**MVP2 Report**

**Litter Smart**

**A web app solution to report, clean, and track littering.**

A green recycle bin full of garbage

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| Company: | Wiley Edge |
| Tutor Name(s): | Eugene O’Regan |
| Year: | 2023 |
| Project Title: | Group Final Project: Litter-Snap |
| Student Name : | Verejan Vasile  Kevin Zefi  Csaba Vadasz |
| Submission Date: | 02/06/2023 |

Introduction

In this report, we will focus on explaining the steps taken when creating the Minimum Viable Product in building the Litter App for a local cleaning company. We will be using a set of values and principles (Agile). Agile is the most popular approach in software development and project management which helps teams deliver results to the customers quickly and quickly. Requirements and results are evaluated continuously so any change in implementation could be adapted fast without any difficulties. We use a great combination of Kanban Board and Scrum methodology, that helps our development focus on tasks and improving time management.

The report will cover the product log, sprints, user stories, acceptance criteria for all stories, test cases, and Kanban board and sprint review meetings. Due to the fact that there is such a short time available for the implementation of the project, the team agreed on 2-day sprints. The basic developer rules are flexible, set up by the team, but after the agreement, all the developers must keep them.

Basic developer rules

During the development we have the following events:

* Sprint Planning.
* Daily Scrum.
* Sprint Review.
* Sprint Retrospective.
* Backlog Refinement.

The team agreed to appoint Csaba Vadasz as lead developer. The team created the GitHub repository and agreed on the following:

* Pull requests have been set to the GitHub repository
* Merging into the master branch (production code) after the pull request is required the lead developer’s approval
* Merging into the developer branch requires code review, 2 of the developers must make the code review (that forces all of us to be familiar with the whole code)

We agreed on continuous and mutual support of each other.

Scrum events are handled by the team leader.

All developer has the freedom to add extra fields, and functions to fulfill the requirement. Fields do not need to discuss, but extra features and functions require basic negotiations with others, and cannot risk the

completion of the project!

