SOFTWARE ARCHITECTURE PART 2

Code It Up Online Vol. 10

FOR LIVE QUESTIONS

Check The Link In The Description

Resources

https://github.com/ivaylokenov/Software-Architecture-Series

LIVE STREAM TROUBLESHOOTING

- Sometimes issues happen during a live stream…
  - And sometimes disasters happen, this is how memes are born…
- If my Internet goes down and the stream stops:
  - Wait for 5 minutes, I have another one on a different network
- If YouTube is showing "stream ended":
  - I will post a new link in the comments section below this video
- If something else happens unexpectedly:
  - Well, I will add a solution to this slide during my next event… ☹
- If a major showstopper is happening – no electricity, for example
  - I will create a new event and we will schedule a new live stream…
- In any case – write to wewritesoftware@gmail.com

THE PRESENTER

- **Ivaylo Kenov – Quality Code Advocate**
  - Various job titles at the same time:
    - Organizer & Speaker @ Code It Up
    - CTO @ SoftUni
    - Full Stack Technical Trainer @ Everywhere
    - Code General @ https://docs.mytestedasp.net/
    - Meme Copy Machine @ Daily Programming Fun
    - {*Insert Job Title Here*}
  - Contacts
    - https://github.com/ivaylokenov
    - https://facebook.com/ivaylo.kenov
    - https://linkedin.com/in/kenov
    - https://www.instagram.com/ivaylokenov/
  - YouTube & Blog
    - https://www.youtube.com/c/CodeItUpWithIvo
    - https://codeitup.today/

SPONSORS

- This lecture is free thanks to our sponsors
  - Which will interrupt the lecture here and there
  - Because it takes quite a lot of personal time to prepare the materials
- You will help the initiative a lot if you visit their web sites
  - And consider their propositions to you
- I personally select various premium jobs to present them during the talk
- These are the current ones:
  - INDEAVR - https://indeavr.com
  - Americaneagle.com - https://www.americaneagle.com
  - SmartIT - https://smartit.bg

ABOUT CODE IT UP

CODE IT UP

- The Code It Up initiative:
  - Aims to provide detailed knowledge on advanced software development topics
  - Aimed at people with at least 1 year of programming experience (mostly C#)
  - Sort of acquired by SoftUni last year
- Code It Up Online
  - Free live-streamed online events (2+ hours long)
  - Led by me – mainly .NET, architecture, and infrastructure
  - We have 6 more lectures before the initiative ends
- Code It Up Workshop
  - Paid events containing theory & practical exercises for the attendees
  - No more paid lecture planned for now

THANKFUL IF YOU SHARE A STORY

- You can be extremely helpful to the initiative
- Just share a story on Facebook or Instagram during the lecture
- Make sure you tag me so that I can reshare your post - **@ivaylokenov**
- Bonus – add the **#codeitup** hashtag
- Thank you! You rock!

ABOUT THE SERIES OF EVENTS

ABOUT THIS SERIES OF CODE IT UP EVENTS

- Theoretical lectures on software architectures
  - Widely-used design patterns and concepts in production
  - Depending on your level, you may be familiar with some of the topics
  - A real-life project example
- A practical guidebook for architecting various solutions
  - 3 real-life projects on more than 70 pages
  - Legacy systems, vast data load, lots of concurrent users, working with critical data, and more
  - Please report to wewritesoftware@gmail.com, if you find any "bugs" in the book!
- And I have a lot more to add in the future!
  - You receive free updates of the book!
- MOST IMPORTANTLY – HUGE THANK YOU! <3

THE CONTENT OF THE SERIES – A FREE COURSE

- Why Software Architecture
- What Is Software Architecture?
- Unified Modeling Language
- Designing Solution Architectures
- Common Technology Stacks
- Architecture Design Patterns
- Choosing The Right Patterns
- Architecture Quality Attributes
- System-Wide Considerations
- Deployment Considerations
- Monolithic Architecture
- Domain-Driven Design
- Microservices
- Event Sourcing
- The Architecture Document
- The Architect And The Team
- What Makes A Great Architect
- Designing A Real-Life Solution

IN THE PREVIOUS PART

- Why Software Architecture
- What Is Software Architecture?
- Unified Modeling Language
- Designing Solution Architectures

- It is not required to watch the parts in order
  - But it is strongly advised!
- Get the previous recording from here:
  - https://www.eventbrite.com/e/software-architecture-fundamentals-essentials-code-it-up-online-vol-9-registration-222550182587

IN THIS PART

- Common Technology Stacks
- Architecture Design Patterns
- Choosing The Right Patterns

- Don't Forget The Optional But Practical Guide-Book
  - 3 Real-World Scenarios
  - 70+ Pages
  - Free Updates
  - Get It From The Event's Page or write to wewritesoftware@gmail.com
    - https://www.eventbrite.com/e/software-architecture-technology-patterns-code-it-up-online-vol-10-registration-244365432587

ABOUT THIS TOPIC

- HUGE DISCLAIMER! THIS TOPIC IS LIKE THE LITTLE PRINCE BOOK!
- **There are a lot of things to know about software architectures**
  - **You may or may not recognize some of the patterns**
  - **And we most probably will not mention all of them**
  - **If you are a beginner, just try to absorb what you can**
- The technology world is moving very fast
  - Some of the examples shown here may be considered anti-patterns in the future
  - But the overall concept and process stays the same
- As all my other topics – this one is super intense too!
  - Even though the lectures will be a bit shorter
  - So, give yourself time and if you get the book – finish it!
  - And don't worry! The knowledge provided here will save you weeks of reading!

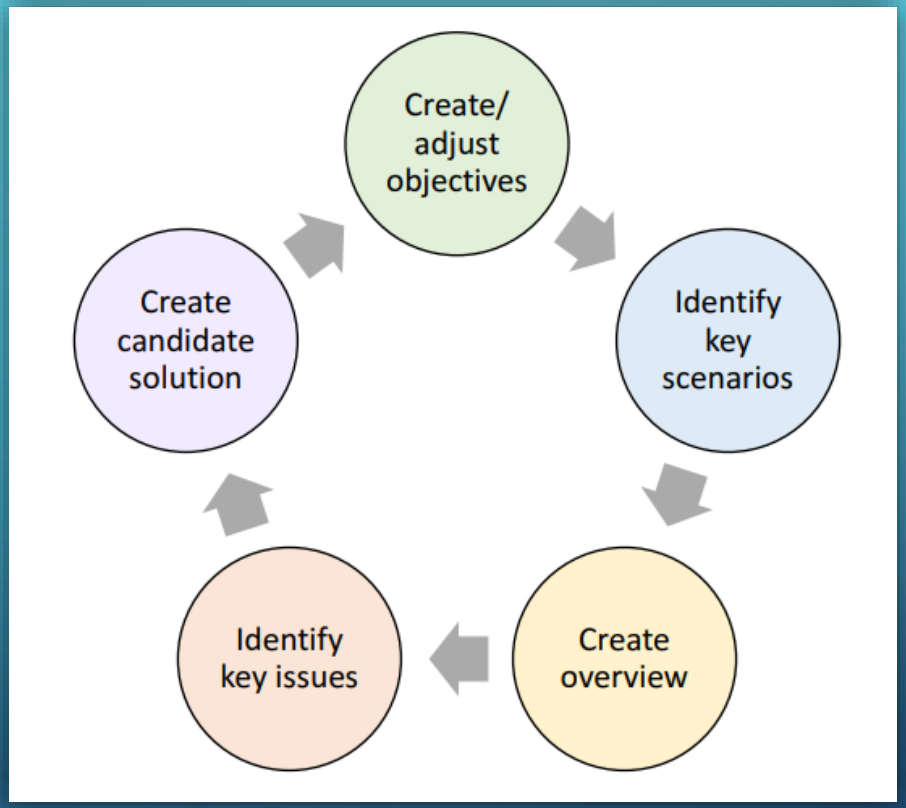
INDEAVR – THE EVENT'S DIAMOND SPONSOR

- They provide technology services focused on the Digital, Data, Cloud and Advanced Software Engineering expertise.
- They are always in search for creative and passionate people with the combination of a sharp strategic mind, emotional maturity, entrepreneurial instincts, and the ability to deliver results.
- https://www.indeavr.com/en/technology/application-services
- https://www.indeavr.com/en/careers

