

# Asynchronous Domain Model in Javascript

Asynchronous Idioms

Exploit Concurrency

Asynchronous Domain Models

# Different Kinds of Data

```

- <country>
  - <![CDATA[
    Portugal
  ]]>
</country>
- <region>
  - <![CDATA[
    Lisboa
  ]]>
</region>
<latitude>38.717</latitude>
<longitude>-9.133</longitude>
<population>51779</population>
- <weatherUrl>
  - <![CDATA[
    https://www
  ]]>
</weatherUrl>
</result>
- <result>
  - <areaName>
    - <![CDATA[
      Lisbon
    ]]>
  </areaName>
  - <country>
    - <![CDATA[
      United States
    ]]>
  </country>
  - <region>
    - <![CDATA[
      Maine
    ]]>
  </region>
  <latitude>44.031</latitude>
  <longitude>-70.105</longitude>
  <population>9392</population>

```

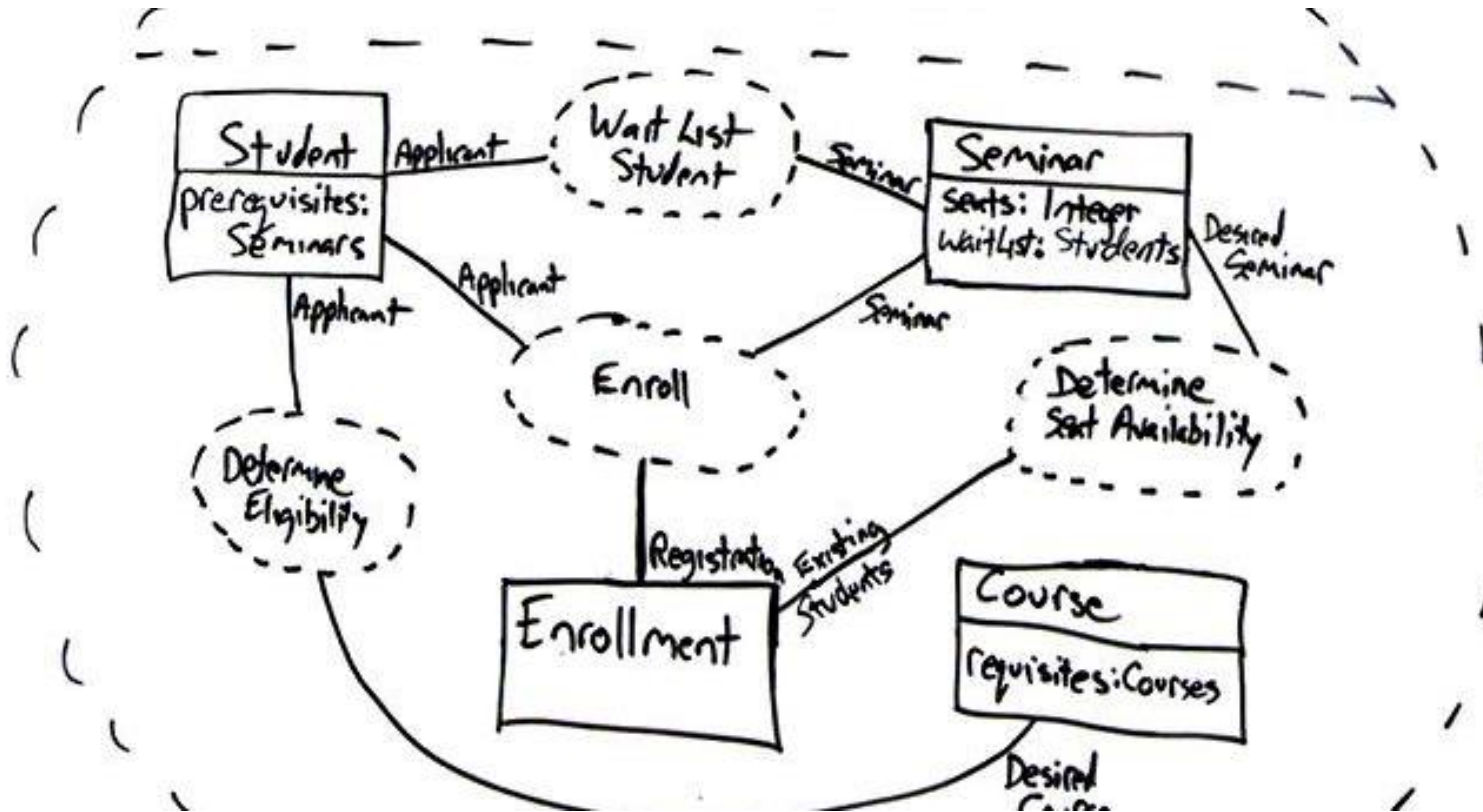
```

{
  "toptracks": {
    "track": [
      {
        "name": "Panic Station",
        "playcount": "2804955",
        "listeners": "331488",
        "mbid": "a7ec9417-5b63-4995-9c3e-d80c593d0d53",
        "url": "https://www.last.fm/music/Muse/_/Panic+Station",
        "streamable": "0",
        "artist": {
          "name": "Muse",
          "mbid": "fd857293-5ab8-40de-b29e-55a69d4e4d0f",
          "url": "https://www.last.fm/music/Muse"
        }
      }
    ]
  }
}

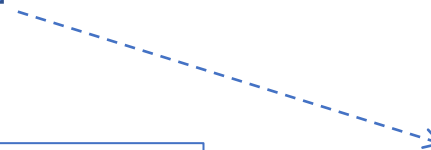
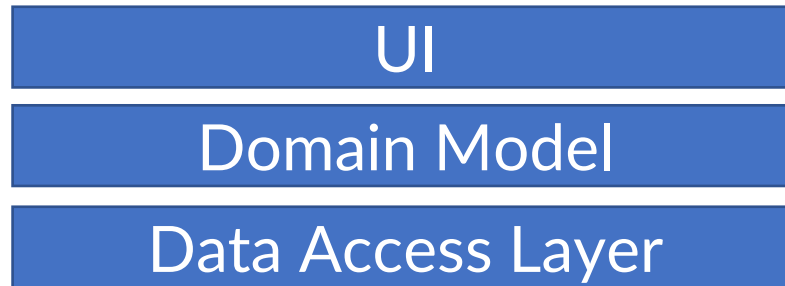
```

|   | EmployeeID | LastName | FirstName | Title  |
|---|------------|----------|-----------|--|
| 1 | 5          | Buchanan | Steven    | Sales Manager  |
|   |            | Buchanan | Steven    | Sales Manager  |
|   |            | Buchanan | Steven    | Sales Manager  |
|   |            |          |           | weather.ashx?q=37.017,-7.9338  |
|   |            |          |           | entative   |
|   |            |          |           | entative   |
|   |            |          |           | nt, Sales  |
|   |            |          |           | nt, Sales  |
|   |            |          |           | nt, Sales  |
|   |            |          |           | entative   |
|   |            |          |           | entative   |
|   |            |          |           | ning Crescent,26   |
|   |            |          |           | line.net/images/wsymbols   |
|   |            |          |           | 6  |
|   |            |          |           | ning Crescent,19   |
|   |            |          |           | ne.net/images/wsymbols01   |
|   |            |          |           | 6  |
|   |            |          |           | ning Crescent,11   |
|   |            |          |           | images/wsymbols01_png_64/  |
|   |            |          |           | 7,45,12,54,11,18,12,54   |
|   |            |          |           | 2019-01-04,16,60,13,55,07:46 AM,05:28 PM,06:16 AM,04:22 PM,Waning Crescent,4 |

# Domain Model



# Domain Model <-> Data Layer

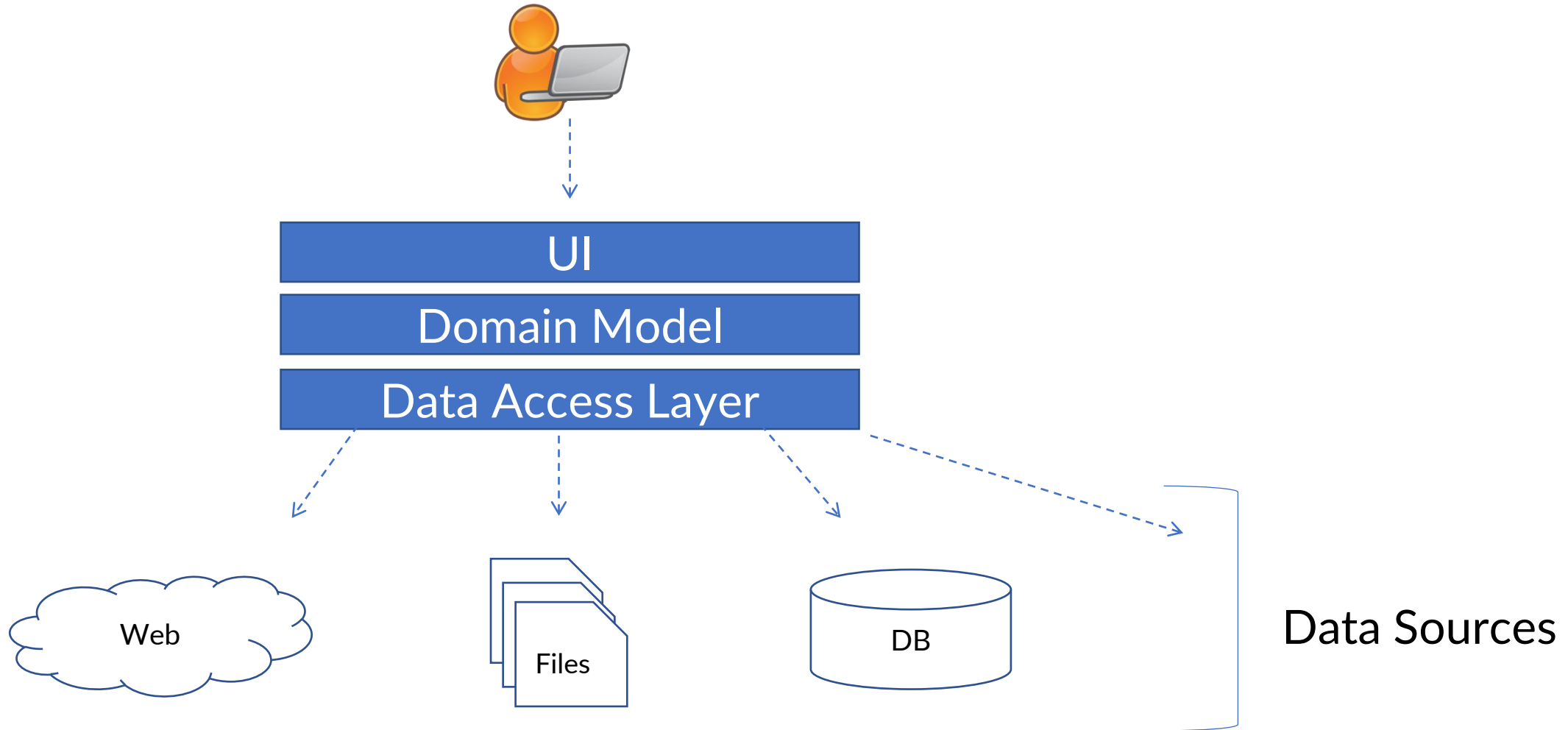


```
2019-01-01,17,63,12,54,07:45 AM,05:2  
2019-01-01,24,17,63,6,10,74,ENE,116,  
cloudy,0.0,0.0,59,10,6,1031,31,43,14  
2019-01-02,18,64,12,53,07:45 AM,05:2  
2019-01-02,24,18,64,6,9,179,S,116,ht  
cloudy,0.0,0.0,57,10,6,1030,31,15,14
```