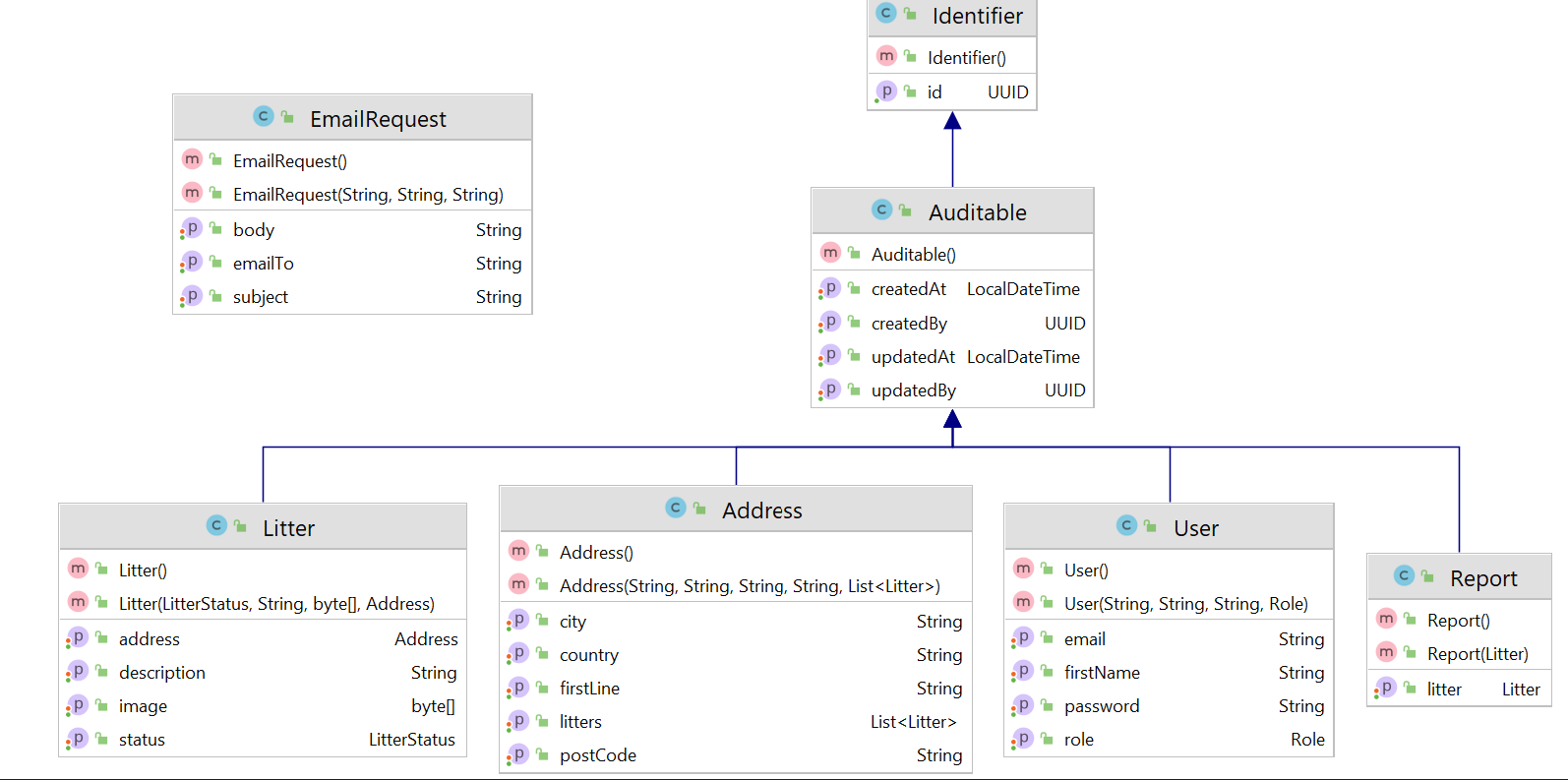
*MUST be kept in any cases:*

* MVC approach
* Naming conventions (e.g all packages are singular)
* Achieve uniform code with naming, structuring
* Clean Code (code readability is the main focus!)
* SOLID principles
* JavaDoc type of comments (on classes, fields and methods)
* The minimum test coverage is 80% (unit and integration tests)
* Expect a unified URL (where the displayed entity is plural in the URL)
* Continous documentation and manual & readme update (both GitHub and in the documentation)
* Any changes in pom.xml or application.properties are allowed only after the agreement
* Future hosting plan on our existing domain: csaba79coder.com (presented in the project structure)
* Enjoy coding ☺

*Future plans:*

* Improve test coverage (unit tests, integration tests, end-to-end tests)
* Implement login form and Spring Security
* Create a separate table for roles (and set a list of roles to the users)
* After roles are implemented, we plan to make a separate REST API & Thymeleaf for the user (now only admin implemented)
* We plan to use our existing domain: csaba79coder.com
* We plan to make registration with social media
* We also plan to make a mobile app (and use Google map's API there for the localization of the users)

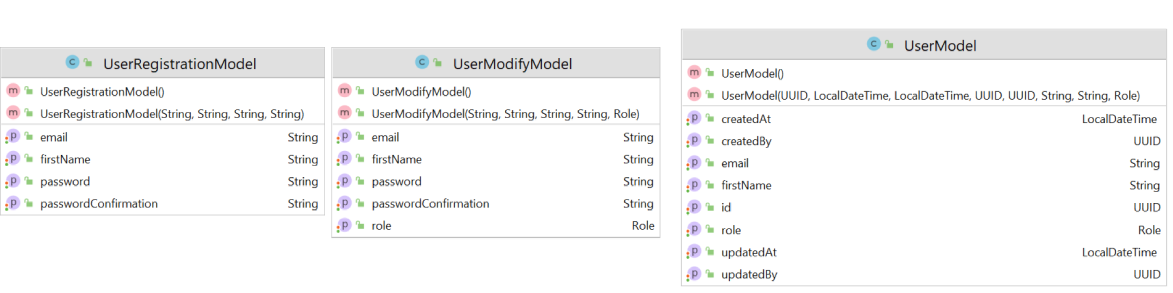
*User Stories:*



***As all Entity has their own ID, that is a UUID, for avoiding code repetition, we set up a base entity, that has an ID field, for auto-generating it, that is extended by the Auditable class (contains: createdAt, createdBy, updatedAt, updatedBy fields) as this is also must be set for all the entities, that reason these Super Classes must be extended (inherited) by the following entities: user, address, litter, and report.***

