Opera

Data races when writing to a db without locks

Opera

Introduction

Who am I?

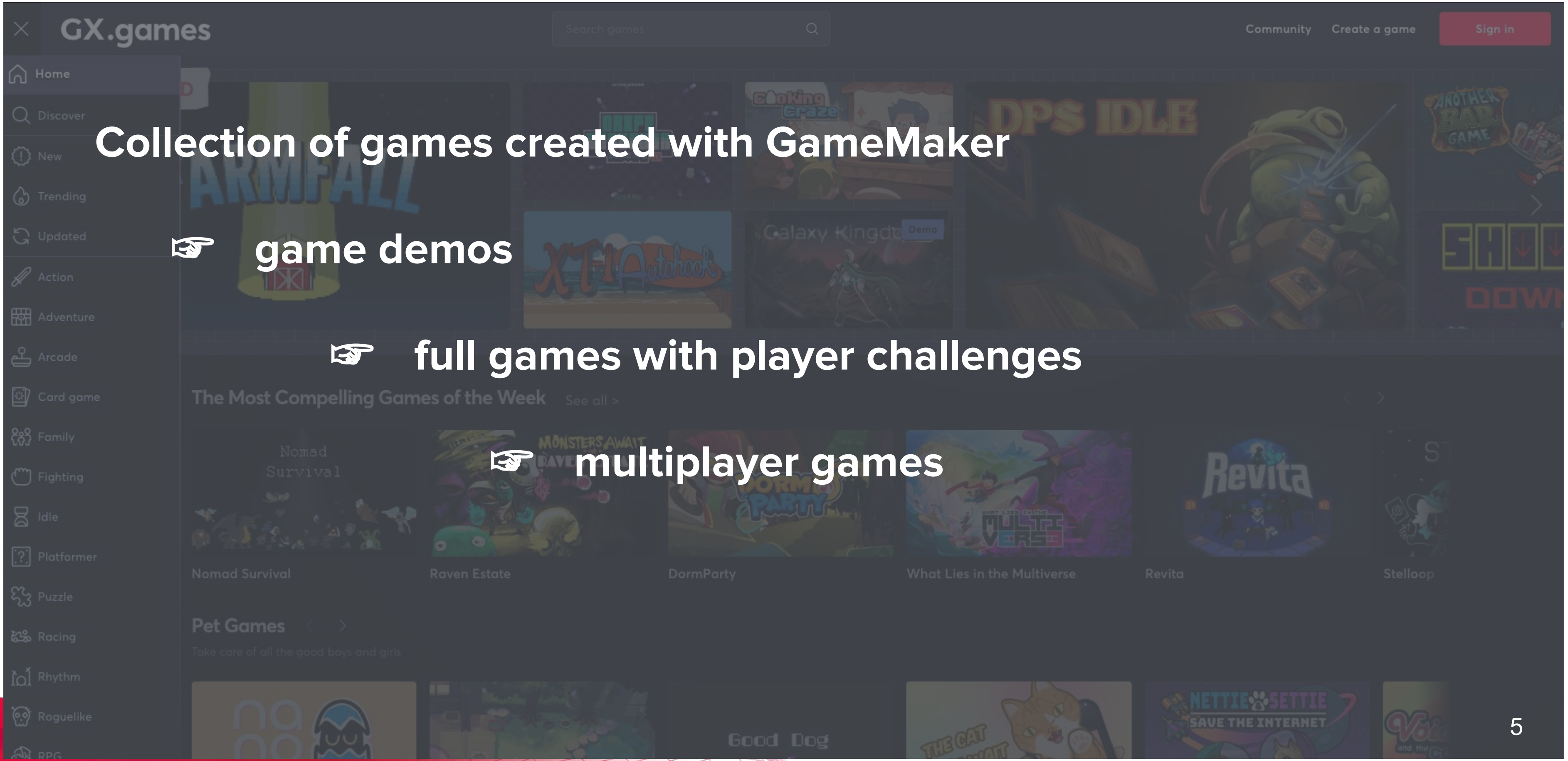
- Denis Furian
- previously Chalmers student (MPALG 2017–2020)
- nowadays working at Opera in Gothenburg
    - Android developer some projects ago
    - now back end engineer for GX.games

Introduction

Opera

- founded in Norway 1995
- 1997: first browser (Opera 2.1 for Windows)
- 2005: browser for mobile phones (Opera Mini)
- 2019: new browser for gamers (Opera GX)
- 2021: acquired YoYo Games and GameMaker
- 2022: launch of GameMaker storefront GX.games
- 2023: launch of mods support for Opera GX

