Internet Oriented Systems

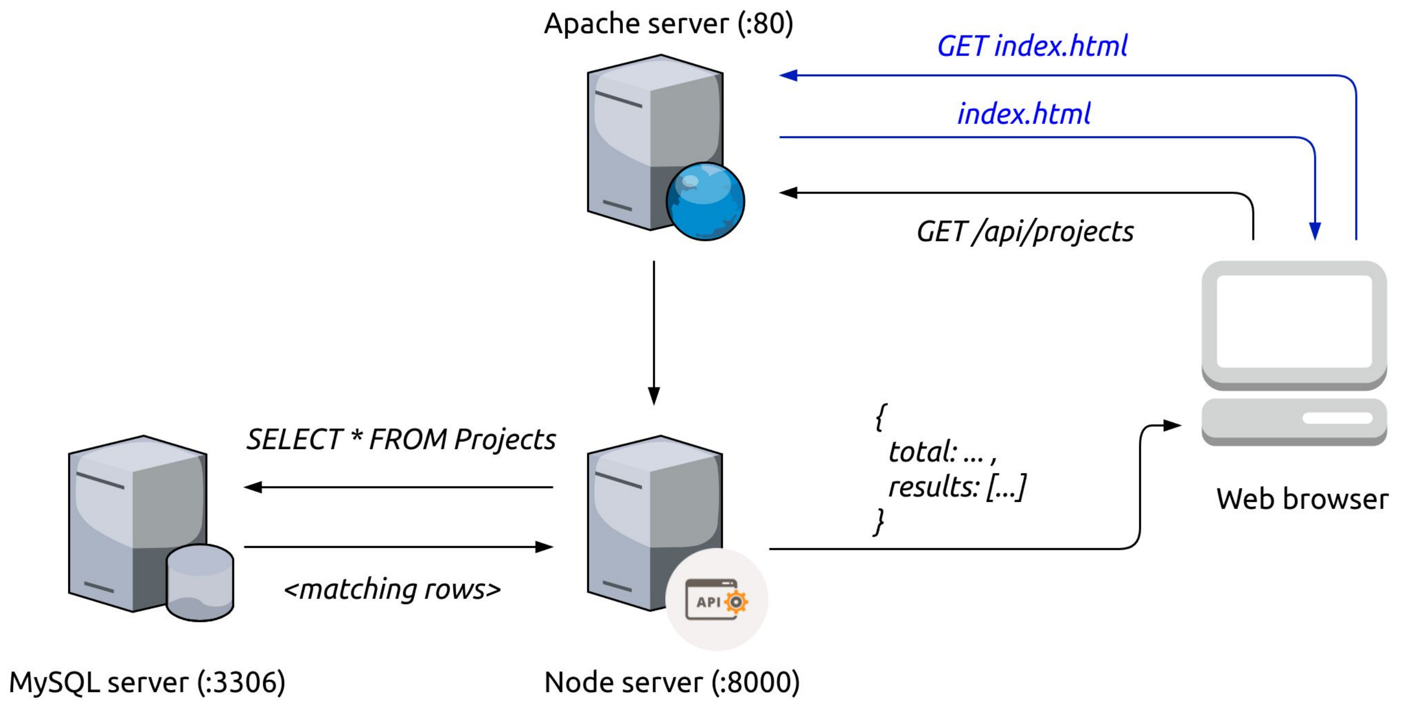
University of Parma - A.Y. 2020-2021

Collaborative Data Curation Platform

Architectural design and implementation

Giuseppe La Gualano [giuseppe.lagualano@studenti.unipr.it]
Francesco Vetere [francesco.vetere@studenti.unipr.it]

