Software Architecture & Design

Folio 3 — Regional Director, UEP

Senior Software Architect.

- heads app development → focuses on custom development.
- 16 years of experience.
- worked at Ibex for 6 months
- class participation + do actual work ⇒ Grade A
- Software Architecture:
  - blueprint
  - not talk about detailed components or how we will do this work.
  - e.g. there will be a cloud, database, yes/no data warehouse, serverless/server-ful, event-driven or three-layer etc.
- Software Design:
  - implementation level details
  - how to implement architecture e.g. what will be classes, types of classes etc.
- The goal of an ecommerce application or nature of such an application is to make it scalable so we need to choose an architecture that allows scalability.
- Singleton, Observer pattern
  - ↳ these are design patterns.

- Being a software architect, you should have an idea of the ~~application~~ application domain's challenges, design patterns, project manager, pain areas of customer (you should be able to technically translate them), cost of design.
- Software Architecture is saying you want serverless system; Software Design is saying whether to use Azure etc for implementing serverless system.
- KYC → Know Your Customer.
- Food for thought: kya aik jahaaz chaand par jaa sakta hai woh bhi fuel kay baghair?

  Answer: Possible to hai, ab socho, Ash.
- Software architect understands or decides how complex or flexible to keep the system.
- "Ghalti ko accept karnay ka courage paida karein."
  → life lesson by Sir Anees
- Architect deals with strategic decisions.

- What helps us architect a plan are use-cases and scenarios ki realization.
- D/f b/w functional and quality requirements : quality requirements include covering all use-cases, corner cases.
- External confidence vs. internal confidence
  - ↓ client's confidence in you & your design
  - ↓ your team's confidence in your & your design
- Design diagram is a part of your documentation and needs to be showed to the client whether they are technical or not.
- Design documentation certification : Togaph???
- UAT : user-acceptance testing.
- Dev testing : development complete (basic use cases) and then just basic testing.
- Golden Rule : Developer cannot do testing. QA cannot do development
- Dev testing vs. unit testing.
- We will not know if the implemented architecture is good until it is implemented → limitation (lack of analysis....)
- Design reviews are used for senior-level promotions. In this, if you showcase a design that is not in production, the first question you will be asked is that it's not in testing. (so you're fired hehehe!)
- Design mein coupling nahin honi chahiyay ; classes should be independent.
- Architectural Design Principles :
  1. Separation of concerns → each part is responsible for separate concern → no coupling → no overlap ≠ b/w modules ; everything independent. → e.g. Repository design pattern, ORM, N-tier architectures achieves this separation of layers.
     - Presentation Layer (display data)
       - ↑ Business layer (business logic/rules)
       - ↑ Data Access layer (persistent storage ; take/give data)
  - Middleware : two separate systems with separate interfaces want to exchange data so they use this.
  - Decoupling services also allows us to assign separate teams to work independently on the same product.

- Aspect separation of concerns: a thing (logging) you are using throughout your application for many services (these are called cross-cutting concerns) but it has a separate module for it.

② Modularity: similar functions grouped into a system of independent modules → in layers context, every layer could be a separate module.

③ Abstraction: what a class does.
- e.g. interface (Note: how a class does something is called implementation).

— High coupling → bad (dependency)
High cohesion → good (relationships or links / concerning each other).

— "Coupling aap as much as possible reduce kardein lekin uski kabhi zero nahin hogi. Zaisi si baat hai unki koi relationship hai to hai." → Cohesion vs. coupling.

— One class, one purpose → SRP

— ~~If the same class~~ D/f b/w interface & abstract class

— LSP: even if you pass child in place of parent, the code should still work. And otherwise, this should not even allow child to be passed.

— ISP: every time you are forcefully having to implement an interface method you don't want to, you are violating this principle → huh??

— Dependency Inversion Principle: high-level module should not depend on low-level module.

— NOTE-TO-SELF: Do proper error-handling once; SHUT HIS DAMN MOUTH!!)