INDEAVR
Serving the high achievers

DESIGNING SOLUTION ARCHITECTURES

THE PROCESS FOR DESIGNING ARCHITECTURES

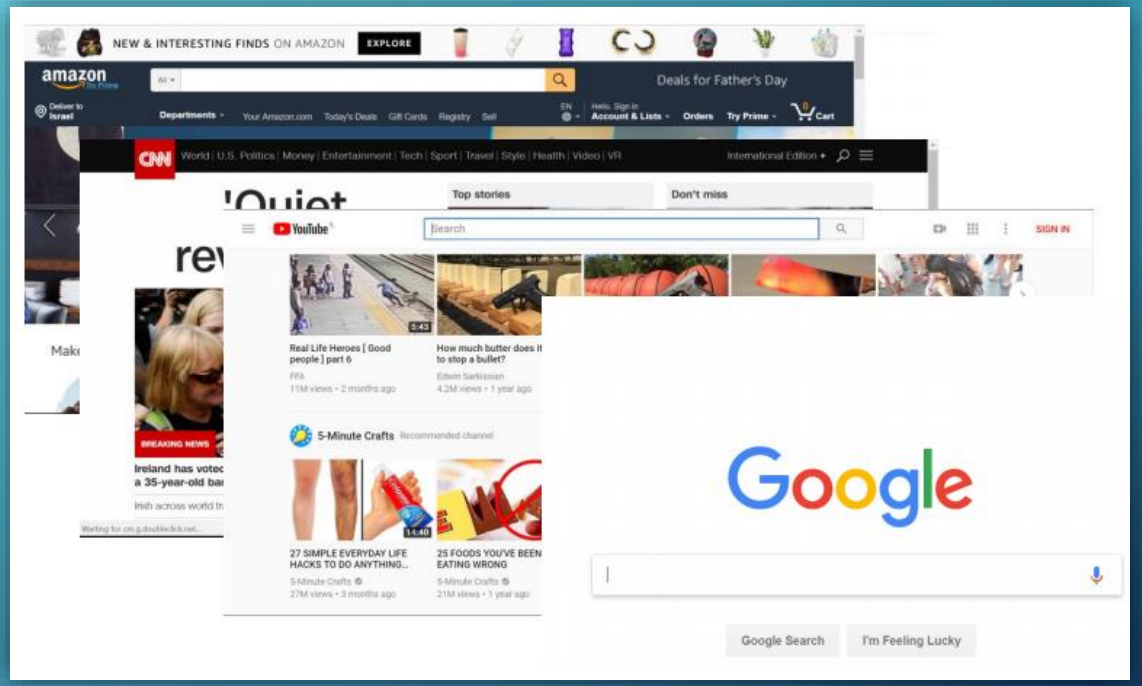


CREATE APPLICATION OVERVIEW

- Determine application type
  - Bot? Console? Service? Web? Desktop? Maybe Serverless?
- Identify deployment constraints
  - Where are you going do deploy the application?
- Identify architecture pattern
  - Layered? Component? Microservices?
- Determine technologies
  - What technologies are available? Libraries? Third-party tools?
  - A lot of factors play here… Choose wisely!
- Create your first diagram

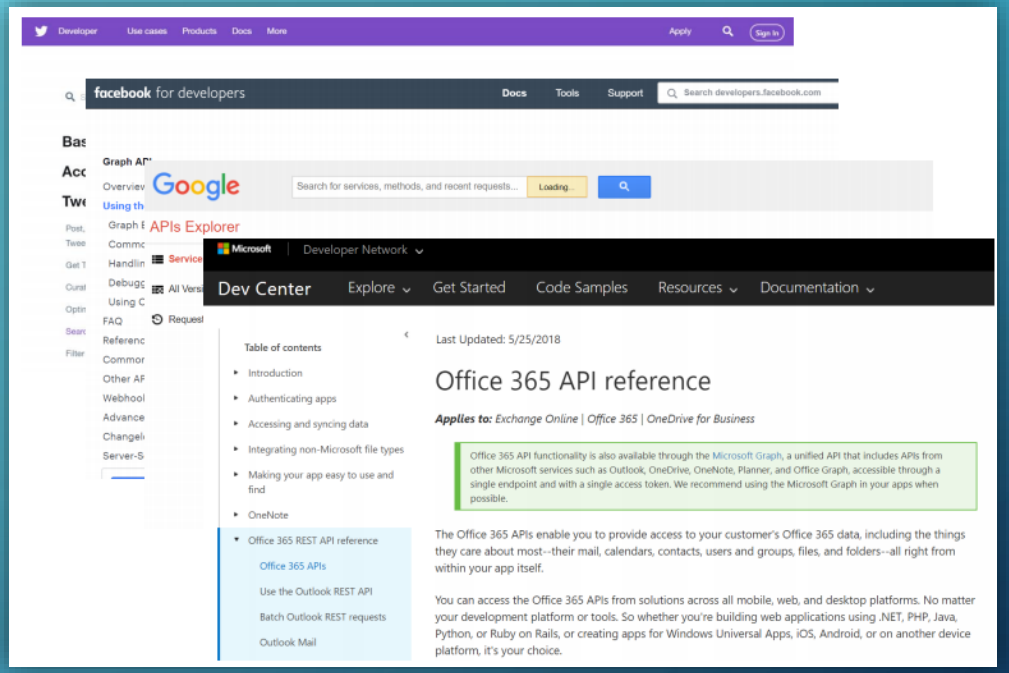
APPLICATION TYPES – WEB APPLICATION

- Best suited for:
  - User interface
  - User initiated actions
  - Large scale
  - Short, focused actions
- Request-Response based



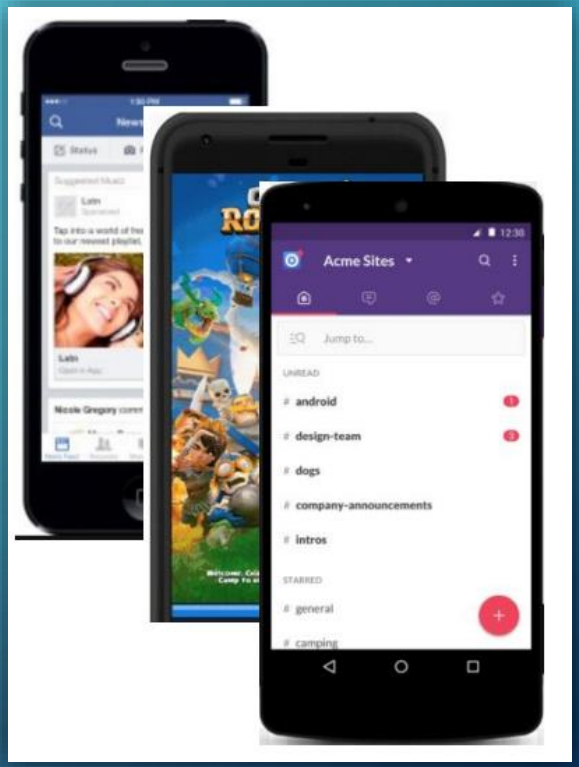
APPLICATION TYPES – WEB API

- Best suited for:
  - Data retrieval and storage
  - Client initiated actions
  - Large scale
  - Short, focused actions
- Usually REST based
- Returns data, not HTML
- Combination of:
  - URL (https://www.mysite.com/api/orders)
  - Parameters (date=10/10/2017)
  - HTTP Verb (GET)



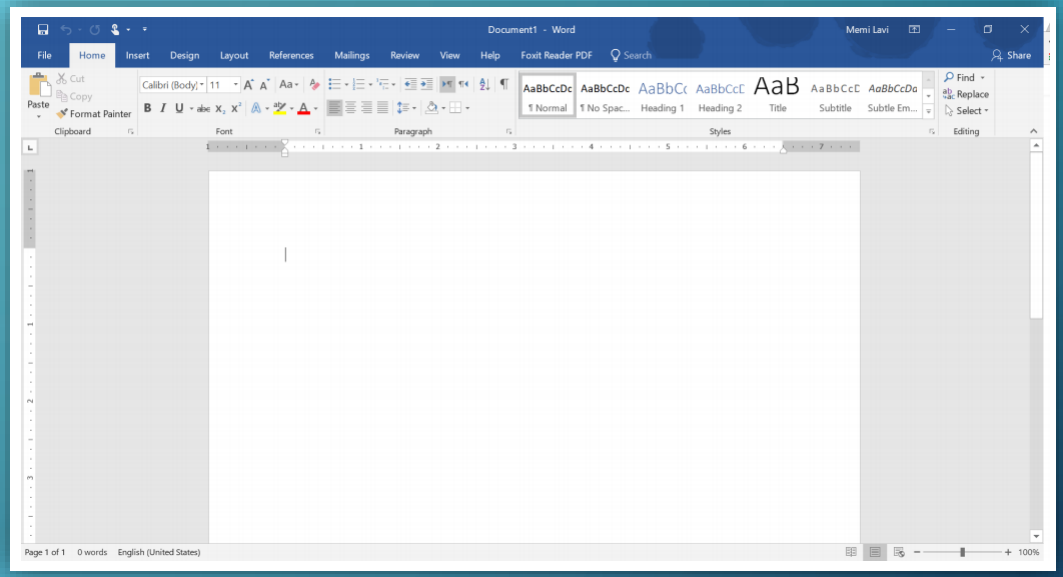
APPLICATION TYPES – MOBILE APPLICATION

- Best suited for:
  - Well, mobile devices ☺
  - User interaction (games, social apps)
  - Front end for Web API
  - Location & camera-based systems
- Usually work with a Web API