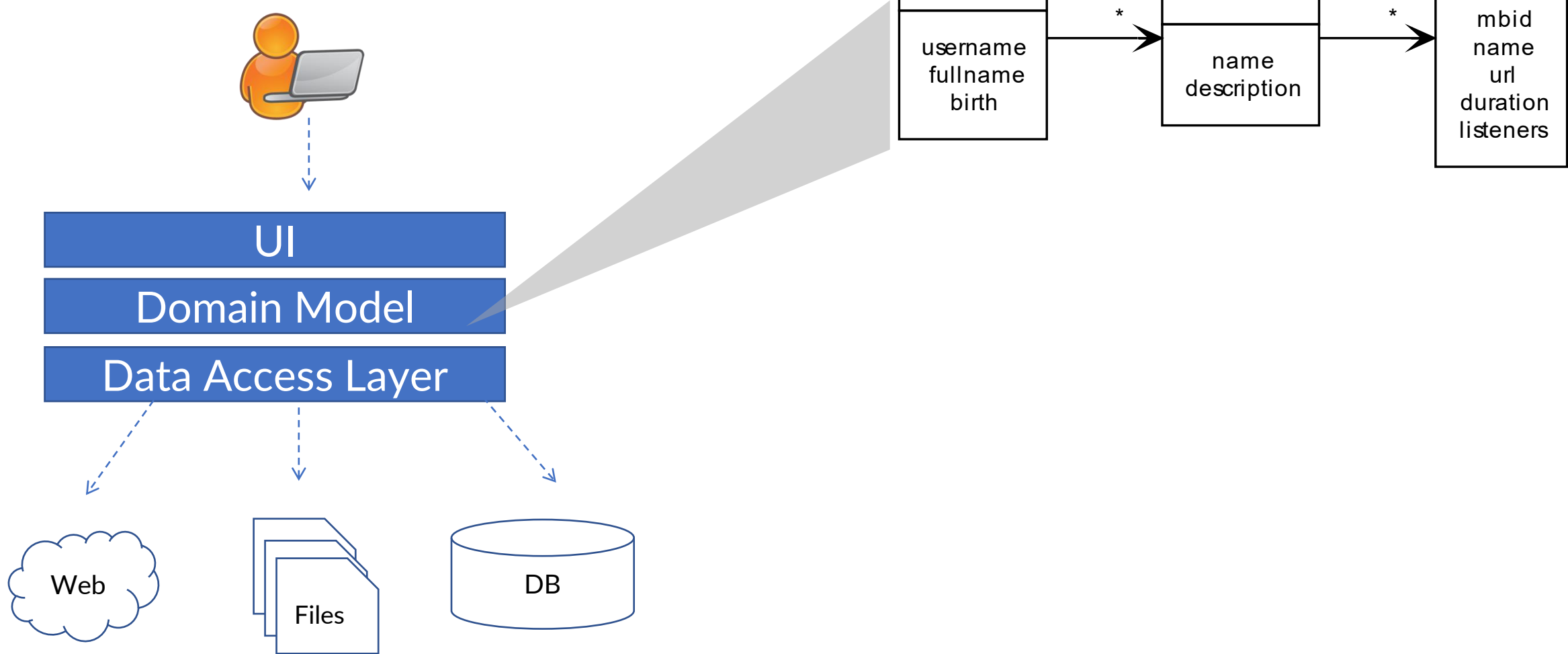
```
"track": [  
  {  
    "name": "Panic Station",  
    "playcount": "2804955",  
    "listeners": "331488",  
    "mbid": "a7ec9417-5b63-49  
    "url": "https://www.last.  
    "streamable": "0",
```

```
- <country>  
- <![CDATA[  
  Portugal  
]]>  
</country>  
- <region>  
- <![CDATA[  
  Lisboa  
]]>  
</region>  
<latitude>38.717  
<longitude>-9.13
```

