ScrumSpace Server Submission (#5)

<u>Overview</u>

This report details our transition from a mock server/database to a fully functional node server with an Express REST API. Following, we'll be exploring the route design in our API, mentioning individual's contributions to this transition and going over additions and fixes to the client side of the application.

<u>Running the Application</u>

Just as before, our application is using Gulp to build and package our application. After dependencies are installed with `npm install`, running an `npm run serve` will trigger the Gulp task and then start the server synchronously.

<u>State of Authentication</u>

Currently we're implementing a very basic understanding of authentication. In the (distant) future, we'll need to implement a role-based user system and expand the complexity of our authentication logic. For the time being however, a user is considered authenticated for project specific actions if they are assigned to that project.

<u>HTTP Requests/Routes</u>

<u>Static Content</u>

As mentioned in the previous report, we've pushed access to all static assets (like JS, CSS and images) to a virtual path prefix, `/static`, which provides static access to the `dist` directory.

<u>REST API</u>

The API lives at `/api`, providing access to all of our application's entities. We'll explore this section in more detail shortly.

<u>Entry</u>

All other unmatched routes are forwarded to our server-side entry router, allowing for react-router to work as intended.

The REST Routes

GET /api/init
author: Abhay Vatsa

This route returns the global application state for the user that is authenticated. We return a collection with two root-level objects: **user** and **projects. User** contains user personal data and **projects** contain all projects the user has access to. The projects object contains all the data needed for the application from stories, sprints, to tasks.

Users

GET /api/user/:id
author: Dylan Fischler

This route returns a “cleansed” (only public information) user object belonging to the ID supplied in the parameters. This is a public route, meaning it is not protected by authentication. Because our application provides the ability to add any user in the system to a project, all user’s public information is available. Currently this route isn’t being used for anything but we’ve kept it in for future use. A user’s personal data is instead sent over with our /api/init route.

GET /api/user/search?searchStr=someString&key=someKey
author: Dylan Fischler

This route allows for a search over the users collection with some search string, and an optional key to search by. While we are aware of the contested use of GET or POST for searching, we’ve opted to use GET because of its specific use in our project. For the reasons listed above in the previous route, this route is also unprotected. It is important to note however that this search functionality also returns “cleansed” user objects. The route is being used in the Project Settings page to allow for a user to add additional users to a project, and has replaced the collection search from the mock server.

PUT api/user /:user_id
Body {user["_id": string, "first_name": string, "last_name": "string", "email": "string", "display_name": "string", "old_password": "string, "new_password": "string", "avatar_url": "string"] }

author: Supriya Kankure

This route works the following way. The route takes a user id and updates information about that particular user. We shall assume that the user is logged in, so when they are taken to this Page, the id is reflective of that user. The user can update their first name, last name, display name, email, and password. If the user does not chose to update those fields, then they will remain the same in the database. If the the user does choose to update these fields, then they will be updated in the database. In the case of passwords, the user must type their old password. If this password matches what is in the database, the new password the user types in will update and replace the old password. This page falls under the user page under api. The server routes for this function is working correctly, but client side is to be complete. The schemas are being validated, but the user is not being authorized because this is a lingering bug that will be soon fixed. In postman, if you take the following url after localhost:8080, the server correctly returns an updated user object. However since the client side is not working, the database is not being updated with the newly updated user object.

Tasks

All Task routes below follow the above mentioned authentication pattern, only allowing action if the performing user is present on the parent project. Additionally, all Task PUT routes below are part of the expansion of the monolithic serverUpdateTask function from the mock server. The expansion came out of the need to support concurrent task actions.

PUT /api/project/:project_id/story/:story_id/task/:task_id
Body { description: String, status: String (Enum) }
author: Dylan Fischler

This route allows for the updating of first-level properties (description and status) on a Task. The route is used in the Task Detail to modify the description of a task, as well as on the Scrum Board when a task is moved to another place on the board.

PUT /api/project/:project_id/story/:story_id/task/:task_id/assigned_to
Body { users: [userIDs], replace: Bool }
author: Dylan Fischler

This route allows for a user to assign users to the associated task. The users are provided in an array from the request's body. Additionally, an optional replace boolean can be sent with the body. This allows for the route to be used to <u>set</u> the assigned users on a task, while also supporting the <u>addition</u> of users to the already existing array. The route is used in the Task Detail panel to add users to a Task, as well as in the Assign User Modal that presents itself when dragging an unassigned Task to the Doing state on the Scrum Board. Due to how the client-side functions, this route's replace functionality enables us to "remove" users from a task, allowing us to omit a specific DELETE route.