Architectural design



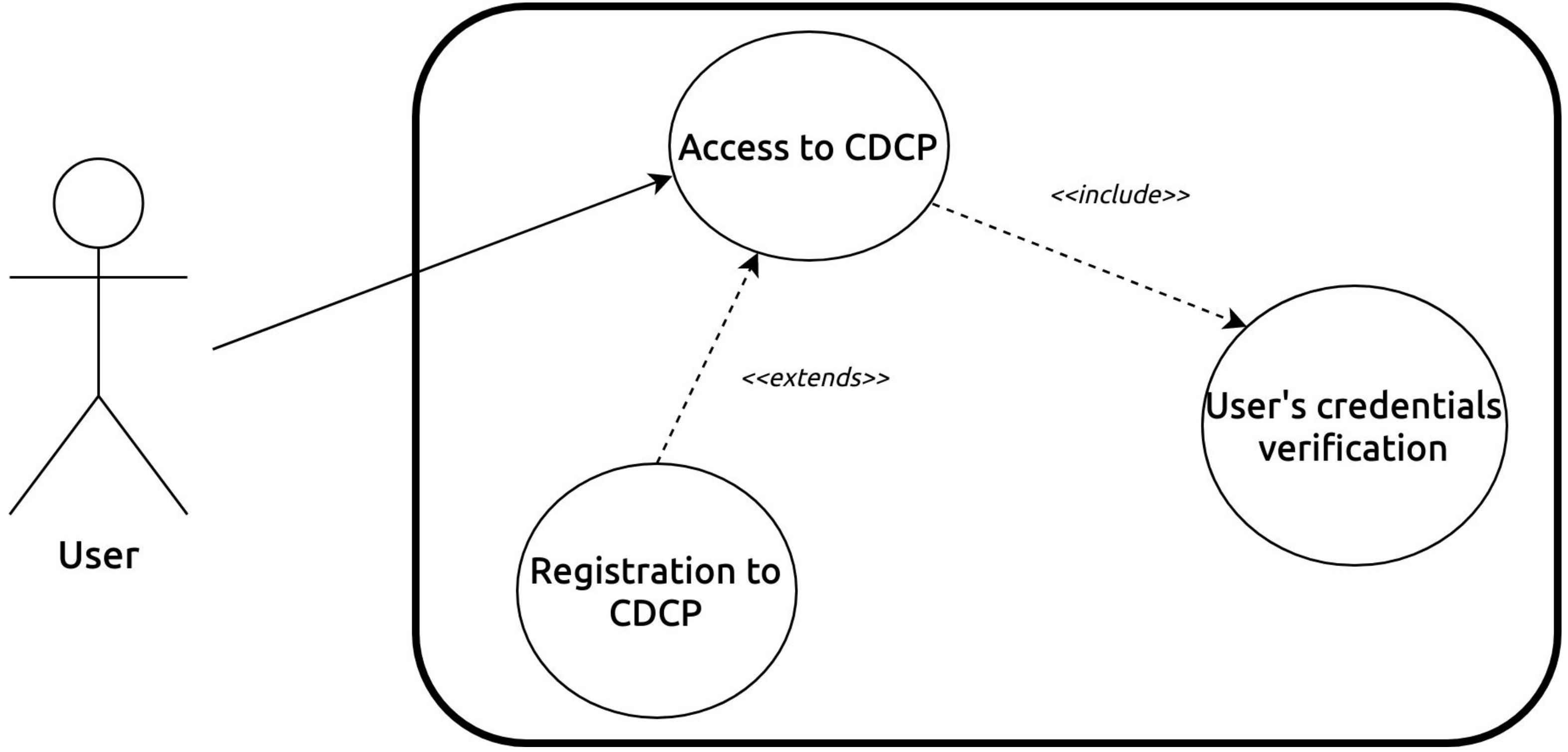
Giuseppe La Gualano and Francesco Vetere

