

# Bimble

## Github URL

<https://github.com/huseinashr/bimble>

## Requirements

### Functional

- F1. User can create new account with an email (Implemented)
- F2. User can verify its newly created account's email
- F3.1. Existing user can login with email and password (Implemented)
- F3.2. System can authenticate user on authenticated endpoint
- F4. Authenticated user can customize their dating profile (OOS but MVP)
- F5. Authenticated user can set their dating preferences (OOS but MVP)
- F7. Authenticated user can see list of personalized dating profiles (OOS but MVP)
- F8. Authenticated user can pass a dating profile
- F9. Authenticated user can like a dating profile
- F10. Regular user is limited to 10 pass+like per day
- F11. Authenticated user won't be able to see the same dating profile in the same day
- F12.1 Regular user can purchase premium packagees which unlock one premium feature of their choosing. The features are:
  - No swipe quota
  - Verified label
- F12.2 System can verify the purchase payment

## System Design

### Tech Stacks

- Go
  - as programming language.
  - Simple, Fast, Compile time type checking. Suitable for rapid development.
- Proto driven HTTP endpoint
  - for defining and generating api docs
  - Can Generate both server stub and OpenAPI spec
- Mockery
  - for unit test
  - Can Generate mock both internal and 3rd party library. Mock with dependency injection not monkey patching
- PostgreSQL
  - for storing transactional data that require ACID or persistent
  - Open source, community driven, and matured ecosystem.
- Redis
  - for storing login session and profile view
  - The goto in-memory datastructure store. Great API, mature ecosystem, reliable
- Kafka
  - notifying when account, profile, preference changes
  - persistent, ordered, scalable, and great ecosystem
  - can rebuild elasticsearch index from the kafka when needed
- Elasticsearch
  - for storing and distributing dating profile that support personalization, re-ranking and boosting certain profile
  - The goto indexing engine for content recommendation. Great API, mature ecosystem, reliable
- GitHub Action
  - integrate natively with github. Many community based 'action' that we can use in our pipeline
- K8s
  - Deployment environment
  - Cloud native, mature ecosystem, easier to setup CI/CD, as well as managing resources such as postgresSQL, redis, elastic

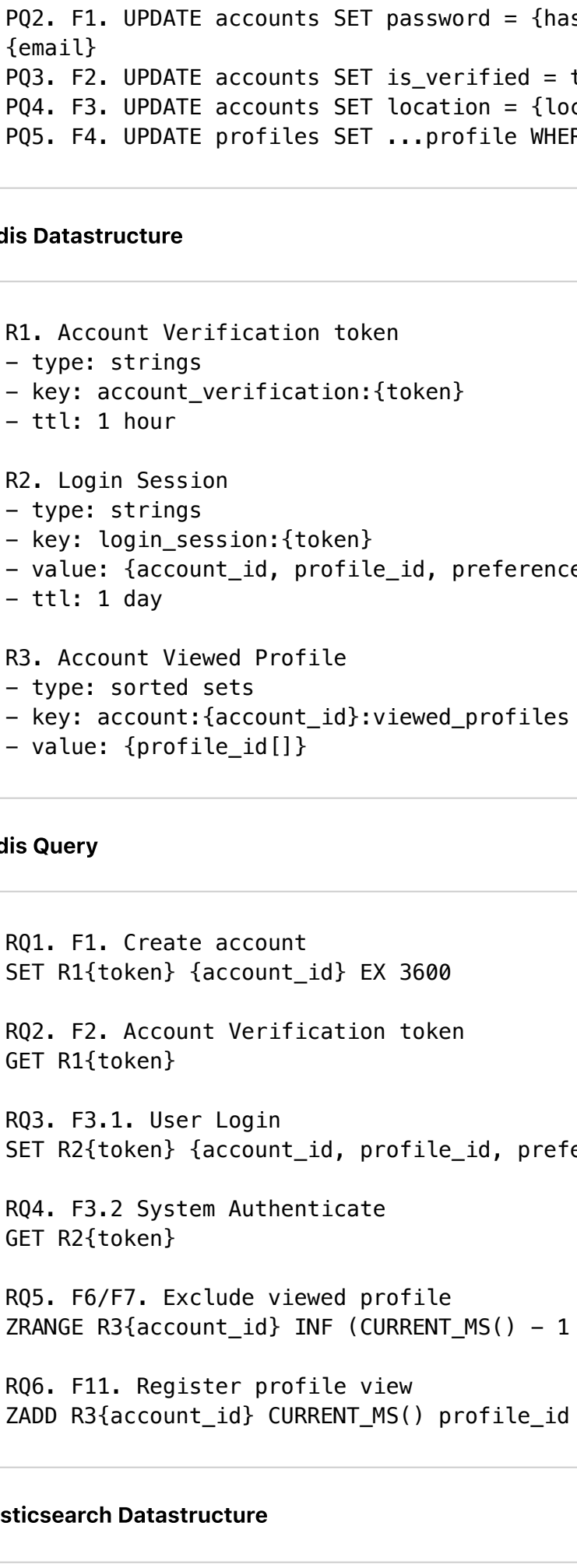
### Endpoints

These endpoints loosely follow <https://google.aip.dev/1> api guidelines. The endpoints numbering match with corresponding functional requirement.