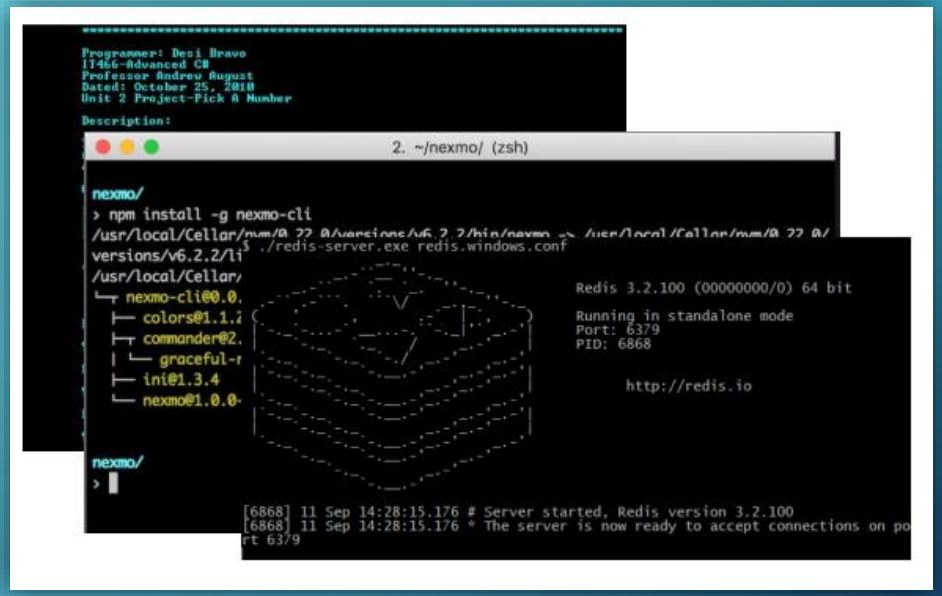
APPLICATION TYPES – DESKTOP APPLICATION

- Best suited for:
  - User centric actions
  - Gaming
- Has all its resources on the local PC
- Might connect to the web
- Great UI



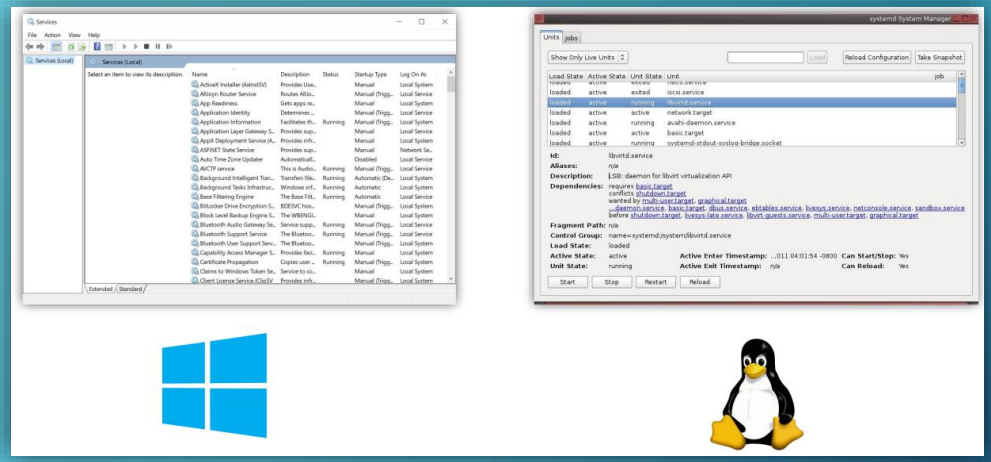
APPLICATION TYPES – CONSOLE APPLICATION

- Best suited for:
  - Long-running processes
  - Short actions by trained power-users
- No fancy UI
- Require technical knowledge
- Limited interaction
- Long or short-running Processes



APPLICATION TYPES – SERVICE

- Best suited for:
  - Long-running processes
- No UI at all
- Managed by the OS service manager



SELECTING TECHNOLOGY STACK

- This is a super important decision (especially if you are a consultant)
  - Mostly because it is almost irreversible
  - And developers get emotional to their favorite tools
- The decision must be:
  - Made with a clear mind, heavily documented and based on a group effort
- Main considerations besides performing the task:
  - Community – check Stack Overflow
  - Popularity – check Google Trends
  - Current developer skills – unknown technologies produce delay and low quality
  - Deadline – advanced technologies take more time
  - Support – developers should develop, be careful with shiny new tools
  - Products – use external and existing tools but always estimate the cost

GENERAL TIPS AND TRICKS

- Do not choose on recommendation
  - Or marketing tricks from vendors
- Always do a CBA and ROI (or at least a pros/cons list)
  - Which type system – static or dynamic?
  - What is our infrastructure?
  - Is the community using the tool?
  - Is performance critical?
  - Do we need to analyze the learning curve?
  - Can you easily hire a new team member?
- Stay rational and don't argue with your colleagues!
  - Use the right tool for the job!
  - And not vice-versa!

POPULAR TECHNOLOGIES

- Web applications – back-end
  - Mature - .NET, Java, PHP
  - Newer - Node.js, Python, .NET Core, Go
  - Serverless – Amazon Lambda, Azure Functions
- Web applications – front-end
  - HTML, CSS & JavaScript (or maybe Web Assembly, if you are adventurous)
  - Which JavaScript framework? Angular? React? Vue? Svelte?
- Mobile applications
  - Native? Hybrid? Cross-Platform?
  - The battle is development time versus capabilities
- Desktop applications
  - WinForms, WPF, UWP

BACK-END AND SERVICE TECHNOLOGIES

	App Types	Type System	Cross Platform	Community	Performance	Learning Curve
.NET	All	Static	No	Large	OK	Long
.NET Core	Web Apps, Web API, Console, Service	Static	Yes	Medium and growing rapidly	Great	Long
Java	All	Static	Yes	Huge	OK	Long
node.js	Web Apps, Web API	Dynamic	Yes	Large	Great	Medium
PHP	Web Apps, Web API	Dynamic	Yes	Large	OK -	Medium
Python	All	Dynamic	Yes	Huge	OK -	Short

MOBILE TECHNOLOGIES

	Native	Hybrid	Cross Platform
Development Language & IDE	**iOS** – Objective-C or Swift, with X-Code & iOS SDK **Android** – Java with Android Studio & Android SDK	Thin wrapper around HTML, JavaScript, CSS	Xamarin (C#, Visual Studio) React Native (JavaScript)
Access to Phone's Features	Full control, no limits	Very limited	Catch-up with latest versions
User Experience	Exceptional	Inferior	Good, with limitations

Keep an eye on PWA!

DESKTOP TECHNOLOGIES

	WinForms	WPF	UWP
Founded	2001	2006	2015
UI Flexibility	Limited	Unlimited	Unlimited, but runs in a sandbox
Learning Curve	Short	Long	Long
Runs on…	PCs	PCs	PCs, XBOX, IOT

DATABASE TECHNOLOGIES

- Key-value
  - Useful for cache, publish/subscribe, leaderboards
  - Not your primary database
  - Redis, Memcached
- Wide column
  - Each key has multiple columns without a schema
  - Cannot do joins and scales easily
  - Useful for time-series (IoT), historical records, high-write/low-read scenarios
  - Not your primary database
  - Cassandra, HBase

DATABASE TECHNOLOGIES

- Document
  - Each document is a container for key-value pairs
  - Fields can be indexed
  - Columns can be collection – query relational data without joins
  - Reading data is usually super fast, but writing data tends to be more complex
  - General purpose – applications, games, IoT, content-management
  - If you have unstructured data – a document database is a great start
  - No standard language for queries
  - Not suitable for huge graphs of data – social networks, for example
  - MongoDB, Firestore, DynamoDB, CouchDB

DATABASE TECHNOLOGIES

- Relational
  - Tables are connected through relationships
  - Requires a schema
  - ACID compliant – atomicity, consistency, isolation, durability
  - Good for stable data
  - More difficult to scale
  - Supports SQL – a query language
  - General purpose - perfect for most applications which does not have unstructured data
  - Some databases support JSON columns
  - MySQL, PostgreSQL, SQL Server
  - Modern databases like CockroachDB – designed for scale

DATABASE TECHNOLOGIES

- Graph
  - Data is represented as nodes
  - Relationships between the nodes are edges
  - Instead of many-to-many tables, you have direct connections
  - Languages like SQL
  - Better performance than relational databases when datasets are large
  - Useful for – recommendation engines and well… graphs
  - Neo4j, CosmosDB

