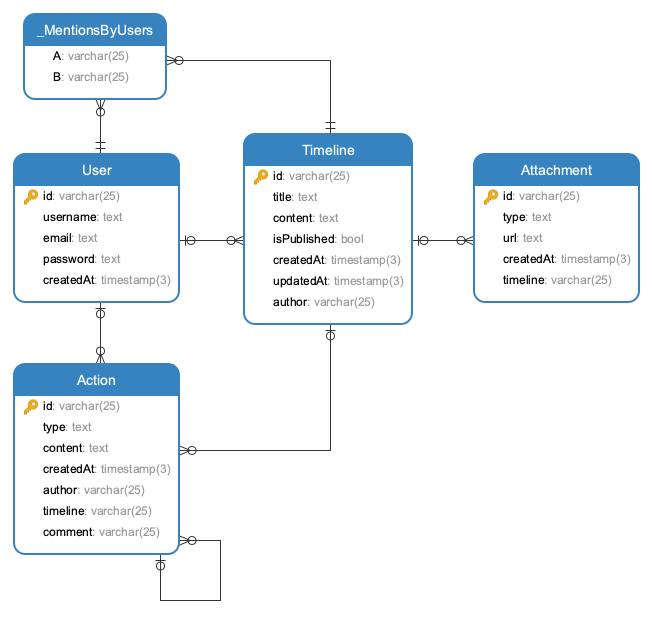
**Timeline management System design**

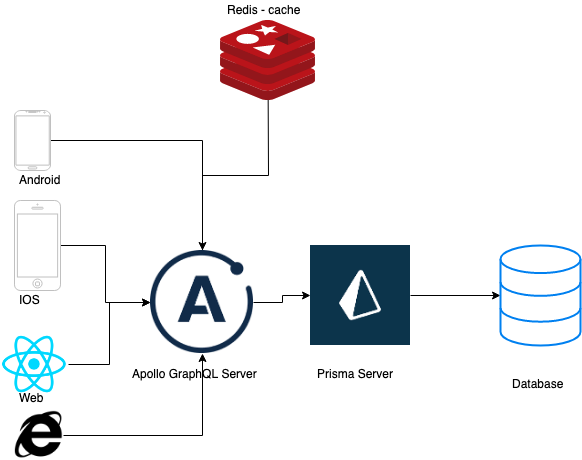
**1- ER Diagram**



* Our action table has type field: “LIKE”, “SHARE”, “COMMENT”
* Our attachment table has type field: IMAGE, VIDEO, URL
* Images, Videos sticker is uploaded to server and store url string to database so that app can retrieve url to display appropriate content

Detail database design can be found in Docs folder

**2 - System design architecture**



**GraphQL architecture**

* All request will go to Apollo graphql server
* Redis will be used as cache response
* Prisma server act as ORM for database manipulation and realtime feature support through websocket

**3 - Tech-stack**

**A- Backend**

* Backend: GraphQL (Apollo server): <https://www.apollographql.com/docs/apollo-server/>
* Backend-Cache: Redis: <https://redis.io/>
* Backend performance metric + logging: Apollo Engine: <https://www.apollographql.com/docs/references/apollo-engine/>
* Backend logging: Kibana
* Database: Postgres or mysql or mongodb
* Database ORM: prisma <https://www.prisma.io/>
* Deployment
  + Local: docker (dockerfile + docker-compose)
  + Production: Google Kubernetes engine (GKE)
    - Auto scaling depend on use