Technologies

Frontend
- HTML5
- CSS3 and Bootstrap
- JavaScript and JQuery

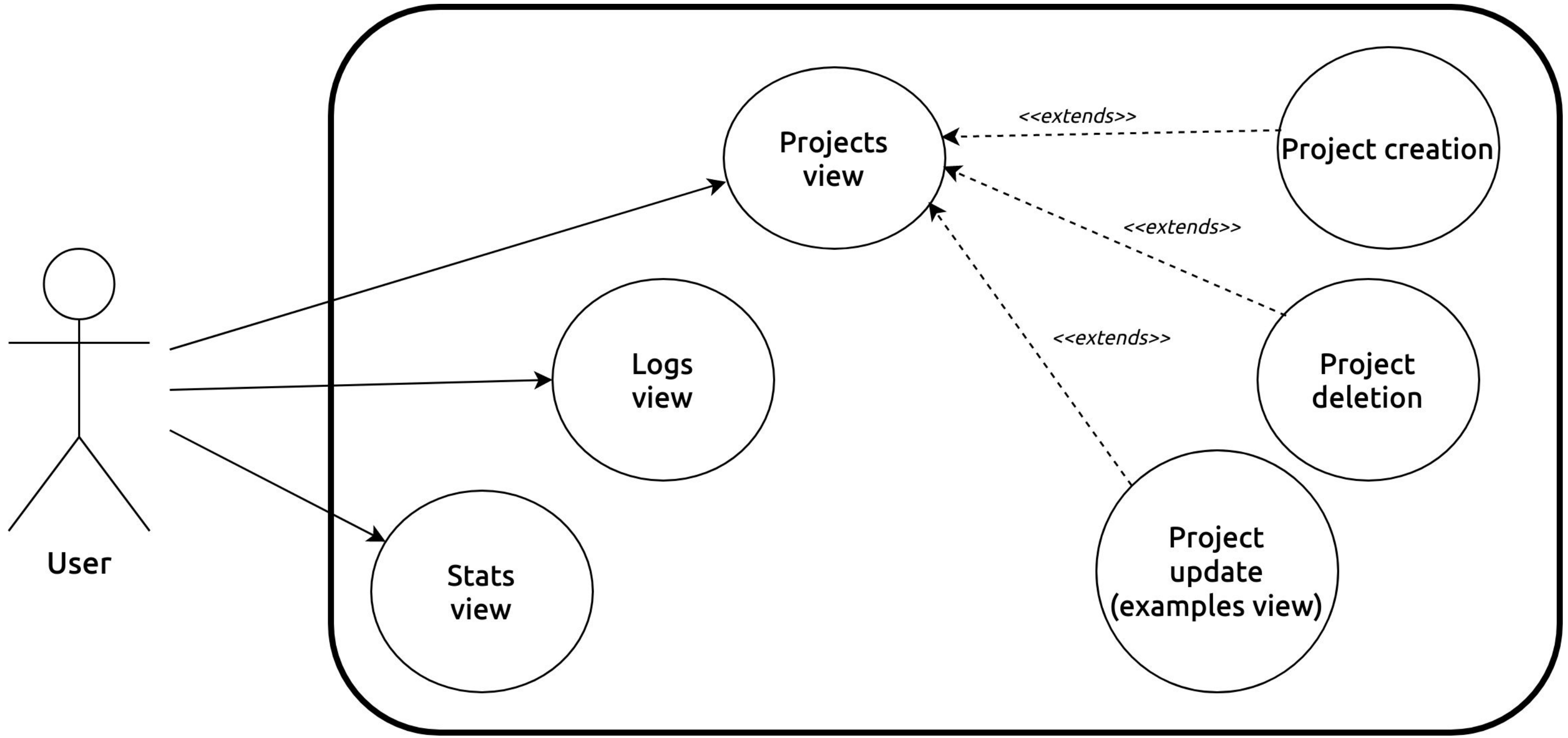
Backend
- Node.js
- Express.js
- MySQL

Use case scenarios



Giuseppe La Gualano and Francesco Vetere

Use case scenarios



Database logical schema

Users (id, nickname, email, password, registrationDate)

Projects (id, title, inputType)

Examples (id, Projects.id, inputType, inputValue)

TagNames (Projects.id, Examples.id, tagName)

TagValues (Projects.id, Examples.id, TagNames.tagName, tagValue)

Logs (id, Users.Nickname, Projects.id, actionType, details, timeStamp)

TokenAuth (id, nickname, token, expirationDate)

Single Page Application

- Client performs **just one request** to the Apache server for all the static contents, at the very beginning (index.html, JS/CSS files, images).

- Client has now its own state and logic, needed in order to display contents on screen: in particular, **no other static resource** is required from the Apache server.

- All other requests will be /api/ requests, performed as **AJAX calls**: form submits will not cause any page reloading!

APIs

- **RESTful** APIs are used, and implementation is done with Express.js using JSON as representation format.

- For almost every DB entity 4 APIs are implemented, in order to perform CRUD operations on them.

- **“Stateless”** requirement is considered (e.g. project examples update).

- API’s don’t need to worry about deleting referenced objects: **“on delete cascade”** logic is already implemented in DDL commands that build up the DB.

Database interactions

- Queries on the database are performed server-side using **promise-mysql.js** library (async/await 🙂).

- Moreover, **prepared statements** are used: every query is processed in order to to prevent SQL Injection attacks.

- A class **DBManager.js** has been written, which takes care of opening a connection, performing a query, and closing the connection.

Cookies

- Two different cookies are stored.

- **Authentication cookie** (tk_auth): if its value is setted ("Remember me" toggled), user bypasses login page and is automatically redirected to the home page.

- **Session cookie** (id_session): used if the first cookie is not setted. Useful for maintaining the user session active even if page reloading occurs.

Authentication

- HTTP is used as application layer protocol: this implies, for instance, having an **insecure authentication** scheme.

- However, passwords are not stored in clear in the DB.
  In fact, a simple **hash function** is applied before storing them, using **bcrypt.js** library.

Security issues/future developments

- **HTML escaping** properly implemented on both client and server side, in order to prevent attacks like XSS.
- **HTTPS** instead of HTTP: confidentiality, integrity and authentication.
- Specific actions should be taken against **DDoS** (WAF, decreasing TCP timeout) and other possible attacks.
- Possibility to switch to a more **compact visualization** for project's examples (huge amount of data expected).

Software implementation

Frontend

- index.html
- custom.css
- application_logic.js, <name>_page.js, cookies.js

Backend

- server.js
- options.js
- routes.js
- DBManager.js

DEMO