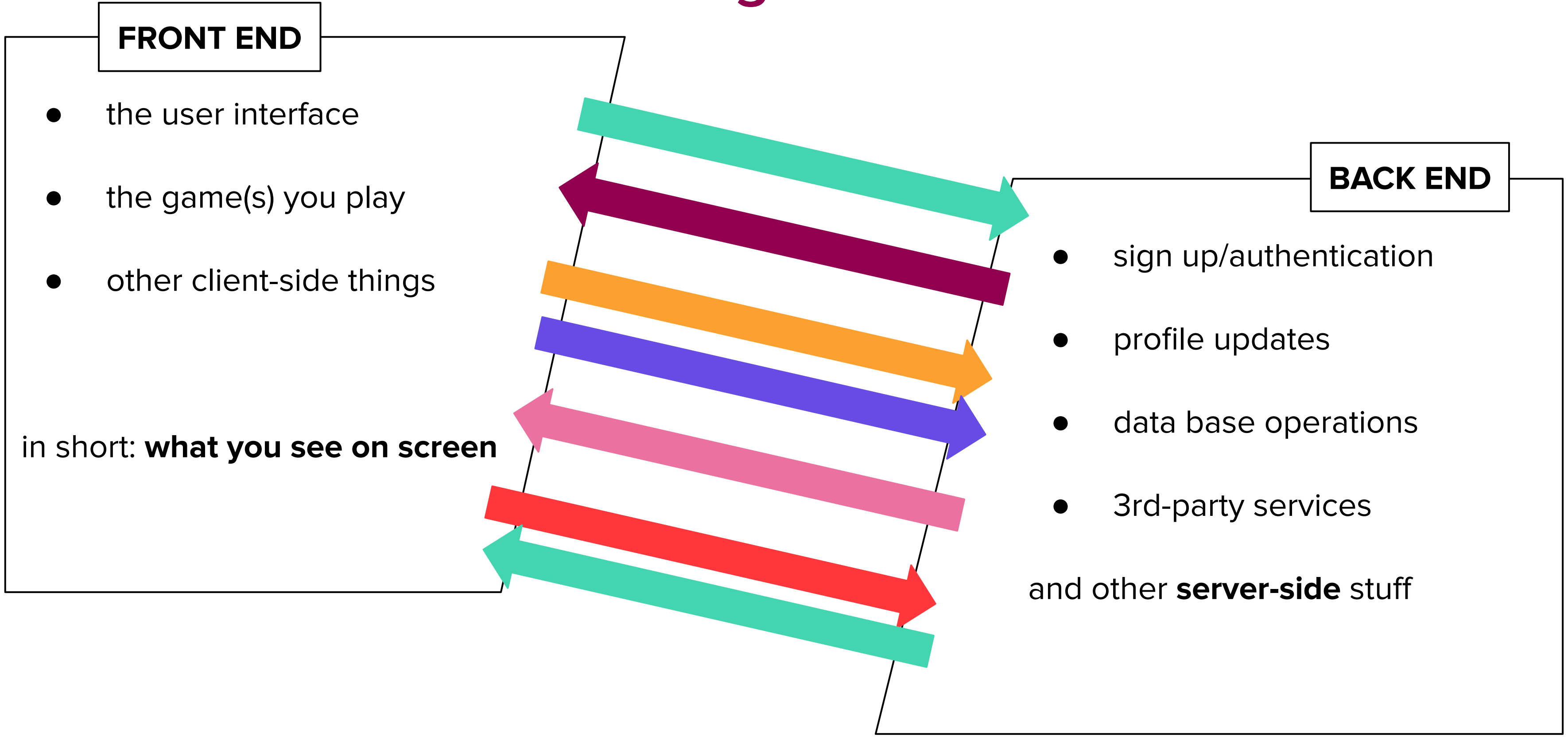
GX.games



Collection of games created with GameMaker

- ☞ game demos
- ☞ full games with player challenges
- ☞ multiplayer games

GX.games



FRONT END

- the user interface
- the game(s) you play
- other client-side things

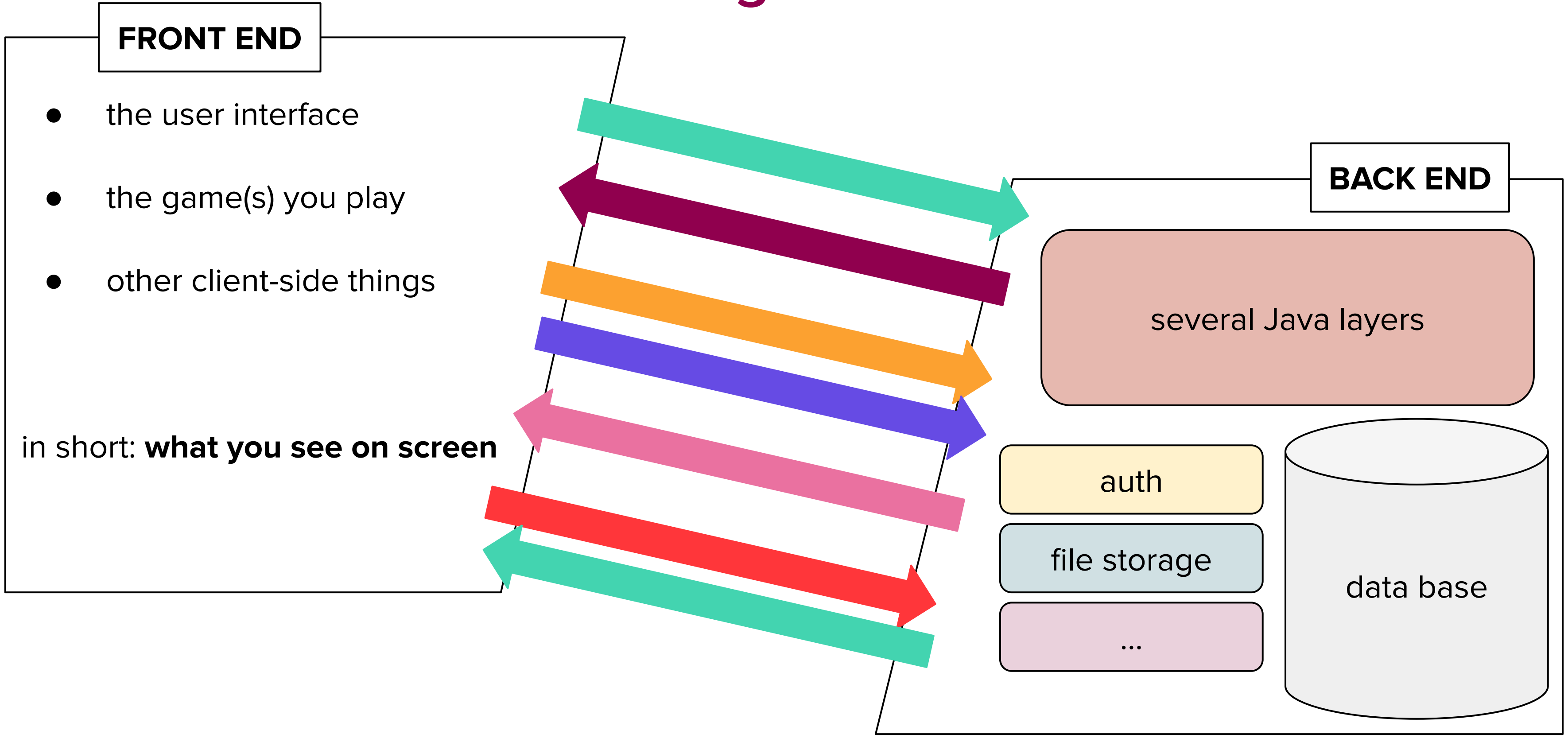
in short: **what you see on screen**

BACK END

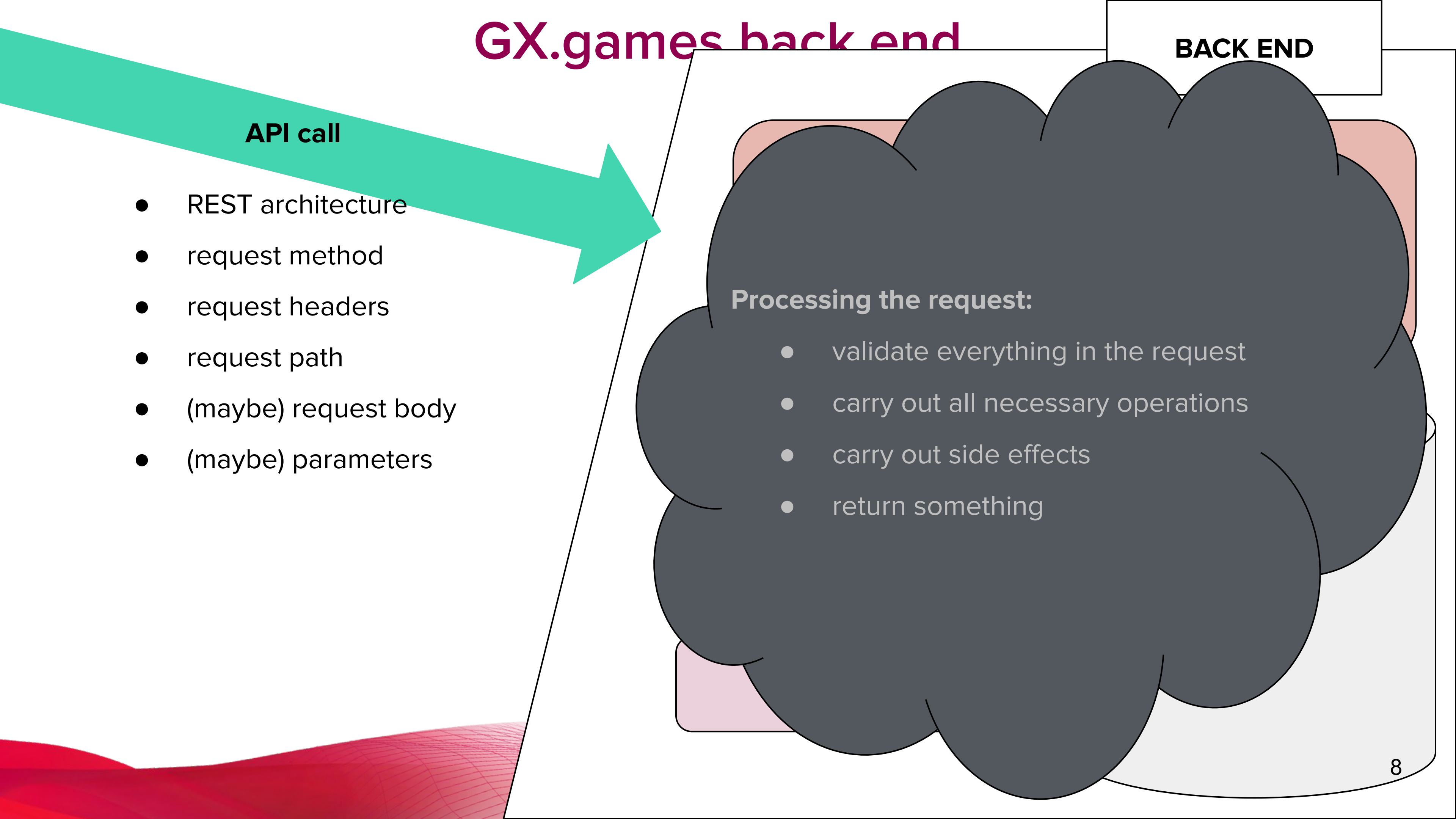
- sign up/authentication
- profile updates
- data base operations
- 3rd-party services

and other **server-side** stuff

GX.games


FRONT END
