Team info & policies

- **List each team member and their role in the project.**
  - Aidan
    - Backend / Firebase
  - Mitchell
    - Frontend / Backend
  - Sofia
    - Backend
  - Lance
    - Frontend / UI Design
  - Tye
    - Backend

- **Link to each project relevant artifact such as your git repo (this can be empty for now).**
  - **https://github.com/Get-It-Done-403**

- **List communication channels/tools with corresponding use policies (e.g., a slack channel)**
  - Slack (Main)
    - General Announcements and questions
  - Text (Secondary)
    - General Announcements and questions
  - Slack Huddle
    - Project meetings / standups and project progress check ins

Product description

A productivity web application that allows users to plan, track assignment progress, and view their calendar. By inputting an assignment's name, due date, and approximate time commitment, *Get it Done*, allocates hours into a user's schedule to work on an assignment based on previously user-declared availability. *Get it Done* helps users be more productive and beat procrastination by breaking down long working sessions into more manageable ones while making sure that assignments are completed in a timely manner.

Features

Major features:

- Login system
- Ability to reschedule/edits time slots
- Ability to view calendar and calendar events
- Project progress tracker

Stretch goals:

- Notification system
- Ability to import to personal calendar
- Works on any platform and syncs data between them.
- Syncs data between teammates

Use Cases (Functional Requirements)

For User search:

ACTORS: User

TRIGGER: User wants to search and find a specific task

PRECONDITION: User has already inputted tasks and is at the tasks page

POSTCONDITION: User successfully finds the task they were looking for and are able to edit and view them now

LIST OF STEPS:

1. User clicks the search button
2. User inputs matching title / substrings of the title that they're looking for
3. All matching tasks are displayed to the user
4. User is able to pick the task they were looking for and view and edit it now

EXTENSIONS/VARIATIONS:

- User can also use filters to narrow their searches
- User is able to search for subtasks within a task
- User can completely clear their current search with a button
- Displays users search history (with options to remove or clear it)

EXCEPTIONS:

- The specified search does not exist
    - Show result of nothing found
- Unable to connect to database
    - Suggest to re-login or check WiFi connection

For the log in system:

Actors: User of the app

Triggers: User is at the login page and initiates the log in process

Preconditions: User has an account with the app

Postconditions: User is successfully logged in and able to access their calendar and add assignments to their schedule

List of steps:

1. The user enters their username and password.
2. System verifies the credentials and validates.
3. User is granted access to the dashboard page

Extensions/variations:

- Remember me feature
- forgot password feature

Exceptions:

- Invalid login credentials
  - Return to login page and try again
- locked account due to too many failed login attempts
  - Send request to reset password
- network connection error
  - Show error and tell User to restart app
- Account doesn't exist
  - Send user to create account i

For the ability to enter new task:

ACTORS: User

TRIGGER: User
wants to schedule a new task

PRECONDITION: User has availability on the calendar page
POSTCONDITION: User
successfully enters new task and can view it on the calendar
page.

LIST OF STEPS:

1. User clicks new task button
2. User inputs tasks information such as title, due date, and approximate time commitment.
3. User saves the new task and can view it on the calendar page

EXTENSIONS/VARIATIONS:

- User can set a task to repeat daily weekly or monthly

EXCEPTIONS:

- Scheduling conflicts (doing two tasks at the same time)
  - Warn user no schedule match was found. No time slots available.
- Unable to connect to database
  - Suggest to refresh page, re-login or check WiFi connection
- User is not logged in
  - Suggest the user log in to view their tasks

For the ability to reschedule a time slot:

ACTORS: User

TRIGGER: User
wants to reschedule a task

PRECONDITION: User has
existing tasks and is already on the calendar page

POSTCONDITION: User
successfully reschedules their task and can view their changes on the calendar
page.

LIST OF STEPS:

1. User clicks task
2. User clicks the tasks edit button
3. User inputs desired date and time for task
4. User saves the edits to the task and can view their changes on the calendar page

EXTENSIONS/VARIATIONS:

- User can click and hold a task and drag it to their desired date
- User can edit a task to repeat daily weekly or monthly

EXCEPTIONS:

- Scheduling conflicts (doing two tasks at the same time)
  - Warn the user they are creating a scheduling conflict and double check they wish to reschedule a task to that time
- Unable to connect to database
  - Suggest to re-login or check WiFi connection
- User is not logged in
  - Suggest the user log in to view their tasks

For user project timeline:

Actors: User

Trigger: User wants to see timeline of when they need to work on it

Preconditions: user has submitted tasks and is at task page

Postcondition: User is able to successfully see a timeline schedule to finish the project

List of steps:

1. User clicks on task they want to make a schedule for
2. User selects times that work for them
3. Algorithm figures out the best plan to space out the work
4. User can see and edit the created schedule

Extensions/Variations:

- User can change the schedule the algorithm gave
- User is able to make updates to how long they need or what it requires
- Users can click and focus on the project to see important details

Exceptions:

- Unable to connect to database
    - Suggest to re-login or check WIFI connection
- Algorithm fails and cannot correctly show the project plan
    - Show error not found or re-login or check wifi

For user calendar view:

Actors: User

Trigger: User wants to see their calendar and all schedules tasks

Preconditions: user has submitted tasks and/or has available time slots

Postcondition: User is able to successfully view their calendar on dashboard

List of steps:

1. User goes to dashboard page
2. User can see calendar (scheduled tasks and free time slots)

Extensions/Variations:

- User can drag and drop any task into a new day and time.
- User can click and focus on the project to see important details

Exceptions:

- Unable to connect to database
    - Suggest to refreshing page, re-login or check WIFI connection
- User is not logged in
    - Suggest the user log in to view their tasks

Non-functional Requirements

1. Add events to google calendar as opposed to just web app
    a. The app is web based but users could still access their schedule from their phones
2. Scalability in app traffic (database and servers)
    a. How to handle and increase our capabilities during large volume traffic times
3. Log-in (security requirements)
    a. Encryption for personal data (user's schedule)
    b. Manage log in credentials
    c. 2FA

External Requirements

In addition to the requirements stated above, the course staff imposes the following requirements on your product:

- Our product must be robust against errors that can reasonably be expected to occur, such as invalid user input.
- The server must have a public URL accessible to users.
    - We plan on deploying our website URL to the public using services such as Vercel when the project reaches that point.
- We will need to provide instructions for someone else setting up a new server. Our system should be well documented to enable new developers to make enhancements.
- The scope of the project must match the resources (number of team members) assigned.

Team process description

Specify and justify the software toolset you will use.

- Reactjs for frontend and framework
    - We are all familiar with React and it works well with building web applications
- Java for backend and Spark
    - We have the most experience with Java and Spark in creating the backend and server side
- Google Calendar API
    - Google's calendar api is the most widely used calendar and well documented
- Firebase for database
    - We have the most experience with firebase and wanted a relational database and it also has a lot of features already that we could use such as login authentication and two step verification
- Figma to design the UI
    - Currently one of the most popular UI design tools and the one we have the most experience with
- IntelliJ IDE for coding
    - The one that we're all already currently using
- TailwindCSS for css
    - Helps with css and keeps it organized and simpler.
- React router

Team Structure

- Aidan
    - Backend / Firebase
    - High knowledge of java and data structures
    - Knowledge of firebase in mobile apps and web applications
    - Some experience with API integration
    - Knowledge of Python and ReactJS
- Mitchell
    - Frontend / Backend
    - Lots of experience with Reactjs and javascript
    - Some experience with databases (mongodb) and API integration
- Sofia
    - Backend
    - Knowledge of Java and data structures
    - Knowledge of of Python and React JS/TS
    - Experience with API integration
- Lance
    - Frontend / UI Design
    - Experience with frontend and react
    - Has made a functional website
    - Has taken classes focused on UI design

- Tye
    - Backend
    - Experience in building APIs and API integration
    - Knowledge of Java, Python, and SQL

Risk Assessment

- Technical Risks
    - Meeting performance requirements
    - Coming up with the scheduling algorithm could be challenging
        - Medium
- Resources
    - Learning firebase and how to use it with Java could be challenging.
        - Low
- Time management
    - We might not be able to finish all of the implementation in time
        - Low
- UI Design
    - Coming up with a UI design that is scalable and able to change with our goals would be the most challenging creative part
        - Low
- Modularity and Adaptability
    - Likelihood of occuring
        - Medium
    - Impact if it occurs
        - Medium
    - Evidence
        - Since our designs and code are still in the development stage, we are still constantly implementing things and then changing things which have had different difficulties depending on how modular the certain part of code is.
    - Plans for detecting the problem
        - We can see this problem whenever we decide to change things or add more designs.
    - Mitigation plan
        - We can make it easier for ourselves in the future by identifying what spots could probably be changed and make it modular so that implementing the change is easier and won't require changing everything else as well.