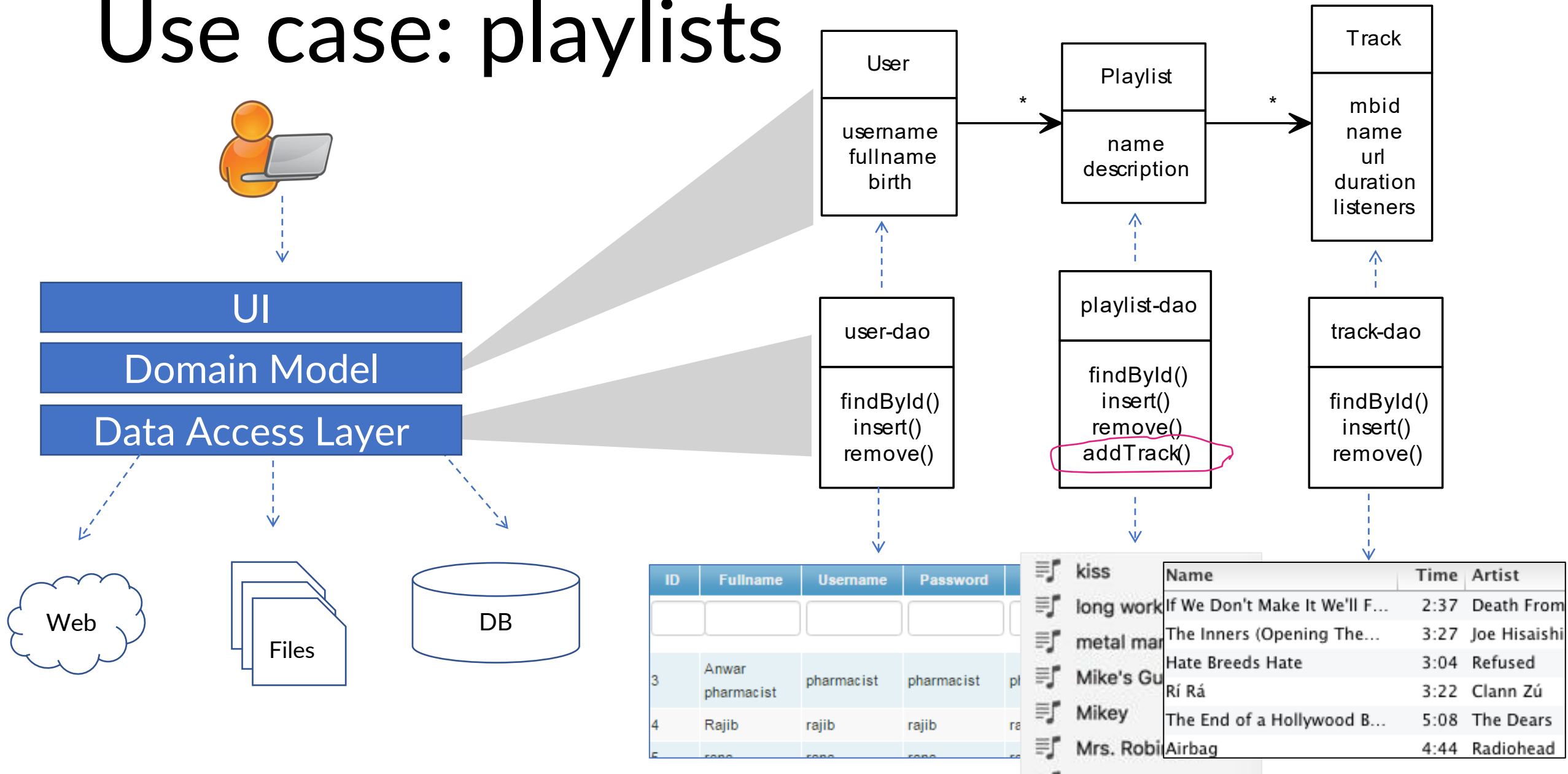
# Domain Model <-> Data Layer



# Use case: playlists



# Use case: playlists



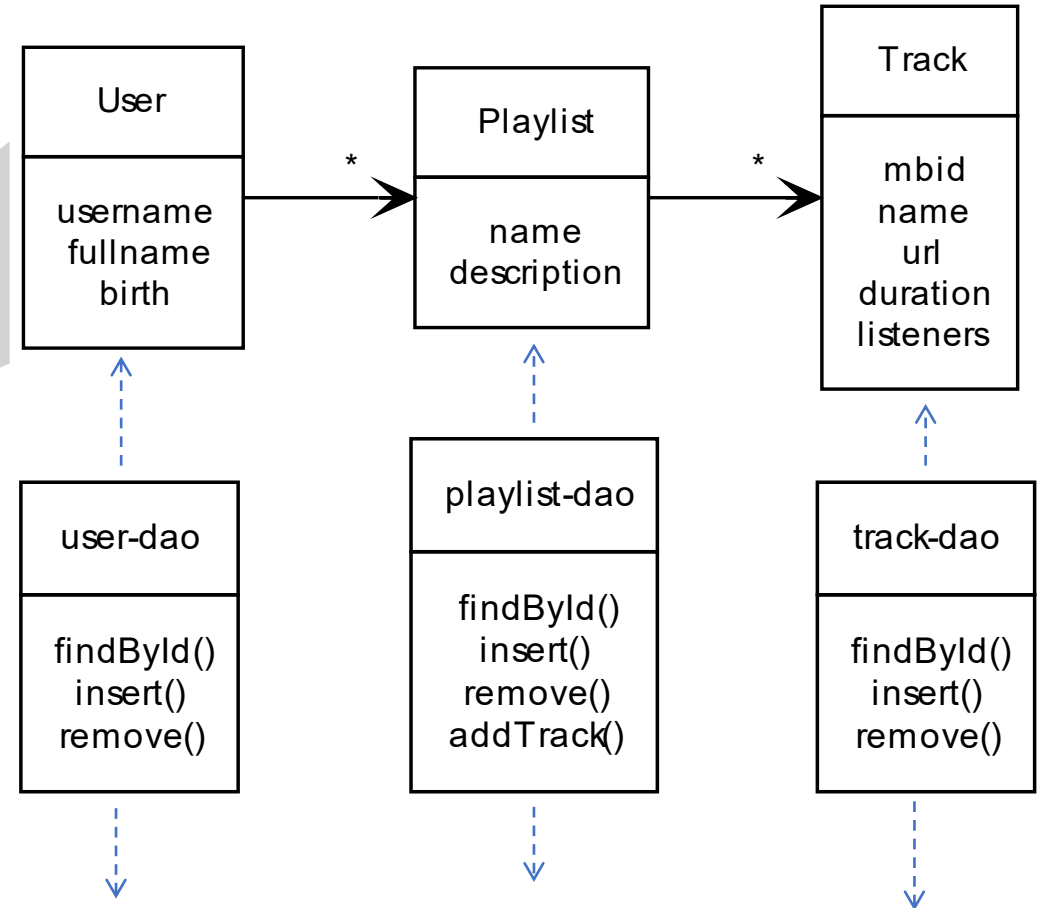
# Use case: playlists



UI

Domain Model

Data Access Layer





# Use case: playlists



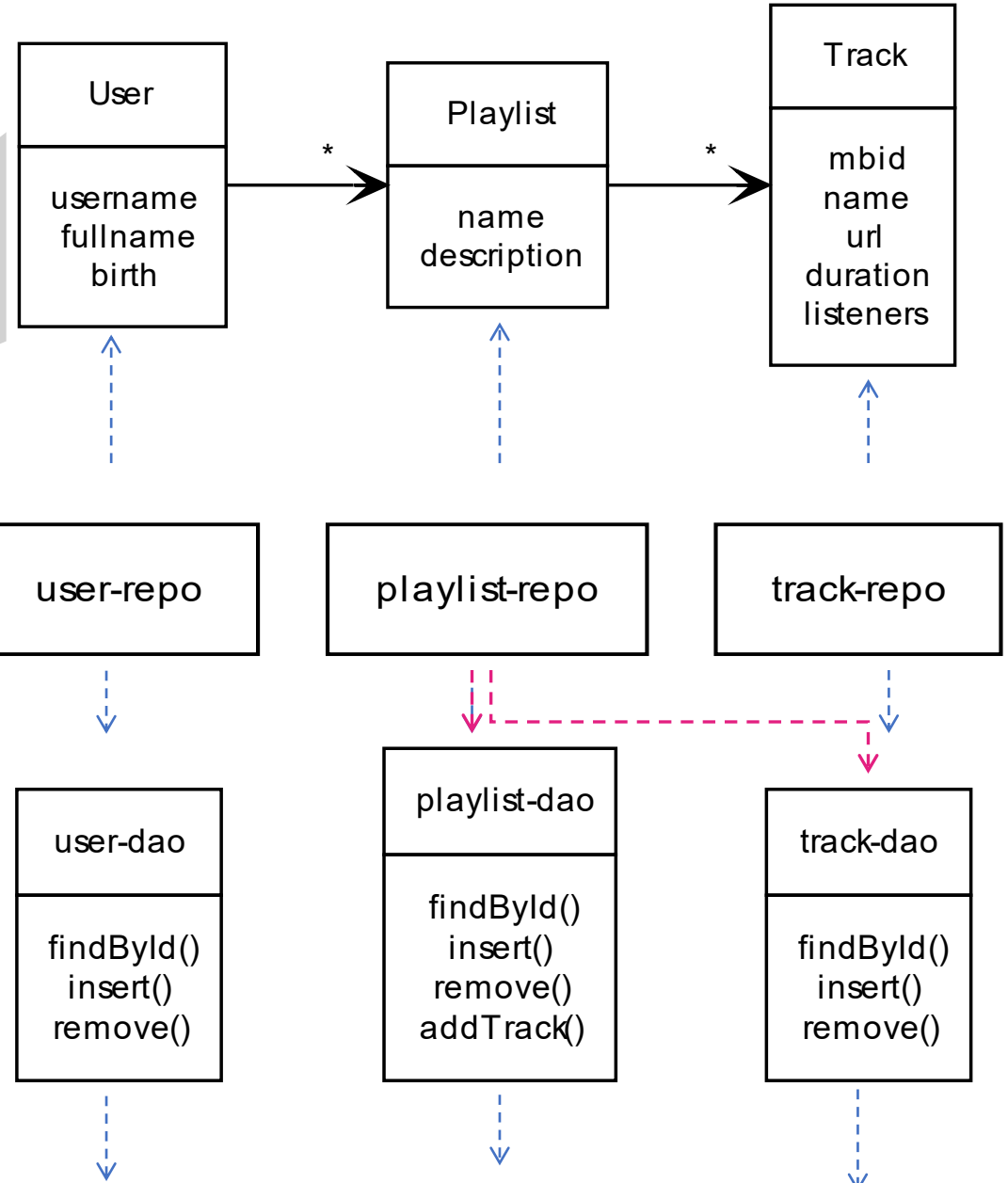
UI

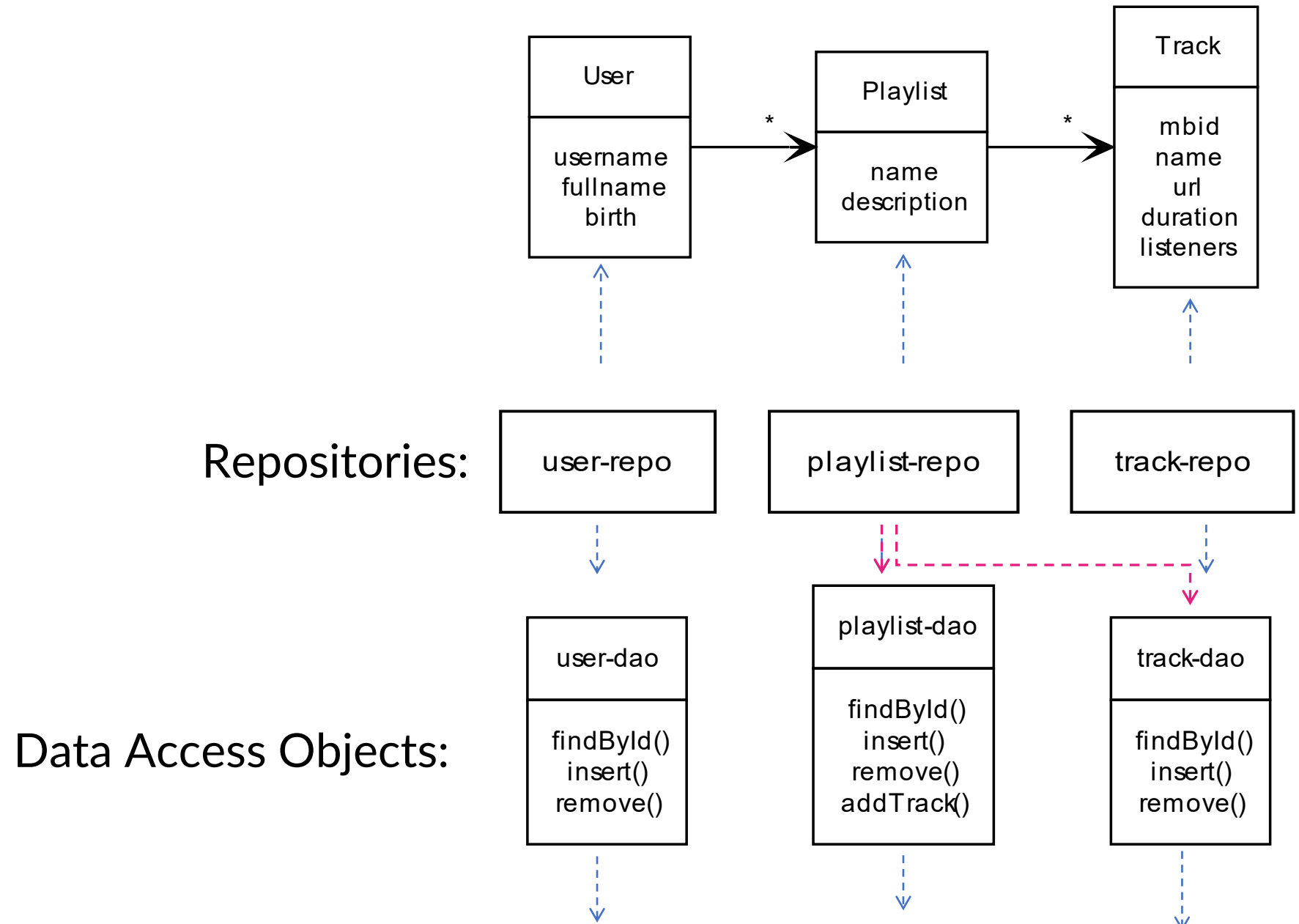
Domain Model

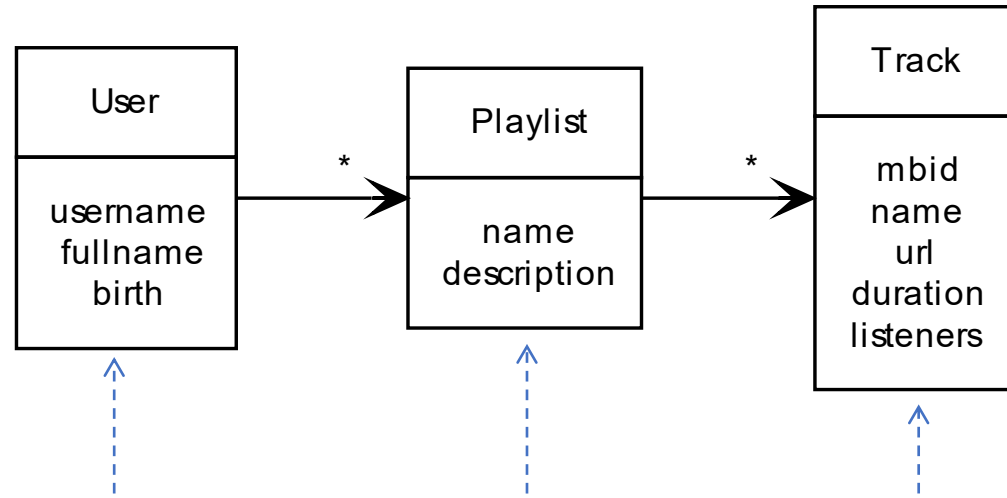
Data Access Layer

Repositories:

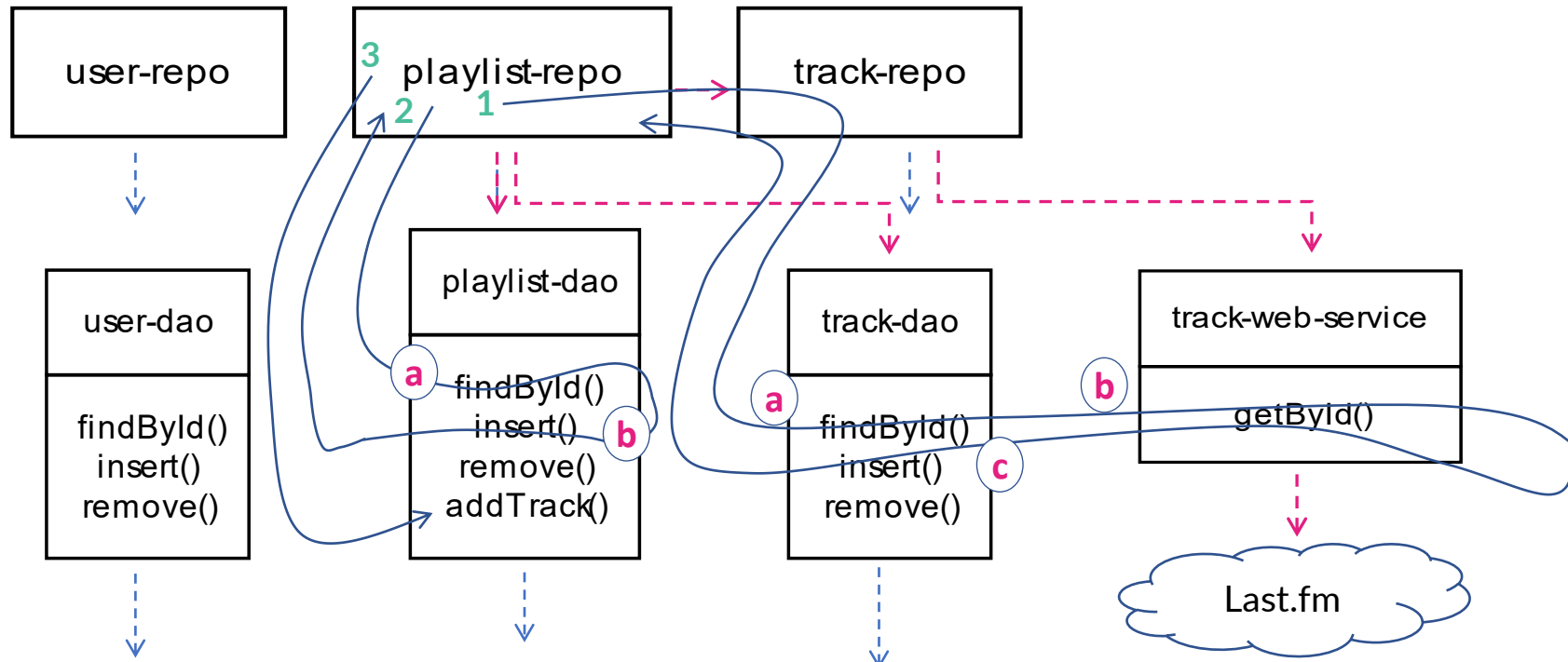
Data Access Objects:







Repositories:



Data Access Objects:

# playlist-repo: addTrack()... naïf

playlist-repo.js

```
export function addTrack(username, plstName, mbid, cb) {  
  /**  
   * 1. Checks for track with mbid in Dao.  
   * If it doesn't find the track in Dao then fetchs the track from the Web.  
   */  
  tracksRepo.findById(mbid, (err, t) => {  
    if(err) return cb(err)  
    /**  
     * 2. Checks for the playlist and creates it if it doesn't exist.  
     */  
    findById(username, plstName, (err, playlist) => {  
      if(err) return cb(err)  
      /**  
       * 3. Insert track mbid in the playlist if both exist.  
       */  
      dao.addTrack(username, plstName, mbid, cb)  
    })  
  })  
}
```

# playlist-repo: addTrack()... concurrently

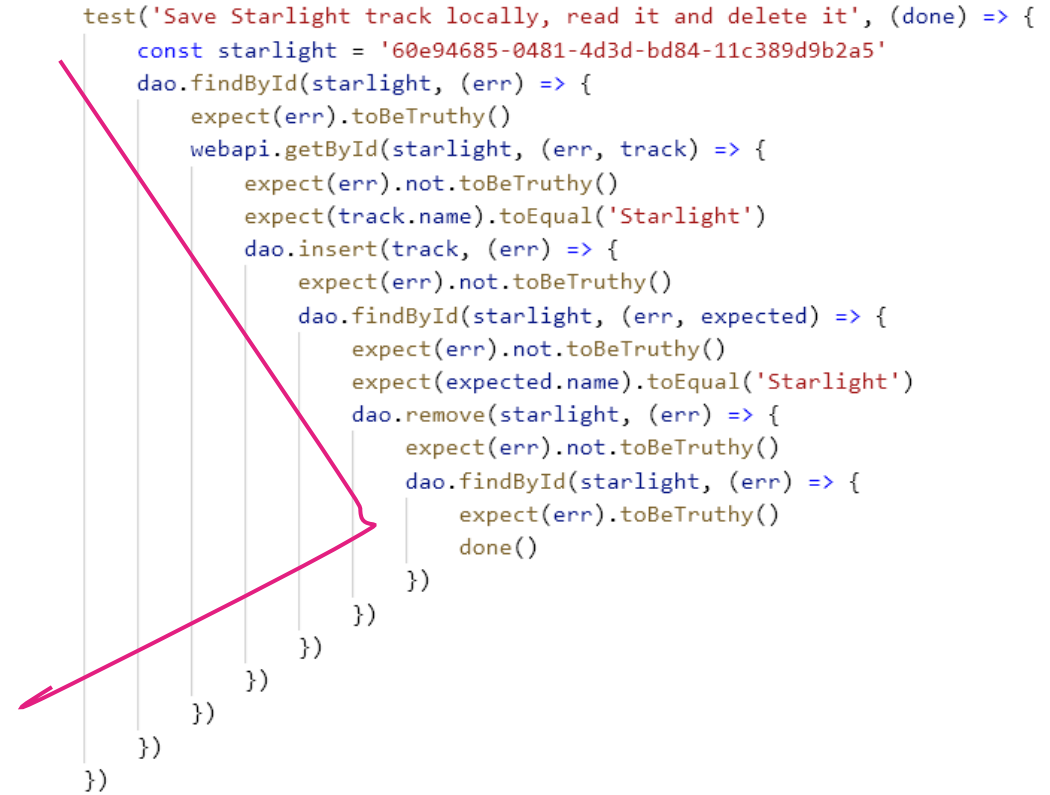
playlist-repo.js

```
export function addTrack(username, plstName, mbid, cb) {  
  let plst, track  
  const handler = err => {  
    if(err) return cb(err)  
    if(plst && track)  
      /**  
       * 3. Insert track mbid in the playlist if both exist.  
       */  
      dao.addTrack(username, plstName, mbid, cb)  
    }  
    /**  
     * 1. Checks for track with mbid in Dao. If it doesn't find the track in Dao then fetch...  
     */  
    tracksRepo.findById(mbid, (err, t) => { if(err) handler(err); else track = t; handler(null) })  
    /**  
     * 2. Checks for the playlist and creates if it does not exist.  
     */  
    findById(username, plstName, (err, p) => { if(err) handler(err); else plst = p; handler(null)})  
  }  
}
```

# Callback Hell

- Pyramid shape
- Hard to read
- Verbose
- bug-prone, e.g. `cb(data)` rather than `cb(null, data)`
- Hard to parallelize

```
test('Save Starlight track locally, read it and delete it', (done) => {
  const starlight = '60e94685-0481-4d3d-bd84-11c389d9b2a5'
  dao.findById(starlight, (err) => {
    expect(err).toBeTruthy()
    webapi.getById(starlight, (err, track) => {
      expect(err).not.toBeTruthy()
      expect(track.name).toEqual('Starlight')
      dao.insert(track, (err) => {
        expect(err).not.toBeTruthy()
        dao.findById(starlight, (err, expected) => {
          expect(err).not.toBeTruthy()
          expect(expected.name).toEqual('Starlight')
          dao.remove(starlight, (err) => {
            expect(err).not.toBeTruthy()
            dao.findById(starlight, (err) => {
              expect(err).toBeTruthy()
              done()
            })
          })
        })
      })
    })
  })
})
```

A diagram illustrating the concept of 'Callback Hell'. It shows a series of nested function calls represented by closing curly braces '}' on the left side of the code. The braces are arranged in a pyramid shape, with each level of nesting adding another brace. A pink arrow points from the top of the pyramid down to the bottom, indicating the flow of execution and the increasing complexity of the nested structure.