the user interface
the game(s) you play
other client-side things
in short: **what you see on screen**
BACK END
several Java layers
auth
file storage
...
data base

GX.games back end



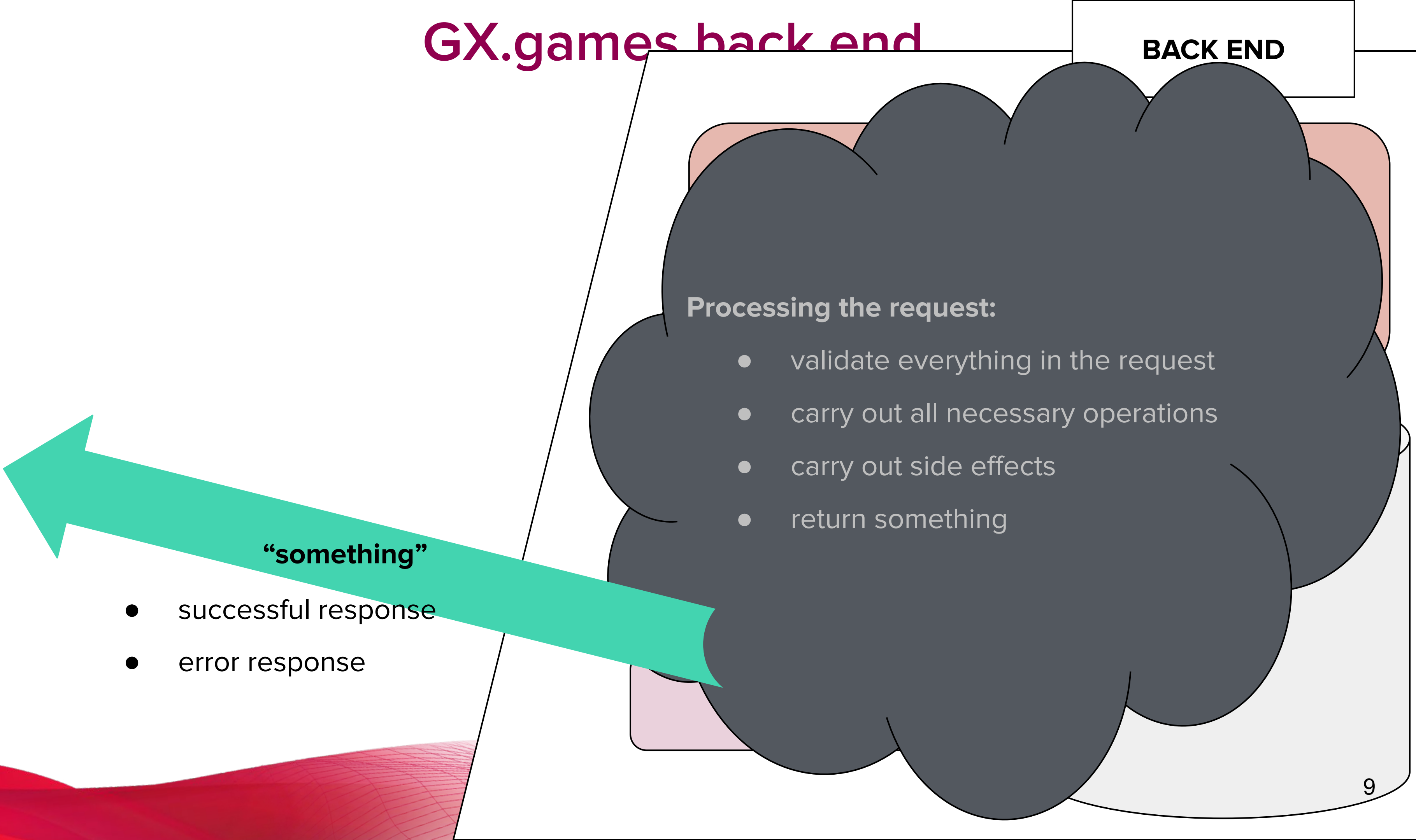
API call

- REST architecture
- request method
- request headers
- request path
- (maybe) request body
- (maybe) parameters

Processing the request:

- validate everything in the request
- carry out all necessary operations
- carry out side effects
- return something

GX.games back end



Processing the request:

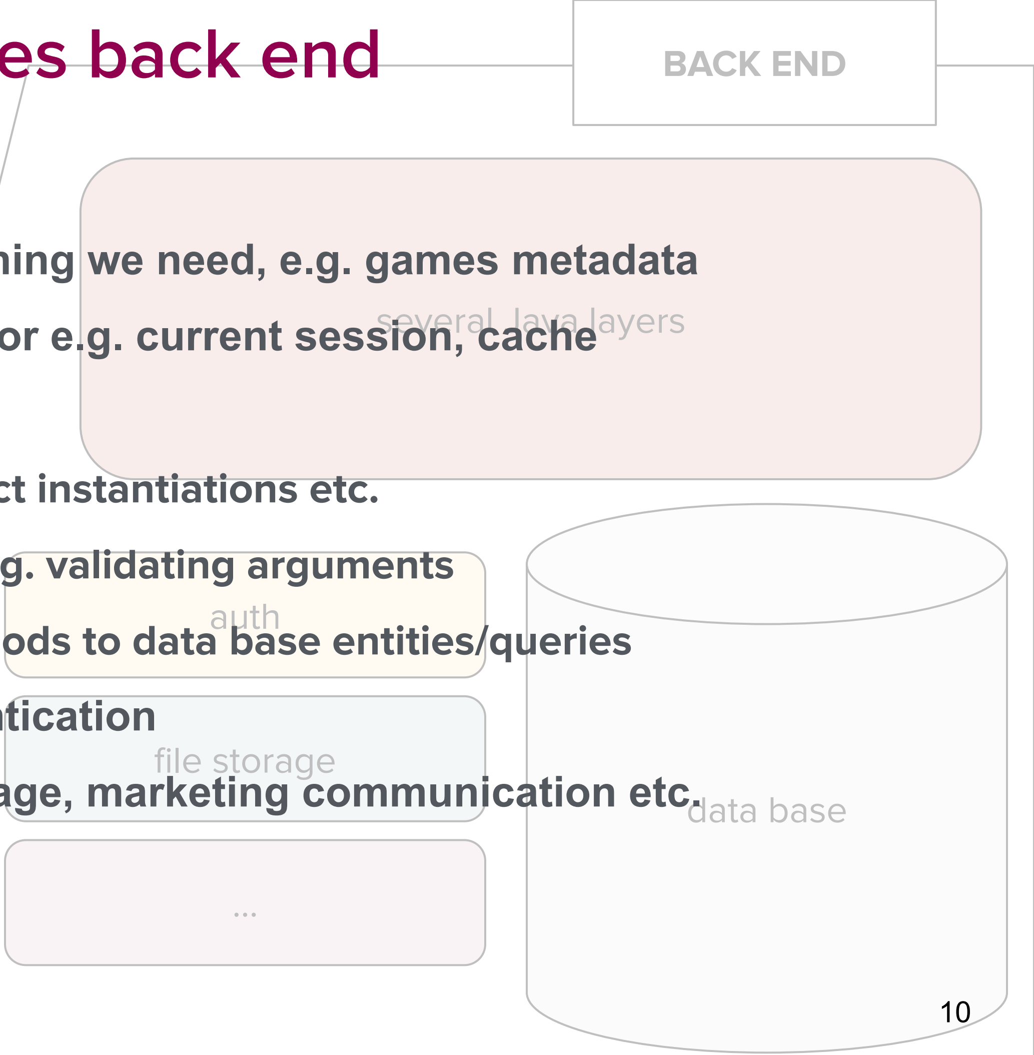
- validate everything in the request
- carry out all necessary operations
- carry out side effects
- return something

- successful response
- error response

GX.games back end

The BE infrastructure, simplified:

- PostgreSQL data base: storing everything we need, e.g. games metadata
- Redis: temporary in-memory storage for e.g. current session, cache
- Several Java frameworks:
  - Spring: handling transactions, object instantiations etc.
  - AspectJ: aspect-oriented checks, e.g. validating arguments
  - Hibernate: maps Java classes/methods to data base entities/queries
- Other Opera services, e.g. user authentication
- Several 3rd-party services for file storage, marketing communication etc.



GX.games back end

Today's focus on:

data base and OO implementation

“object-oriented”

robustness against data races

PostgreSQL: tables and relations

A table:

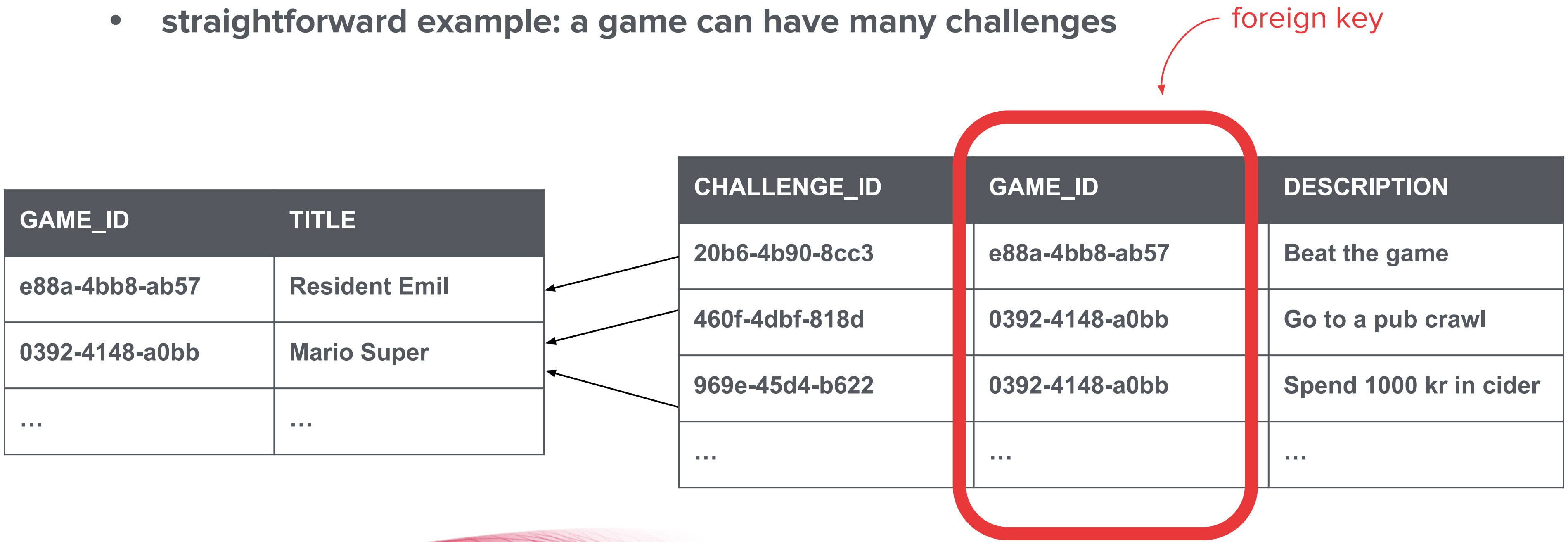
- generally identifies an entity (user, game, score, mod etc.)
- contains data about that entity (user: name, email, birthdate, …)
- each table row is a separate entity

USER_ID	NAME	EMAIL	BIRTHDATE
fc95-43f8-bc85	Emilia Emilsson	emili@mail.se	2001-12-03
6081-4613-b547	Someone Elsson	som1els@gmail.com	1998-03-16
bad9-4008-a292	Åså Vidarsson	osv@mail.com	2004-09-27
…	…	…	…

PostgreSQL: tables and relations

A relation:

- is a relationship between two entities:
  - straightforward example: a game can have many challenges



GAME_ID	TITLE
e88a-4bb8-ab57	Resident Emil
0392-4148-a0bb	Mario Super
…	…

CHALLENGE_ID	GAME_ID	DESCRIPTION
20b6-4b90-8cc3	e88a-4bb8-ab57	Beat the game
460f-4dbf-818d	0392-4148-a0bb	Go to a pub crawl
969e-45d4-b622	0392-4148-a0bb	Spend 1000 kr in cider
…	…	…

PostgreSQL: tables and relations

A relation:

- is a relationship between two entities:
  - straightforward example: a game can have many challenges
  - slightly more complex: a user can have friends

USER_ID	NAME
fc95-43f8-bc85	Emilia Emilsson
6081-4613-b547	Someone Elsson
bad9-4008-a292	Åså Vidarsson
...	...

a relationship can be a table

USER_1_ID	USER_2_ID	DATE_ADDED
fc95-43f8-bc85	6081-4613-b547	2023-08-13
bad9-4008-a292	fc95-43f8-bc85	2023-10-02
...	...	...

