For each endpoint write up the following based on a local copy of your database using fake data (with at least a million rows):

- **A profile of the endpoints using EXPLAIN before any indexes are added**
- **The SQL on relevant indexes that will impact the performance of that endpoint and WHY those indexes are the right indexes**
- **A profile of the endpoints using EXPLAIN after all indexes are added**

BEFORE INDEXING

POST /users/

```
explain analyze
insert into users (starting_lbs, name, height_inches, avg_calorie_intake,
birthday, gender, password, salt)
values (200, 'jonny', 78, 2250, '2002-10-07', 'M', null, null)
```

Insert on users (cost=0.00..0.01 rows=0 width=0) (actual time=0.415..0.416 rows=0 loops=1)
  -> Result (cost=0.00..0.01 rows=1 width=160) (actual time=0.230..0.231 rows=1 loops=1)
Planning Time: 0.081 ms
Execution Time: 0.524 ms

GET /users/{user_id}

```
explain
SELECT id, name, starting_lbs, height_inches, avg_calorie_intake, birthday,
gender FROM users WHERE id = 2600009
```

Index Scan using users_pkey on users (cost=0.42..8.44 rows=1 width=48)
Index Cond: (id = 2600009)

POST /users/login

```
explain
SELECT salt, password FROM users WHERE name = 'Jennifer Lopez'
```

Seq Scan on users (cost=0.00..4797.00 rows=3 width=64)
filter: (name = 'Jennifer Lopez'::text)

GET /workouts/{user_id}

```
explain
SELECT id FROM users WHERE id = 200000
```

Index Only Scan using users_pkey on users (cost=0.42..4.44 rows=1 width=4)
 Index Cond: (id = 200000)

```
explain
SELECT id, workout_name, weight, distance_ft, repetitions, seconds, sets,
times_per_week FROM workouts WHERE user_id = 200000
```

Seq Scan on workouts (cost=0.00..4562.00 rows=1 width=56)
filter: (user_id = 200000)

GET /user/{user_id}/logs

```
explain
SELECT id, name FROM users WHERE id = 197859
```

Index Scan using users_pkey on users (cost=0.42..8.44 rows=1 width=18)
 Index Cond: (id = 197859)

```
explain
SELECT id, current_lbs, time_posted FROM logs WHERE user_id= 197859
```

Gather (cost=1000.00..131635.34 rows=8 width=20)
  Workers Planned: 2
  -> Parallel Seq Scan on logs (cost=0.00..130634.54 rows=3 width=20)
filter: (user_id = 197859)
Jit:
functions: 4
options: Inlining false, Optimization false, Expressions true, Deforming true

POST /user/{user_id}/logs

```
Explain analyze
SELECT id FROM users WHERE id = 2600025
```

Index Only Scan using users_pkey on users (cost=0.42..4.44 rows=1 width=4) (actual time=0.033..0.034 rows=1 loops=1)
  Index Cond: (id = 2600025)
  Heap Fetches: 0
Planning Time: 0.344 ms
Execution Time: 0.118 ms

```
Explain analyze
INSERT INTO logs (user_id, current_lbs, time_posted)
```

```
VALUES (2600025,145, CURRENT_TIMESTAMP)
RETURNING id
```

Insert on logs (cost=0.00..0.02 rows=1 width=28) (actual time=1.079..1.080 rows=1 loops=1)
  -> Result (cost=0.00..0.02 rows=1 width=28) (actual time=0.909..0.910 rows=1 loops=1)
Planning Time: 0.090 ms
Execution Time: 1.151 ms

POST /goals/

```
explain
SELECT starting_lbs, birthday, gender, height_inches FROM users WHERE id =
2600105
```

Index Scan using users_pkey on users (cost=0.42..8.44 rows=1 width=22)
  Index Cond: (id = 2600105)

```
Explain analyze
INSERT INTO workouts (workout_name, weight, distance_ft, repetitions, seconds,
sets, times_per_week, user_id)
VALUES ('Run', 0, 64114, NULL, NULL, NULL, 7, 178599);
```

Insert on workouts (cost=0.00..0.01 rows=0 width=0) (actual time=0.469..0.469 rows=0 loops=1)
  -> Result (cost=0.00..0.01 rows=1 width=92) (actual time=0.181..0.182 rows=1 loops=1)
Planning Time: 0.090 ms
Execution Time: 0.541 ms

```
Explain analyze
INSERT INTO goals (type_id, user_id, target_weight, workout_id)
VALUES (0, 2600105, 55, 2800006);
```

Insert on goals (cost=0.00..0.01 rows=0 width=0) (actual time=0.385..0.386 rows=0 loops=1)
  -> Result (cost=0.00..0.01 rows=1 width=36) (actual time=0.085..0.085 rows=1 loops=1)
Planning Time: 0.092 ms
Execution Time: 0.455 ms

POST /user/{user_id}/projection

```
Explain analyze
SELECT FROM users WHERE id = 2600324
```