— Even if you feel that code is exception-giving-unprone, still write exception-handling block.

— Catch errors on high level in business layer for errors in data access layer.

— Code should first be functional. Search for already implemented solutions. Then verify. Then implement. Then look at code to see if it's following design principles.

— Most important to do SRP and OCP.

— Logging types: DBlogging, file logging, ELK.
elastic, kabana

- Requirements gathering; Business objective outlined; Pain points identified

⬇

Define scope

⬇

Evaluate (MUST) technologies, then make pros and cons list of each option, then choose.

⬇

Communicate trade-offs, feasibility to client properly.

⬇

Implement

⬇

Review, review, review design!! Incorporate changes / fixes to design ASAP → it is for the best.

- LSP: replacing child with parent without any modifications.
- Fault tolerant: will the entire system shut down on an error?
- UML: visual way of explaining design
  - learn all 5 UML diagram types → NOTE-TO-SELF !!!
  - Use case diagram → must know notations; this diagram tells us features.
  - Designers / Industry people are VERY impressed by knowing all design diagrams.
  - Class diagram: also mention signatures.
  - Composed object's life is connected to parent's life. Aggregated object's life is independent of parent's / class life.
  - Aggregation & composition → diamonds.
  - State design pattern → also a design pattern that exists

Architectural Style & Patterns

- Observer pattern → search!

② Client server architecture:

— Main components:

client, server, network

- client → generates requests, can be more than one
- server → responds to requests, can also be more than one
- network → connects client & server

— Presentation/UI layer (where data is showing)

↕

Processing/Business layer (where data is processed)

↕

Data Access layer (where data is persistently stored)

These are layered to make maintaing our application easier. Maintain separation of concern in our application.

— Client-server architecture

- 2-tier
  - Thin client → application processing on client-side
  - Fat client → app. processing done on server side
- 3-tier

Not good practice to distribute logic evenly on client & server.

— Thin client prefered for applications where user experience needs to be fast b/c data is rendered quickly. Example is web browser. Disadv is that it only works if there is a network connection.

— In both thin & fat client models, server is used for data management. In thin client, application processing is also done on server.

— Fat client example is any app that needs to be install-ed in general. Disadvantage is that security is compromised because software can be cracked, and second problem is that updating app is an issue because you can not make sure everyone updates it and until this is done, it is a security threat. (This is why we keep "force update" option).

— Is it worth it? Is life really worth the effort of getting up in the morning each day and striving for something I don't even know I want? Is it easier to kill oneself? Easier, for sure. Cowardly? So much more...

— Fat-client model problems is summarized as

difficult to maintain (maintain-ability is the issue).

— What is greater? The loss of the chance at being more than your -self, or losing yourself in the process of being a new self? Who do I discuss these thoughts in my head with? I am surroun-ded by such intellectually mundane company at this university. Not one individu-al I know who is trained in the etiquettes of thought and speech...

— ATM → thin-client

— "All those days, watching from the window, all those years, outside looking in..."

— All kiosks are thin-client.

— Adv. & disadv of 2-tier are adv. & disadv. of thin & fat client model.

— In 3-tier architecture, each layer has a separate process -or and these processors may or may not be on the same machine.

— 3 tier is easily scalable ca-use all layers are separate. 2 tier can also have 3 application layers but in 3 tier, each layer can also be has-ted separately as well (completely independent).

— MVC is NOT 3-tier architecture.

— NOTE-TO-SELF: Don't be the kind of girl that draws stars in the corners of her notebooks.

— 3-tier components can be: Client(s) → Web server → Database server

— If you think scalability is a goal later on, choose 3-tier.

(2) MVC Architecture Pattern

— goal of using MVC is to achieve separation of concern

— MVC is NOT 3-tier. (IMP!!!)

— Model: it is NOT just the class that defines entity & its attributes. Everything concerning the manage-ment of data is model.