PUT /api/project/:project_id/story/:story_id/task/:task_id/blocked_by
Body { blocking: [taskIDs], replace: Bool }
author: Dylan Fischler

This route allows for a user to assign blocking tasks to the associated task. The tasks are provided in an array from the request's body. Additionally, as in the assigned_to route, an optional replace boolean can be sent with the body, allowing for the route to be used to <u>set</u> the blocking tasks on an associated task, while also supporting the <u>addition</u> of blocking tasks to the already existing array. The route is used in the Task Detail panel to add blocking tasks to a Task, as well as in the Assign Blocking Modal that presents itself when dragging a Task to the Blocked state on the Scrum Board. As stated above, the route's replace functionality enables us to "remove" blocking tasks from the associated Task, therefore allowing us to again omit a specific DELETE route.

DELETE api/project/:project_id/story/:story_id/task/:task_id
Body{}
author: Niharika Venkatathri

This route allows the user to delete a particular task from a story in a project. The body for this request is empty because all of the data needed to locate the story and task is provided in the req.params. The route checks to make sure that the user is a part of the project before the user is allowed to delete a task.

POST api/project/:project_id/story/:story_id/task
Body{status, description}
author: Niharika Venkatathri

This route allows the user to post a new task to the story. This will only work if the user posting the task is a part of the project. This route assumes that the 'history' field of the task object is null since it is a newly created task that doesn't have a history. The modified project with the additional task is sent back to the front end for the state tree to be updated. The route can also be used to update the status or description of an existing task.

Sprints

POST /api/project/project/:project/sprint
Body{name: String, scrum_time: String, duration: Integer}
author: Ryan Jerue

This route allows for the user to create a new sprint. The route will only run in the user token sent to it matches that of a user on the project. The route is initiated by the "done" button inside of the modal that creates a new sprint. It replaces the mock server method for posting a new sprint. A modified project is returned back to the front end so that the redux state tree may be updated. In terms of errors, if one tries to post to a project they do not have permission for or does not exist, they will receive a 401 error (they don't need to know it doesn't exist).

PUT/api/project/project/:project/sprint/:sprint
Body{name: String, scrum_time: String, duration: Integer}
author: Ryan Jerue

This route allows for a user to edit a sprint's name, duration, and scrum time. Like the POST route, this route will only run if the user token matches one of the current project. This route is initiated by the "done" button in the sprint creation modal appears when the "edit" button if pressed for a particular sprint. It replaces the functionality of the mock server's postNewSprint function. Previously, posting and putting a sprint used the same function. Now, while they both still use the same helper function, they have separate routes. A modified project is returned back to the front end so that the redux state tree may be updated. In terms of errors, if one tries to put to a project they do not have permission for or does not exist, they will receive a 401 error (they don't need to know it doesn't exist). If the sprint they try to put to doesn't exist, they will receive a 400 error.

PUT /api/project/:projectid/sprint/:sprintid/start
author: Dylan Fischler

This route allows for a user to initiate the start of a Sprint. It's authorized for users on the parent project. The route is used on the Sprint Planning page, with the Start Sprint button existing on sprints with at least one story in them. This route is a new introduction, therefore does not replace any mock server methods.

REMOVE /api/project/project/:project/sprint/:sprint
author: Ryan Jerue

This route allows for users to delete a particular sprint. In the project planning window, close buttons will appear next to the panels for each sprint (except for the active sprint) allowing for the user to delete them from their project. Upon pressing the button, the route is called if the user token is one that is assigned to the particular project, and the sprint is removed from the database. Any stories in that sprint are moved to the backlog. This replaces a the former mock server method for deleting sprints. A modified project is then returned back to the front end so that the redux state tree may be updated. In terms of errors, if one tries to delete from a project they do not have permission for or does not exist, they will receive a 401 error (they don't need to know it doesn't exist). If the sprint they try to delete doesn't exist, they will receive a 400 error.

Stories

POST /api/project/:project_id/story/
Body {title, description, tasks: [task]}
author: Abhay Vatsa

This route will allow a user to create a new story for a project. On creation, it's not assigned to a sprint, so it be assigned a null sprint_id in the route. The title, description, and an array of tasks. Only users authorized for the project will be able to create the story. This is used by sprint-planning.

This replaces the body of serverMakeNewStory function used by the client.

PUT /api/project/:project_id/story/:story_id/
Body {title, description, tasks: [task]}
author: Abhay Vatsa

This route allows updating properties in the story object. In the request body we provide properties that need to need to be updated. Only users authorized for the project will be able to update the story. This is used on the scrumboard as well as on sprint-planning.

This replaces the body of serverMakeNewStory function used by the client.