PostgreSQL: operations with tables

Tables can be created

modified (by e.g. adding constraints or altering columns)

deleted

updated (i.e. you can add/remove rows or edit a specific one)

joined together

queried (i.e. you can look up the data in one or more tables)

Hibernate: mapping tables

Table

- one or more columns
- each column has a type
- some columns have constraints
- some can reference other tables

CHALLENGE_ID	GAME_ID	DESCRIPTION
...	...	...

Class

- one or more fields
- each field has a type
- some fields have complex types

```
public class Challenge {
   @Id @Column
   public String challengeId;
   @Column
   public String gameId;
   @Column
   public String description;
   @ManyToOne @JoinColumn
   public Game game;
}
```

Hibernate: mapping operations

Repository class:

- each method is equivalent to a data base operation
- the method arguments (if any) become parameters
- the method return type will depend on the operation:
    - if it's a query, it might be an object (or a list of objects)
    - otherwise, it might be the number of rows inserted/deleted/updated

How to implement all this?

Spring Data JPA library

- **Implements repository classes under the hood**
- **Provides base operations off the bat**

```
save(), delete(), findAll(), findById(String id), ...
```

- **Tools for creating db operations using keywords**

```
Optional<Challenge> findFirstByGameIdOrderByNameAsc(String gameId);
```

can return a record, or nothing

only return the first record

search a specific column

sort A–Z based on another column

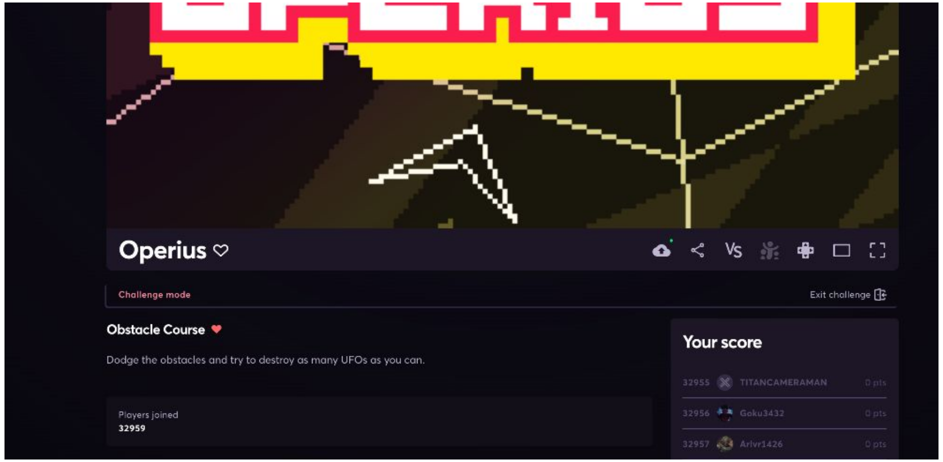
- **...or you can write your own SQL queries via annotations**

```
@Query("SELECT description FROM challenges WHERE game_id = :gameId")
List<Challenge> foo(String gameId);
```

Now for a practical example

What we want to do:

- let’s allow a user to save one or more challenges as “favorites”;
- when a user views a challenge on GX.games, we should tell if it’s a favorite.



Obstacle Course ♥

Dodge the obstacles and try to destroy as many UFOs as you can.

Players joined
32959

Your score

32955	TITANCAMERAMAN	0 pts
32956	Goku3432	0 pts
32957	Arlvr1426	0 pts

Now for a practical example

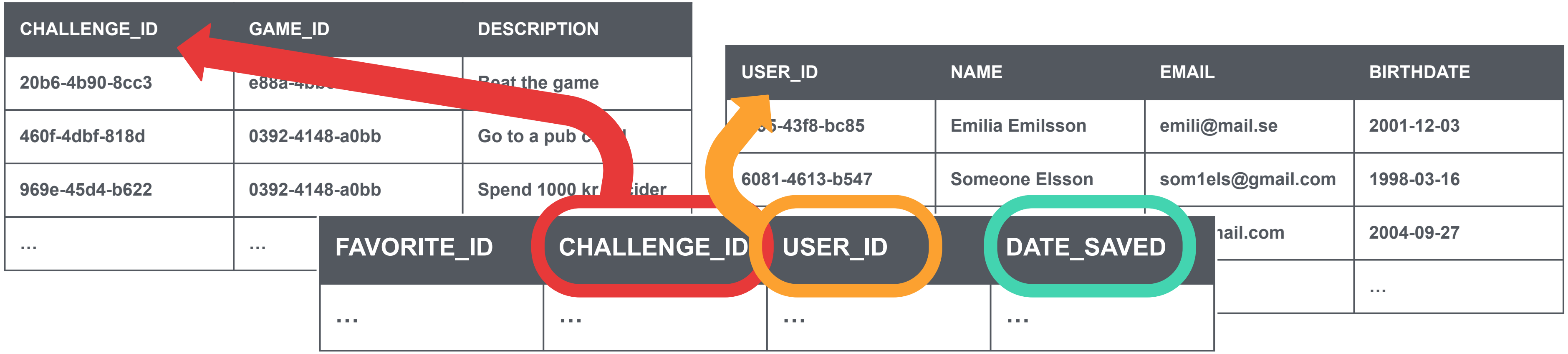
How do we do it?

1. we create a table for “favorites”
   - it must contain information about the challenge, and the user!
2. we create an API for managing favorites
   - users should be able to save a favorite
   - they should also see if a given challenge is a favorite

Now for a practical example

1. Creating a table for favorites

- ☞ it must contain information about the **challenge**, and the **user**!
- ☞ plus a **timestamp** for when the challenge was saves as favorite



CHALLENGE_ID	GAME_ID	DESCRIPTION
20b6-4b90-8cc3	e88a-4bb	Beat the game
460f-4dbf-818d	0392-4148-a0bb	Go to a pub c
969e-45d4-b622	0392-4148-a0bb	Spend 1000 kr ider
…	…	

USER_ID	NAME	EMAIL	BIRTHDATE
5-43f8-bc85	Emilia Emilsson	emili@mail.se	2001-12-03
6081-4613-b547	Someone Elsson	som1els@gmail.com	1998-03-16
		ail.com	2004-09-27
			…

FAVORITE_ID	CHALLENGE_ID	USER_ID	DATE_SAVED
…	…	…	…

Now for a practical example

1. Creating a table for favorites

☞ it must contain information about the challenge, and the user!

