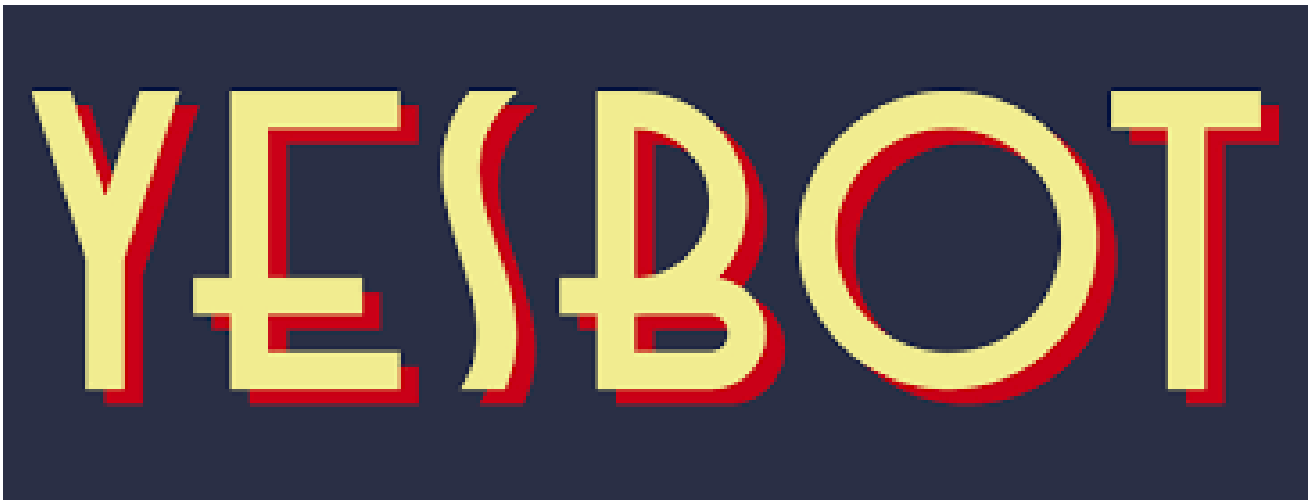
Report for Assignment 1

Hugo Berg - 2779175
Mark Duijn - 2778293
Tomas Karosas - 2552044



Name: YesBot

URL: https://github.com/Yes-Theory-Fam/yesbot-ts
URL to our forked repository: https://github.com/HugoVU/yesbot-ts

Number of lines of code and the tool used to count it: 13.4k lines, counted with lizard

Programming language: TypeScript

We used Vitest to measure the coverage of the project, with v8 as provider. The project already included Vitest for unit testing purposes. All we had to do was add the flag coverage in package.json and we get a nice report.

All files

22.92% Statements 3454/15066	39.22% Branches 111/283	12.75% Functions 31/243	22.92% Lines 3454/15066

The entire project had an initial coverage of 39.22%. A more detailed overview of the analysis can be found on the github, on the assignment1 branch.

Own coverage tool

Hugo

ensureGuildMemberOrNull
For my first test I picked the file helper.ts in src/event-distribution/helper.ts, and wanted to add the test case for this particular function:

```
export const ensureGuildMemberOrNull = (
  member: GuildMember | APIGuildMember | null,
  client: Client,
  guild: Guild | null
): GuildMember | null => {
  if (!member) {
    ensureGuildMemberOrNullCoverage[0] = 1;
    return null;
  }

  if (member instanceof GuildMember) {
    ensureGuildMemberOrNullCoverage[1] = 1;
    return member;
  }

  if (!guild) {
    ensureGuildMemberOrNullCoverage[2] = 1;
    throw new Error(
      "Could not instantiate GuildMember from raw data; missing guild from
button interaction"
    );
  }
```

The commit with the branch coverage tracking can be found here: https://github.com/HugoVU/yesbot-ts/commit/683a2733362a3a5aea08ebbf51e98be1736512ba

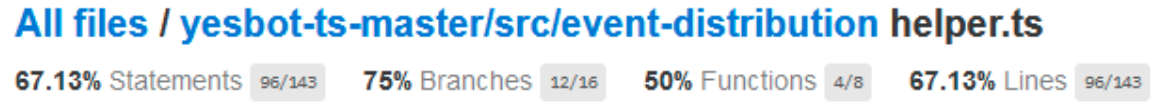
This function had no unit tests at all, so we got a branch coverage of 0% for this particular function:

```
stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns
 null if member is null
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns
 the member if it is a GuildMember
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Throws
an error if the member is not a GuildMember and guild is null
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns
 a GuildMember if the member is not a GuildMember and guild is not null
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%
```