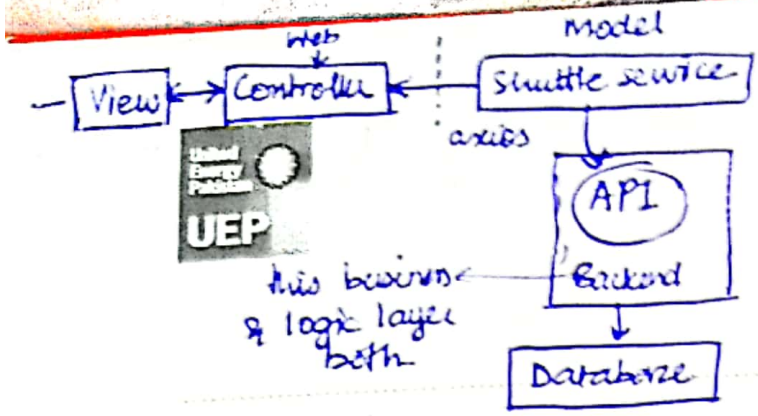
— View: shows data?

— Controller: communicates b/w model & view.

— MVC is only part of the Presentation layer of 3-tier.

— Controller should have NO business logic.

— Business layer or service layer will be the one applying business logic and then passing it to contro-ller.



- NOTE-TO-SELF: Ask Sir to explain MVC again ....
- Using MVC, we can easily hand over view to UI engineer b/c view is independent of complex code logic this way.
- NOTE-TO-SELF: You had to learn TDD, dumbo....
- MVC is usually applied on 3-tier architecture.
- API layer is View ... ??? in Django whereas View is actually supposed to be UI in MVC so Django is not following MVC really.
- Think of what the impact is to the one who's paying you (business company); don't just give them a technical solution.
- Main advantage of MVC is being able to do unit testing (b/c everything must be independent to be able to do unit testing).
- "Kuch cheezein aap chahtay hain lekin nahin kar paatay kyun-kay aap ko woh scenarios nahin milta jiss mein aap kar sakein." ~ Sir Aneeq.
- Unit testing does not eliminate the need for QA.
- Model: classes and entities where data is fetched and set.
- Controller: middleman that takes data from model layer and gives it to view.
- Model layer has a model class whose object is passed.
- View: represents data. (Same data may have d/f views).
- See GeeksForGeeks code examples shown in class
- Controller sets the data in View.
- <Model value = model>

    <text value = model.name> ← variable

- Where user interaction updates model and we do not need to show it on view, we use

MVC (this is best use of MVC). & if this not the case and view is being updated as well and we still use MVC, we will have to mock both the UI & controller for testing and so unit testing we will not be able to do easily.

- One controller can corresponds to many views.

- صبر اور انتظار، پر کبھی مایوسی نہیں۔ یہ ہے دعاؤں کا نتیجہ

- MVP: model-view-presenter
  - MVP needed over MVC b/c view and controller are too tightly coupled (one controller, multiple views). & in MVP one presenter corresponds to exactly one view.
  - Model does not set views data this time, model gives data to ~~view~~ presenter and presenter gives it to view.
  - Presenter is connected to both view & model.
  - & the Android example given, main activity is the view.
  - MVP will have a HUGE presenter layer if SRP is not followed properly.

Midterm:
- MCQs • One/question to two • 30 minute exam • No theory (definition, benefits, disadv) etc. • Apply this principle or identify which is applying/not applying and then give modified code. • MVC implementation can come. • Event-driven architecture question

* MVVM:
- In MVP, there will be too many presenters. And there is too much coupling b/w view & presenter.
- 1 view model handles many views (Many to 1).
- View and ViewModel are completely independent of each other.
- Data-binding means page re-rendered whenever data changed (listener/observer keeps listening to data changes) → React follows this → this is MVVM.
- Publisher/Subscriber pattern — Same → this is "Property changed event" in GeeksforGeeks diagram.
- In MVP, presenter has reference of view; in MVVM, view-model has

no reference of view i.e. they are only bound together.