# Callback Hell... yet...

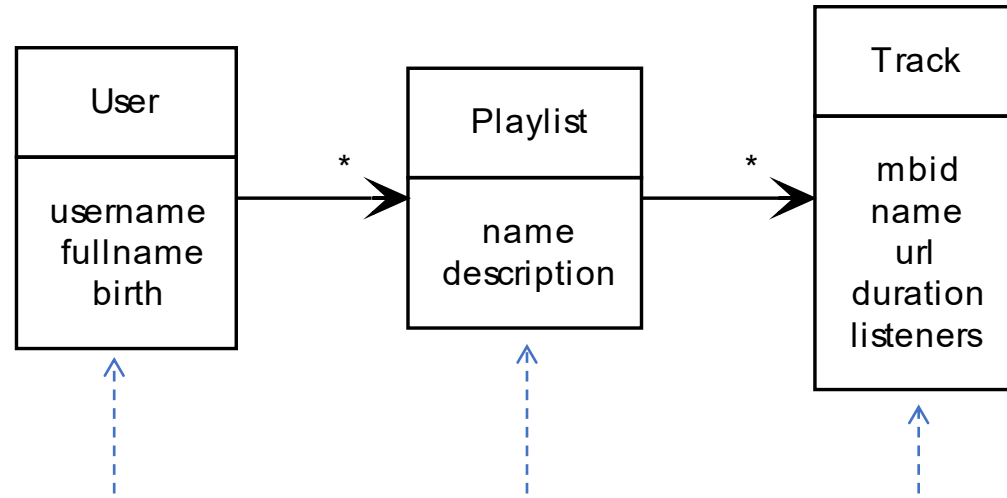
*“This is the most important topic to understand if you want to understand how to use node. Nearly everything in node uses callbacks. They weren't invented by node, they are just part of the JavaScript language.”*

Max Odgen in Art of Node about Callbacks

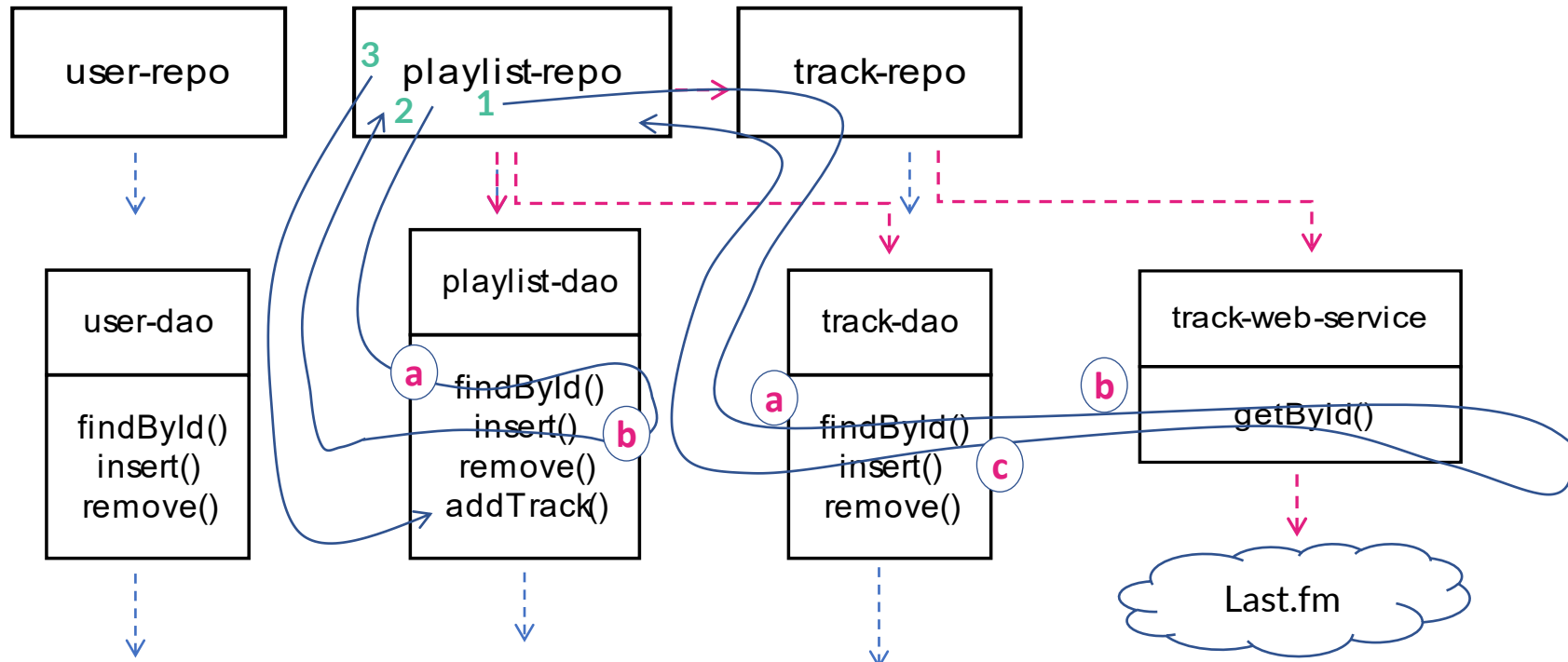
# Revisit playlists architecture

Domain Model  
Repository  
Data Access Objects

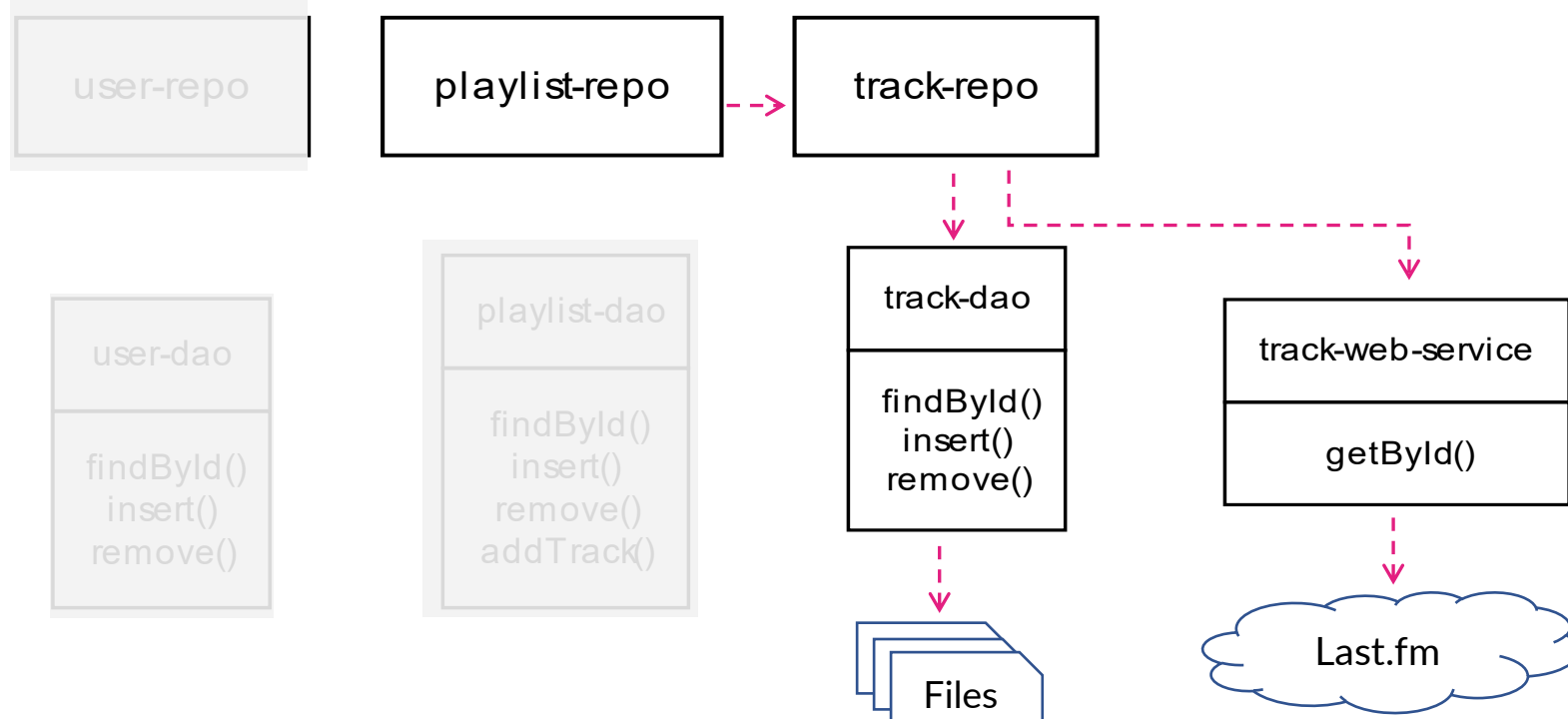
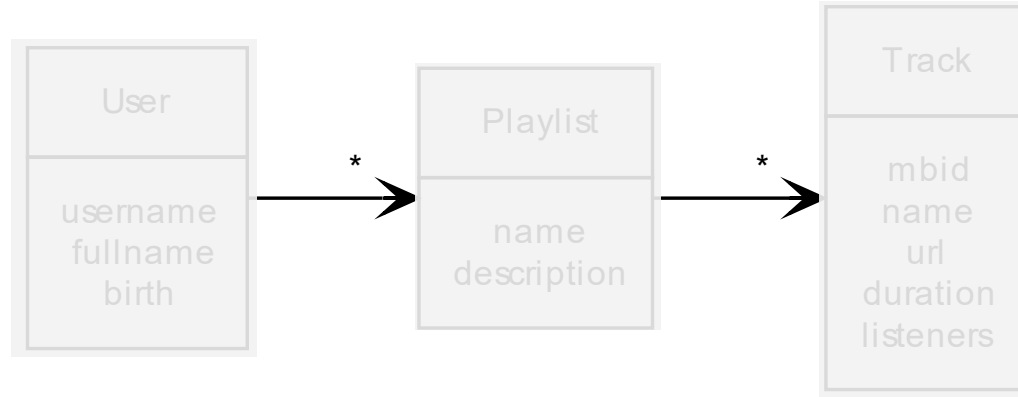


