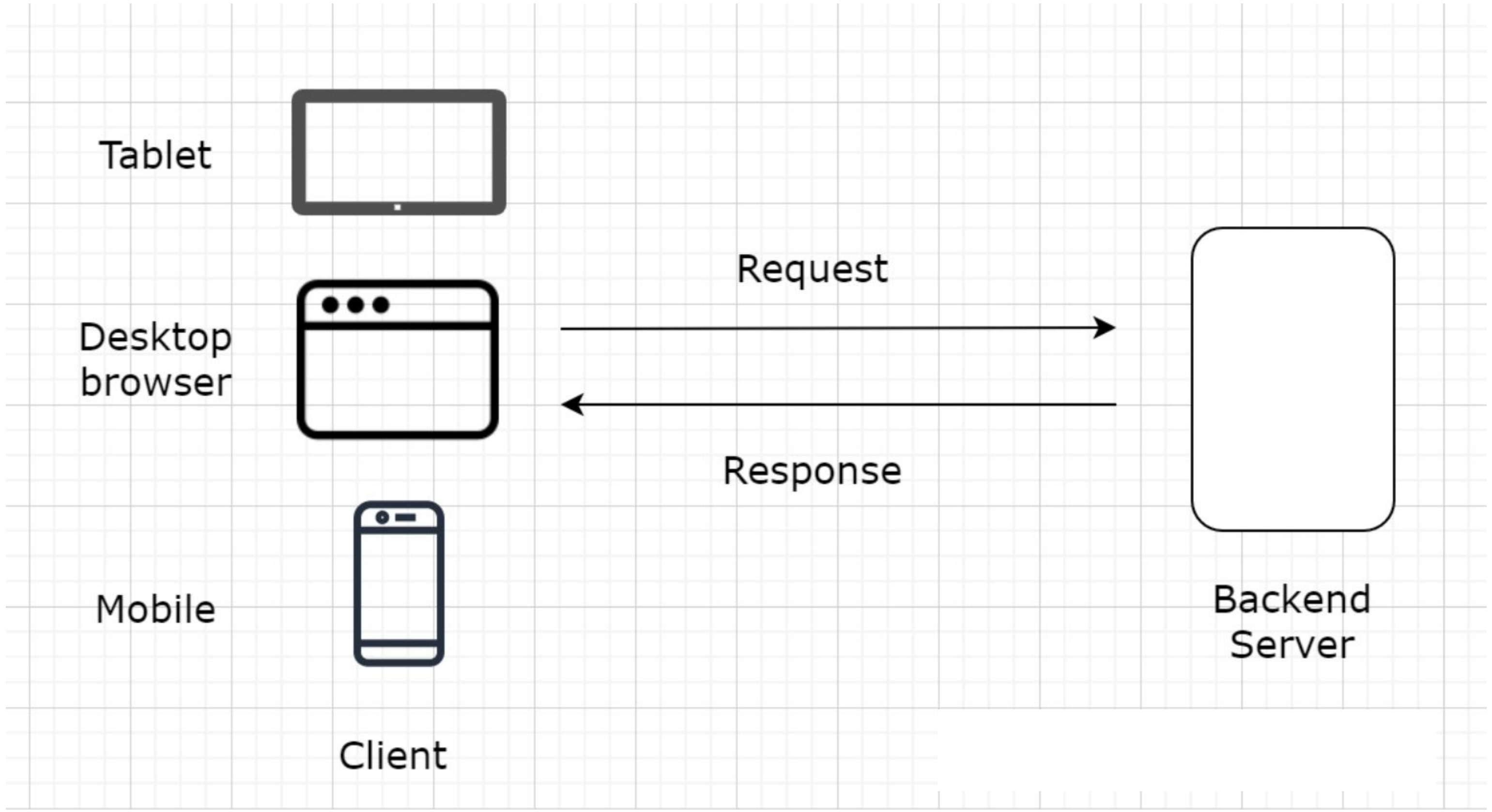
Cons of Single Tier Applications

- Application’s publisher has no control over the application.
  - Once the software is shipped, no code or feature updates can be made until the customer manually updates it
  - Example: ship a buggy game? wait for customer to update :(
- Code vulnerable to being tweaked and reversed engineered.
  - Product security for the app publisher is minimal.
  - A bad actor (with some effort) can get access to the application’s source code
    - Modifying or copying it for profit.
    - This is unlikely in an architecture where the company controls the application server and implements security to fend off the hackers.
- Performance and look and feel can be inconsistent with older machines or weird configurations of user machine
  - App rendering largely depends on the configuration+ performance of the user’s machine.
  - Can lead to bad user experience