DELETE /api/project/:project_id/story/:story_id/
Body {}
author: Abhay Vatsa

This route lets a user delete a story attached to a particular project. No additional data is provided in the body, as the req.params will provide enough information to locate the story. Only users authorized for the project will be able to delete the story. This route is used by Sprint Planning.
This replaces the body of serverRemoveStory function used by the client.

Projects

POST /api/project
Body {title, description, user_ids, membersOnProj}
author: Rachana Lingutla

This route will allow a user to create a new project. On creation, the project is assigned a title, description and users. When the creator adds users, the client side simply stores the ID of the added members to the project. This makes it easier to validate the data coming through. It is not too difficult to re-map the full user object to the project. The membersOnProj array stores the first names of the users added onto the project. This array is used exclusively for the git stats feature. Anyone can create a project and add any users to it, which means that this feature doesn't need authentication.

This replaces the body of serverPostNewProject function used by the client's mock server.

PUT /api/project/:projectid
Body {project_iD, title,users}
author: Rachana Lingutla

This route allows user to update a project's users or title. In the request body we provide the properties we want to update along with the project id so the server knows which project we want to update Only users working on the project will be authorized to update it(basically users whose ids' exists in the users array of a project).

This is a new functionality that did not exist in the old version of the project. We opted to give the users more flexibility in managing their projects and thus built this functionality.

DELETE /api/project/:project_id
Body {}
author: Rachana Lingutla

This route lets a user delete a project . Only users who are assigned to the project will be authorized to delete it (basically users whose ids' exists in the users array of a project). This function can't be reversed. This is a new functionality that did not exist in the old version of the project. We opted to give the users more flexibility in managing their projects.

<u>Git-Stats (Rachana Lingutla)</u>

So for my git stats feature, I did not need to create any/special new http routes. As with the previous submission, my feature is simply querying the state tree for the information it needs. The one http route that is used is the **GET** /api/init route, as this function gives us the tree. After the feature extracts the information the way it did before. Because of this, I also have no mock server methods to replace. Users will only be able to see the data for the projects they are working on. Once I get project validation properly working, this should be a non issue. Since all the issues with data are handled before we get to render the graphs, there are no error banners that appear directly on the graphs page, except perhaps a 'could not get' banner. There is also no hard-coded data in the client side.

I also wanted to respond to this comment that from the last assignment's feedback:
"Commits in the DB should be linked with user accounts in some way, e.g. with a database reference. You should fix this by the next submission."

I unfortunately noticed this comment very late in the assignment and was not able to transfer to the data into the accounts. This process involves restructuring the way git-graphs will extract data, and due to the amount of time it took to get the group assignment functions working, I was not able to get this feature working according to our's (and your's) standards. I aim to have this fixed for the next assignment.

<u>**Client-Side Additions/Fixes**</u>

<u>Scrum Board (Dylan Fischler)</u>

While the Scrum Board was functionable in the previous submission, the Task movement handling was completely rewritten to support logical actions on move. Currently, there are three logical functions included in the board. The first occurs when a user drags an Unassigned Task to a Doing state. When the Task is dropped, instead of moving the Task, a prompt appears for the user to first assign user/s to the Task, pushing the move action to trigger on a successful assign. The second functionality occurs when a Task is moved to a Blocked state. Performing in a consistent manner, a prompt appears for the user to assign blocking task/s, again pushing the move action to the assign. The third logical functionality occurs when a Task is moved out of a Blocked state. When the Task is dropped and saved, all blocking tasks are cleared (server-side).

Task Detail (Dylan Fischler)

Vacant in the last submission, the Task Detail modal now functions as intended (Niharika Venkatathri was responsible for some of the UI). Task descriptions can now be edited by clicking and modifying, users can be assigned/removed, and blocking tasks can be assigned/removed if the task is in a Blocked state.

Project Item Graph (Dylan Fischler)

Also vacant in the last submission, the Project Item (on the Project master page) now renders actual history data in its graph.

Start a Sprint (Dylan Fischler)

As we've continued to explore the usage of the application, we realized some additional flaws in our datetime-boxed sprint model. As such, we've transitioned to a duration based model. Sprints now exist as a duration of time (in days), rather than a date range. This allows for a number of different improvements. We realized that in the real world, committing our users to rigid schedules could be incredibly frustrating when these schedules do not pan out as originally foreseen. With a duration model, a sprint exists in a planning phase indefinitely until a user clicks the Start Sprint button on the sprint planning page, then lasting for the supplied duration. When a sprint's duration has run out, it transitions itself to a Review state (this review transition is in the works still). This flexibility fits much more easily in a real world application.

