# Project Desperado
# Test Document

**Actor: User**

**Use Case**: User Register

**Physical Testing** (Manual): To test the registration on the front end we use physical testing. A tester should visit the site, click to register, follow the registration process and make sure that through finishing the process they can log in.

**Automated Testing**: Registration implies the creation of a User model. API Backend tests should contain a test to ensure a User object can be created. Also the POST to /api/users should be able to accept a Json User Object and create it. There should be a test to ensure these things.

**Use Case**: User Login

**Physical Testing** (Manual): One a user has registered, using the login button a physical tester should be able to enter correct credentials and log in to the system.

**Automated Testing**: Login is tested indirectly on the backend. A user can not interact without the correct token generation which is associated with the login process. A test process should

**Use Case**: User Edit Profile

**Physical Testing** (Manual): The front end application will be physically tested for User edit. A button on the menu bar labeled edit will be available to users. A physical tester will click on the edit button, change their information, and the be able to view the changed information.

**Automated Testing**: The back end should have a test to ensure that a PUT request with a JSON User object and the correct authorization parameters to /api/users/{id} can update the attributes to a user. The tests should also handle a request with incorrect parameters passed returns the correct HTML response.

**Use Case**: User Delete Account

**Physical Testing** (Manual): A user should be able to delete their own account through the front end application through a physical tester. Under the user edit there will be a button to remove the account. After the button is clicked the tester should attempt to login and  the login should fail.

**Automated Testing**: The backend should be able to receive a DELETE request at /api/users/{id}. After the request is received the user should get a correct HTML response and the User should no longer persist. There should also be a test to cover that a delete can only happen with the correct permissions.

**Use Case**: User Add Project

**Physical Testing** (Manual): From the front end there is a projects tab. The physical tester should enter the Projects tab, then click on new project. After filling in the description and name of the project the tester will submit for creation. After clicking submit the user should see the project.

**Automated Testing**: The should be a unit test on a User model to make sure that a Project can be added to the User. Indirectly this also calls for the unit tests on the Project. We must have a valid Project model in order to add one to a user. There also needs to be controller tests to route the traffic interacting with these models. The Project controller needs a test to ensure that a POST to /api/projects with the correct JSON Project parameters, as well as authentication tokens will create the project and have the correct response. We also need to test that incorrect requests give the correct failure response.

**Use Case**: User Add Other User to Project & Apply for a project

**Physical Testing** (Manual): The front end will be tested physically for this use case. There will be an application to a Project, and also an invitation to a project. If a user applies to the project they will be put on the project. A physical tester should apply to a project then check that the project was added to their projects. The tester should also create their own project, then invite another user. That user should now be on their project.

**Automated Testing**: The introduction of adding and inviting users creates new models. The Membership model will serve as a connection between the Project and User models. There are two subclasses of the Membership model, Invitation and Application. All three of these models need unit tested. There need to be controller

tests testing that a POST  to /api/projects/:project_id/users with a User JSON model and correct params will add a user to a project. The invitation model has its own POST url, /api/projects/:project_id/applications which needs the same criteria. Both need both the positive and negative responses tested.

**Use Case**: User Remove Other User from Project

**Physical Testing** (Manual): The front end test will be done physically. A user who owns a project should navigate to the list of users on the project and click to remove. The user should no longer be on the project. A tester should then navigate to a project they do not own and not be able to remove users.

**Automated Testing**: In order to remove a user from a project we need a controller test. The test should ensure that a DELETE request to /api/projects/:project_id/users/:id removes the user and has the correct response. It should also check that if a user doesn't have permission to perform the delete they receive and unauthorized response.

**Use Case**: User Edit Project

**Physical Testing** (Manual): A user should only be able to edit a project if they are the owner. When physically testing the front end the tester should enter a project, then click the edit button. After changing the values the value should persist. After the tester should then navigate to a project they do not own and they should not have an edit button available.

**Automated Testing**: In order to edit a project a user must be the owner. There needs to be a controller test that checks that PUT to /api/projects/:id with valid Project parameters and the correct users authentication can update a Project and have the correct HTTP response. If the User does not have access to edit the project the update should fail and the HTTP response mirror the failure.

**Use Case**: User View Other Users

**Physical Testing** (Manual): The front end is physically tested for this use case. Upon log in there should be a list of Users under the users tab.

**Automated Testing**: The needs to be a test on the User controller that a GET to api/users/ gets a list of all active users and a 200 response.

**Use Case**: User Open Projects

**Physical Testing** (Manual): The front end is physically tested for this use case. Upon log in there should be a list of Projects under the project tab.

**Automated Testing**: The needs to be a test on the Project controller that a GET to api/projects/ gets a list of all active projects and a 200 response.

**Actor: System Admin**

Simple by having an Admin actor we need to have a role on the User object that distinguishes a regular user from the Admin user. We need a test to test to correctly identify whether a user is an admin or not. This would live in the user model test.

**Use Case**: Add User to System

**Physical Testing** (Manual):The physical tester, logged in as admin, should observe the ability to register a user other than themselves. They will be able to view the added user after adding.

**Automated Testing**: There should be a test to ensure that a POST to /api/users can be made for a user other than the one logged in(admin).

**Use Case**: Modify Users

**Physical Testing** (Manual): The physical tester, logged in as admin, should observe the ability to edit all users. They will be able to view and edit each user from the Users page.

**Automated Testing**:There should be a test to ensure that a PATCH to /api/users/{id} can be made to all users.

**Use Case**: Delete Users

**Physical Testing** (Manual): The physical tester, logged in as admin, should observe the ability to edit all users. They will be able to view and edit each user from the Users page.

**Automated Testing**: There should be a test to ensure that a DELETE to /api/users/{id} can be made to all users.

**Use Case**: Manage/Edit a Project

**Physical Testing** (Manual):The physical tester, logged in as admin, should observe the ability to edit all users. They will be able to view and edit each user from the Users page.

**Automated Testing**: There should be a test to ensure that a PATCH to /api/projects/{id} can be made on projects in which the admin does not belong too.