- F1. POST /accounts:signup
  - Request
    - Body {email: string, password: string}
  - Response OK
    - Body {message: string}
- F2. POST /accounts:verify
  - Request
    - Body {token: string}
  - Response OK
    - Body {message: string}
- F3.1. POST /accounts:login
  - Request
    - Body {email: string, password: string}
  - Response OK
    - Body {token: string}
- F4. PATCH /accounts/profiles
  - Request
    - Header {Authentication: Bearer Stoken}
    - Body {Profile}
  - Response OK
    - Body {message: string}
- F5. PATCH /account/preference
  - Request
    - Header {Authentication: Bearer Stoken}
    - Body {Preference}
  - Response OK {message: string}
- F6. GET /profile:random
  - Request
    - Header {Authentication: Bearer Stoken}
  - Response OK
    - Body {array of Profile}
- F7. GET /profile:personalized
  - Request
    - Header {Authentication: Bearer Stoken}
  - Response OK
    - Body {array of Profile}
- F8. POST /profile/{profile\_id}:pass
  - Request
    - Header {Authentication: Bearer Stoken}
  - Response OK
    - Body {message: string}
- F9. POST /profile/{profile\_id}:like
  - Request
    - Header {Authentication: Bearer Stoken}
  - Response OK
    - Body {message: string}
- F11. POST /profile/{profile\_id}:view
  - Request
    - Header {Authentication: Bearer Stoken}
  - Response OK
    - Body {message: string}
- F12.1. POST /feature/{feature\_id}:purchase
  - Request
    - Header {Authentication: Bearer Stoken}
  - Response OK
    - Body {payment\_url: string}
- F12.2. POST /payment:confirmation
  - Request
    - Header {Authentication: Bearer Stoken}
    - Body {payment\_token: string}
  - Response OK
    - Body {message: string}

### Entity Relation

#### PostgreSQL Table



#### PostgreSQL query

- P01. F1/F3. SELECT accounts WHERE email = {email}
- P02. F1. UPDATE accounts SET password = {hashedPassword} WHERE email = {email}
- P03. F2. UPDATE accounts SET is\_verified = true WHERE id = {account\_id}
- P04. F3. UPDATE accounts SET location = {location} WHERE id = {account\_id}
- P05. F4. UPDATE profiles SET ...profile WHERE id = {profile\_id}

### Redis Datastructure

- R1. Account Verification token
  - type: strings
  - key: account\_verification:{token}
  - ttl: 1 hour
- R2. Login Session
  - type: strings
  - key: login\_session:{token}
  - value: {account\_id, profile\_id, preference\_id}
  - ttl: 1 day
- R3. Account Viewed Profile
  - type: sorted sets
  - key: account:{account\_id}:viewed\_profiles
  - value: {profile\_id[]}

### Redis Query

- RQ1. F1. Create account
  - SET R1{token} {account\_id} EX 3600
- RQ2. F2. Account Verification token
  - GET R1{token}
- RQ3. F3.1. User Login
  - SET R2{token} {account\_id, profile\_id, preference\_id}
- RQ4. F3.2 System Authenticate
  - GET R2{token}
- RQ5. F6/F7. Exclude viewed profile
  - ZRANGE R3{account\_id} INF (CURRENT\_MS() - 1 Day) REV BYSCORE
- RQ6. F11. Register profile view
  - ZADD R3{account\_id} CURRENT\_MS() profile\_id

### Elasticsearch Datastructure

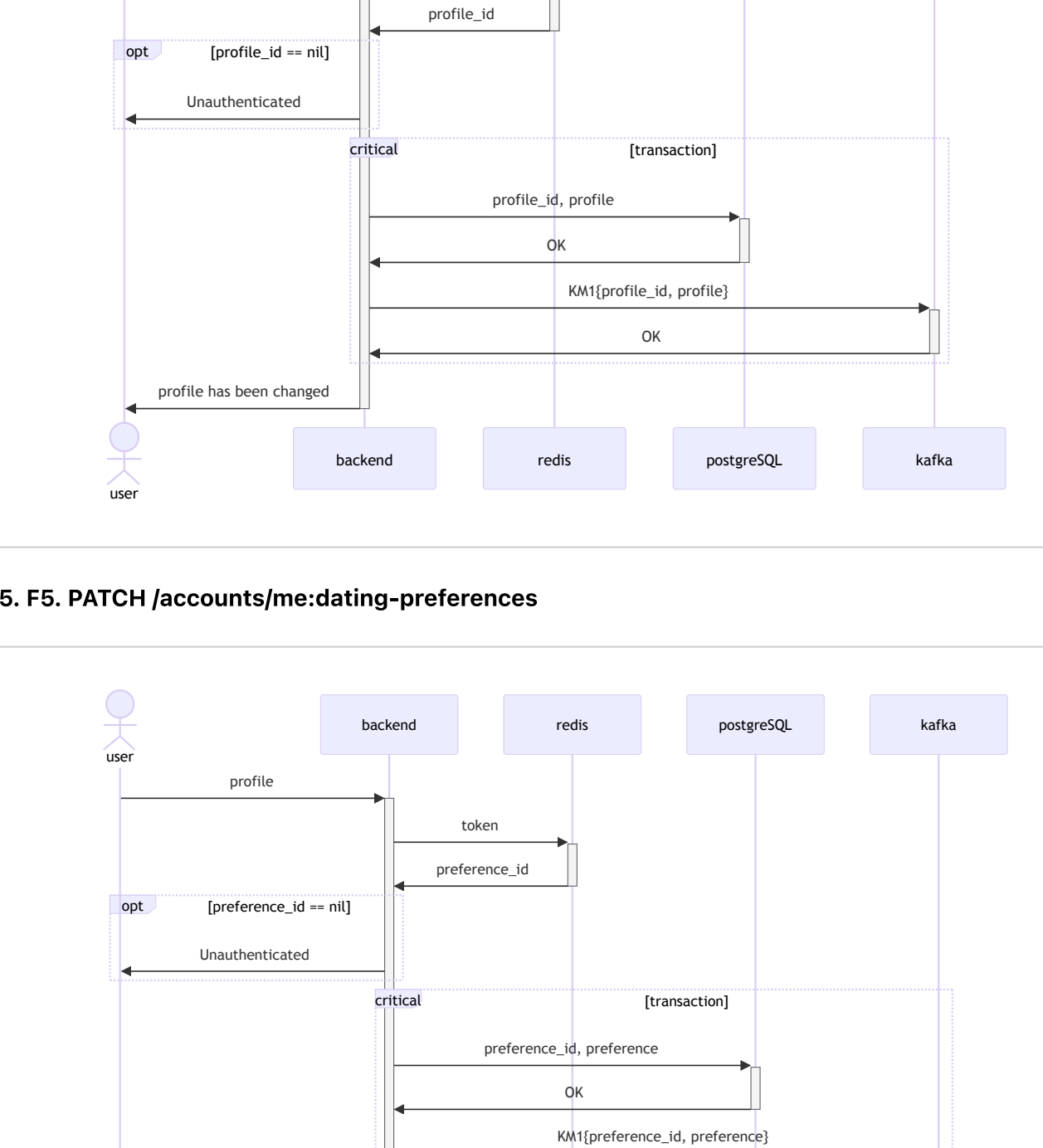
- ED1. F7. index dating\_profiles
  - {  
  "id": 1 // profile.id  
  "account\_id": 1 // account.id  
  "preference\_id": 1 // preference.id  
  "email": "janetest.com" // account.email  
  "location": [1.23, 4.5] // account.location  
  "is\_verified": true // account.is\_verified  
  "name": "Jane" // profile.name  
  "hobbies": ["drawing"] // profile.hobbies  
  "about": "my-about" // profile.about  
  "max\_distance": 200 // preferences.max\_distance  
  "min\_age": 25 // preferences.min\_age  
  "max\_age": 27 // preferences.max\_age  
  "verified\_label": true // features.feature = VERIFIED\_LABEL  
  "no\_swipe\_quota": true // features.feature = NO\_SWIPE\_QUOTA  
}