Project Schedule

FRONTEND

- ~~Designing the UI (1/23)~~
- Implement the UI Design (2/3)
- Connect with backend (2/10)
- Usability testing/review (2/12)

Currently, we have the majority of the UI designed already and some of it implemented already, as these two tasks are dependent on each other. Because they're dependent on each other both of these tasks have to be completed in parts where once parts of the frontend are done, it can be passed off to the backend part and connected to the database and other parts. Once parts start getting connected with the backend we can start testing those singular parts and find room for improvements.

BACKEND

- ~~Design how data is stored to send to database (1/23)~~
- Connect to database (1/27)
    - ~~Build Firebase structure~~
    - Connect with frontend user data
    - Transmit data between backend and frontend
- Scheduling Algorithm (2/3)

Right now we have the Task class done and we have the structure of the database setup. We haven't yet connected the front and backend as we are at that point as of this update. The Database file is being built currently. Spring boot has been installed and started to be implemented. Once the frontend and backend is connected we can start sorting out any issues that may arise.

GENERAL

- QA testing (2/8)
- Host on server and release (2/12)

Test Plan & Bugs

- We will seek external feedback after a basic frontend version has been implemented, and the backend has some basic functionality enough to perform usability testing and UI testing. As both the frontend and backend get refined, we will continue to seek external feedback. This will allow us to make changes early in the process, and continue to iterate on those changes.
- We plan on testing
- Write a basic testing suite (Basic security and functionality testing)

Documentation Plan

- Our plan for developing documentation with our system is having help pages within our website as well as having a tutorial video showcasing all of the features and how to use them for our users.
- For Admins, we plan on having thorough documentation as well as man pages, clearly showing what everything does and what their purposes are and how they can be used.

Testing and Continuous Integration

Frontend:

- Test-automation infrastructure: **Jest**
    - For our rendering testing for our frontend code we will be using Jest. We are using Jest because it is already installed by default with our react app, and is able to do rendering testing to make sure everything is visually correct. Testing with jest, will make sure everything is rendered correctly when called and clicked on without exposing any of the backend code. To add a new test to the code base, depending on the page that you are testing you enter the file "(pagename).test.js" and use test() and expect().

Backend testing:

- Test-automation infrastructure: **JUnit**
    - To enable unit tests of all of our Java classes as units and isolate the functionality of each component that has dependencies. We will use Mockito to mock such dependencies
- Backend server and database testing: **Postman**
    - Postman allows us to test our server and database by connecting to the spring boot server at its localhost and then allows us to send commands through the server such as Get, Post, Put, Patch, Delete and more.

Continuous integration of the project:

- **GitHub Actions**: we chose to use GitHub Action because it is a low maintenance CI tool that would allow us to develop our project in the same place where our code lives (GitHub).
- Note for developers: See Managing issues and pull requests guide for details

A pros/cons matrix for at least three CI services that you considered.

- GitHub Actions
    - Pros
        - Offers a free plan
        - Low maintenance CI tool
        - Convenient as our code lives in GitHub already
        - Analytics/Status Overview
    - Cons
        - Limited configuration options compared to other CI tools
        - Not as feature-rich as other CI tools
        - Limited community support for troubleshooting and solution sharing
- Jenkins
    - Pros
        - Highly customizable

        - Extensive community support for troubleshooting
        - Supports many different types of projects and build tools
    - Cons
        - High maintenance and requires a dedicated server or instance
        - Complex setup and configuration process
        - Steep learning curve for beginners
- CircleCI
    - Pros
        - User-friendly interface and setup process
        - Integrates well with multiple platforms and tools
        - Multiple pre-built workflows and examples available
    - Cons
        - Expensive pricing model
        - Limited configuration options
        - Lacks some of the advanced features seen in other CI services

Which tests will be executed in a CI build.

- Unit tests
- Integration tests
- React tests

Which development actions trigger a CI build.

- On push/merge

Software Architecture

The main software components of our project are our request handler, service handler, and database handler. Our system would get a request from the client - in this case our frontend- and such request will be handled by the request handler. Then our service handler will route the request to the correct service depending on the request type. We also will have a database handler to take care of any read or write request to the database. Finally, once the request has been processed we return a JSON response to our client (frontend).