☞ plus a timestamp for when the challenge was saves as favorite

CHALLENGE_ID	GAME_ID
20b6-4b90-8cc3	e88a-4bb
460f-4dbf-818d	0392-4148-a0bb
969e-45d4-b622	0392-4148-a0bb
…	…

FAVORITE_
…

```
public class Favorite {
  @Id @Column(name = “favorite_id”)
  public String favoriteId;

  @Column(name = “challenge_id”)
  public String challengeId;

  @Column(name = “user_id”)
  public String userId;

  @Column(name = “date_saved”)
  public Timestamp dateSaved;
}
```

D

EMAIL	BIRTHDATE
emili@mail.se	2001-12-03
som1els@gmail.com	1998-03-16
nail.com	2004-09-27
	…

Now for a practical example

2. Creating the API

☞ **need to save a challenge as favorite!**

repository class

entity class from the last slide

```
// Initialized by Spring.
@Autowired
private FavoriteRepository favoriteRepository;

// ...

// Create new entity class with the data we need.
Favorite newFavorite = new Favorite();
newFavorite.challengeId = myChallenge.challengeId;
newFavorite.userId = currentUser.userId;
newFavorite.dateSaved = Timestamp.now();

// Hibernate provides a method to save the new object to the db.
favoriteRepository.save(newFavorite);
```

Now for a practical example

2. Creating the API

- ☞ need to save a challenge as favorite!

Done! ✔
Let's call it "**saveFavorite**"

- ☞ need to return info about favorite…

JSON response example:

```
{
  "isFavorite": true | false,
  "dateSaved": "2023-10-13" | null
}
```

✔ **we can have Spring Data JPA take care of this!**

Now for a practical example

2.

☞ nee

```
// Initialized by Spring.
@Autowired
private FavoriteRepository favoriteRepository;

// ...

// Look up a favorite challenge, if it exists.
Optional<Favorite> maybeFavorite =
    favoriteRepository
        .findByChallengeIdAndUserId(
            myChallenge.challengeId, currentUser.userId);

Response response = new Response();
if (maybeFavorite.isPresent()) {
  response.isFavorite = true;
  response.dateSaved = maybeFavorite.get().dateSaved;
} else {
  response.isFavorite = false;
}
return response;
```

repository class (same as before)

query generated using keywords

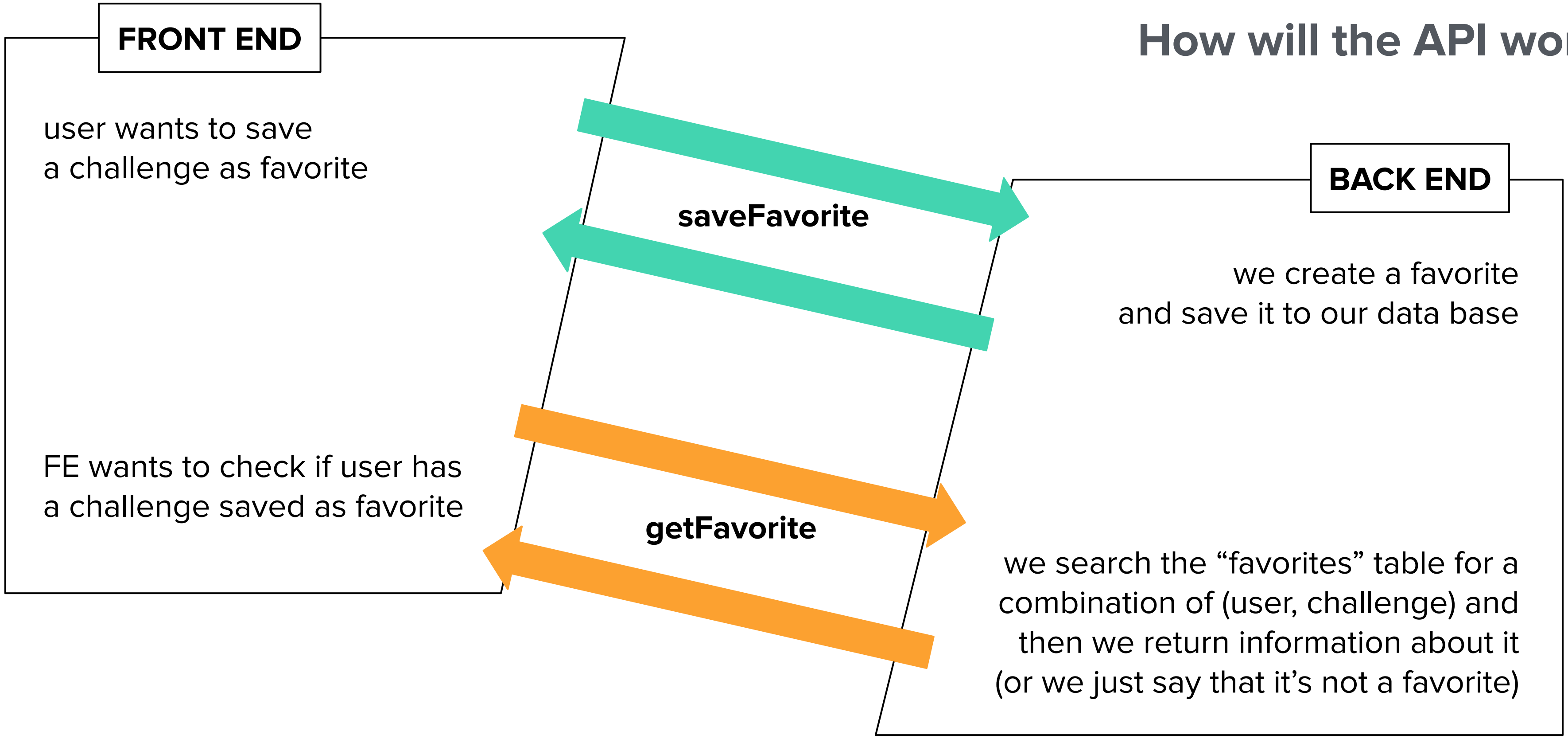
Now for a practical example

2. Creating the API

- ☞ need to save a challenge as favorite! Done! ✔ Let’s call it “**saveFavorite**”
- ☞ need to return info about favorite… Also done! ✔ Let’s call it “**getFavorite**”

Now for a practical example

How will the API work?