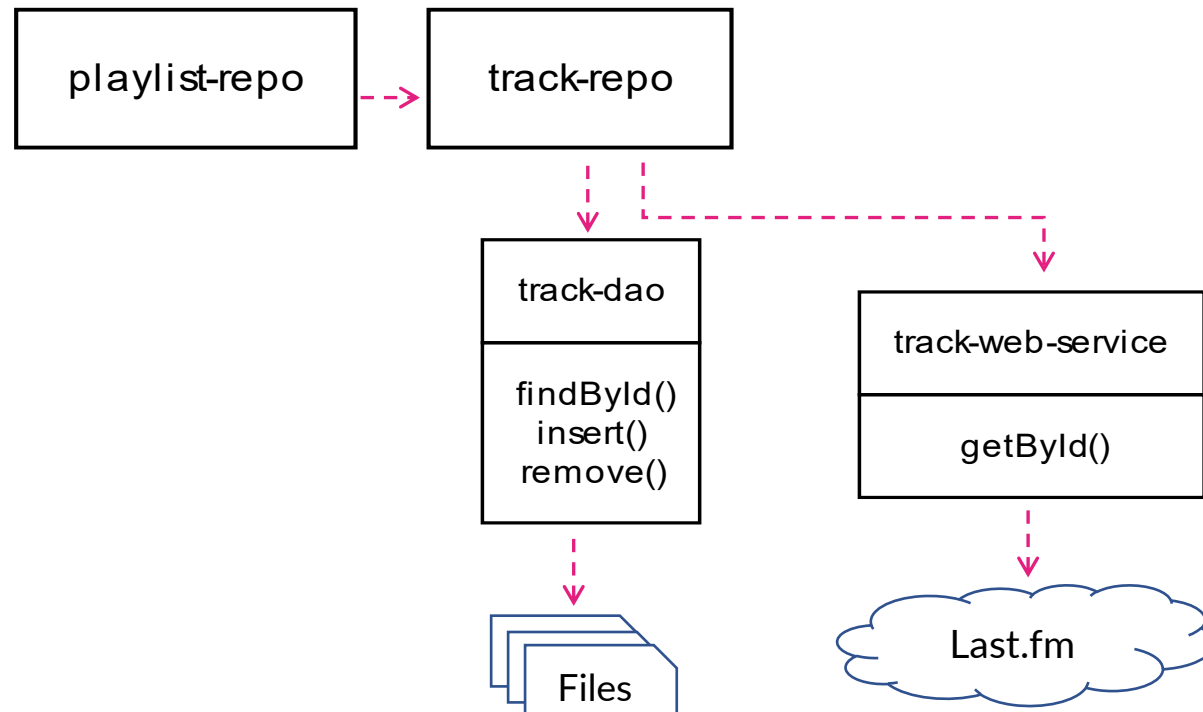


Repositories:



Data Access Objects:





# Callback pipeline

```
export function addTrack(username, plstName, mbid, cb) {  
  ...  
  tracksRepo.findById(mbid, (err, track) => { /* do something */ })  
  ...  
}
```

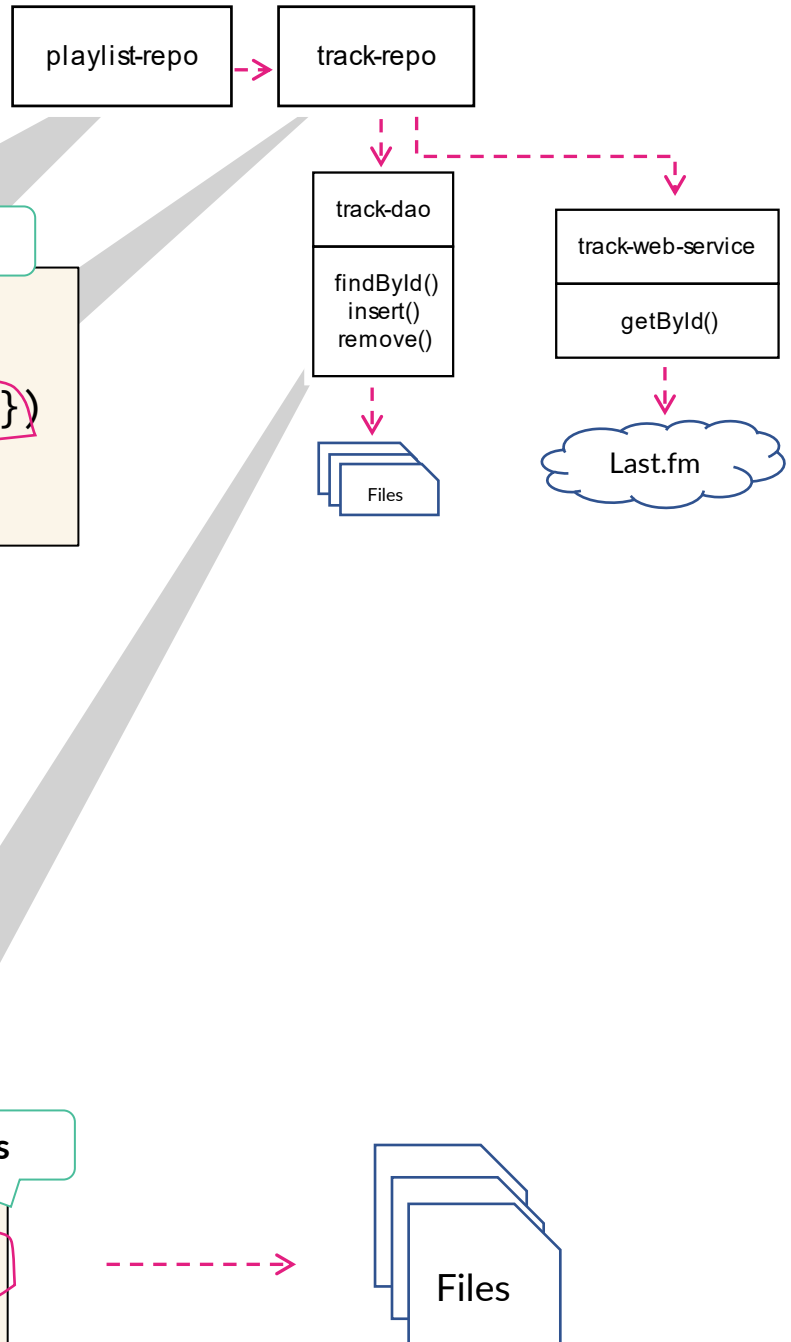
playlist-repo.js

```
export function findById(mbid, cb) {  
  dao.findById(mbid, (err, track) => {  
    if(!err) return cb(null, track)  
    webapi.getById(mbid, (err, track) => {  
      if(err) return cb(err)  
      dao.insert(track, (err) => {  
        if(err) return cb(err)  
        cb(null, track)  
      })  
    })  
  })  
  ...  
}
```

tracks-repo.js

```
export function insert(track, cb) {  
  fs.writeFile(trackPath(track.mbid), JSON.stringify(track), cb)  
}
```

tracks-dao.js



# ECMAScript 6 introduced Promises

Asynchronous idioms:

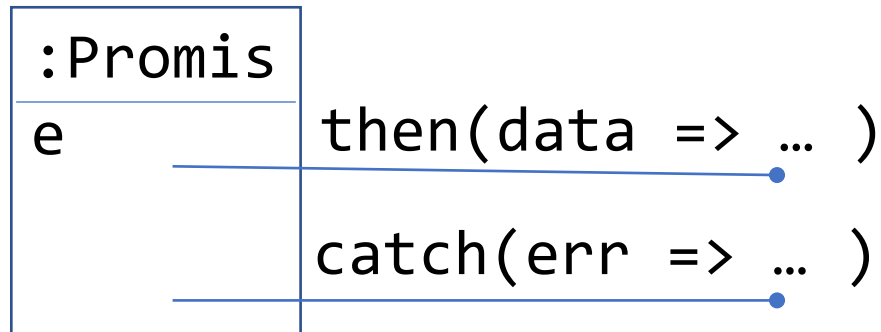
- callbacks
- Promises
- `async/await`
- suspend functions

# Callback <versus> Promise

callback: (err, data) => {...}

```
foo((err, data) => {...})
```

Promise:



```
foo()  
  .then((data) => ...)   
  .catch(err => ...)
```

*Represents the eventual result of an asynchronous operation*

# Callback <versus> Promise

```
export function insert(track, cb) {  
  const filename = trackPath(track.mbid)  
  fs.writeFile(filename, JSON.stringify(track), cb)  
}
```

(err) => { ... }

succeed → cb(null)

failed → cb(error)

```
export function insert(track) {  
  const filename = trackPath(track.mbid)  
  return writeFile(filename, JSON.stringify(track))  
}
```

:Promise

e

then(data => ...)

catch(err => ...)

succeed

failed

# Promise API

**playlist-repo.js**

```
export function addTrack(username, plstName, mbid, cb) {  
  ...  
  ...  
}
```

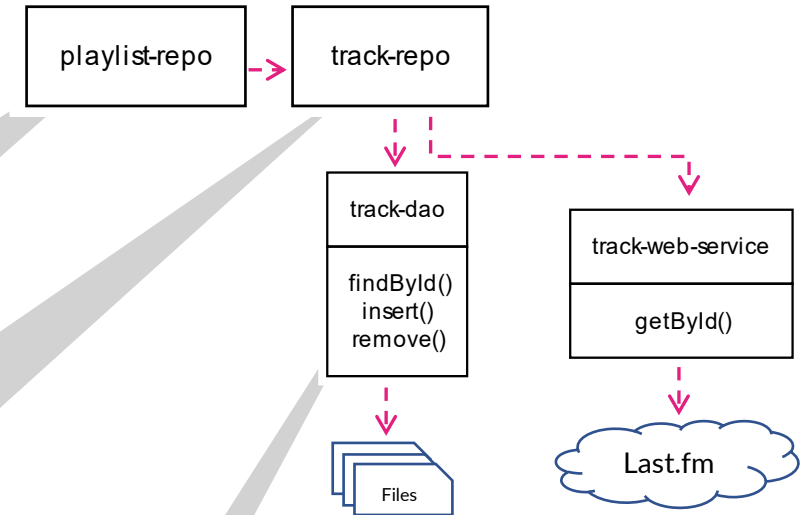
**tracks-repo.js**

```
export function findById(mbid, cb) {  
  dao.findById(mbid, (err, track) => {  
    if(!err) return cb(null, track)  
    webapi.getById(mbid, (err, track) => {  
      if(err) return cb(err)  
      dao.insert(track, (err) => {  
        if(err) return cb(err)  
        cb(null, track)  
      })  
    })  
  })  
  ...  
}
```

