```java

package acme.features.manager.project;

import java.util.stream.Stream;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import acme.client.data.models.Dataset;
import acme.client.services.AbstractService;
import acme.entities.project.Project;
import acme.entities.project.acceptedCurrency;
import acme.roles.Manager;

@Service
public class ManagerProjectUpdateService extends AbstractService<Manager, Project> {

    // Internal state ---------------------------------------------------------

    @Autowired
    private ManagerProjectRepository repository;

    // AbstractService<Manager, Project> --------------------------------------


    @Override
    public void authorise() {
        boolean status;
        int masterId;
        Project project;

        masterId = super.getRequest().getData("id", int.class);
        project = this.repository.findProjectById(masterId);
        status = project != null && project.isDraft() && super.getRequest().getPrincipal().hasRole(Manager.class) && project.getManager().getId() == super.getRequest().getPrincipal().getActiveRoleId();

        super.getResponse().setAuthorised(status);
    }

    @Override
    public void load() {
        Project project;
        int id;

        id = super.getRequest().getData("id", int.class);
        project = this.repository.findProjectById(id);

        super.getBuffer().addData(project);
    }

    @Override
    public void bind(final Project project) {
        assert project != null;

        super.bind(project, "code", "title", "abstractText", "hasFatalErrors", "cost", "link");
    }

    @Override
    public void validate(final Project project) {
        assert project != null;
```

```
60
61          if (!super.getBuffer().getErrors().hasErrors("cost")) {
62              boolean isCurrencyAccepted = Stream.of(acceptedCurrency.values()).map(x ->
    x.toString().toLowerCase().trim()).anyMatch(currency -> currency.equals(project.getCost
    ().getCurrency().toLowerCase().trim()));
63
64              super.state(isCurrencyAccepted, "cost",
    "manager.project.form.error.incorrectConcurrency");
65              System.out.print(Stream.of(acceptedCurrency.values().toString().toLowerCase
    ()));
66              super.state(project.getCost().getAmount() >= 0, "cost",
    "manager.project.form.error.negative-cost");
67          }
68
69          if (!super.getBuffer().getErrors().hasErrors("code")) {
70              Project existing;
71
72              existing = this.repository.findOneProjectByCode(project.getCode());
73
74              super.state(existing == null || existing.equals(project), "code",
    "manager.project.form.error.duplicated");
75          }
76
77      }
78
79      @Override
80      public void perform(final Project project) {
81          assert project != null;
82
83          this.repository.save(project);
84      }
85
86      @Override
87      public void unbind(final Project project) {
88          assert project != null;
89          boolean userStoriesPublishables;
90          boolean isDraft;
91
92          userStoriesPublishables = this.repository.findAllUserStoriesOfAProjectById
    (project.getId()).stream().allMatch(x -> x.isDraft() == false) &&
    this.repository.findAllUserStoriesOfAProjectById(project.getId()).size() > 0
93              && project.isHasFatalErrors() == false;
94          isDraft = project.isDraft() == true;
95
96          Dataset dataset;
97
98          dataset = super.unbind(project, "code", "title", "abstractText", "hasFatalErrors",
    "cost", "link", "isDraft");
99          dataset.put("masterId", project.getId());
100         dataset.put("publishable", userStoriesPublishables);
101         dataset.put("isDraft", isDraft);
102
103         super.getResponse().addData(dataset);
104     }
105
106 }
107
```