Two Tier

- Two tiers: **Client** and **Server**
- Client:
    - Contains the user interface with rendering logic in one machine.
    - Views and rendering in client
- Server:
    - Serves code to client upon request
    - No Database
    - Example: https://www.comeshave.com/

Two Tier


Tablet
Desktop browser
Mobile
Client
Request
Response
Backend Server

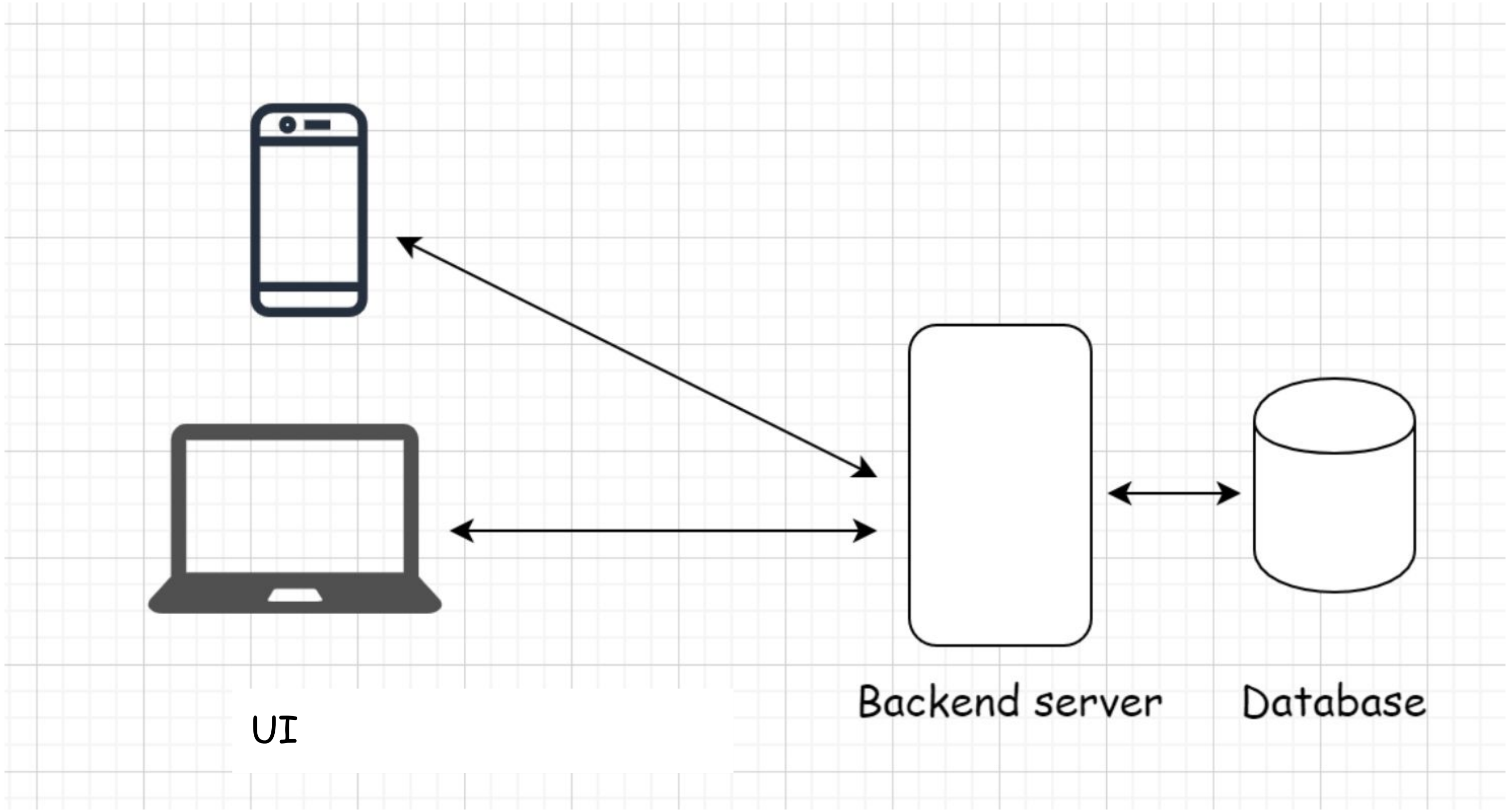
Two Tier: Pros and Cons

- Application code vulnerable to being accessed by a third person
    - In these scenarios, even if the code is accessed by a third person, it won’t cause the business much harm.
    - Since the business logic is executed on client side, backend can be simple
    - Lightweight and cheap to host
- Often used for browser and mobile app-based games.
    - The game files are pretty heavy, and they only get downloaded on the client once when the user uses the application for the first time.
    - Run in browser

