```java

package acme.features.manager.project;

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.roles.Manager;

@Service
public class ManagerProjectPublishService 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", "isDraft");
    }

    @Override
    public void validate(final Project project) {
        boolean condition = this.repository.findAllUserStoriesOfAProjectById(project.getId()).stream().allMatch(x -> x.isDraft() == false) && this.repository.findAllUserStoriesOfAProjectById(project.getId()).size() > 0
            && project.isHasFatalErrors() == false;
        super.state(condition, "*", "manager.project.form.error.publishable");
        if (!super.getBuffer().getErrors().hasErrors("hasFatalErrors"))
            super.state(!project.isHasFatalErrors(), "hasFatalErrors", "manager.project.form.error.fatal-errors");

        if (!super.getBuffer().getErrors().hasErrors("cost"))
            super.state(project.getCost().getAmount() >= 0, "cost", "manager.project.form.error.negative-cost");

        if (!super.getBuffer().getErrors().hasErrors("code")) {
            Project existing;

            existing = this.repository.findOneProjectByCode(project.getCode());

            super.state(existing == null || existing.equals(project), "code", "manager.project.form.error.duplicated");
        }

    }

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

        project.setDraft(false);
        this.repository.save(project);
    }

    @Override
    public void unbind(final Project project) {
        assert project != null;
        boolean userStoriesPublishables;
        boolean isDraft;

        userStoriesPublishables = this.repository.findAllUserStoriesOfAProjectById(project.getId()).stream().allMatch(x -> x.isDraft() == false) && this.repository.findAllUserStoriesOfAProjectById(project.getId()).size() > 0
            && project.isHasFatalErrors() == false;
        isDraft = project.isDraft() == true;

        Dataset dataset;

        dataset = super.unbind(project, "code", "title", "abstractText", "hasFatalErrors", "cost", "link", "isDraft");
        dataset.put("projectId", project.getId());
        dataset.put("publishable", userStoriesPublishables);
        dataset.put("isDraft", isDraft);

        super.getResponse().addData(dataset);
    }

}
```