André Restivo

LTW Project

Project description for the 2023 edition of the Web Languages and Technologies course.

Objective

Develop a website to streamline and manage **trouble tickets** effectively. The system should enable users to **submit**, **track**, and **resolve** tickets promptly and efficiently. Additionally, the website should have intuitive user interfaces and reporting functionalities to provide real-time insights into ticket status and performance metrics.

To create this website, students should:

- Create an SQLite **database** that stores information about users, tickets, departments, hashtags, and a database of frequently asked questions (FAQ).
- Create documents using **HTML** and **CSS** representing the application's web pages.
- Use **PHP** to generate those web pages after retrieving/changing data from the database.
- Use **Javascript** to enhance the user experience (for example, using Ajax).

Workgroups

- Students will complete this project in **groups of three**.

  In classes where the number of students is not a multiple of three, one or two groups of **two students** will be created.
- Students should contact their practical class teachers during the **weekly class** (or using **Slack**) to establish these workgroups.

Requirements

Note: Requirements are **sparsely defined** so that each group has some **freedom** to create a different website. Use your imagination!

In this ticket tracking website, there are three types of users: **clients** that want to submit and track new tickets (*e.g.*, "Someone changed my password, and now I cannot log in to the website"), **agents** that get assigned to tickets and solve them, and **admins** that have complete control over the website.

The **minimum** expected set of requirements is the following:

- **All users** should be able to (users can simultaneously be clients and agents):
    - **Register** a new account.
    - **Login** and **Logout**.
    - Edit their **profile** (at least name, username, password, and e-mail).
- **Clients** should be able to:
    - **Submit** a new ticket optionally choosing a department (*e.g.,* "Accounting").
    - **List** and **track** tickets they have submitted.
    - **Reply** to inquiries (*e.g.*, the agent asks for more details) about their tickets and add more information to already submitted tickets.
- **Agents** should be able to (they are also clients):
    - **List** tickets from their departments (*e.g.,* "Accounting"), and **filter** them in different ways (*e.g.*, by date, by assigned agent, by status, by priority, by hashtag).
    - **Change** the department of a ticket (*e.g.,* the client chose the wrong department).
    - **Assign** a ticket to themselves or someone else.
    - **Change** the status of a ticket. Tickets can have many statuses (*e.g.,* open, assigned, closed); some may change automatically (*e.g.,* ticket changes to "assigned" after being assigned to an agent).
    - **Edit** ticket hashtags easily (just type hashtag to add (with autocomplete), and click to remove).
    - **List** all changes done to a ticket (*e.g.,* status changes, assignments, edits).
    - **Manage** the FAQ and use an answer from the FAQ to answer a ticket.
- **Admins** should be able to (they are also agents):
    - **Upgrade** a client to an agent or an admin.
    - **Add** new departments, statuses, and other relevant entities.
    - **Assign** agents to departments.
    - **Control** the whole system.

Students should also make sure that:

- The following **technologies** are all used:
  HTML, CSS, PHP, Javascript, Ajax/JSON, PDO/SQL (using sqlite).
- The website should be as **secure** as possible.
  Have special attention to SQL injection, XSS and CSRF attack protection, and sound password storage principles.
- The code should be **organized** and **consistent**.
- The design does not need to be *top-notch* but should be **clean and consistent** throughout the site. It should also work on **mobile** devices.
- Frameworks are **not allowed**.
- Small helper libraries (*e.g.*, displaying a gallery of pictures) **might be** allowed (talk with your practical

class teacher).

Some suggested **additional** requirements. These requirements are a way of making sure each project is unique. You **do not have** to implement all of these:

- Tickets can have documents attached to them (both by clients and agents).
- Admins should be able to see key performance indicators and other statistics (*e.g.*, number of tickets closed by agent, number of open tickets per day).
- Agents can belong to more than one department.
- Agents can see a client's history.
- Agents can watch tickets not assigned to them (*e.g.*, when transferring a ticket, the agent can check a box stating that he still wants to follow the ticket).
- Tickets can be merged together or marked as duplicates from another ticket.
- Tickets can have to-do lists that must be completed before the ticket is closed.
- Tasks can also be assigned to agents.

If you prefer, you can create your own additional requirements.

Work Plan

This is a proposed plan to guide your work. **No deliverables are expected** or will be evaluated on these dates:

- Week 6: Create mockups and navigation diagrams and a first draft of the database design.
- Week 7: Finalize the database script, create the database, and implement most main pages.
- Week 8: Implement all main pages.
- Week 9: Start working on secondary features.
- Week 10: Continue working on secondary features and start working on Javascript and Ajax.
- Week 11: Work on REST API or other secondary features, testing, and code cleanup.
- Week 12: Finish secondary features and focus on security aspects.
- Week 13: Final testing and code cleanup.

We recommend that students adopt an **agile** methodology. Don't start by planning every little detail right from the start, as you risk ending up with a great plan but a poor implementation, but be aware of code organization and quality from the beginning.

Evaluation

Evaluation will be done on the following **topics**:

- Complexity (*e.g.*, implemented features).
- Security (*e.g.*, XSS, CSRF, injection, password storage).

- Technology (*e.g.*, correct usage of HTML, CSS, Javascript, Ajax, No frameworks).
- Quality (*e.g.*, code quality, file organization, consistency).
- User Interface (*e.g.*, usability, consistency).

The goal is to implement the requirements in a **unique way** that meets the project objectives. Using code from other sources **without proper attribution** will lead to the student failing the class (possibly with an RFR mark).

Delivery

- Delivery until the 19th of may at 23:59 (WEST).
- Demo in last week's practical class (using the delivered version).

Created using Sitory. Copyright 2022 André Restivo.

🕒	✓
17/3	☐ Mockups
17/3	☐ Diagramas de Navegação
17/3	☐ Rascunho da Base de Dados
24/3	☐ Base de Dados
24/3	☐ Implementação das Páginas Principais
31/3	☐ Implementação das Páginas
14/4	☐ Início das funcionalidades secundárias
21/4	☐ Continuação das funcionalidades secundárias
21/4	☐ Javascript & Ajax
28/4	☐ APIs e outras funcionalidades secundárias
28/4	☐ Testagem e Limpeza do Código
5/5	☐ Fim dos funcionalidades secundárias
5/5	☐ Seguransa
19/5	☐ Teste e Limpeza Finais