- all views become independent of each other so switching between views becomes easier.

* Event-Driven Architecture (EDA)

- No coding examples in this because ... ask Sir why.
- (Q) What is the problem with using triggers for deleting?
  (A) New developer will not know what delete is triggering in other tables.
- Using triggers is not considered good practice b/c we're shifting business logic from code to database → ~~to~~ prevent this, proper documentation is the solution.
- Events MUST be independent of each other → DECOUPLED.
- Event generators do not know consumers.
- SOA: service-oriented architecture; similar to microservices; every service should be independently developed.
- Programatically telling to lessen inventory when checking out is not event-driven; just checking out and emitting inventory event so it is listened by inventory and lessens by itself, this is event-driven.
- Event-driven architecture relates to Publisher-Subscriber pattern.
- Hovering. That is what bothers me. Hovering upon mind, hovering near body. Annoying. And invasive. Why do people pay attention to things that mean nothing to them — should mean nothing to them.
- Used in distributed, decoupled system.
- button.click += new Event(Handler)
  → button is publisher
  - button.click means subscribing to the event.
- (Q) What is meant by self-contained event? is a property of an
- Events are managed ASYNCHRONOUSLY in event-driven architecture.
- "Fire And Forget" → UDP protocol (no receipt of acknowledgment)
- EDA is very scalable; if one event is happening too much, increase consumers for it. But drawback is that this is very complex.

- EDA makes debugging harder b/c failure in one step may result in some important step not happening which means you need to handle this which means code is more complex.
- See slides on adv. & disadv for sure → Sir has said so.

* Real Time System Architecture:
  - correctness is not enough; results are time-based.
  - Performance is based on correct result and timely results
  - Deadlines are important to this system.
  - Types:

Hard	Soft	Firm
hard deadline ↓ late results have drastic effects, not acceptable	we can survive with late results	late results have serious effects but you can survive with that

  - RTS are always concurrent b/c executions are being done in parallel.
  - have feedback concept.
  - fail safe state & fault tolerance

- NOTE-TO-SELF: Add SOLID principles and software architecture to CV??
- Push notifications possible fire and forget; web sockets are a good alternative if you want RTS.
- Fault Tolerance
  ① N-version programming blak: select most 'voted' answer by running different versions of algorithms (?) on same input.
  ② Recovery block: Acceptance Testing done iteratively after executing each version of application and if failing

* Data Stream Processing:
  - Continuous intelligence: looking at real-time data to make better business decisions.
  - Batch processing vs. Stream processing

Batch processing	Stream processing
process group by group as bit come in and form group	process bit by bit as each bit comes in. e.g. applying filter while filming video.

  - In streaming services, data stream size is huge but data record size of stream and timestamp & value are its most important fields
  - Event processing (real time low velocity) example is AC on/off

based on incoming temperature.

— In batch processing, data is processed first and stored later. And the processing is not so complex. In-flow and out-flow is almost identical so data does not pile up.

— Cron jobs are an example of batch processing.

x ——— x ——— x ——— x

Thinking time. Agar tutorial 'manage' ho he jaata, to mein tutorial say pehlay milnay ka bolti he kyun? Koi ajeeb si schemes, naaarzigiyaan, chalaakiyon ka combination bana kar ye harkatein banti hain jo ye log kar rahay hain. End mein phasta kon hai? Mein. Agar itnay he faaigh hain to ye blii kartein, kon si uniki zindagii haraam ho rahi ha? Ajeeb. Useless log. Useless to nahin, but just frustrating.

x ——— x ——— x

— Cold storage: data that is retrieved frequently → can be archived.

