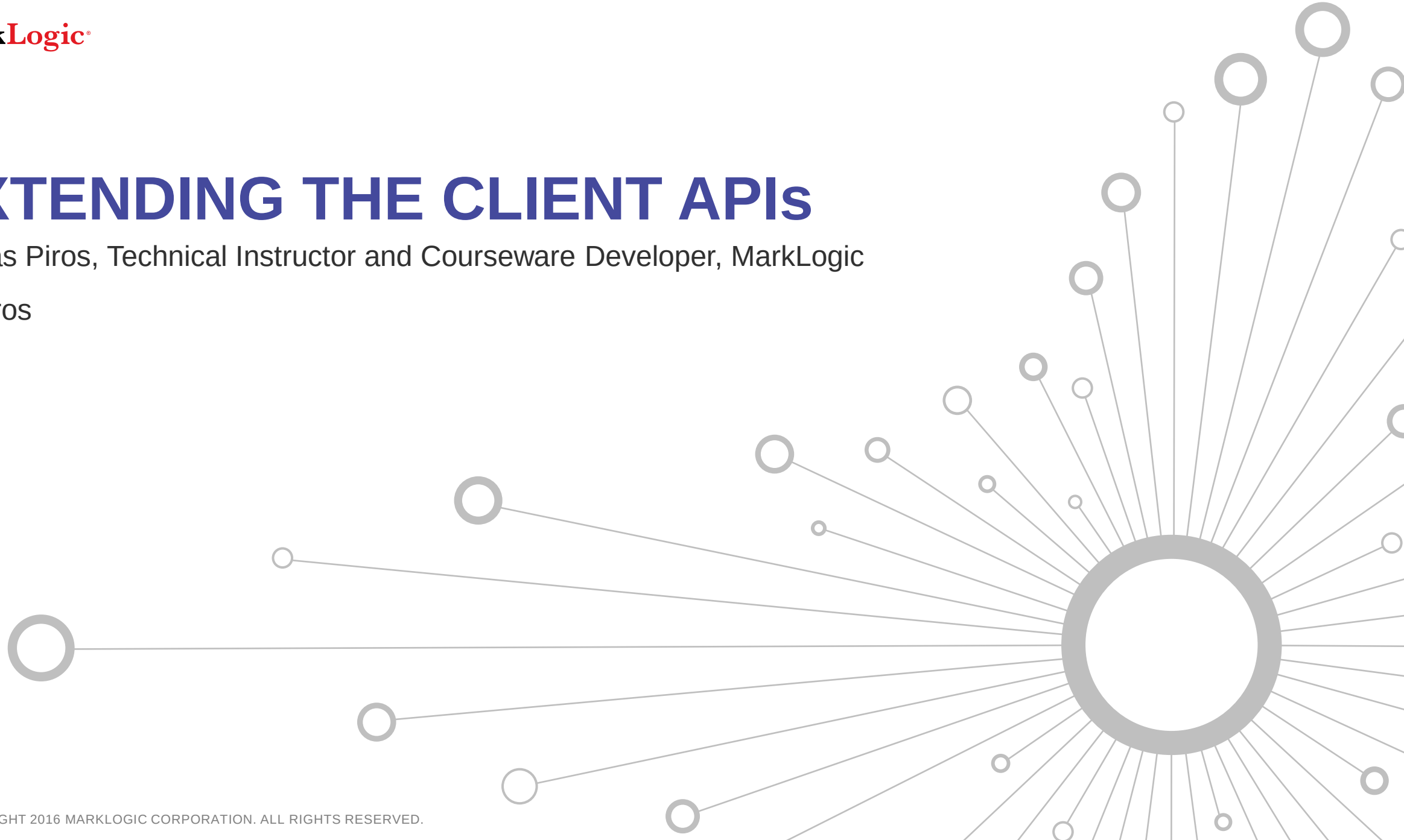
MarkLogic

EXTENDING THE CLIENT APIs

Tamas Piros, Technical Instructor and Courseware Developer, MarkLogic

@tpiros



MarkLogic

/me

- Senior Technical Instructor @ MarkLogic
- 10+ years of full stack web development
- 5+ years of technical training experience
- Prolific blogger on the “latest and greatest” web tech

- Get in touch via Twitter (@tpiros) or via email tamas.piros@marklogic.com

MarkLogic

What Are the Client APIs?

- The Client APIs allows developers to access MarkLogic using the programming language of their choice (Java or Node.js)
- Set of methods that allow managing and searching of documents as well as managing MarkLogic
- You can also define your own custom functionality and reuse that in a Java or Node.js application

MarkLogic

Resource Service Extensions

- Accessing extension services is possible via the standard HTTP verbs
  - GET
  - PUT
  - POST
  - DELETE
- Extensions can be written in either JavaScript or XQuery
- Once an extension is created and saved in the Modules database, it can be used via the Node.js Client API or the Java Client API

MarkLogic

Example

/squad/rom.json

```
{
"name": "AS Roma",
"id": 100,
"crest":
"http://upload.wikimedia.org/wikipedia
/de/3/32/AS_Rom.svg",
"value": 261700000
}
```

/player/francescototti.json

```
{
"name": "Francesco Totti",
"position": "Centre Forward",
"jerseyNumber": 10,
"dateOfBirth": "1976-09-27",
"nationality": "Italy",
"contractUntil": "2016-06-30",
"marketValue": 1000000,
"teamId": 100
}
```

MarkLogic

Example

- Server-side JavaScript needs to be saved to modules database
- Can define GET, POST, PUT, DELETE functions as well as other helper functions (full JavaScript or XQuery functionality)
- Must export functions accordingly
- Parameters can be accessed from the request in the application
- Once saved in modules database, endpoint can be invoked from Node.js or Java

```
function get(context, params) {
  var results = [];
  var team = params.team;
  var teamId = parseInt(fn.doc(fn.baseUri(cts.search(team))).next().value.root.id, 10);
  var jsearch = require('/MarkLogic/jsearch');
  var players = jsearch.collections('players');
  var results =
    players.documents()
    .where(cts.jsonPropertyRangeQuery('teamId', '=', teamId))
    .slice(0, 50)
    .result().results;

  var players = [];
  results.map(function(player) {
    players.push(player.document.name);
  });

  context.outputStatus = [200, 'OK'];
  return xdmp.arrayValues(players);
};

module.exports = {
  GET: get
};
```

MarkLogic

Example

- Each HTTP verb can be called via `db.resources` followed by the actual verb, e.g. `db.resources.get`

§ The call accepts a `name` and a `params` property

```
db.resources.get({
  name: 'show-players',
  params: { team: 'AS Roma'}
}).result().then(function(response) {
  response.map(function(document) {
    return console.log(document.content);
  })
}).catch(function(error) {
  console.log(error);
});
```

```
→  mlw node players.js
Gerson
Vasilis Torosidis
Kevin Strootman
Salih Ucan
Leandro Castán
Wojciech Szczesny
Alessandro Florenzi
Norbert Gyömber
Antonio Rüdiger
Bogdan Lobont
Diego Perotti
```

MarkLogic

Summary

- Extending the client APIs means that you can create your own custom endpoints, with custom functionality
- Custom functionality can be created for HTTP methods (GET, POST, PUT, DELETE)
- Custom extensions can be called from Node.js and Java applications
- Allows developers to concentrate on application development without having to worry about developing / changing the database

THANK YOU