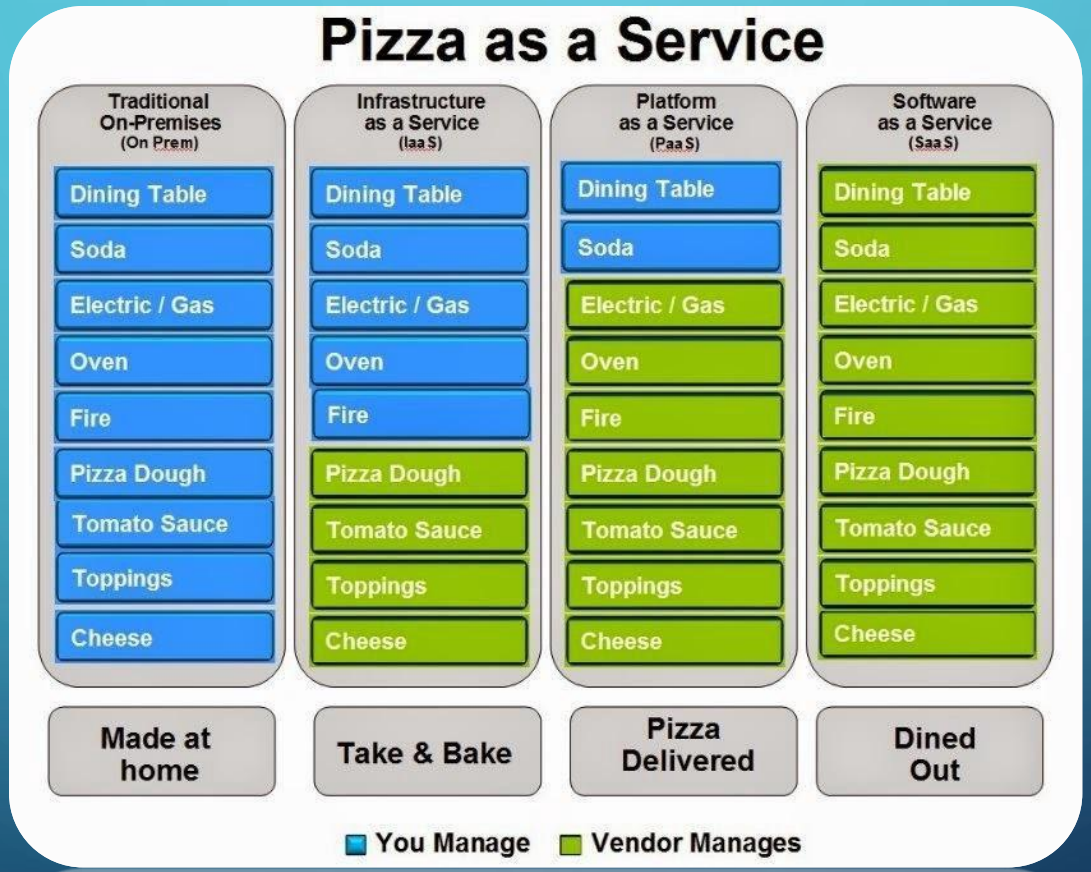
DATABASE TECHNOLOGIES

- Search engine
  - Full-text search engine
  - Like document databases but under the hood all the text is indexed
  - Works like an index at the back of a book
  - Can easily rank results
  - Useful for search engines and typeahead scenarios
  - Elasticsearch, Algolia, MeiliSearch

DATABASE TECHNOLOGIES

- Multi-model
  - You do not think about data modeling, schemas, shards, replications
  - You describe how you want to use the data with GraphQL
  - It chooses the database type behind the scenes
  - ACID compliant, extremely fast and you do not care about provisioning infrastructure
  - Useful for… everything?
    - But is it mature enough?
  - FaunaDB

CLOUD TECHNOLOGY



OTHER IMPORTANT TECHNOLOGIES

- Web servers
  - Nginx & Apache
- Containers & orchestrators
  - Docker & Kubernetes
- Message brokers
  - RabbitMQ & Kafka
- Logging
  - ELK Stack – Elasticsearch, Logstash, Kibana
- CDN

IDENTIFY KEY ISSUES

- Quality attributes:
  - System, run-time, design, user
  - Performance, stability, manageability, configurability, etc.
- System wide concerns:
  - Authentication & authorization
  - Caching
  - Communication
  - Configuration management
  - Logging & exception management
  - Validation

CREATE CANDIDATE SOLUTION

- Create a Baseline architecture
- Create a Candidate architecture
- Develop Architectural Spikes
  - Proof of concept projects to validate unknown concepts
- Create Activity, Sequence, State, and Component Diagrams
- Create Class Diagram if needed
  - They are too low-level
  - Document only the most important classes
  - Everything else should be done by the Lead Developer

DURING EACH CYCLE

- Do not introduce new risk
  - Do not introduce untested technologies and concepts
- Mitigate more risk than baseline
  - Each cycle should reduce the risk
- Meet additional requirements
  - Cover more and more requirements until you meet them all
- Enable more key scenarios
- Address more key issues
- Start communicating architecture when you exceed 50% coverage
  - If you have the feeling you are half done, invite the developers
  - You don't want to be the bottleneck

BEFORE WE CONTINUE…

HUGE THANKS FOR YOUR SUPPORT & TRUST!

- Everyone who got a paid ticket – 91 people in total! Thank you!
- Top supporter – **Georgi Kermekchiev – 100 BGN**! Thank you, you rock! <3
- Thanks to – **Nikolay, Valentin, Boyan, Sonya, Miroslava, Borislava, Plamen, Iliya, Hristo, Ангел, Ivan, David, Илина, Marin, Jordanka, Vanya, Stoycho, Stoyan, Angel, Georgi , Albena, Aneliya, Ivan, Vladimir, Teodor, Калин, Hristina, Georgi , Radoslav, Рая, Veselin, Maria, Petar, Dinyo, Hristo, Julia, Sonya, Pavel, Alexander, Dobromir, Zlatko, Teodor, Stoil, Petar, Diana, Mariyana, Ирина, Svetoslav, Mira, Yavor, Ivan, Dobromir, Dimitar, Ivan, nikolay, Борислав, Anna, Ivayla, Daniel, Velizar, Ivaylo, Vladimir, Nikolay, Petar, Hristo, Dimitar, Evgeniya, Kalina, Georgi, Pencho, Plamen, Ivan, Iva, Angel, Svetoslav, Aleks, Pavel, Hristo, Eduard, Lyuboslav, Hristo, Kiril, Mihail, Bozhidar, Pavel, Tugay, Mihail, Dimitar, Ventsislav, Borislav, Hristo, Ivan, Hristo, Vasil, Iliyan, Emiliyan**
- And everyone who supported the initiative during the years!

THESE EVENTS ARE NOT EXACTLY FREE

- I prefer to call them "Pay what you want"
  - Depending on how much you value the provided knowledge
- It takes me a considerable amount of free time to prepare these lectures
  - And I want them to be perfect and complete!
  - I put my soul in them!
- For this reason, I will be extremely thankful, if you decide to support me and my projects!
  - It is never expected, but always appreciated!
- The easiest way is via
  - PayPal: **http://paypal.me/ivaylokenov**
  - Revolut: **@ivaylokenov**

A SAMPLE PROJECT

LEARNING SYSTEM

- Here are our business requirements from the Functional Analyst:

*A **student** visits the solution and **logs in**. He or she is presented with a **list of courses**. When a student clicks a course, he or she is taken directly to the last visited lecture in that course. The lecture detail page has 3 panels and shows the **curriculum** on the left, the **lecture contents** in the middle, and a **Q & A panel** on the right. The student can use the curriculum to navigate to different lectures and **submit questions** to the instructor in the Q & A panel.*

*An **instructor** visits the solution and logs in. He or she is presented with a **list of courses**. When the instructor clicks on a course, he or she is taken directly to a **course management** page. The page shows all **questions** in the course, with the unanswered questions highlighted. By clicking on a question, the instructor navigates to a new page where he or she can **answer the question**.*

LEARNING SYSTEM

- And here are our business requirements from the CTO:

> *Lectures need to load in 1 second or less. **Performance is key**, our USP is to be the fastest platform in the business, and our students and instructors will abandon our platform if we are too slow. We also cannot **afford to be offline**. Every hour we are offline would cost our business thousands of dollars in lost revenue. And we want to scale to **millions of students**, the platform must be able to accommodate for that with ease.*

- Basically, our key attributes are:
  - Performance
  - Availability
  - Scalability

INITIAL DESIGN

- We have not learned about architecture patterns yet
  - So, it's a bit premature to start creating a detailed design
  - But we know about the process and can get started with our initial diagrams
- Our tasks for now are in the first 3 steps of the design process:

  1a. What are our objectives? What are our scopes?

  1b. Who is the key audience? Any constraints?

  2a. What are the key business scenarios? Create a Use Case diagram.

  2b. Create an Activity diagrams for the student and the instructor.