- Database ORM: prisma <https://www.prisma.io/>

- CI/CD: Gitlab + Jenkins

**B- Frontend**

* IOS: swift
* Android: kotlin
* Web: ReactJS or VueJS

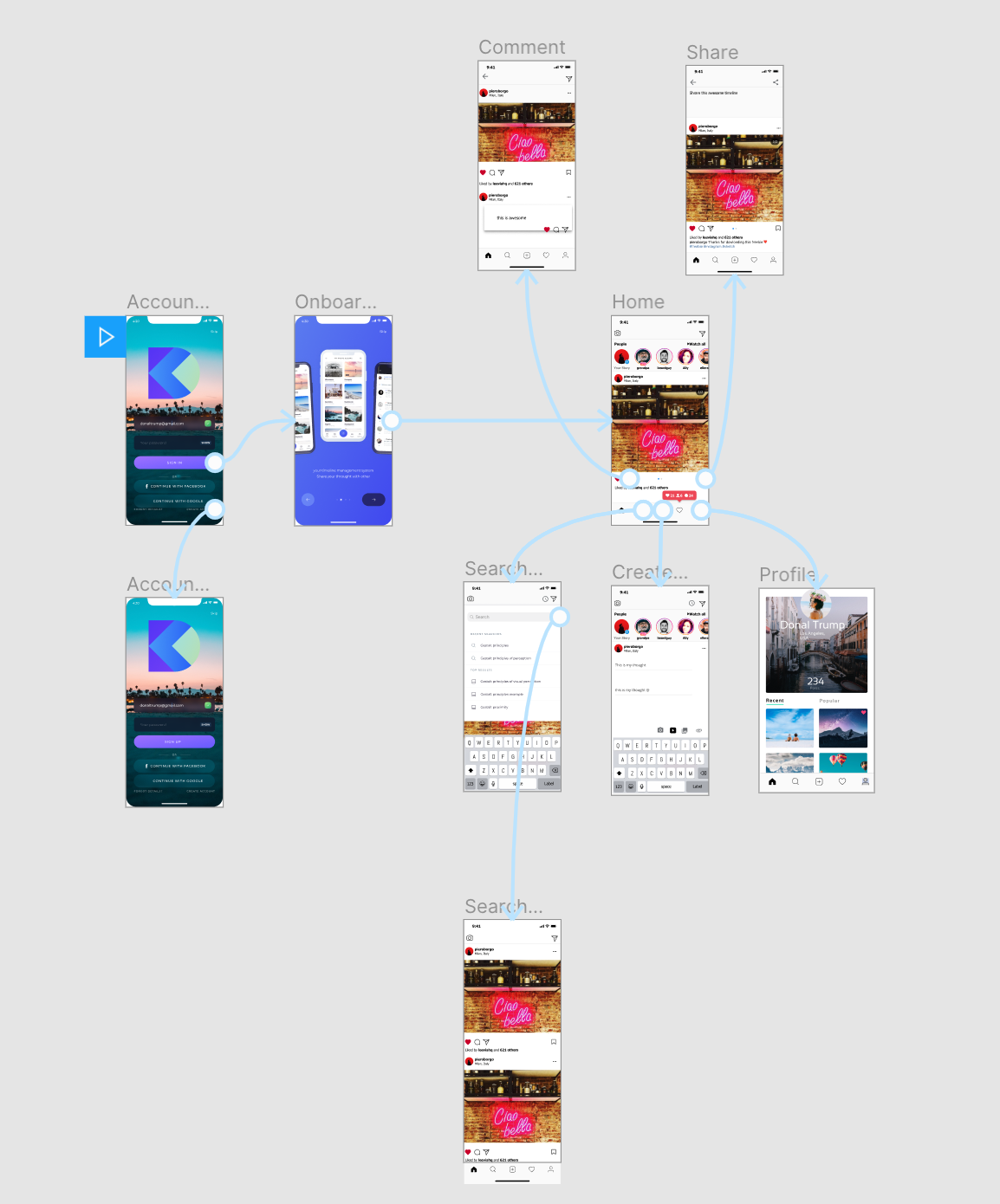
Note: can consider react native if development time is short

All base on Apollo-client: <https://www.apollographql.com/docs/react/api/apollo-client/>