**User:**

*Please check in the project structure the class diagram for Entity, and Models, as that was the base of the User.*

**

Must be done: CRUD operations, with necessary error handling

Nice to have: creating roles, and setup admin and user roles with different operations.

Important:

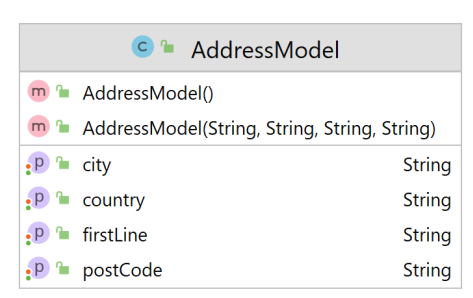
Email and Password validation, using Enums for roles (set up as a base: ROLE\_USER)

As admin, we want to add a new user, delete a user, find a user by email, and by id, and we want to update a user.

Future: we want that the user sees only his data!

**Address:**

*Please check the class diagram for Entity, and Model in the project structure.*



We need to check if all fields are matching in case of an existing address, if it is, save that Address to any of the Entity that requires it.

If an Address does not exist, we have to save it as a new one and set it to the required Entity.

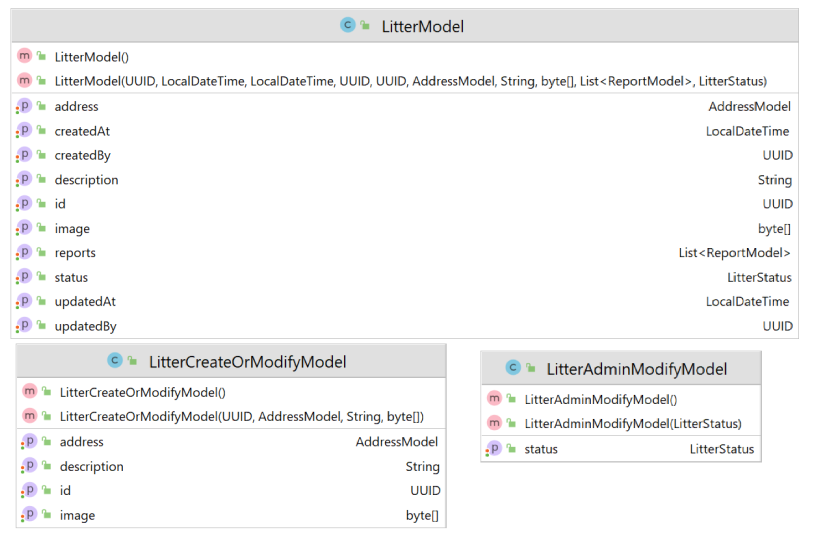
We need also here a required common error handling.

The address does not require a full CRUD operation, just the create we need, as the project does not require changes in address afterward! We do not want to update or delete, and we don’t need a list of addresses, as it is just additional information in the Litter’s report.

Nice to have feature: later for mobile app. to integrate Google map’s API, that is generating the current location (in case the user is accepting it, he does not have to manually add the address itself!)

**Litter:**

*Please check the class diagram for Entity, and Models in the project structure, as that was the base of the Litter.*



The litter needs all the CRUD operations. We have to create a new one and update an existing one, for finding it, we have to implement the findById, and in case of deletion, in the report, we just set the status as COMPLETED.

Automatically a Litter is created by REPORTED status. (that can modify by the cleaning firm, due to the status’s changes.

In Litter required the common error handling.

Nice to have a function: to separate roles, and regarding roles, the admin can see all and can modify all, but the user can only list his litter reports.