**tracks-dao.js**

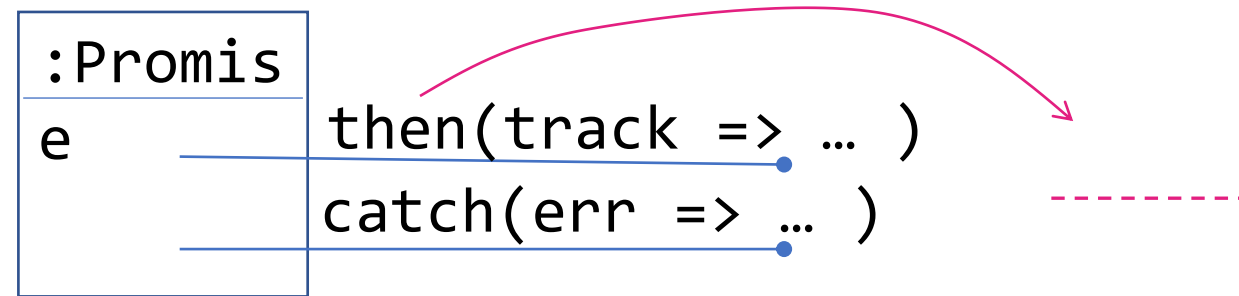
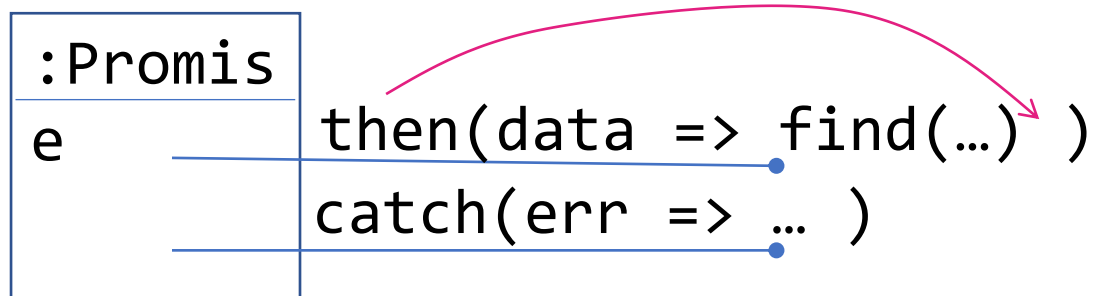
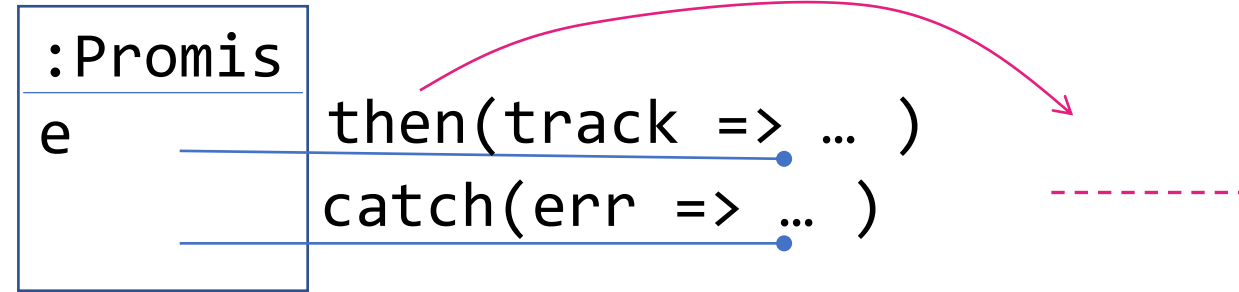
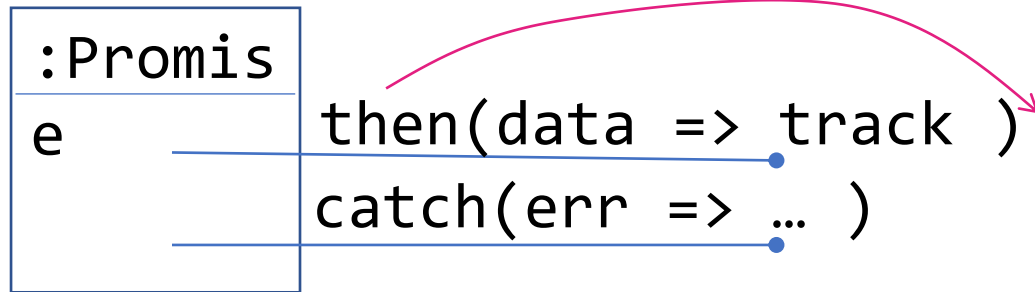
```
export function insert(track) {  
  return fs.writeFile(trackPath(track.mbid), JSON.stringify(track))  
}
```

`dao.insert(track)  
 .then(() => cb(null, track))  
 .catch(err => cb(err))`





# Promise API: The power of composing



```
export function find(...) {  
  return ... // returns a Promise of a Track  
}
```

# Promise API

**playlist-repo.js**

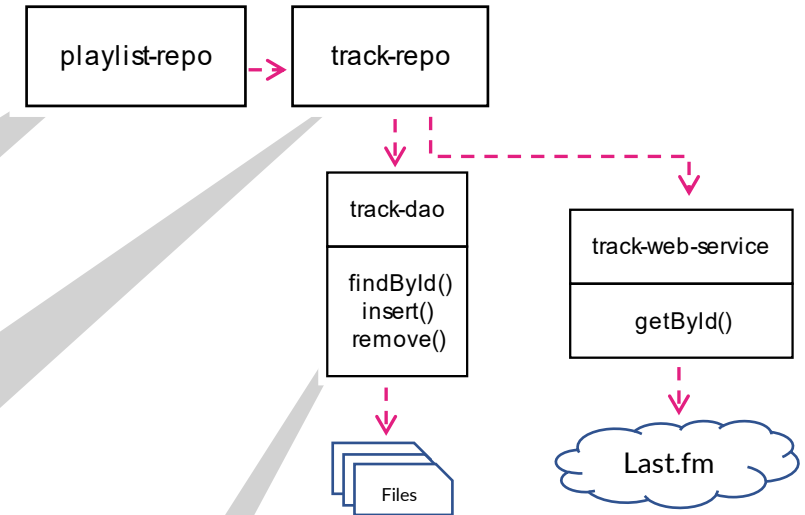
```
export function addTrack(username, plstName, mbid, cb) {  
  ...  
}
```

**tracks-repo.js**

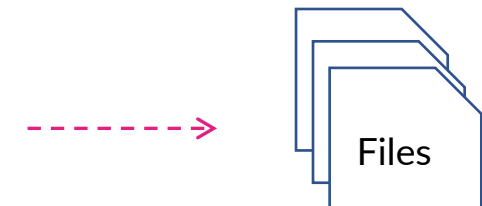
```
export function findById(mbid, cb) {  
  dao.findById(mbid, (err, track) => {  
    if(!err) return cb(null, track)  
    webapi.getById(mbid, (err, track) => {  
      if(err) return cb(err)  
      dao.insert(track, (err) => {  
        if(err) return cb(err)  
        cb(null, track)  
      })  
    })  
  })  
}
```

**tracks-dao.js**

```
export function insert(track) {  
  return fs.writeFile(trackPath(track.mbid), JSON.stringify(track))  
}
```



```
return dao  
  .findById(mbid)  
  .catch(err => webapi  
    .getById(mbid)  
    .then(track => dao  
      .insert(track)  
      .then(() => track)  
    ))
```



# addTrack() now with Promises

playlist-repo.js

```
export function addTrack(username, plstName, mbid, cb) {  
  let plst, track  
  const handler = err => {  
    if(err) return cb(err)  
    if(plst && track)  
      /**  
       * 3. Insert track mbid in the playlist if both exist.  
       */  
      dao.addTrack(username, plstName, mbid, cb)  
    }  
    /**  
     * 1. Checks for track with mbid in Dao. If it doesn't find the track in Dao then fetch...  
     */  
    tracksRepo.findById(mbid, (err, t) => { if(err) handler(err); else track = t; handler(null) })  
    /**  
     * 2. Checks for the playlist and creates if it does not exist.  
     */  
    findById(username, plstName, (err, p) => { if(err) handler(err); else plst = p; handler(null)})  
  }  
}
```

# addTrack() now with Promises

playlist-repo.js

```
export function addTrack(username, plstName, mbid, cb) {
  let plst, track
  const handler = err => {
    if(err) return cb(err)
    if(plst && track)
      /**
       * 3. Insert track mbid in the playlist if both exist.
       */
      dao.addTrack(username, plstName, mbid, cb)
  }
  /**
   * 1. Checks for track with mbid in Dao. If it doesn't find the track in Dao then fetch...
   */
  tracksRepo.findById(mbid, (err, t) => { if(err) handler(err); else track = t; handler(null) })
  /**
   * 2. Checks for the playlist and creates if it does not exist.
   */
  findById(username, plstName, (err, p) => { if(err) handler(err); else plst = p; handler(null)})
}
```

# addTrack() now with Promises

playlist-repo.js

```
export function addTrack(username, plstName, mbid, cb) {
  let plst, track
  const handler = err => {
    if(err) return cb(err)
    if(plst && track)
      /**
       * 3. Insert track mbid in the playlist if both exist.
       */
      dao.addTrack(username, plstName, mbid, cb)
  }
  /**
   * 1. Checks for track with mbid in Dao. If it doesn't find the track in Dao then fetch...
   */
  const ptrack = tracksRepo.findById(mbid)
  /**
   * 2. Checks for the playlist and creates if it does not exist.
   */
  const pplst = findById(username, plstName)
}
```

# addTrack() now with Promises

playlist-repo.js

```
export function addTrack(username, plstName, mbid, cb) {  
  /**  
   * 1. Checks for track with mbid in Dao. If it doesn't find the track in Dao then fetch...  
   */  
  const ptrack = tracksRepo.findById(mbid)  
  /**  
   * 2. Checks for the playlist and creates if it does not exist.  
   */  
  const pplst = findById(username, plstName)  
  /**  
   * 3. Insert track mbid in the playlist if both exist.  
   */  
  ...  
  dao.addTrack(username, plstName, mbid, cb)  
}
```

# addTrack() now with Promises

playlist-repo.js

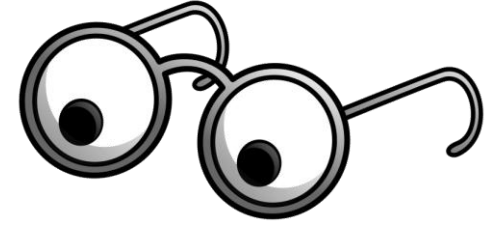
```
export function addTrackSeq(username, plstName, mbid) {  
  /**  
   * 1. Checks for track with mbid in Dao. If it doesn't find the track in Dao then fetch...  
   */  
  const ptrack = tracksRepo.findById(mbid)  
  /**  
   * 2. Checks for the playlist and creates if it does not exist.  
   */  
  const pplst = findById(username, plstName)  
  /**  
   * 3. Insert track mbid in the playlist if both exist.  
   */  
  return ptrack  
    .then(track => pplst)  
    .then(plst => dao.addTrack(username, plstName, mbid))  
}
```