### Kafka Messages

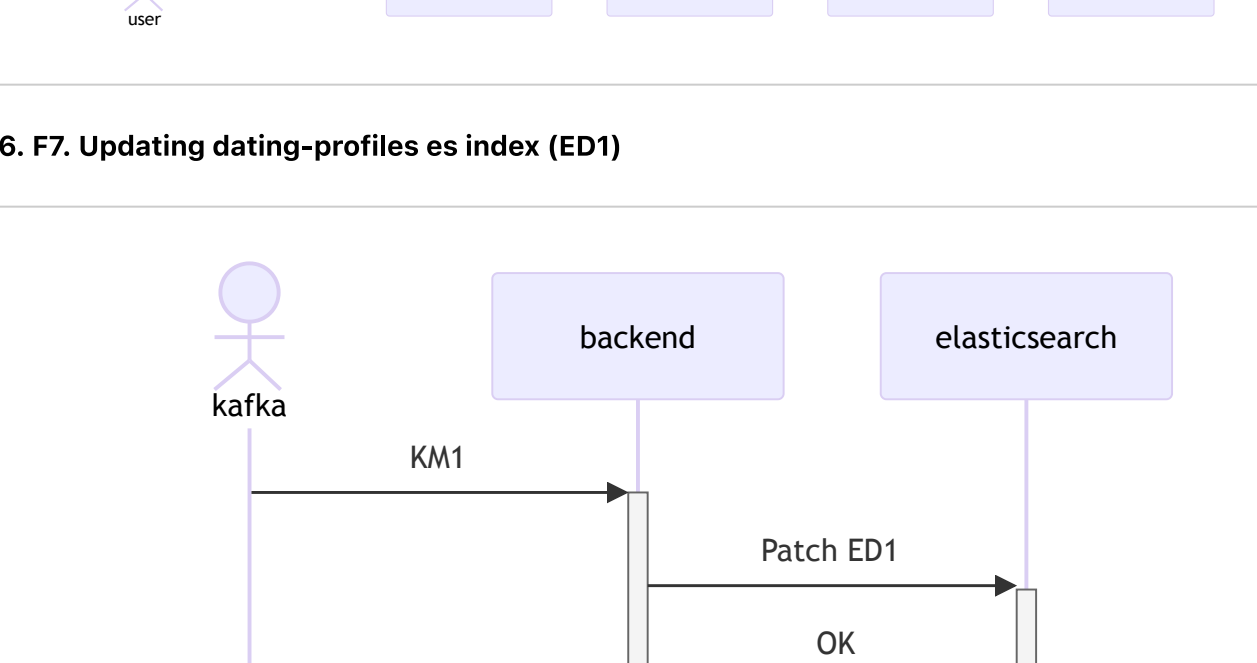
- KM1. F7. event account\_changed
  - account {  
  ...partial,  
  profile {  
    ...partial,  
    preference {  
      ...partial  
    }  
  }  
},  
}