* Project:

- Implement 3-tier architecture or MVC properly
- Architecture diagram of project
- Viva / Presentation at end.
- Goal is to do industry-level work (industry = standard quality ka)
- Social media web/mobile app:
  1. Sign up (email or Google sign in + FB sign in)
  2. User can post anything.
  3. Privacy of post is managed. So only visible to followers.
  4. Users follow / not follow.
     → No need to make this permission-based i.e. no request send/accept, just follow.
  5. User management
  6. Feed management

  } Two important things in project

- Database Diagram
- Authentication process identified.
  → Encryption???
- Do NOT use Firebase — V. Imp!!!

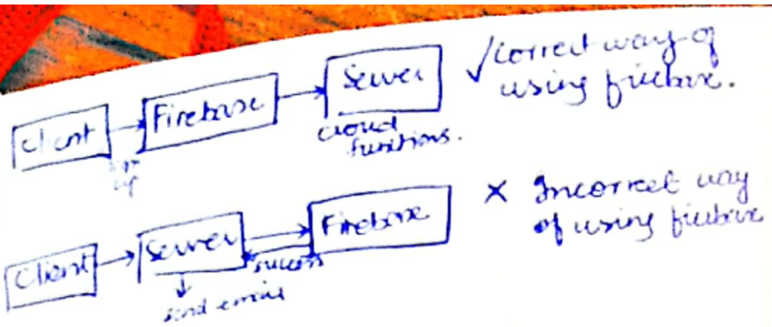
Don't sleep. Don't sleep. Don't sleep.

- User Management → first milestone

sign up
sign in
authentication
security ...?
forget password
refresh token ...?
Google sign in
Facebook sign in ...?

Database — user profile
(make diagram)
↓ ERD
NEXT CLASS.

authorization/roles.
relational database
followers/following
feed structure (Posts)



* FireStack architecture → Search!!!

* **Cloud Computing**

- Manage service: someone else manages the tool and you just party with it e.g. EC2 machines, or all cloud-based services.
- Deployment model: where your hardware for the cloud is located and how you will access it.
- **VPC** (virtual private cloud)
  → search it!
- **Service Model**
  - **IaaS**
    e.g. cable service, internet service, servers, storage space
    → provides only hardware.
    e.g. like providing storage for backup but no backup button.
    → used by developers to develop software
  - **PaaS**
    provides server + utilities
    e.g. amazon, Azure, GCP, Database itself
    provides hardware + tools
  - **SaaS**
    has platform + actual software
    → e.g. Gmail, ERP, WordPress
    users are normal people, not developers (:

— NOTE-TO-SELF: Pay more attention in class. Think . Ask more questions. Ask being ALIVE in class.

— NOTE-TO-SELF: Ask after Aaiy's ittekaaf.

— Biggest disadvantage of IaaS, PaaS and SaaS is that they MUST have Internet to work.

— NOTE: Don't use subqueries unless you absolutely have to.

— NOTE-TO-SELF: Don't be friends with people who talk during class.

کیوں اتنا شور ہے میرے سانسوں میں؟ —

— Data is physically stored on the basis of the column that has been cluster-indexed so

People look. They become curious and then they pry. They can be as interested in our lives as possible, that's fine, but the moment someone starts asking me questions that's where the annoyance starts. Don't draw attention to yourself. That's the rule.

First row mein beth kar he ye saaray kaam karna yaad aatein hain aap ko!!!! Issi ka toh maza hey!!! —

hehe, annoyed you? mission accomplished

:) :) :) *✓*

there is only one cluster index in table → interview questions → answers → V. Imp!

— Where is a non-clustered index preferred? Hmmm. Let's think. Where lexicographic order is preferred? Or contains information? Nope. Maybe where we want to point to specific rows rather than whole group of ordered records. Nah, the REAL reason is that when two systems are being integrated, to avoid collisions between ids if they were kept integer, we instead generate GUIDs throughout both systems and these are non-integer, non-clustered indices.

— Database design of survey management system:

① Surveys → has status. If published, then can fill it

② Survey questions. (Can be of multiple types)

③ Survey responses. only if survey is published.