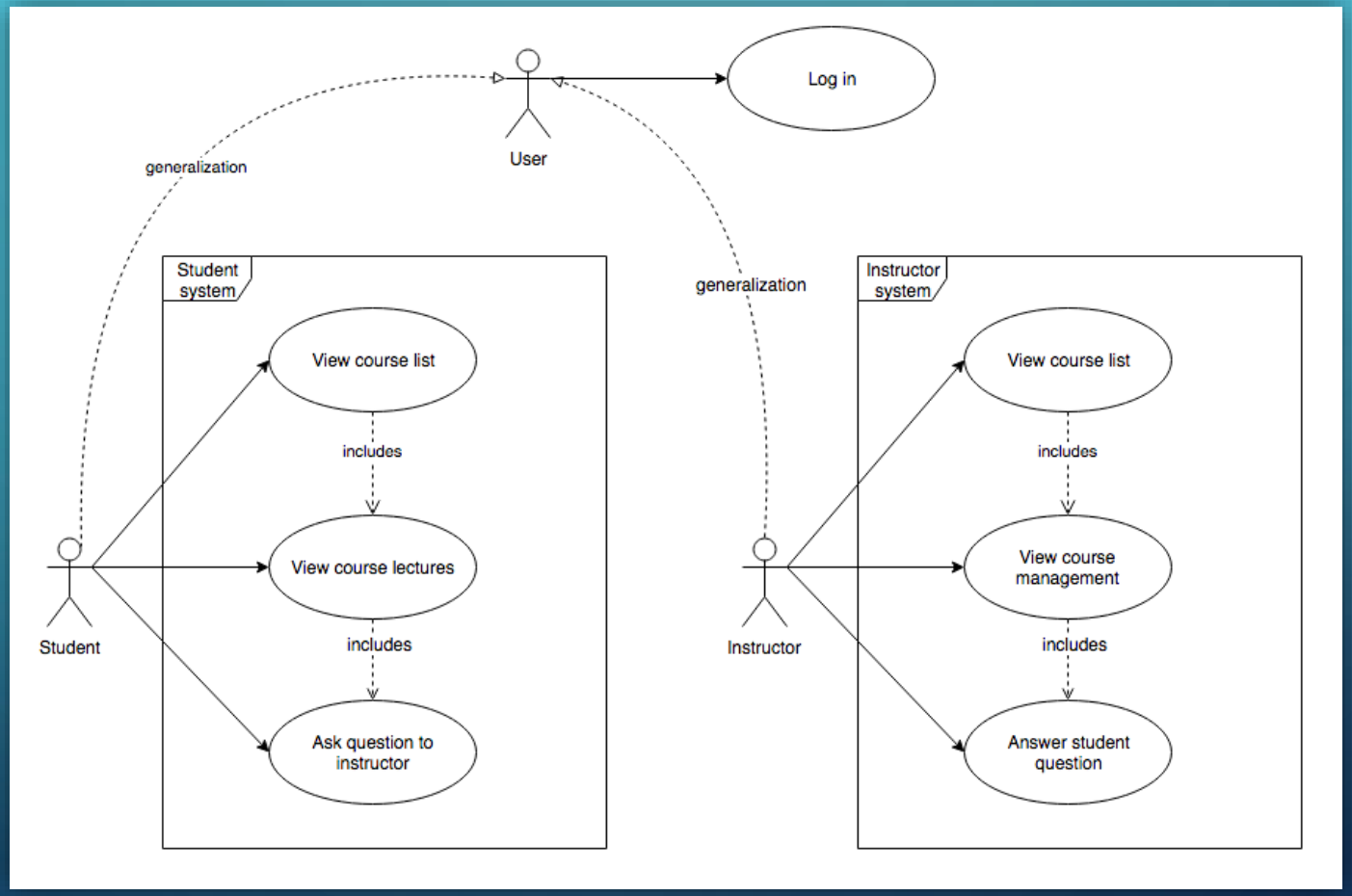
  3a. Think about the architectural overview. What is the application type?

  3b. Decide the technology stack. Back-end? Front-end?
- You can use https://draw.io or https://lucid.co/product/lucidchart for the diagrams

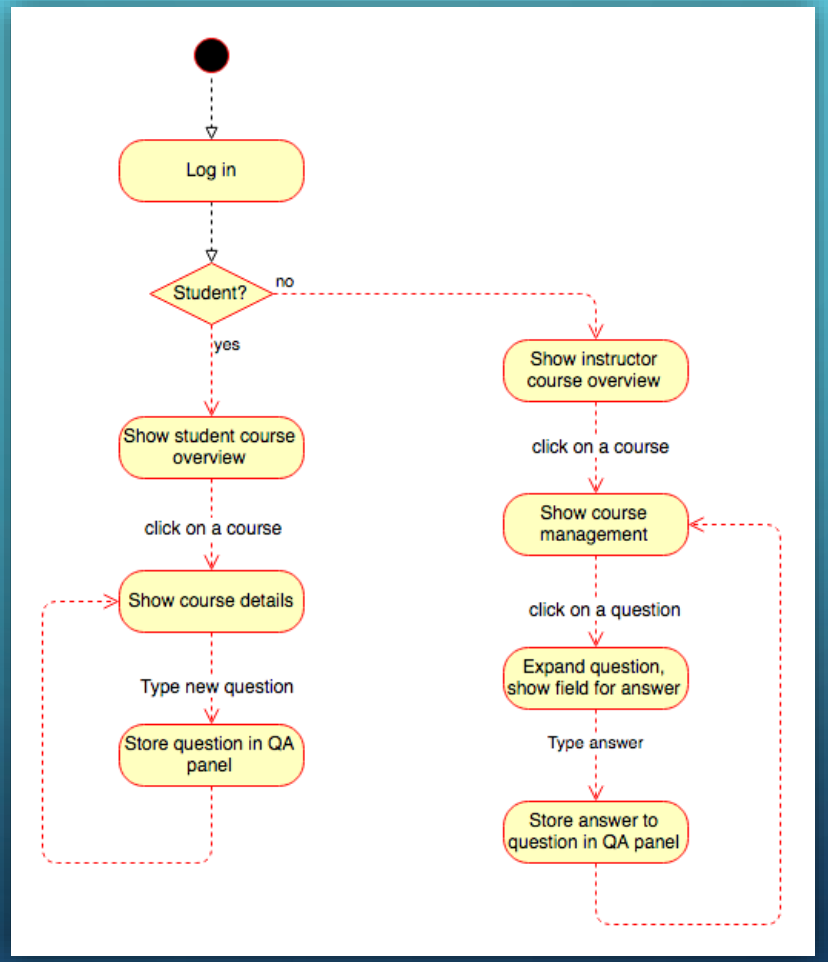
SCOPE AND OBJECTIVES

- The scope of this architecture contains:
  - Login screen for both student and instructor
  - Student interface for listing courses
  - Watching course lectures
  - Q & A panel for submitting questions
  - Instructor interface for listing courses
  - Course management page
  - Q & A panel for answering student questions
- Time schedule – 1 week
- Audience – CTO and Developers
  - We can easily include technical stuff

KEY SCENARIOS USE CASE DIAGRAM EXAMPLE



KEY SCENARIOS ACTIVITY DIAGRAM EXAMPLE



TECHNOLOGY CONSTRAINTS

- We are building the solution from scratch
  - We choose technologies based on our and the developer skills
  - There is no need to introduce an unfamiliar stack – it will increase the expenses of the project
  - If you are still building the team – you can experiment a bit more
- I am proficient with .NET and JavaScript
  - And my team has very skilled .NET developers
  - My technology stack will be Microsoft oriented
- We want millions of students and huge availability
  - On premises servers will require huge maintainability
  - We are going to use a public cloud
  - Azure is the perfect choice for Microsoft technology
  - But consider the other options as well – they may be cheaper

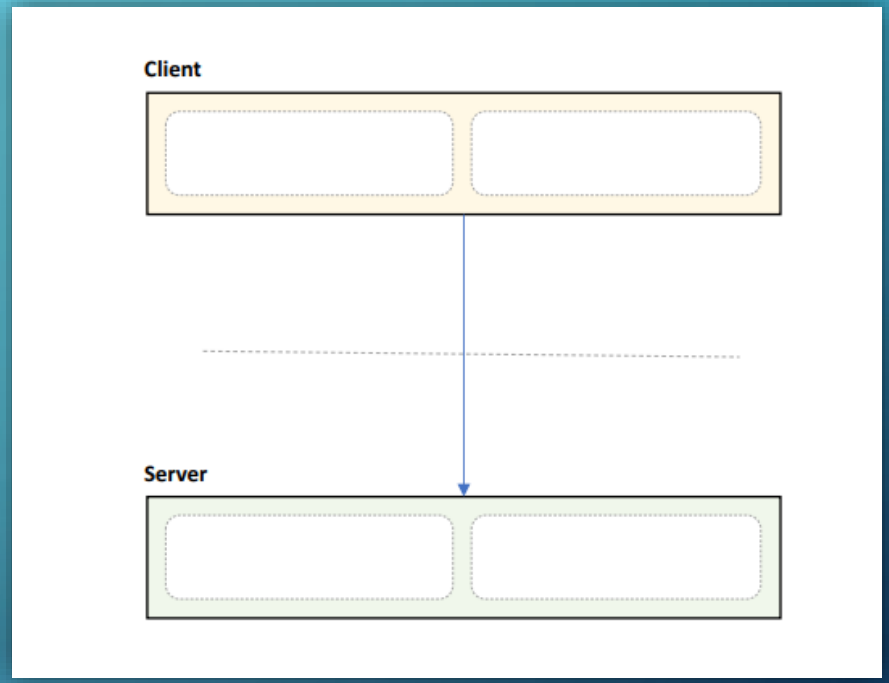
TECHNOLOGY OPTIONS

- Back-end technology options:
  - ASP.NET Core REST API – requires heavy server-side architecting
  - Serverless with Azure Functions – less architecture burden but less overall control
- Front-end technology options:
  - ASP.NET Core MVC
    - Server-side rendering is not very suitable for interactive applications
  - ASP.NET Core Razor Pages
    - Less overhead in terms of the client
  - Blazor
    - Way too new and experimental but the developers may want the bleeding edge
  - React or Vue
    - Depends on the knowledge of the developers