Index Only Scan using users_pkey on users (cost=0.42..4.44 rows=1 width=0) (actual time=0.018..0.018 rows=0 loops=1)
  Index Cond: (id = 1600324)
  Heap Fetches: 0
Planning Time: 0.298 ms
Execution Time: 0.106 ms

```
Explain analyze
SELECT current_lbs, time_posted FROM logs WHERE user_id = 2600324
ORDER BY time_posted
```

Gather Merge (cost=131634.59..131635.29 rows=6 width=16) (actual time=3260.010..3268.628 rows=4 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Sort (cost=130634.57..130634.57 rows=3 width=16) (actual time=3228.528..3228.529 rows=1 loops=3)
        Sort Key: time_posted
        Sort Method: quicksort Memory: 25kB
        Worker 0: Sort Method: quicksort Memory: 25kB
        Worker 1: Sort Method: quicksort Memory: 25kB
        -> Parallel Seq Scan on logs (cost=0.00..130634.54 rows=3 width=16) (actual time=3208.455..3228.444 rows=1 loops=3)
filter: (user_id = 2600324)
            Rows Removed by Filter: 3466550
Planning Time: 0.321 ms
Jit:
functions: 12
options: Inlining false, Optimization false, Expressions true, Deforming true
timing: Generation 1.324 ms, Inlining 0.000 ms, Optimization 43.575 ms, Emission 838.024 ms, Total 882.924 ms
Execution Time: 3313.507 ms

```
Explain analyze
```

```
INSERT INTO projection (user_id, projection_lbs, projection_date, date_posted)
VALUES (2600324, 80, '2023-04-07', CURRENT_TIMESTAMP);
```

Insert on projection (cost=0.00..0.02 rows=0 width=0) (actual time=0.231..0.232 rows=0 loops=1)
  -> Result (cost=0.00..0.02 rows=1 width=32) (actual time=0.079..0.079 rows=1 loops=1)
Planning Time: 0.081 ms
Trigger for constraint projection_user_id_fkey: time=0.746 calls=1
Execution Time: 1.058 ms

INDEXES:

GET /user/{user_id}/projection

CREATE INDEX ON projection(user_id)

- This will improve the performance of this endpoint as the executor previously had to do a full table scan of the projection table in order to find all the projections with the same user_id. This will no longer occur as the indexes will have pointers to the locations projections which match the user_id

POST /users/

- Due to this being a post, there isn't rally an index that could improve this endpoint since there is no querying the database

GET /users/{user_id}

CREATE INDEX ON users (id)

- This will improve the performance of this endpoint as it queries for the specific user based on their id. Therefore, the executor won't have to traverse the whole page in a sequential scan and can rather use the indexing of the id

POST /users/login

CREATE INDEX ON users(name)

- In this endpoint, a query is run to find the user with the associated name, therefore by adding an index for the users table based on the name, there will be a faster lookup rather than a full table scan

GET /workouts/{user_id}

CREATE INDEX ON workouts(user_id)

- This will improve the performance of this endpoint as the associated workout will be indexed via the user id, where as before the endpoint would have to traverse the entire page of workouts to find the right workout, but now that is not an issue as the index will provide a reference to the proper workout

GET /user/{user_id}/logs

CREATE INDEX ON logs(user_id)

- This will improve the performance of this endpoint as it queries the logs table to find the logs with the same user_id as the one passed in. As a result, the executor won't have to do a full scan of the logs table and will be able to look up the logs based on the

POST /user/{user_id}/logs

- This endpoint would not benefit from any indexes as the only relevant index would to be to index the user table based on id to find the user, but this won't provide a large performance impact as the id is the users table's primary key

POST /goals/

- Much like the previous endpoint, there is only a query to search for the associated user, which is handled by the primary key of the id

POST /user/{user_id}/projection

CREATE INDEX ON logs(user_id)

- This will improve the performance of this endpoint as it queries the logs table to find the logs with the same user_id as the one passed in. As a result, the executor won't have to do a full scan of the logs table and will be able to look up the logs based on the

GET /user/{user_id}/projection

CREATE INDEX ON projection(user_id)

- This will improve the performance of this endpoint as the executor previously had to do a full table scan of the projection table in order to find all the projections with the same user_id. This will no longer occur as the indexes will have pointers to the locations projections which match the user_id

AFTER INDEXING

POST /users/

```
explain analyze
insert into users (starting_lbs, name, height_inches, avg_calorie_intake,
birthday, gender, password, salt)
values (200, 'jonny', 78, 2250, '2002-10-07', 'M', null, null)
```

Insert on users (cost=0.00..0.01 rows=0 width=0) (actual time=0.485..0.485 rows=0 loops=1)
  -> Result (cost=0.00..0.01 rows=1 width=160) (actual time=0.159..0.159 rows=1 loops=1)
Planning Time: 0.082 ms
Execution Time: 0.552 ms

GET /users/{user_id}

```
explain
SELECT id, name, starting_lbs, height_inches, avg_calorie_intake, birthday,
gender FROM users WHERE id = 2600009
```

Index Scan using users_id_idx on users (cost=0.42..8.44 rows=1 width=48)
  Index Cond: (id = 2600009)

POST /users/login

```
explain
SELECT salt, password FROM users WHERE name = 'Jennifer Lopez'
```