### Sequence

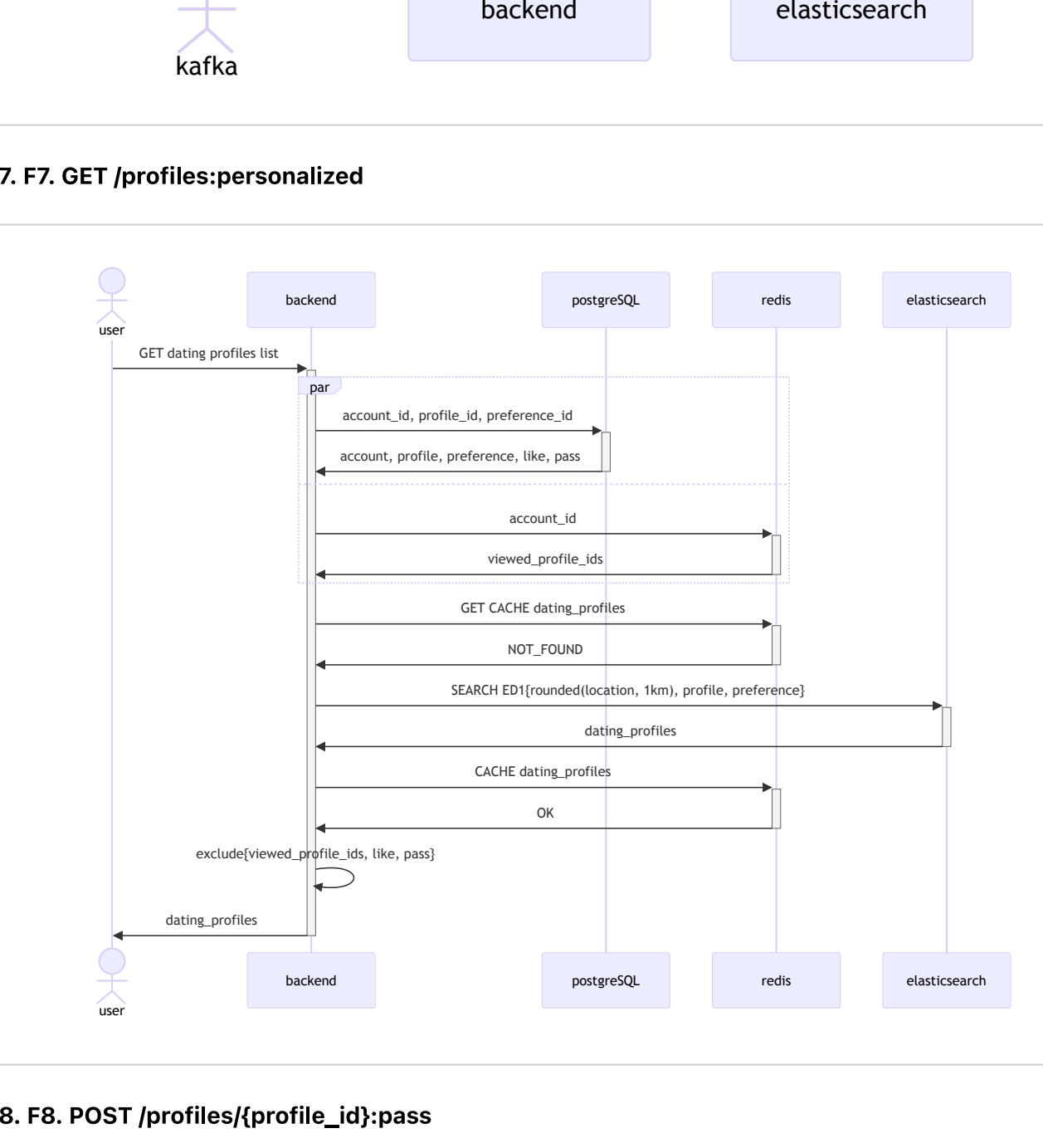
#### SD1. F1. POST /accounts



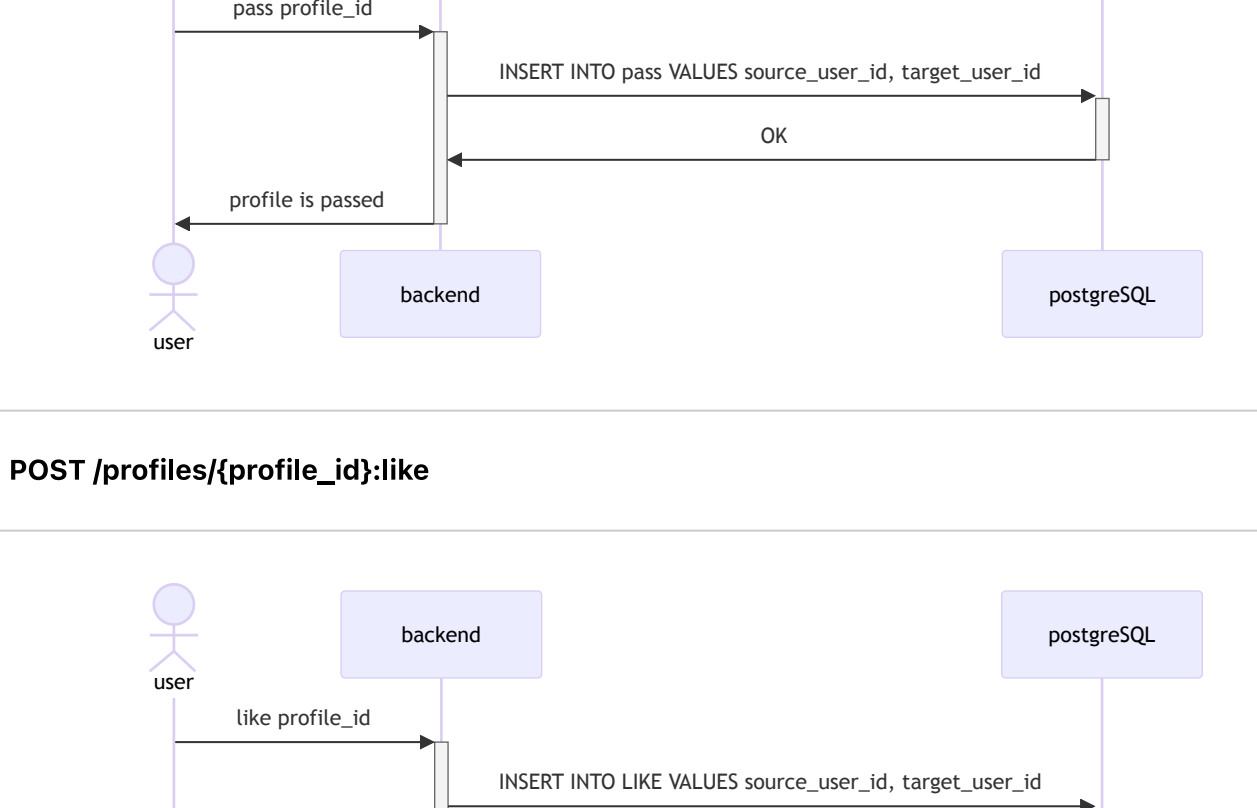