# With Promises

- ~~Pyramid shape~~
- ~~Hard to read~~
- ~~Verbose~~
- ~~bug-prone, e.g. `cb(data)` rather than `cb(null, data)`~~
- ~~Hard to parallelize~~

```
test('Save Starlight track locally, read it and delete it', (done) => {
  const starlight = '60e94685-0481-4d3d-bd84-11c389d9b2a5'
  dao.findById(starlight, (err) => {
    expect(err).toBeTruthy()
    webapi.getById(starlight, (err, track) => {
      expect(err).not.toBeTruthy()
      expect(track.name).toEqual('Starlight')
      dao.insert(track, (err) => {
        expect(err).not.toBeTruthy()
        dao.findById(starlight, (err, expected) => {
          expect(err).not.toBeTruthy()
          expect(expected.name).toEqual('Starlight')
          dao.remove(starlight, (err) => {
            expect(err).not.toBeTruthy()
            dao.findById(starlight, (err) => {
              expect(err).toBeTruthy()
              done()
            })
          })
        })
      })
    })
  })
})
```





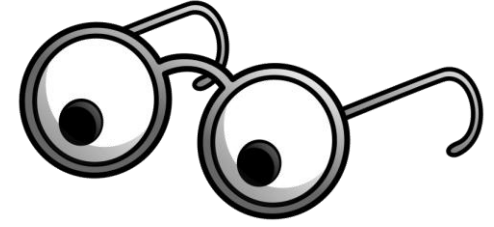
# Next... Overview

async/await

Promise.all()

Asynchronous Domain Model

# Overview...



## With async/await

Like synchronous calls but **without blocking**

tracks-repo.js

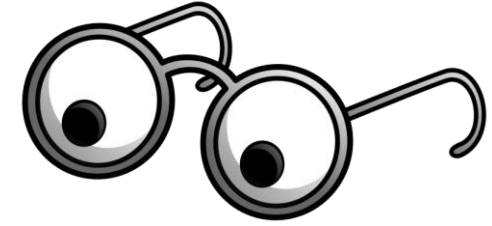
```
return dao
  .findById(mbid)
  .catch(err => webapi
    .getById(mbid)
    .then(track => dao
      .insert(track)
      .then(() => track)
    ))
```



tracks-repo.js

```
try {
  return dao.findById(mbid)
}
catch (err) {
  const track = await webapi.getById(mbid)
  await dao.insert(track)
  return track
}
```

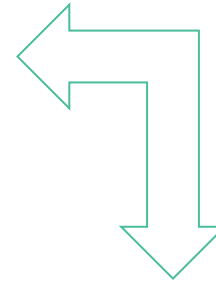
# Overview...



playlist-repo.js

```
export function getTracks(username, plstName, cb) {
  dao.getTracks(username, plstName, (err, arr) => {
    if (err) return cb(err)
    const res = []
    arr = arr.filter(mbid => mbid !== '')
    arr.forEach(mbid => {
      tracks
        .findById(mbid)
        .then(t => {
          res.push(t)
          if (res.length >= arr.length)
            cb(null, res)
        })
        .catch(err => cb(err))
    })
  })
}
```

## Promise.all()

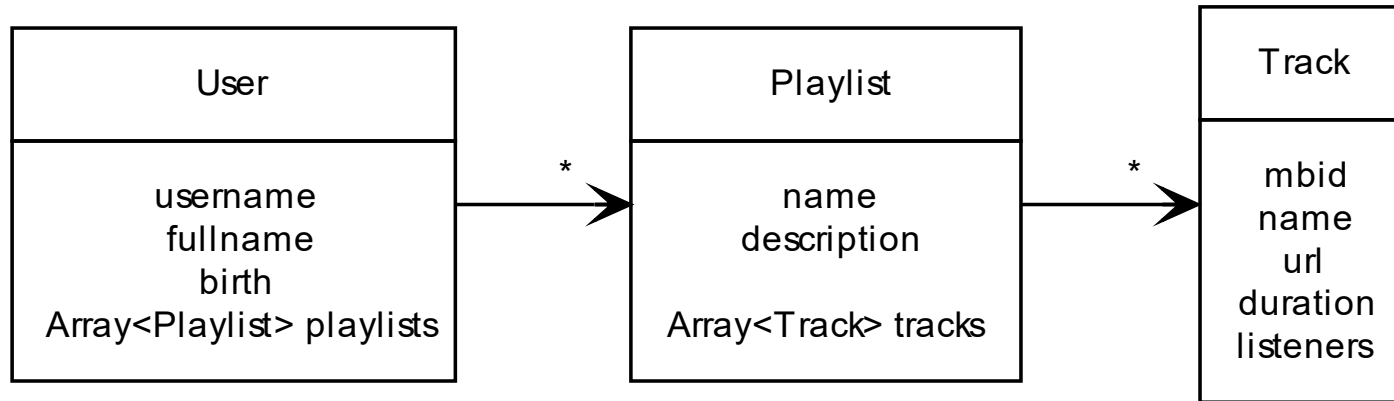


playlist-repo.js

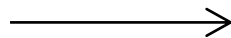
```
export function getTracks(username, plstName) {
  return dao
    .getTracks(username, plstName) // Promise<Array<String>>
    .then(arr => arr // Promise<Array<Promise<Track>>>
      .filter(mbid => mbid !== '')
      .map(tracks.findById))
    .then(arr => Promise.all(arr)) // Promise<Array<Track>>
}
```

# Association

e.g. A user has playlists and a playlist has tracks.



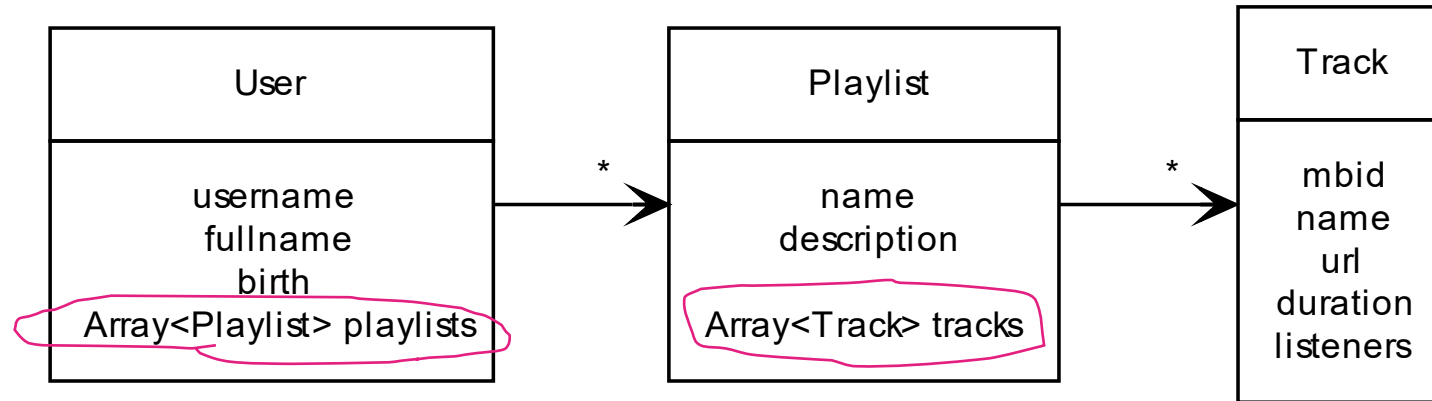
Association:  
relationship



*“has”*

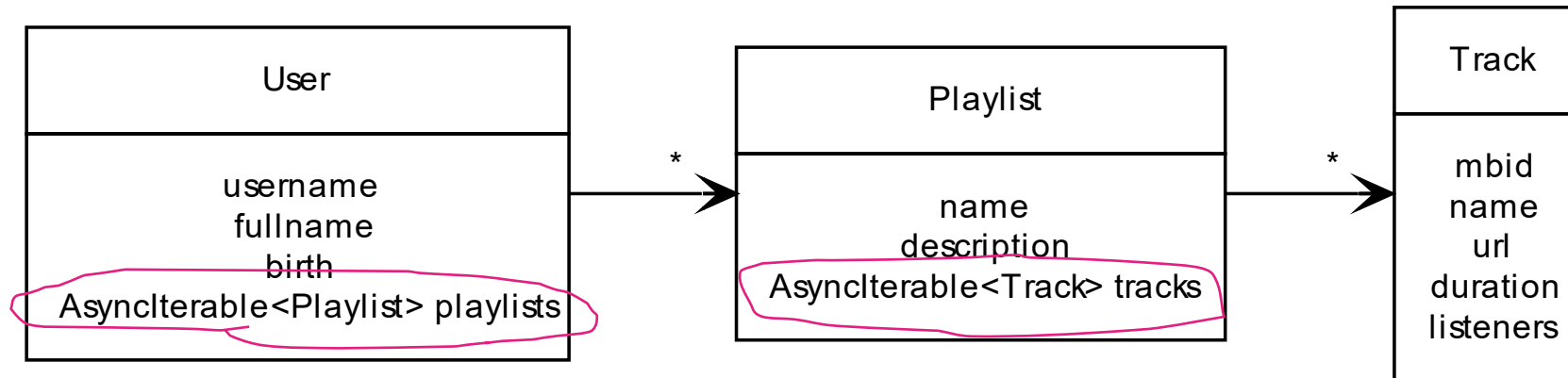
# Eager

The use of Array turns the relationship eager.



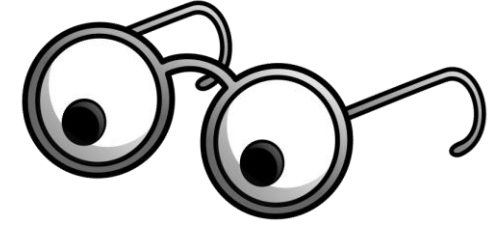
# Asynchronous

Do not wait for relationship completion



ECMAScript 9 introduces asynchronous iteration via the AsyncIterator protocol and async generators.

# Overview...

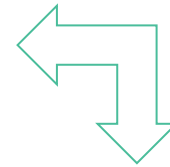


With async iterator

Like synchronous calls but **without blocking**

playlist-repo.js

```
/**
 * @returns {Promise<Track[]>}
 */
export function getTracks(username, plstName) {
  return dao
    .getTracks(username, plstName)
    .then(arr => arr
      .filter(mbid => mbid !== '')
      .map(mbid => tracks.findById(mbid)))
    .then(arr => Promise.all(arr))
}
```



playlist-repo.js

```
/**
 * @async
 * @yields {Track}
 */
export async function* getTracks(username, plstName) {
  for await (const mbid of dao.getTracks(username, plstName)) {
    if(mbid !== '') {
      yield tracks.findById(mbid)
    }
  }
}
```