user wants to save
a challenge as favorite

we create a favorite
and save it to our data base

FE wants to check if user has
a challenge saved as favorite

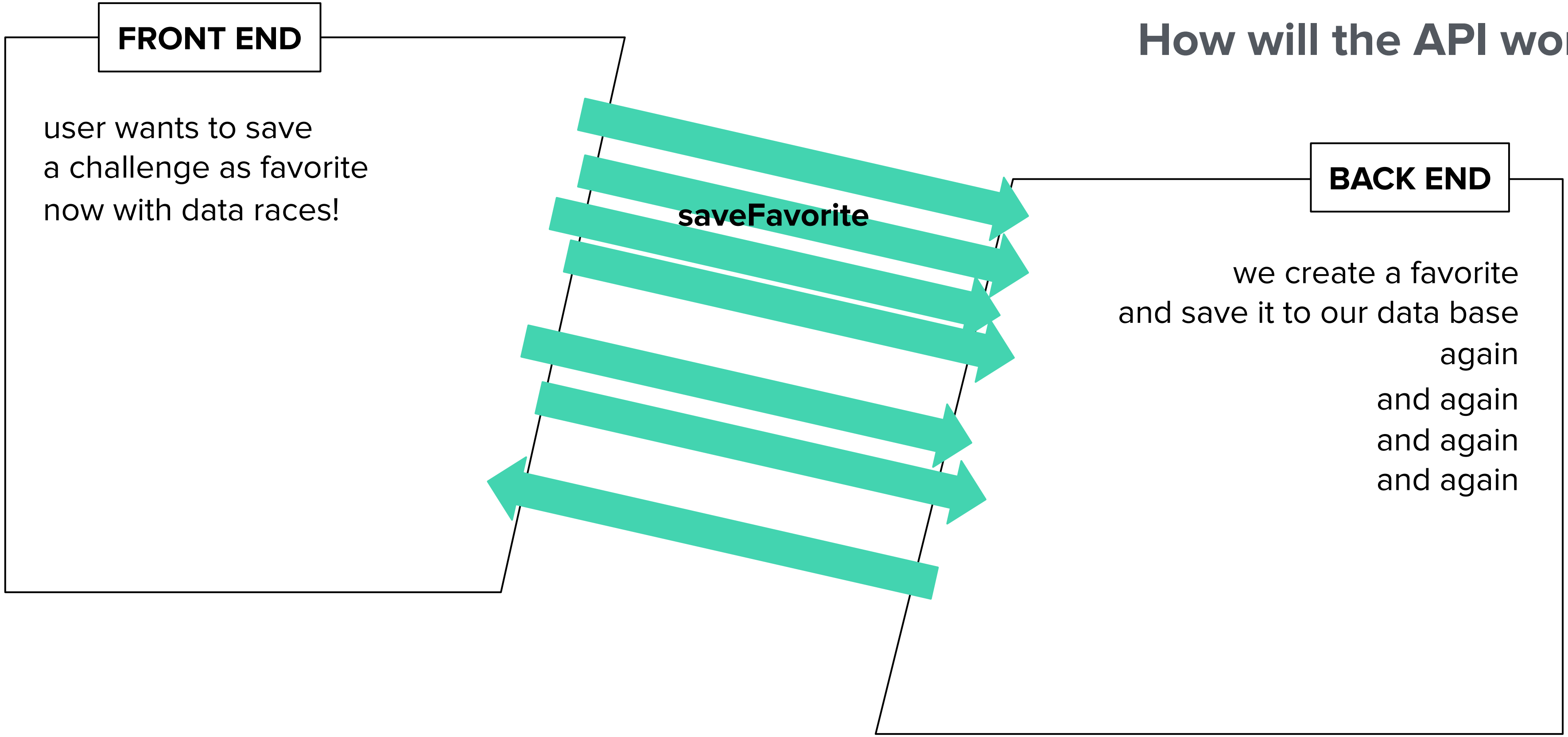
we search the “favorites” table for a
combination of (user, challenge) and
then we return information about it
(or we just say that it’s not a favorite)

Now for a practical example

Real life scenario:
adding data races

Now for a practical example

How will the API work?



Now for a practical example

How will the API work?