#### SD2. F2. POST /accounts:verify



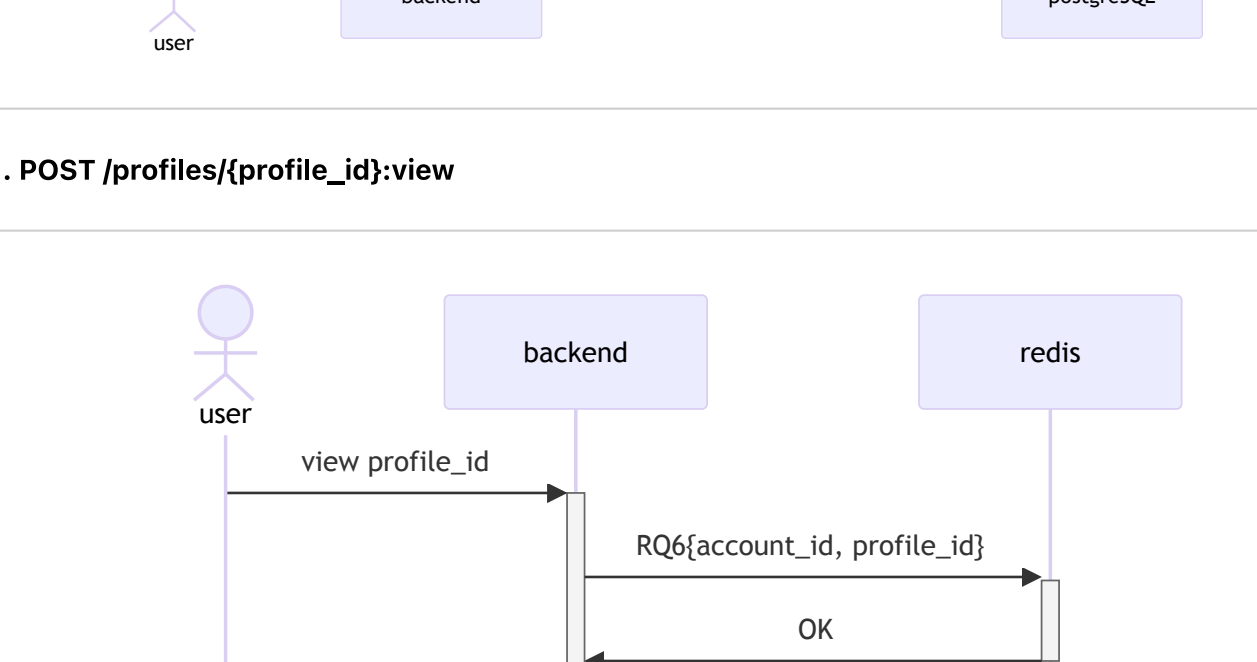
#### SD3. F3.1. POST /accounts:login



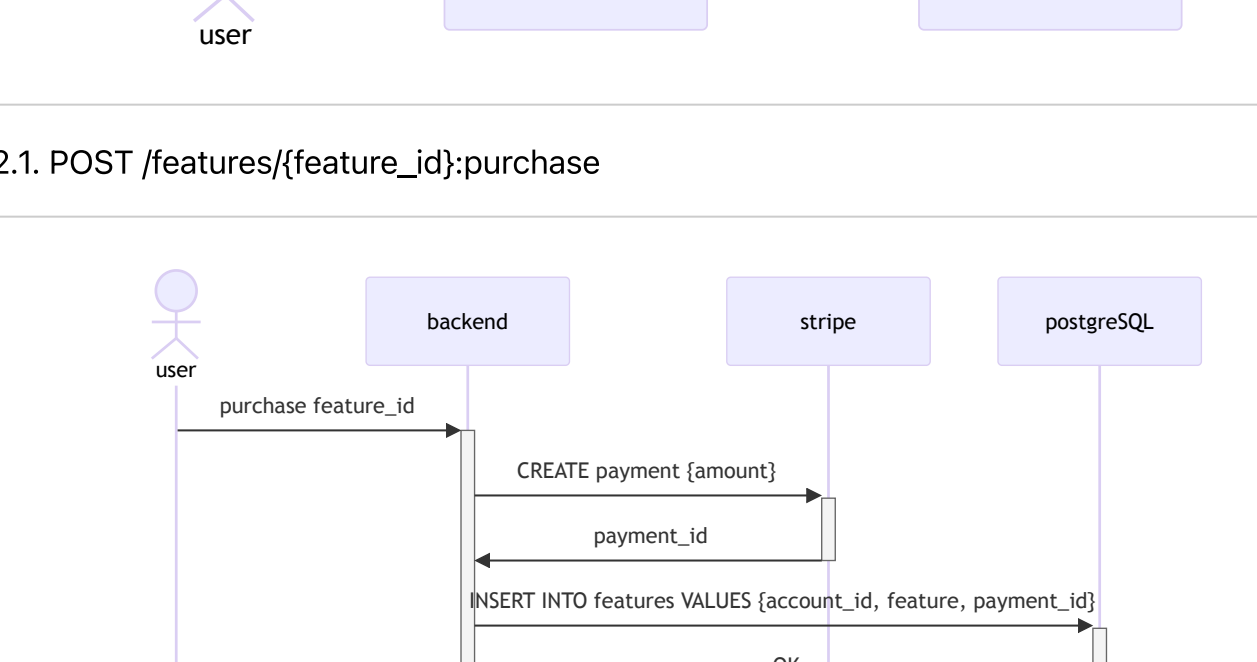