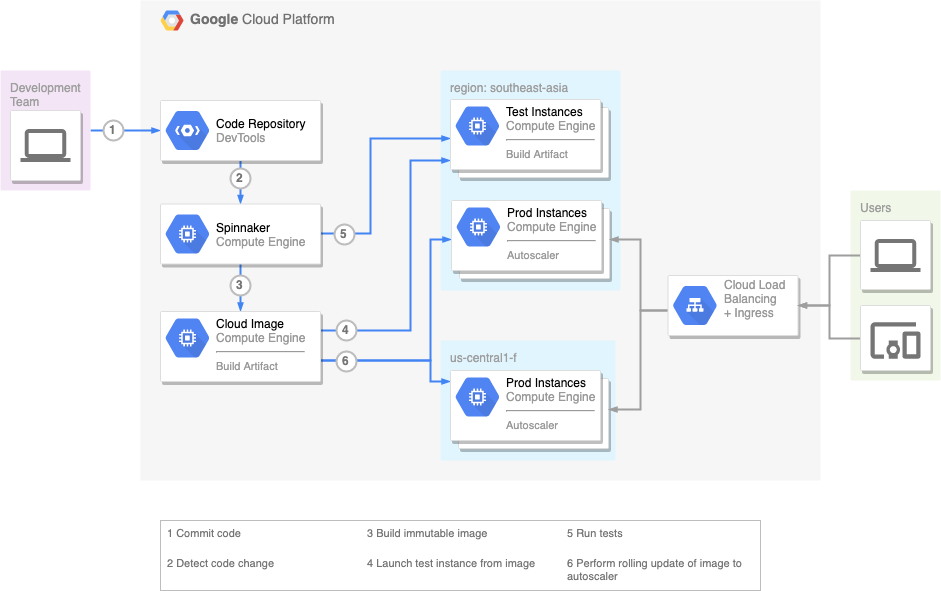
* Analytics + tracking: Firebase analytics + facebook analytics
* Performance monitoring + tracing: Firebase Performance
* Crash reporting: Firebase crashlytics
* CI/CD: bitrise <https://www.bitrise.io/>