④ Can edit/delete/add survey question later but responses for that question should not be deleted. (soft delete concept)

⑤ survey versioning — changes tracked (new questions added, deleted, edits shown) and also tell which version is active. (No question versioning just whether it's there or not and whether it has been edited)

This is an individual assignment.

— NOTE-TO-SELF: When someone tells you a hair color doesn't suit you, listen to them.

— "Nayi nahin to hai, ye nayi nahin to hai kahani, kaisi paheli hai ye kaisi paheli zindagani"

— Mein nay jo kuch khoya hai, kya tumhein panay kay liyay kafi tha?

— Serverless architecture d/f platforms: AWS lambda, Azure functions, Cloud Functions.

— Benefits of serverless:

- quickly develop apps.
- easy scalability
- only pay for usage time, not idle time.

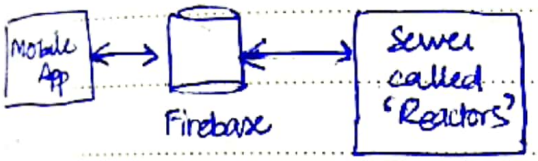
Disadvantages of serverless:

- not efficient for long-running tasks.
- vendor lock-in (difficult to change platforms).

- Was it worth it? Messing up my courses, throwing away my future, denouncing everything I stood for...? I don't blame you. You still are the best thing that ever happened to me. I blame myself. I didn't handle this right.

- Firebase's remote config means you can handle different configurations for different users.

- F.I.Re Stack Architecture:



All communication by client is done though firebase, not directly to reactors.

Only for functions/tasks you really need to use server, you use server, otherwise firebase handles all CRUD operations related to Firestore and related things.

Security was a concern with firebase cause whole database exposed. But Firebase security rules handles this a little bit.

Also gives push-notification functionality.

Web sockets (used by firebase) are much faster than http APIs, by a GREAT significance.

Reactor and client's device are both connected to the stream service b/c for reading data we need client's device, but for writing data the channel needs to be secure so we use the server/reactor for that.

Elastic search used to search for keywords within data quickly, is dependent of searching for posts for feed.

Firebase queue allows you to execute tasks sequentially with separate work functions (?). Firebase queue went obsolete once Cloud Functions was introduced.

Database structure is very important in Firebase.

Having ~~no~~ redundant data in NoSQL is expected.

Bohat saari dostiyaan, bohat saara rishtay tooth gayay hain.

Security rules in firebase help us understand/control authorization.

Misconceptions related to firebase include it is not scalable.

Firebase's main concern is cost. It increments cost according to how u scale. is scaling instead of giving one-off scaling cost every time. It also handles scaling on its own, so in the big picture, you will always be in profit with firebase.

* Survey Assignment

→ There should be a field of no. of versions available.

→ data will become dirty if the version added for a survey is d/f than the ~~version~~ survey whose version that is.

→ adding active/inactive field means that at most, the data will ~~at most~~ have a problem of handling more than one active version of data.

→ Responses field should have ~~question~~ survey which you are filling, uss ki details.

→ Survey questions will be duplicated for newer versions so this kind of duplication is okay ** **

→ Published will not be changed → make a new version now.

Database design is coming in the final exam → V. Imp !!! inclusive of ~~pre~~-midterm stuff too.

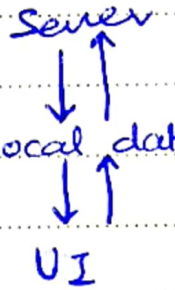
* Colgate – Palmolive case study

– Differential data calls.

– GCM for notifications.

– Yii2 framework???

– Local database means all operations & data are first gone from local database then to server OR from server to local database, and then data loaded/uploaded from/to UI to/from local database.



Always follow this when you want offline support.

– Notification based syncing: when-ever server updates data, you get a push notification to update data. But push notifications can be ignored or not received, so only give differential data's entities names for syncing data of

just those entities. But even after this, just to be safe, give manual syncing option.
- History was also synced.

Big Big Monies $$$