#### SD4. F4. PATCH /accounts/me:dating-profile



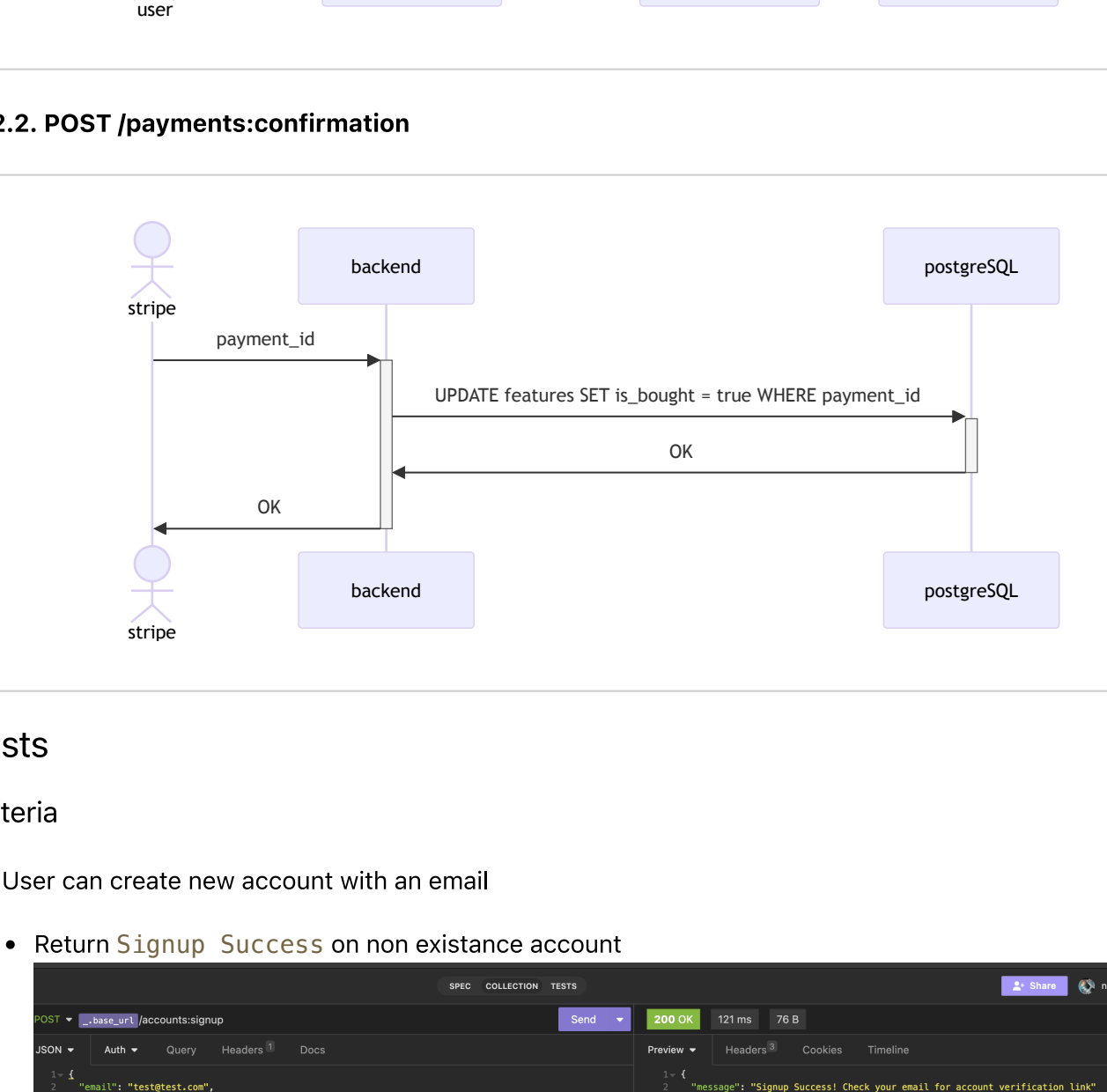
#### SD5. F5. PATCH /accounts/me:dating-preferences