ARCHITECTURE DESIGN PATTERNS

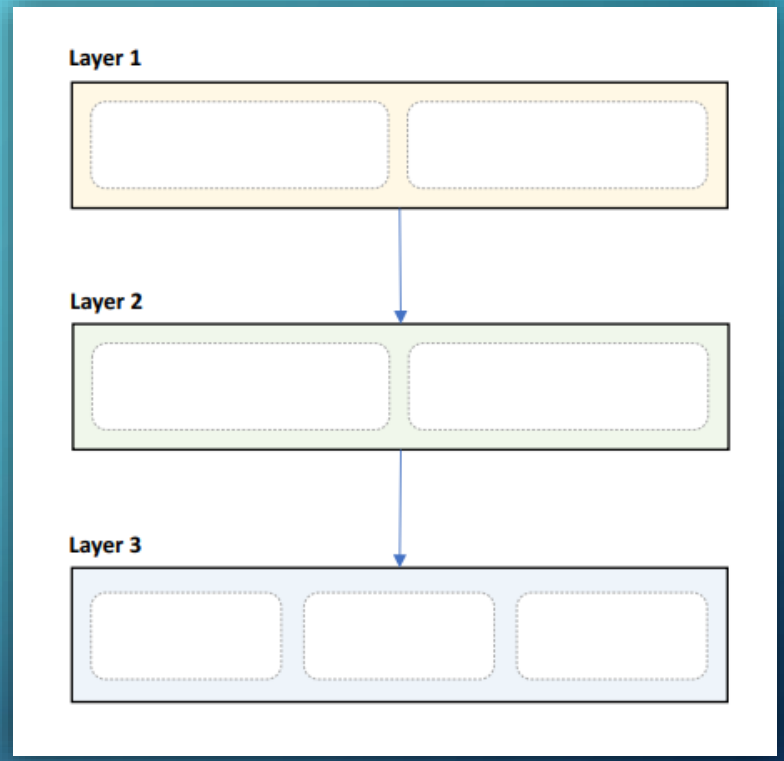
CLIENT/SERVER PATTERN – LAYERED

- Distinct client and server
- Separated by network
- Communication protocol
- Many clients, one server
- Pros:
  - Secure & Simple
  - Centralized Control
  - Easy to manage
- Cons:
  - Requires network, difficult to scale
  - Single point of failure



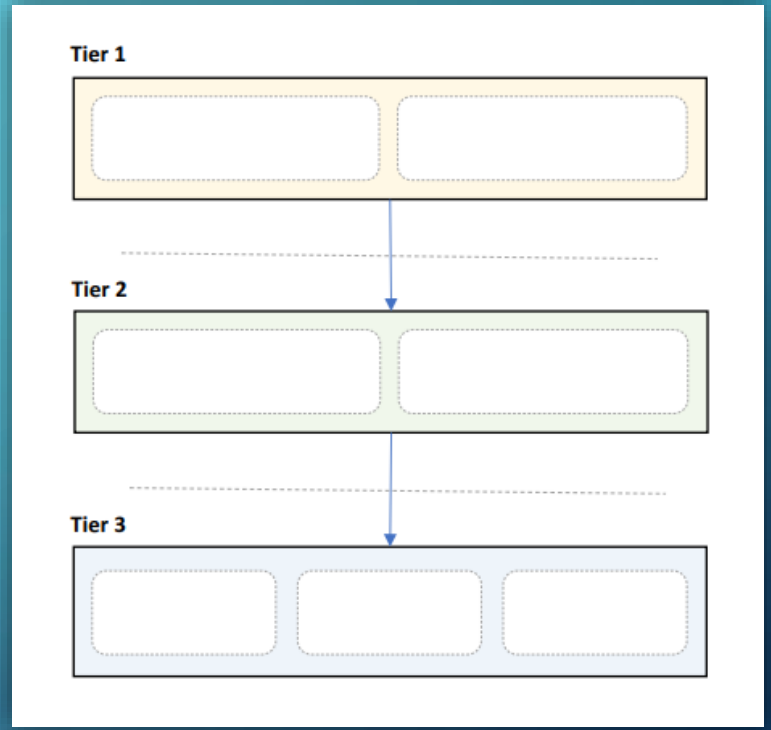
LAYERED PATTERN – LAYERED

- Strict areas of concern
- Layers may only communicate with peers above/below
- Pros:
  - High abstraction & High isolation
  - Structured communication
  - Easy to scale out
- Cons:
  - Deep call chains
  - Can hide complexity
  - May harm performance
  - Lowest layer must cover all use cases



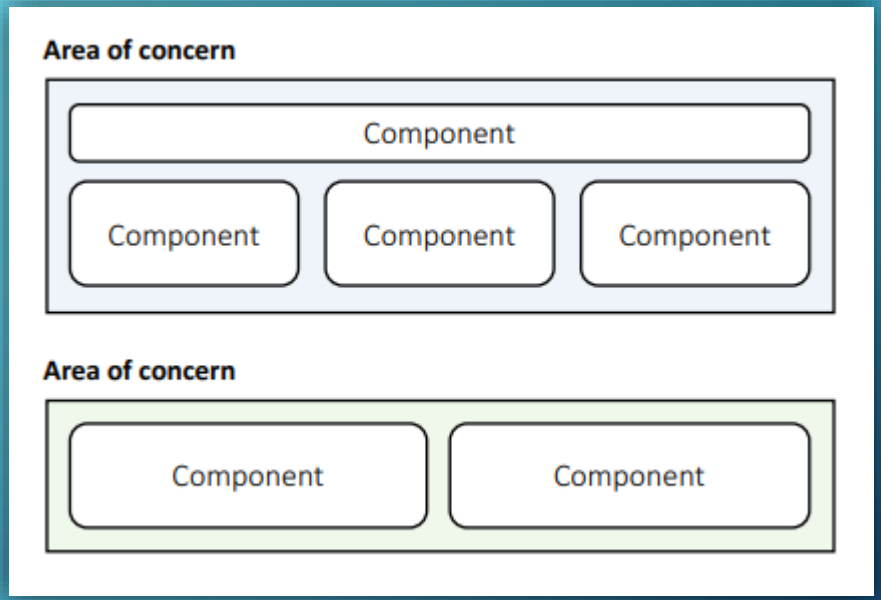
N-TIER PATTERN – LAYERED

- Layers deployed to servers
- Usually - 3 tiers, always start with them
- Communication between layers uses network
- Pros:
  - High abstraction & High isolation
  - Structured communication
  - Easy to scale out
- Cons:
  - Network = point of failure
  - Network may be slow
  - Coarse interfaces
  - Hard to debug



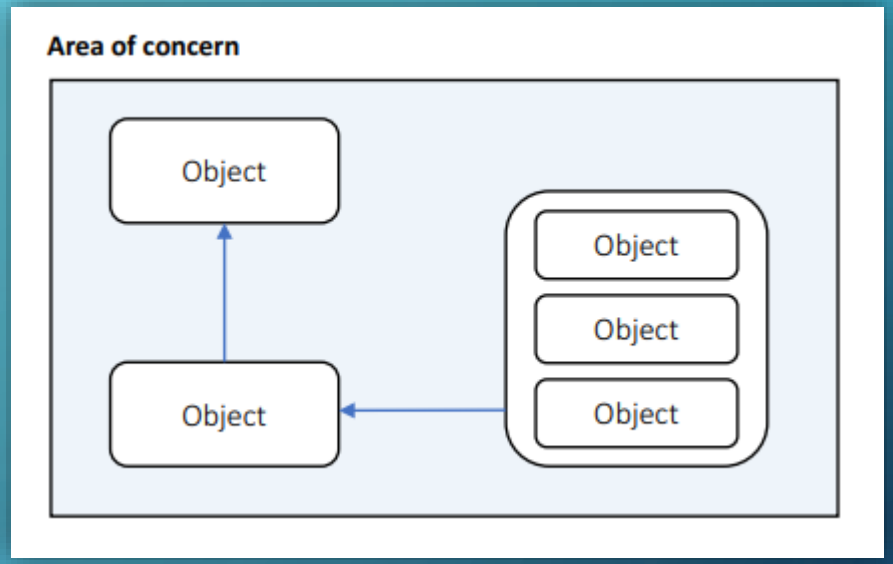
COMPONENT-BASED PATTERN - STRUCTURAL

- Components are modular building blocks of software
- Grouped into areas of concern
- Clearly described interfaces
- Containers provide additional services
- Pros:
  - Easy deployment & Allows 3rd party tools
  - Promotes modularity & Few unanticipated interactions
- Cons:
  - Coarse building blocks & Can be expensive
  - Initialization may be slow & Harder to develop & maintain