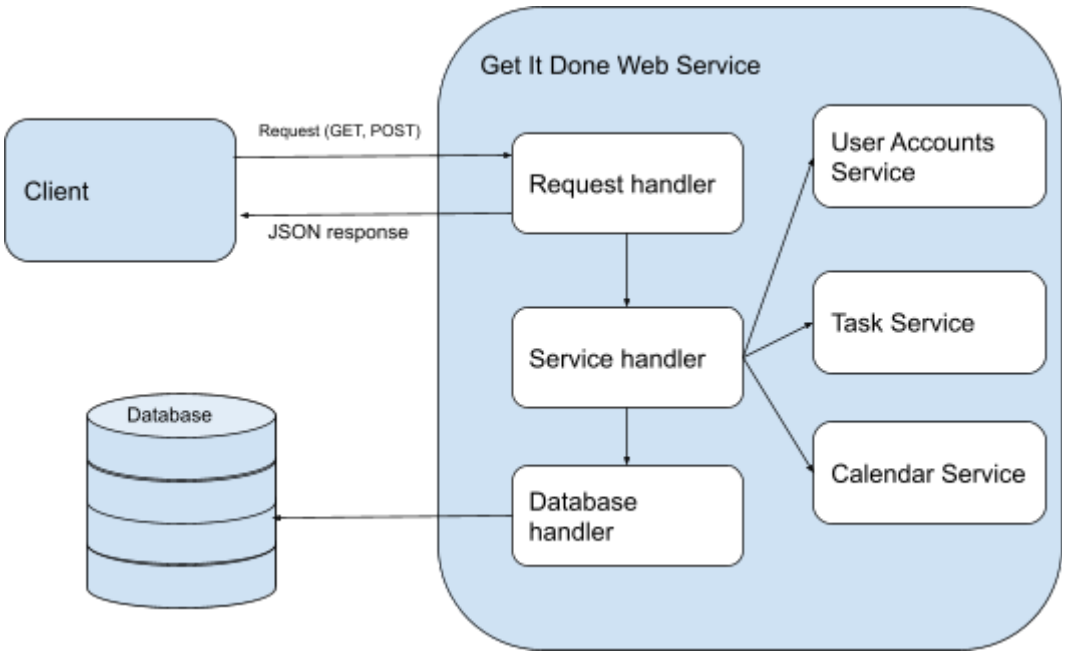
Service handler

Task Service
- Adds, removes, and updates tasks

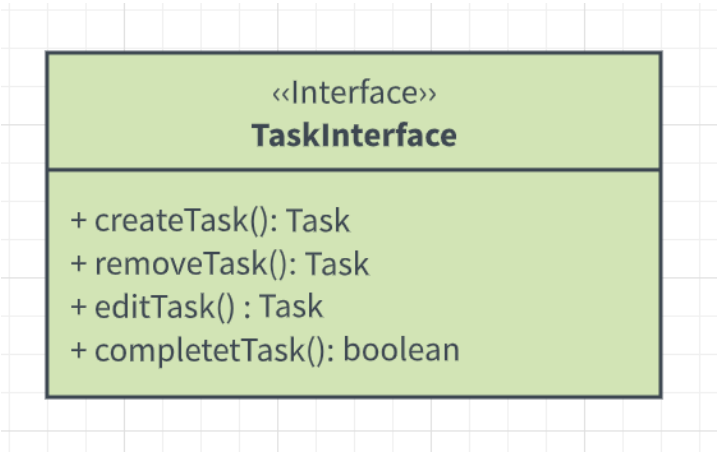
Calendar Service
- Allocates time for each task and adds to the calendar

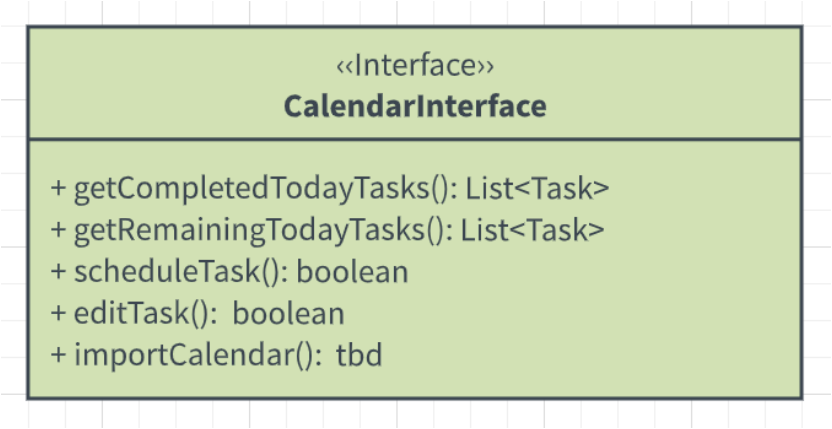
User Service
- Manages account settings for users

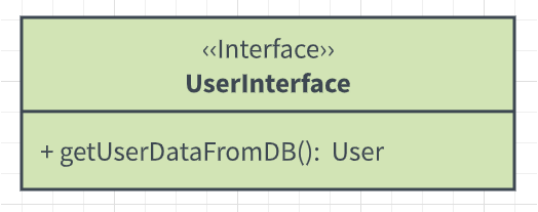
All services will communicate with the database to store or retrieve data.