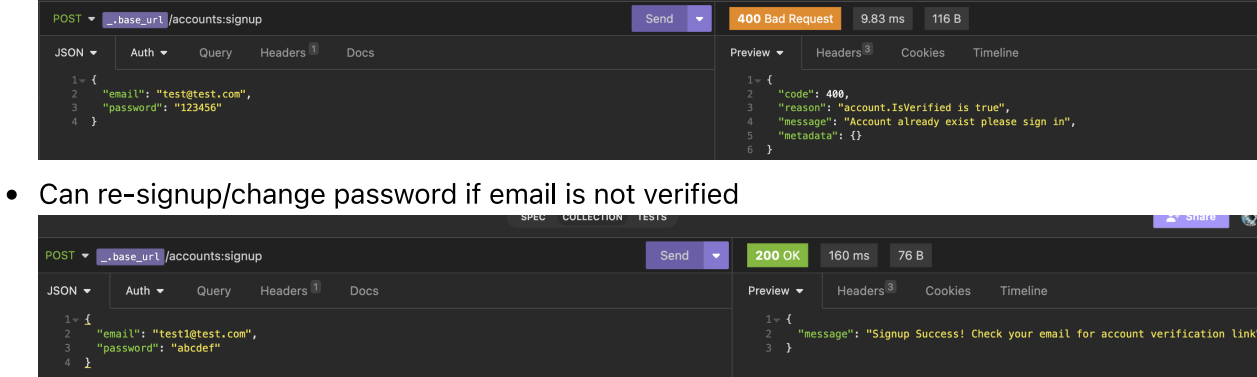
#### SD6. F7. Updating dating-profiles es index (ED1)



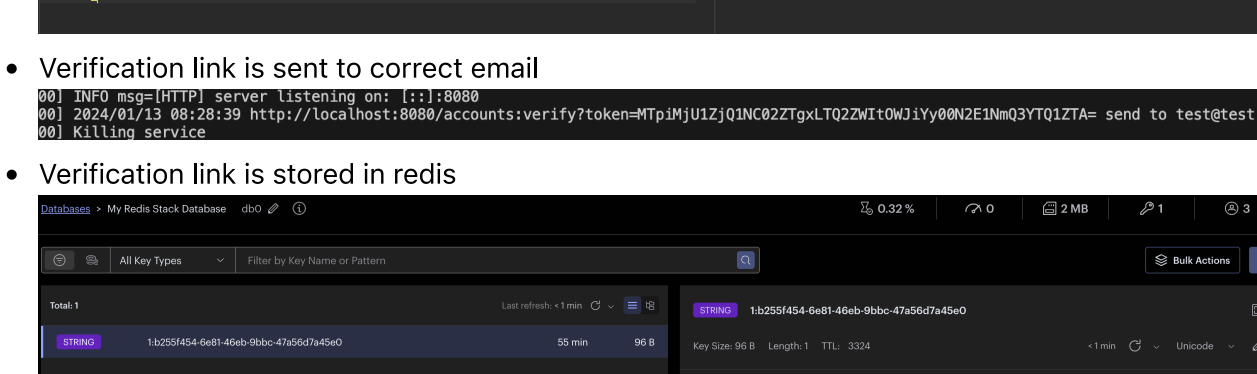
#### SD7. F7. GET /profiles:personalized



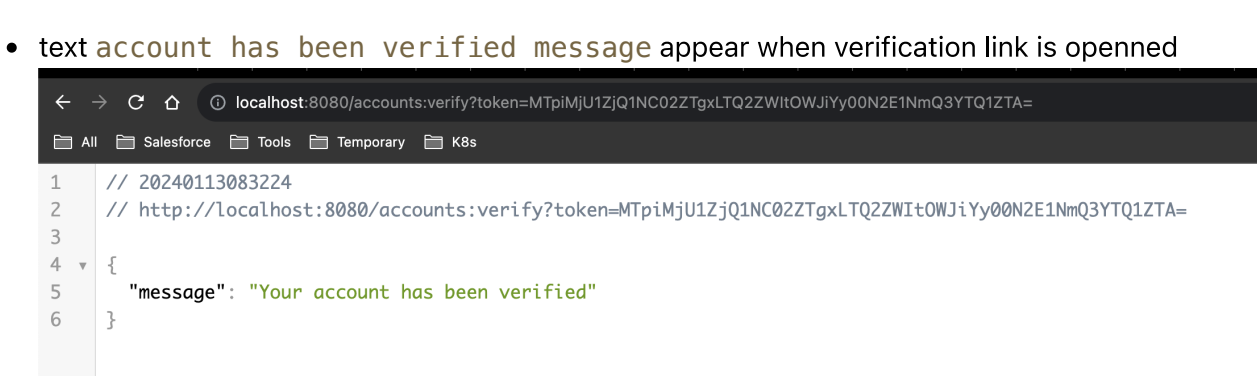
#### SD8. F8. POST /profiles/{profile\_id}:pass