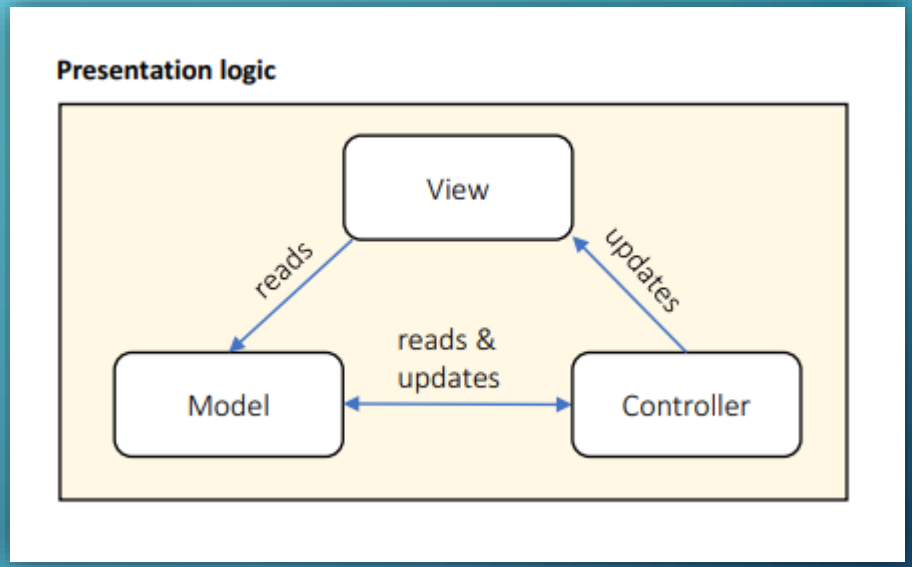
OBJECT-ORIENTED PATTERN - STRUCTURAL

- Uses classes
- Grouped into areas of concern
- Describes public members
- Uses inheritance, composition, aggregation, and associations
- Pros:
  - Easy to understand & Promotes reuse
  - Easy to test & debug & Highly cohesive
- Cons:
  - Inheritance hard to get right & Useful only if you do not have a Lead Developer
  - Many unanticipated communications & Too detailed



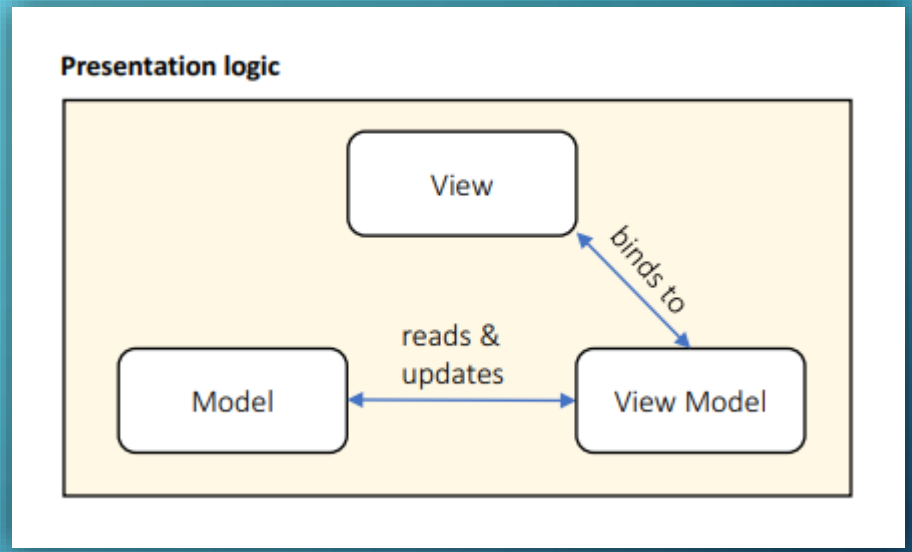
MVC PATTERN - PRESENTATION

- Designed for presentation layers
- View handles output
- Model handles data
- Controller handles interaction
- Pros:
  - Strict separation of concerns
  - Scales well
- Cons:
  - High overhead
  - Scattered code
  - Hard to data-bind



MVVM PATTERN - PRESENTATION

- Derived from MVC pattern
- View handles output
- Model handles data
- View Model is binding source
- Pros:
  - Strict separation of concerns & Scales well
  - Easy to data-bind
- Cons:
  - High overhead
  - Scattered code
  - Controller not separated

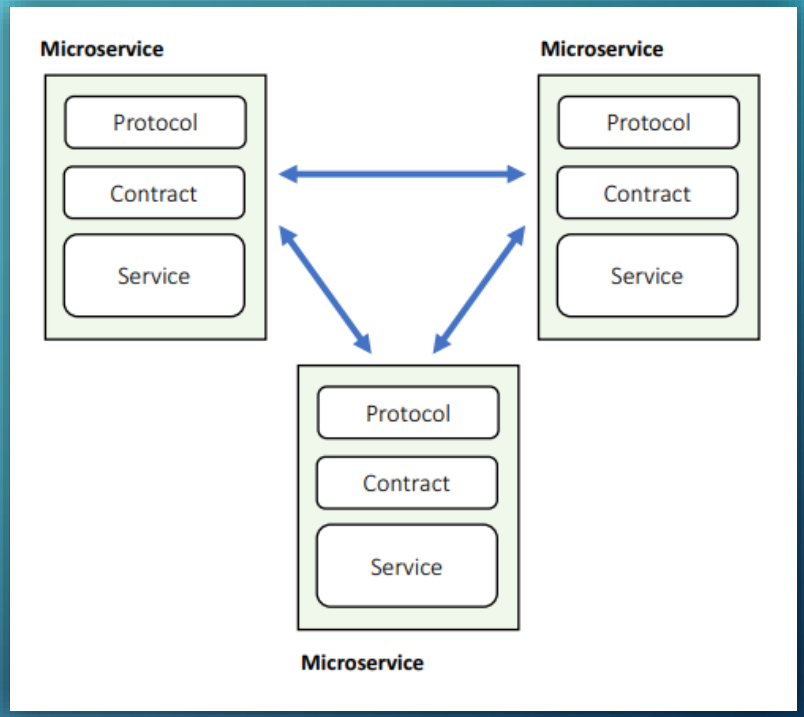


SERVICE-ORIENTED PATTERN - SERVICE

- Discrete business services
- Use network to communicate
- HTTP, XML, SOAP, Binary…
- Pros:
  - Business domain alignment & High abstraction
  - Discoverable, resilient, Allows 3rd party libraries & Cross-platform
- Cons:
  - Clients must handle slow, offline network
  - Coarse interfaces
  - May harm performance
  - Security issues

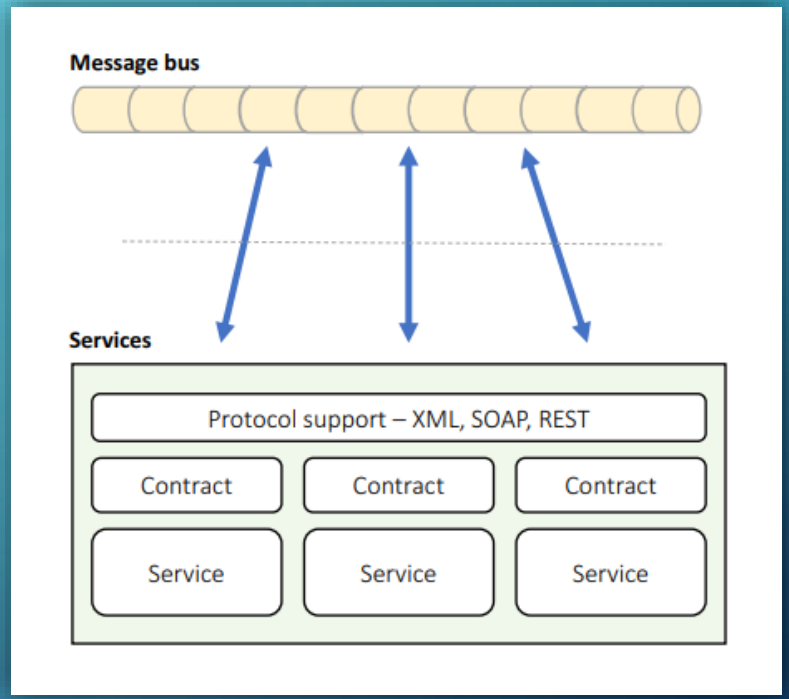
MICROSERVICE PATTERN - SERVICE

- Services calling services
- Can use fast private network and RPC
- Deployed on multiple servers
- Pros:
  - Modular & Reduced abstraction
  - Discoverable, resilient & Less coarse interfaces
  - Can use fast network
- Cons:
  - Must cope with slow, offline network
  - More unanticipated communications
  - Hard to do transactions & Hard to test, debug, deploy