Bitmap Heap Scan on users (cost=4.44..16.16 rows=3 width=64)
  Recheck Cond: (name = 'Jennifer Lopez'::text)
  -> Bitmap Index Scan on users_name_idx (cost=0.00..4.44 rows=3 width=0)
        Index Cond: (name = 'Jennifer Lopez'::text)

GET /workouts/{user_id}

```
explain
SELECT id FROM users WHERE id = 200000
```

Index Only Scan using users_id_idx on users (cost=0.42..8.44 rows=1 width=4)
  Index Cond: (id = 200000)

```
explain
SELECT id, workout_name, weight, distance_ft, repetitions, seconds, sets,
times_per_week FROM workouts WHERE user_id = 200000
```

Index Scan using workouts_user_id_idx on workouts (cost=0.42..8.44 rows=1 width=56)
  Index Cond: (user_id = 200000)

GET /user/{user_id}/logs

```
explain
SELECT id, name FROM users WHERE id = 197859
```

Index Scan using users_id_idx on users (cost=0.42..8.44 rows=1 width=18)
  Index Cond: (id = 197859)

```
explain
SELECT id, current_lbs, time_posted FROM logs WHERE user_id= 197859
```

Index Scan using logs_user_id_idx2 on logs (cost=0.43..8.57 rows=8 width=20)
  Index Cond: (user_id = 197859)

POST /user/{user_id}/logs

```
Explain analyze
SELECT id FROM users WHERE id = 2600025
```

Index Only Scan using users_pkey on users (cost=0.42..4.44 rows=1 width=4) (actual time=0.033..0.034 rows=1 loops=1)
  Index Cond: (id = 2600025)
  Heap Fetches: 0
Planning Time: 0.397 ms
Execution Time: 0.200 ms

```
Explain analyze
INSERT INTO logs (user_id, current_lbs, time_posted)
VALUES (2600025,145, CURRENT_TIMESTAMP)
RETURNING id
```

Insert on logs (cost=0.00..0.02 rows=1 width=28) (actual time=0.470..0.471 rows=1 loops=1)
  -> Result (cost=0.00..0.02 rows=1 width=28) (actual time=0.084..0.084 rows=1 loops=1)

Planning Time: 0.090 ms
Execution Time: 0.548 ms

POST /goals/

```
explain
SELECT starting_lbs, birthday, gender, height_inches FROM users WHERE id =
2600105
```

Index Scan using users_id_idx on users (cost=0.42..8.44 rows=1 width=22)
 Index Cond: (id = 2600105)

```
Explain analyze
INSERT INTO workouts (workout_name, weight, distance_ft, repetitions, seconds,
sets, times_per_week, user_id)
VALUES ('Run', 0, 64114, NULL, NULL, NULL, 7, 178599);
```

Insert on workouts (cost=0.00..0.01 rows=0 width=0) (actual time=0.402..0.403 rows=0 loops=1)
  -> Result (cost=0.00..0.01 rows=1 width=92) (actual time=0.088..0.088 rows=1 loops=1)
Planning Time: 0.091 ms
Execution Time: 0.476 ms

```
Explain analyze
INSERT INTO goals (type_id, user_id, target_weight, workout_id)
VALUES (0, 2600105, 55, 2800006);
```

Insert on goals (cost=0.00..0.01 rows=0 width=0) (actual time=0.235..0.235 rows=0 loops=1)
 -> Result (cost=0.00..0.01 rows=1 width=36) (actual time=0.088..0.088 rows=1 loops=1)
Planning Time: 0.093 ms
Execution Time: 0.304 ms

POST /user/{user_id}/projection

```
Explain analyze
SELECT FROM users WHERE id = 2600324
```

Index Only Scan using users_id_idx on users (cost=0.42..8.44 rows=1 width=0) (actual time=0.036..0.036 rows=1 loops=1)
  Index Cond: (id = 2600324)
  Heap Fetches: 0
Planning Time: 0.392 ms
Execution Time: 0.129 ms

```
Explain analyze
SELECT current_lbs, time_posted FROM logs WHERE user_id = 2600324
ORDER BY time_posted
```

Sort (cost=8.69..8.71 rows=8 width=16) (actual time=0.444..0.445 rows=4 loops=1)
  Sort Key: time_posted
  Sort Method: quicksort Memory: 25kB
  -> Index Scan using logs_user_id_idx2 on logs (cost=0.43..8.57 rows=8 width=16) (actual time=0.412..0.415 rows=4 loops=1)
        Index Cond: (user_id = 2600324)
Planning Time: 0.504 ms
Execution Time: 0.552 ms

```
Explain analyze
INSERT INTO projection (user_id, projection_lbs, projection_date, date_posted)
VALUES (2600324, 80, '2023-04-07', CURRENT_TIMESTAMP);
```

Insert on projection (cost=0.00..0.02 rows=0 width=0) (actual time=0.285..0.286 rows=0 loops=1)
  -> Result (cost=0.00..0.02 rows=1 width=32) (actual time=0.064..0.065 rows=1 loops=1)
Planning Time: 0.074 ms
Trigger for constraint projection_user_id_fkey: time=0.646 calls=1
Execution Time: 1.077 ms