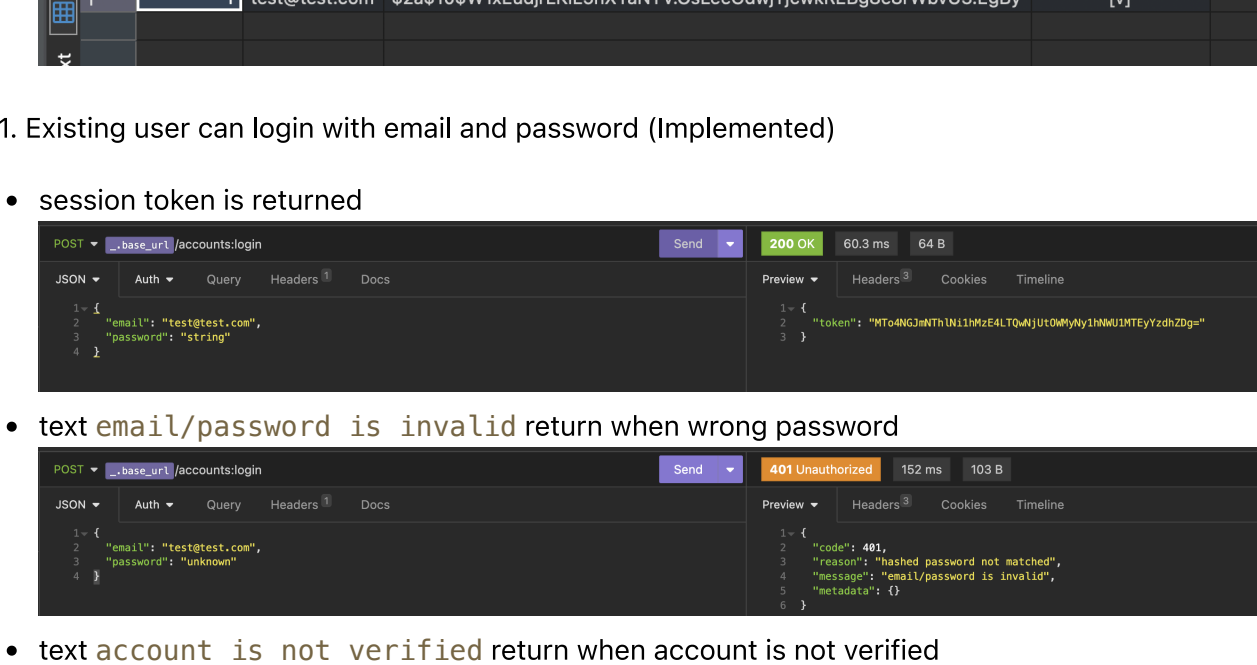
#### F9. POST /profiles/{profile\_id}:like



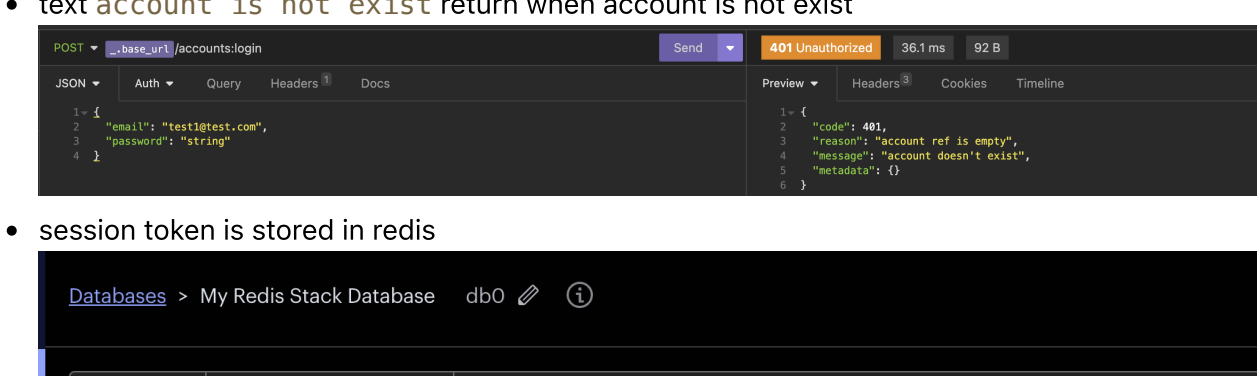
#### F11. POST /profiles/{profile\_id}:view



#### F12.1. POST /features/{feature\_id}:purchase



#### F12.2. POST /payments:confirmation



## Tests

### Criteria

#### F1. User can create new account with an email

- Return Signup Success on non existence account
- Return account already exist on existed verified account
- Can re-signup/change password if email is not verified
- Can login after re-signup

- Verification link is sent to correct email
- Verification link is stored in redis

#### F3.1. Existing user can login with email and password (Implemented)

- session token is returned
- text email/password is invalid return when wrong password
- text account is not verified return when account is not verified
- text account is not exist return when account is not exist
- session token is stored in redis

#### F3.2. System can authenticate user on authenticated endpoint

- endpoint with "Authentication" header not returning Unauthenticated status
- endpoint will return Unauthenticated status when using invalid session token

#### F4. Authenticated user can customize their dating profile (OOS but MVP)

- endpoint return "profile has been changed"

#### F5. Authenticated user can set their dating preferences (OOS but MVP)

- endpoint return "preferences has been changed"

#### F7. Authenticated user can see list of personalized dating profiles (OOS but MVP)

- endpoint return array of dating profile ranked approximately by distance and user profiles/preferences

#### F8. Authenticated user can pass a dating profile

- endpoint return "profile is passed"

#### F9. Authenticated user can like a dating profile

- endpoint return "profile is liked"

#### F10. Regular User is limited to 10 pass+like per day

- endpoint return "cannot passed/liked, quota of 10/day is reached"

#### F11. Authenticated user won't be able to see the same dating profile in the same day

- endpoint exclude viewed profile that day

#### F12.1 Regular user can purchase premium packagees which unlock one premium feature of their choosing. The features are: No swipe quota & Verified label

- endpoint return payment link

#### F12.2 System can verify the purchase payment

- endpoint return OK