**4 Front-end mockup (see .fig file for detail) (Open with frigma app)**

**https://www.figma.com/file/LkWkr13h32ccDYre6KwcW5Bs/Timeline?node-id=0%3A1&viewport=440%2C2414%2C0.1484600305557251**



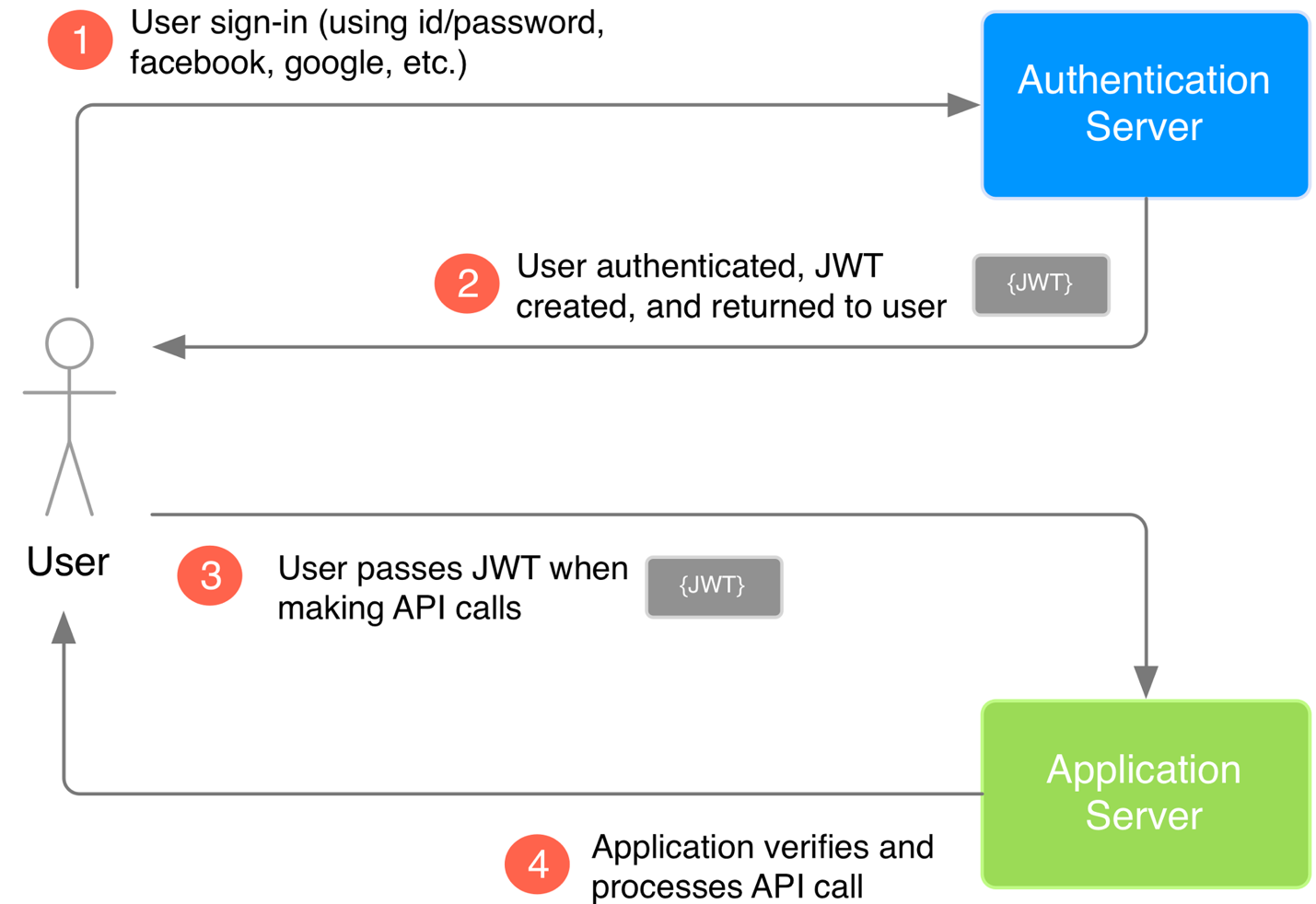
**5 - Deployment workflow**



Scaling note:

* Scaling support by kubernetes autoscale pod
  + Number of replica
  + Load balancer
  + Ingress configuration

**6 - Authentication Flow**



**First phase using simple authentication with jwt (JsonWebToken)**

* User need to signup using email + password
* Password will then be hash with a secret and store on database as random string
* Successful signup will return token
* User can login with email + password (after successfully signup)
* Subsequent request for other endpoint need token as header “Authorization” inorder to get data from server
* Sample request

curl 'http://localhost:4000/' -H 'Accept-Encoding: gzip, deflate, br' -H 'Content-Type: application/json' -H 'Accept: application/json' -H 'Connection: keep-alive' -H 'DNT: 1' -H 'Origin: http://localhost:4000' -H '**Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiJjanc5bjJ3bWYwMDFsMDcxNnp2OGdmODNpIiwiaWF0IjoxNTU5MjA4MzgzfQ.DXRx1Sz4Bu0zU2XF0J2E4XBjxjBPDeet\_OcAJxymejg**' --data-binary '{"query":"# Write your query or mutation here\nmutation {\n createDraft(title: \"newDraft4\", content: \"newContent4\") {\n title\n id\n author {\n username\n email\n id\n }\n isPublished\n createdAt\n updatedAt\n attachments {\n id\n type\n url\n createdAt\n }\n }\n}"}' --compressed

* Authentication System can be improved by provide short-live access-token and longer-live refresh token (better security and more convenient)
* Authentication System can be improved by using SSO (single sign on)

**7 - Security design**

* Backend:
  + server need to have SSL certificate (connect through https)
  + Prisma server is protected by API management: <https://www.prisma.io/docs/prisma-server/authentication-and-security-kke4/>
  + Password is stored in database as hash string (can not be retrieved only can be verified using SECRET string
* Frontend:
  + app make request to server with **certificate-pinning** (ensure no man in the middle attack)
  + User info is encrypted before save to device database

**8 - Sample Backend (database + cache + prisma server + graphql server)**

A **installation**

* Install docker
* Install docker-compose
* Open terminal in root project folder
  + Enter: “docker-compose-up -d” (without quote)
  + first time run my take 5-10 min (pulling docker image from remote)
  + When finish please wait 2 min (prisma deploy create database table)
  + If database model change please run “docker-compose-up” again (2nd run much faster with docker cache)
  + Access graph server at <http://localhost:4000>
* List of api can be seen from graphql playground

(for development playground is enable, playground will be disable in production server)

* Local deployment script is place in “docker-compose.yml” file
* Sampe kubernetes deployment script can be found in k8s-node folder (need to change to actual credential gke in order to work)

**B - Folder structure**

* Generated prisma and nexus is place in generated folder (should not modify)
* Query and mutation is center point for api design