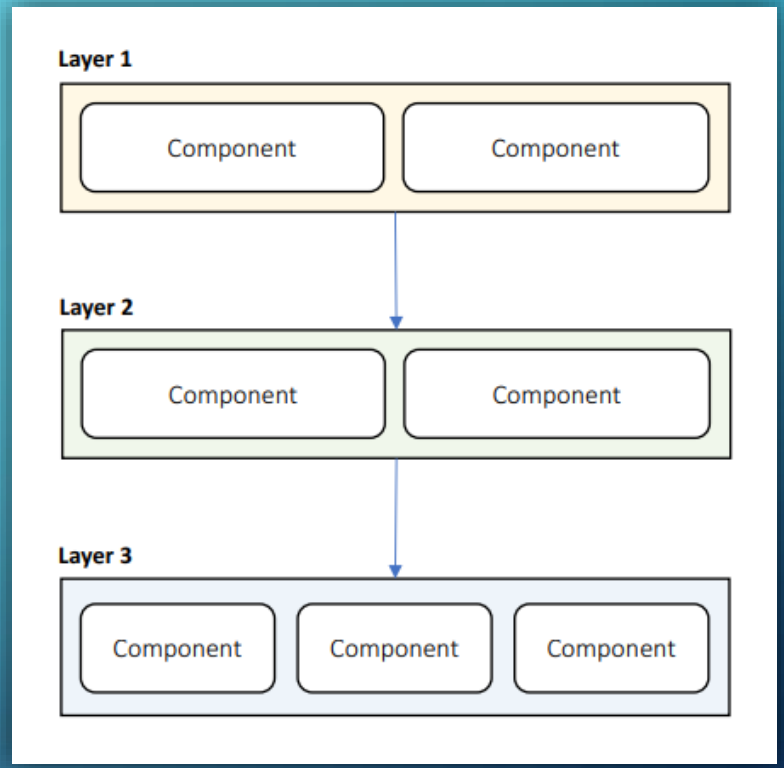
MESSAGE BUS PATTERN - SERVICE

- Services connected to shared data bus
- Uses messages for communication
- Supports discovery, failover
- Pros:
  - Easy to extend & Simple communication
  - Very flexible & Easy to scale
  - Easy discovery, failover
- Cons:
  - Bus = single point of failure
  - Coarse communications
  - Can be slow & Hard to test, debug



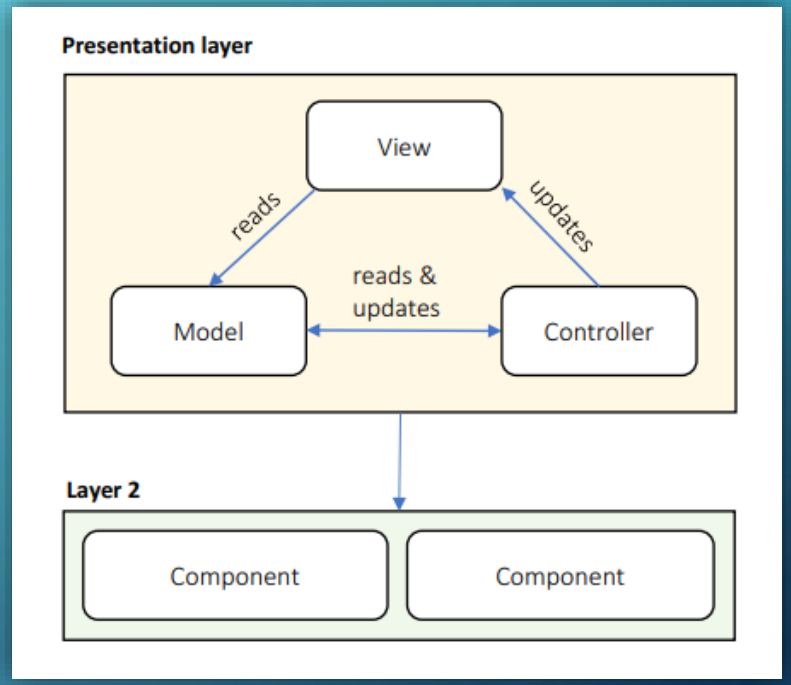
COMPONENTS IN LAYERS - HYBRID

- Layers containing components
- Benefits of components
- With structured communication, isolation, easy scaling & testing



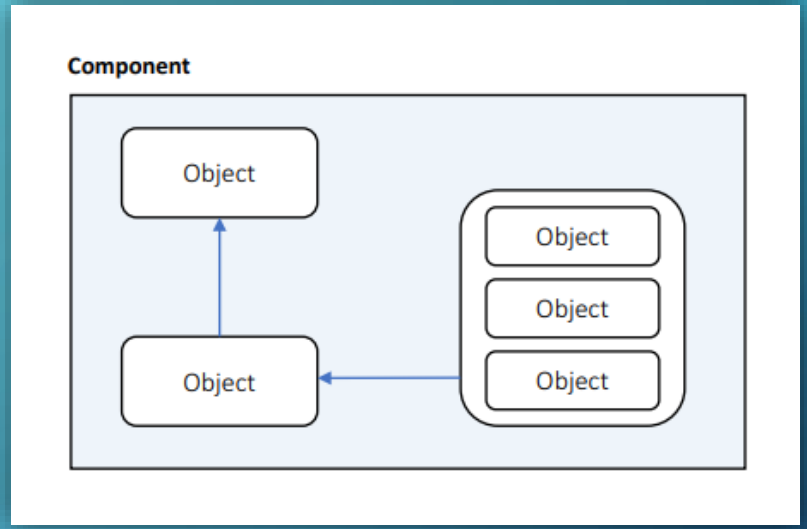
MVC IN PRESENTATION LAYER - HYBRID

- Presentation layer with MVC
- Benefits of layers
- With separation of concerns, presentation code scalability



OBJECTS IN COMPONENTS - HYBRID

- Components contain objects
- Benefits of components
- With cohesion & extensibility
- You are probably using this pattern already



WANT MORE PATTERNS? YOU HAVE THEM!

- Pipe and filter
- Publish/Subscribe
- Stream processing
- Multitenancy
- Sharding
- Circuit breaker
- Event-based architecture
- Many, many more!

MIXING OF PATTERNS

- A software architecture is just a mixture of patterns
- You can safely nest architectural design pattern inside of each other
- For example:
  - Microservices with components inside them
  - Microservices with a layered pattern inside them
- Create your own hybrid patterns and use them in your design
- If the pattern works for you, implement it!

QUESTIONS FOR THE LEARNING SYSTEM

- We already chosen an application type and the technology stack
- We need to pick architectural pattern and draw a diagram

1a. Choose an architectural design pattern. A great place to start is the layered pattern.

1b. Create a diagram with presentation, service, business, data and systemwide layers.

1c. Choose a pattern for the presentation layer. MVC? MVVM? Components?

2a. Consider authentication, caching, communication, configuration, logging, exceptions and validation.

2b. Consider all third-party tools to help you with these cross-cutting concerns.

3a. This is our first baseline architecture. Think about the components in the layers.

3b. Create diagrams for each layer and describe their public interfaces.

3c. Make sure you keep it simple at this stage. You can go into more detail in later cycles.

4. Design the business domain entities. Create a class diagram and define their public properties.

FINAL WORDS BEFORE Q & A

SUMMARY

- Common Technology Stacks
- Architecture Design Patterns
- Choosing The Right Patterns

- Don't Forget The Optional But Practical Guide-Book
  - 3 Real-World Scenarios
  - 70+ Pages
  - Free Updates
  - Get It From The Event's Page or write to wewritesoftware@gmail.com
    - https://www.eventbrite.com/e/software-architecture-technology-patterns-code-it-up-online-vol-10-registration-244365432587

IN THE NEXT PART

- Choosing The Right Patterns
- Common Design Solutions

- Register here:
  - https://www.eventbrite.com/e/software-architecture-common-design-choices-code-it-up-online-vol-11-registration-251121690737
- Don't Forget The Optional But Practical Guide-Book
  - 3 Real-World Scenarios
  - 70+ Pages
  - Free Updates
  - Get It From The Event's Page or write to wewritesoftware@gmail.com
    - https://www.eventbrite.com/e/software-architecture-technology-patterns-code-it-up-online-vol-10-registration-244365432587

OTHER GOODIES

- You can check the Code It Up blog and subscribe:
  - https://codeitup.today
- You can watch some of the free videos:
  - https://www.youtube.com/CodeItUpwithIvo
  - Clean code & The art of testing
  - Docker, CI/CD, Redis, Elasticsearch
  - And many more…
- The source code in all free lessons is available on Patreon:
  - https://www.patreon.com/ivaylokenov
- Unrelated to the IT sector but check out my art T-Shirts:
  - https://way-ve.com/
  - Use CODE10 during checkout for 10% discount

PAST CODE IT UP EVENTS

- You can get the recordings of the past C# events from the event page:
  - C# Async-Await In Detail
  - The C# ORM Battle
  - Docker – From ABC To XYZ
  - Identity Server Demystified
  - Eventual Consistency Done Right
  - Let's Get Functional With C#
  - C# API Scenarios – REST, GraphQL & gRPC
- Past workshops are also available:
  - C# Multithreading
  - Domain-Driven Design With ASP.NET Core
  - Kubernetes For Web Developers
- If interested, you can write to wewritesoftware@gmail.com

ANY QUESTIONS?

- You can support me and my projects:
  - Via PayPal: http://paypal.me/ivaylokenov
  - Via Revolut: **@ivaylokenov**
  - On Patreon: https://www.patreon.com/ivaylokenov
  - On Open Collective: https://opencollective.com/mytestedaspnet
  - Via Buy Me A Coffee: http://buymeacoff.ee/ivaylokenov
  - Crypto: http://bit.ly/ik-sponsors
- Never expected, always appreciated!
- Make sure you check my sponsors!
  - INDEAVR - https://indeavr.com
  - SmartIT - https://smartit.bg

THANK YOU!

RESOURCES:

HTTPS://GITHUB.COM/IVAYLOKENOV/SOFTWARE-ARCHITECTURE-SERIES