FRONT END

user wants to save
a challenge as favorite
now with data rac

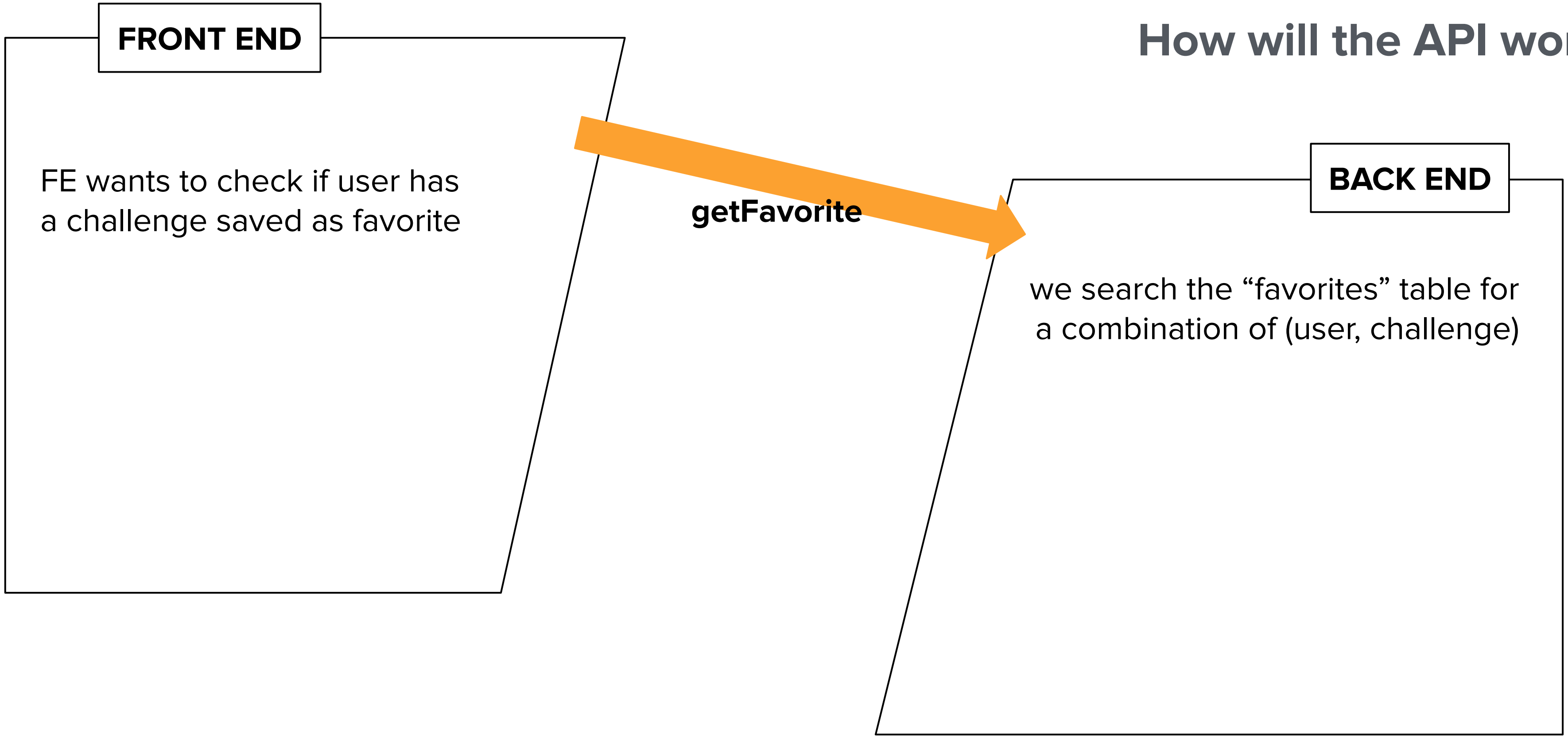
BACK END

e create a favorite
it to our data base
again
and again
and again
and again

FAVORITE_ID	CHALLENGE_ID	USER_ID	DATE_SAVED
23bd-4cb0-b1ff	c55a-497c-89c8	95c8-4dc7-83a7	2023-10-13
b155-4cd0-b570	c55a-497c-89c8	95c8-4dc7-83a7	2023-10-13
3a0b-4b74-b8f0	c55a-497c-89c8	95c8-4dc7-83a7	2023-10-13
71e7-4462-a3a3	c55a-497c-89c8	95c8-4dc7-83a7	2023-10-13
7e46-484e-9daa	c55a-497c-89c8	95c8-4dc7-83a7	2023-10-13
c7ab-4940-94a3	c55a-497c-89c8	95c8-4dc7-83a7	2023-10-13

Now for a practical example

How will the API work?



FE wants to check if user has a challenge saved as favorite

we search the “favorites” table for a combination of (user, challenge)

Now for a practical example

API work?

FE wa
a cha

END

ble for
enge)

```
// Initialized by Spring.
@Autowired
private FavoriteRepository favoriteRepository;

// ...

// Look up a favorite challenge, if it exists.
Optional<Favorite> maybeFavorite =
    favoriteRepository
        .findByChallengeIdAndUserId(
            myChallenge.challengeId, currentUser.userId);

Response response = new Response();
if (maybeFavorite.isPresent()) {
  response.isFavorite = true;
  response.dateSaved = maybeFavorite.get().dateSaved;
} else {
  response.isFavorite = false;
}
return response;
```

Something has gone wrong!

What’s going on?

An `Optional<T>` object can contain an instance of type `T`, or it can be empty.

- **we are looking for one favorite entry at most**
- **Spring “converts” this to an** `Optional<Favorite>`
- **however, there are several entries because of the data race!**
- **Spring can’t convert a list of records to an** `Optional`

↳ **this triggers a casting exception**

☞ **this snowballs into a** `500 internal server error`

⚠️ **no mention of data races or duplicate favorite entries!**

Handling a data race

fix it when it happens

in other words: delete duplicate favorites

- **have to detect them in the first place**
- **not sustainable**

prevent it in the first place

have to make sure only one record exists for every combination (user, challenge)

⬇

we should introduce locks to our code base

Using locks on the Favorites table

SQL data bases use a set of keywords for implementing locks

- `SELECT ... FOR SHARE` → **locks concurrent writes, but allow reads**
- `SELECT ... FOR UPDATE` → **locks concurrent reads and writes**
- ...

On PostgreSQL, this locks the selected record(s)and prevents other operations until the transaction is finished.

Using locks provided by Spring

Repository classes can use keywords for generated query methods

- just add “`forUpdate`” to method name:

```
@Lock(PESSIMISTIC_WRITE)
Optional<User> findForUpdateByUserId(String userId);
```

Spring provides an “`EntityManager`” class to manage entity updates

- this way, we can get a lock on an object:

```
User currentUser = getCurrentUser();
entityManager.refresh(currentUser, PESSIMISTIC_WRITE);
```

Using locks provided by Spring

Repository classes can use keywords for generated query methods

- just add "forUpdate" to method name:

```
@Lock(PESSIMISTIC_WRITE)
Optional<User> findForUpdateByU
```

We can specify the type of lock we want:

- "Pessimistic" locks try and avoid conflicts by locking a row entirely;
- "Optimistic" locks check for conflicts before committing a transaction, and any conflict will cause a rollback.

Spring provides an "EntityManager" class to manage entity updates

- this way, we can get a lock on an object:

```
User currentUser = getCurrentUser();
entityManager.refresh(currentUser, PESSIMISTIC_WRITE);
```

Using locks… But where?

- On a query?
- On a repository method?
- On an object?

The real question is: what concurrent update are we trying to prevent?

☞ we want to prevent a user from saving the same challenge more than once

Using locks on `User` entities

1. **Update "`User`" class to have a list of favorites!**

   How?

   Using the `@OneToMany` annotation:

```
public class User {
  // ...

  @OneToMany
  public List<Favorite> favorites;
}
```

2. **Lock the user object before creating a new favorite.**

   By using the `EntityManager` we can prevent concurrent updates on the whole object, including the new list of favorites.

Saving a favorite, revisited

```
// Initialized by Spring.
@Autowired
private FavoriteRepository favoriteRepository;

// ...

// Create new entity class with the data we need.
Favorite newFavorite = new Favorite();
newFavorite.challengeId = myChallenge.challengeId;
newFavorite.userId = currentUser.userId;
newFavorite.dateSaved = Timestamp.now();

// Hibernate provides a method to save the new object to the db.
favoriteRepository.save(newFavorite);
```

Add the `EntityManager`

Refactor this code: we are locking the `currentUser` and adding a new entry to `currentUser.favorites`

Saving a favorite, revisited

```
// Initialized by Spring
@Autowired
private UserRepository userRepository;
@Autowired
private EntityManager entityManager;

// ...

// Lock the user object to prevent concurrent updates.
entityManager.refresh(currentUser, PESSIMISTIC_WRITE);

// Create new entity class just like before.
Favorite newFavorite = new Favorite();
newFavorite.challengeId = myChallenge.challengeId;
newFavorite.userId = currentUser.userId;
newFavorite.dateSaved = Timestamp.now();

// Save the user (and the new favorite) to the db.
currentUser.favorites.add(newFavorite);
userRepository.save(currentUser);
```

Hindsight

Could we have prevented this?

Of course 🥲

Uniqueness constraint ➔ by making the combination of *(user_id, challenge_id)* unique, we can discard any duplicates automatically!

⬇

We can also “promote” the combination of *(user_id, challenge_id)* to be primary key instead of *favorite_id*, since a primary key is always unique!

⬇

This will prevent duplicates when creating favorites… But we will still need a lock if we want to update an existing favorite.

Hindsight

Should we make *(user_id, challenge_id)* unique?

Sure! 👍

- **one more failsafe**
- **it makes sense for the table**

Different cases will require different solutions:

- what entity should be locked
- what kind of lock should be used

$\left\{ \begin{array}{l} \text{optimistic vs pessimistic} \\ \\ \\ \text{query vs code} \end{array} \right.$

More info

- **PostgreSQL →** https://www.postgresql.org/
- **Java resources**
  - **Hibernate →** https://hibernate.org/
  - **Spring →** https://spring.io/
- **Opera**
  - **GX browser →** https://www.opera.com/gx
  - **GX.games →** https://gx.games/

Thanks for watching!

Opera

Denis Furian
denisf@opera.com