Project Planning (Ryan Jerue)

As mentioned in the previous report, a rework of Project Planning was a goal. As we learned more about React and Javascript, we made it our goal to make Sprint Planning far more dynamic. Sprint Planning has been changed to Project Planning and now has the ability to plan multiple sprints in advance while preserving the creation of stories as before. Both the creation and editing ability of stories and sprints has been placed into modals. The movement of stories to a specific sprint and back to the backlog is achieved using ReactDND. Sprints and stories may also be removed using the remove button (active sprints may not be removed). Sprints may be edited by clicking on the edit button in their panel's footer whereas stories may be edited by simply clicking on them. The editing and creation is done through two new modals. A story removed from a sprint is put into the backlog, and may only be removed if the remove button is pressed in the backlog. Sprints that contain stories that are removed have their stories moved to the backlog. Additionally, if a Project is in the planning phase, a button to start a sprint is available for the user to use to put the project into the "sprint" phase. Ultimately, Project Planning is a product of vastly improved Javascript knowledge, research, and design.

Assigning Users to Projects(Dylan Fischler)

In the previous submission, users could assign any names they wished to a new project. With our updates, the users are given a list of users with accounts via the MultiSelect functionality. Multiselect allows users to see what options they have and as they click each user to add to a project, the options list decreases. Dylan built the functionality of this tool and several members have used it to more effectively map users to a projects and its sprints, stories and tasks.

Resetting Project Creation Modal (Rachana Lingutla)

In the last submission, the new project creation modal was not being reset it's empty fields if the user closed the modal. This functionality has been repaired and the data is not saved if the modal is simply closed.

User Mapping -Gits Stats (Rachana Lingutla - Honors)

In the last submission, users were not being mapped to their projects. Instead all of the users in the database were displayed on the graph in order to demonstrate how data is being displayed. Now when a new project is created, only the users that were selected in the creation have stats appearing in the graph.

Unmentioned Individual Contributions

Movement of Database and Mock Data to Server (Ryan Jerue)

As the server runs Node, several functions in the database needed to be changed in order to work properly with Node. Additionally, a new database function, overwriteCollection allows for an entire collection to be overwritten by a given document. In addition, readDocument has been modified to return an entire collection if just the collection name is called.

Schemas (Abhay Vatsa)

Hierarchical schemas were created in **/src/server/schemas/** for data validation using json schemas. Since our data is heavily nested, we had to define the dependencies between schemas using the es6 module system. This was preferable to the XPATH way described in documentation as we can reuse schemas as needed to validate smaller parts.

Lingering Bugs

Sprint Review and Sprint Ending (Dylan Fischler)

Currently, a sprint can exist in two of the three states possible. When a sprint is created, it is in the “planning” state. When started, the sprint goes into a “sprint” state. The final state, “review”, will automatically be transitioned to when a sprint’s duration runs out. Currently this does not happen, however the foundation is in place to make this possible. When a sprint enters a “review” phase, it will exist in this state indefinitely until a user chooses to accept the review and end the sprint. After this point, the project is free to enter another sprint.

Task Deletion (Niharika Venkatathri)

The task detail modal has a ‘Delete Task’ button that should be calling the delete task route. This route is missing the entire client side functionality that is required in order to cause a task to be deleted. This is missing due to an oversight on my part.

User Settings (Supriya Kankure)

The save buttons in the user settings page should trigger the server routes, but as of now are not connected. This is an issue with the client side, which will be fixed, so the user can interact with the page. I will be working on fixing this. Additionally, the server method is not checking for authorization on the user_id because the authorization method has been producing -1 for the value of the user_id. This is another bug I plan to fix. Additionally, once the client side methods are fixed, the database will be triggered with a new updated user object

Auth for project functions (Rachana Lingutla)

Authorization is proving a bit tricky for the project functions mainly because I couldn’t figure out in time how to extract the user ID of the person logged in from the header and compare it to the existing users in the project. I need this value so I can compare it the value returned by the authorization function on the server side. It was hard to test when there wasn’t an easy way to switch which user we might be logged into. I also need to validate who can actually view, update and delete projects depending on who has been assigned to the project.

Propagating user changes from project updates to all features (Rachana Lingutla)

In the project settings page, users can currently change the project title, change the users on a project and delete the project entirely. While the changing the title and deleting the project functionalities are properly handled from the server side, the ‘changing users’ feature doesn’t function the way we want it to. We currently can’t delete members from a project without causing other issues in the scrum board. The removal and addition of members to a project isn’t being propagated to the git-stats feature either. This is a straight-forward fix that simply involves crossing checking the ‘assigned_to” array in tasks & the membersOnProj array of a project against the new updated list of users. This will be handled for a future submission.

Project and User Avatars

Both Project and User entities have avatars to aid in the visual identification of each. Currently our mock data contains these avatars, however we do not yet have the functionality for a user to upload these avatars for new entities.