Three Tier

- Tiers:
  - User interface
  - Business logic
  - Database
- All reside on different machines and, thus, have different tiers.
  - They are physically separated.
- Example: Simple blog.
  - The user interface will be written using HTML, JavaScript, and CSS
  - The backend application logic will run on a server
  - The database will be MySQL.
- A three-tier architecture works best for simple use cases.
  - Example: https://www.entirestudios.com/

Three Tier



N-Tier

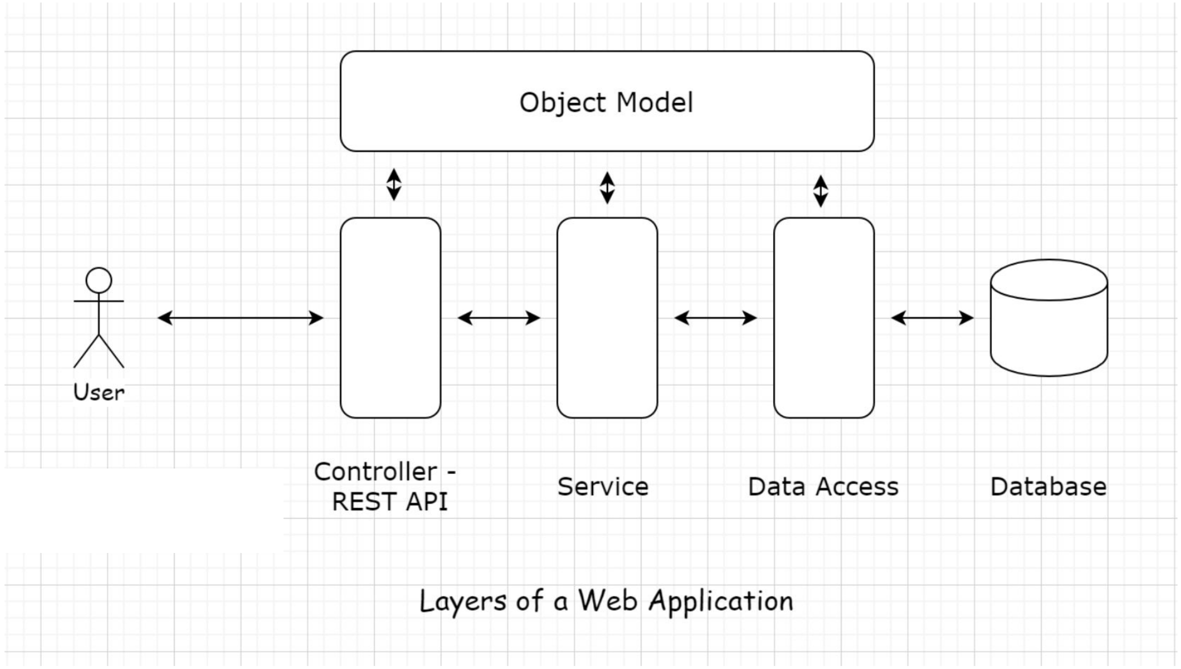
- Architecture has more than three components (beyond business logic, UI, DB).
- Examples:
  - Cache
  - Message queues for asynchronous behavior
  - Load balancers
  - Search servers for searching through massive amounts of data
  - Components involved in processing massive amounts of data
  - Components running heterogeneous tech commonly known as web services, microservices, etc.
- Large applications like Instagram, Facebook, TikTok, Uber, Airbnb, Pokémon Go, Roblox, etc.

Why So Many Tiers?

- Single Responsibility Principle
- Keeping the components separate makes them reusable.
  - Different services can use the same database, messaging server or any other component as long as they are not tightly coupled with each other.
- Scalability
- Robustness
  - If one service goes down due to machine issue, all don’t

Difference between Layers and Tiers

- **Layers** represent the conceptual/logical organization of the code
- **Tiers** represent the physical separation of components.


Object Model
User
Controller - REST API
Service
Data Access
Database
Layers of a Web Application