Interfaces







Database Schema

- Users
    - Calendar
        - Tasks
            - Due Date
                - Date
                - Time
            - Estimated time to complete
            - IsCompleted

            - RepeatingTask
            - Description
        - Upcoming Tasks
        - Google Calendar
    - User Account Data
        - Email
        - Password
        -

Assumptions underpinning the architecture

- Assuming that the users are familiar with using simple websites
    - Users can navigate through the website
    - Users are familiar with the calendar interface on the website
- Assuming that the architecture can handle increasing traffic and data storage demands in the future.
- Assuming that Firebase's security features will protect sensitive user data.
- Assuming that React js and Spring boot will work well on different browsers and devices.

Design Alternatives

Alternatives for ReactJs

AngularJs

Angular is a complete framework with features for handling data binding, data binding, and routing. Uses a component based architecture.

Pros of using AnglualrJs over ReactJs

- Two way data binding: automatic updates to the UI when the data changes. Makes it easier to maange complex applications.
- Built in testing tools: easier to debug the application.

Cons of using AngularJs over ReactJs

- Steep learning curve: it is more complex than react js
- Performance: it is slower than ReactJs

Firebase vs MongoDB

- Mongo/Mongoose is a non-sql database ODM library for mongodb and Node.js. It synergizes very well with Node.js making it very easy to create, and update your databases
- Firebase is a cloud-hosted NoSQL database. It is easy to use and google provides many tools such as an Authentication feature, statistics, analytics, different ways of storage and much flexibility.

Software Design

Front End -
app.js
models.js (For database stuff?)
Controllers

- calendar.js
- profile.js
- home.js
- settings.js

Back End -

Task

- Task.java
    - Store information about a single Task
- TaskController.java
    - Handles HTTP requests from the client
    - Endpoints for add, remove, and update.
- TaskService.java
    - Responsible for creating, deleting, and updating a task from the database

Calendar

- Calendar.java
    - Stores the current view of the calendar
- CalendarController.java
    - Endpoints to edit the calendar
- CalendarService.java
    - Responsible for retrieving today's task, this week's task.
    - Responsible for updating the calendar.
    - Responsible for allocating time for a task in the calendar.

User

- User.java
    - Contains the user account information
- UserController.java
    - Handles HTTP requests to update user settings
    - Endpoints to add, remove, edit, user information

- UserService.java
    - Responsible for adding, removing, and updating user account information.

Coding Guidelines

1. **Java Guidelines**
   We chose these guidelines because we are all for the most part familiar with the style as well as these are well documented guidelines. We will enforce these guidelines through code reviews and checking in when we see something that goes against the provided guidelines and holding each other accountable.
   a. https://google.github.io/styleguide/javaguide.html
   b. Source files should consist of case-sensitive name of the top level class it contains, plus the .java extension
   c. Column limit of 100 characters
      i. Exceptions
         1. Lines where obeying the column limit is not possible (for example, a long URL in Javadoc, or a long JSNI method reference).
         2. Package or import statements
         3. Command lines in a comment that may be copy and pasted into a shell
         4. Very long identifiers
   d. Indentation of at least +4 from the original line
   e. Horizontal alignment: Never required
   f. Meaningful comments that are well written and easy to understand for everyone reading
      i. May be in: /* … */ or // ….
   g. Camel Casing
2. **JavaScript Guidelines**
   We chose these guidelines because this has the most similar style to the java guidelines which we are already familiar with. Also, choosing a similar style guide will allow for consistency within our entire project and make it easier to read. We will enforce these guidelines through code reviews and checking in when we see something that goes against the provided guidelines and holding each other accountable.
   a. https://gist.github.com/antonrogov/1216380/21800f463af3d3a98e98083c4bc109e44f981ef4
   b. JavaScript files should stay as Javascript files and not HTML
   c. Indentation will be one tab
   d. Line length should be smaller than 80 and if breaks are needed,
      i. Break after an operator or after a comma and should be indented two tabs
   e. Make meaningful well-written comments, and keep up to date
   f. Declare variables before use
   g. Declare functions before use
      i. No space between function and parameters ex. Function(x, y) { }