This can be verified with the Vitest tool:

```
export const ensureGuildMemberOrNull = (
  member: GuildMember | APIGuildMember | null,
  client: Client,
  guild: Guild | null
): GuildMember | null => {
  if (!member) return null;

  if (member instanceof GuildMember) {
    return member;
  }

  if (!guild) {
    throw new Error(
      "Could not instantiate GuildMember from raw data; missing guild from button interaction"
    );
  }

  return Reflect.construct(GuildMember, [client, member, guild]) as GuildMember;
};
```

Red means it is not covered, and clearly no unit tests are implemented yet.
4 out of 8 functions in the entire helper.ts file were covered, and 75% of the branches.



resolveEmojis

I added tests for this function, found in src/programs/polls.ts:

```
export const resolveEmojis = (lines: string[], bot: Client): EmojiResolvable[]
=> {
  const emojiEmojis = getEmojis(lines, bot);

  if (emojiEmojis && emojiEmojis.length > 0) {
    resolveEmojisCoverage[0] = 1;
    return emojiEmojis;
  }
```

```
  const letterEmojis = getLetterEmojis(lines);
  if (letterEmojis && letterEmojis.length > 0) {
    resolveEmojisCoverage[1] = 1;
    return letterEmojis;
  }

  resolveEmojisCoverage[2] = 1;
  return ["A", "B"].map(letterToEmoji);
};
```

The commit with the branch coverage tracking can be found here: https://github.com/HugoVU/yesbot-ts/commit/683a2733362a3a5aea08ebbf51e98be1736512ba (Same link as previous, it is one commit)

This function once again had no unit tests at all, so we got a branch coverage of 0%:

```
stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array of e
mojis if they are present
Branch coverage on resolveEmojis after test run: 0.00%

stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array Emoj
i letters of the letters
Branch coverage on resolveEmojis after test run: 0.00%

stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array of d
efault emojis if no emojis are present
Branch coverage on resolveEmojis after test run: 0.00%
```

Verified with the Vitest tool:

```
const resolveEmojis = (lines: string[], bot: Client): EmojiResolvable[] => {
  const emojiEmojis = getEmojis(lines, bot);

  if (emojiEmojis && emojiEmojis.length > 0) {
    return emojiEmojis;
  }

  const letterEmojis = getLetterEmojis(lines);
  if (letterEmojis && letterEmojis.length > 0) {
    return letterEmojis;
  }

  return ["A", "B"].map(letterToEmoji);
};
```

The polls.ts file has 0% coverage in its entirety, including my function:

All files / yesbot-ts-master/src/programs polls.ts

0% Statements 0/139 **0%** Branches 0/1 **0%** Functions 0/1 **0%** Lines 0/139

Mark

addEventHandler & extractEventHandler

Both the coverage tracking was added in the same patch diff, therefore I will squash these 2 functions into the same section.

Github diff: link

After every test it will output the coverage that was reached for both functions.

```
stdout | __tests__/event-distribution/events/events.spec.ts > EventDistribution events >
should call addMemberLeaveHandler on DiscordEvent.MEMBER_LEAVE
Branch coverage on addEventHandler after test run: 7.69%
Branch coverage on extractEventInfo after test run: 0.00%
```

After all tests are ran:

```
stdout | __tests__/event-distribution/events/events.spec.ts > EventDistribution events >
should throw an error no event is provided
Branch coverage on addEventHandler after test run: 100%
Branch coverage on extractEventInfo after test run: 100%
```

Tomas

Function 1: MetaCommand (src\programs\meta.ts):

Code snippet:

```
class MetaCommand implements CommandHandler<DiscordEvent.CONTEXT_MENU_MESSAGE> {
  async handle(command: MessageContextMenuCommandInteraction): Promise<void> {
    const message = command.targetMessage;

    if (command.user === message.author) {
      throw new Error(MetaErrors.SELF_META);
    }

    const emojiByName = (name: string) =>
      message.guild?.emojis.cache.find((e) => e.name === name);
    const metaEmoji = emojiByName(metaEmojiName);

    const didBotReact = message.reactions.cache.some((reaction) => reaction.me);
    if (didBotReact) {
      throw new Error(MetaErrors.ALREADY_METAED);
    }

    await message.reply({
      content:
        "https://user-images.githubusercontent.com/17064122/122255708-ae6e9680-c
    });
    await message.react(metaEmoji ?? "🤚");

    await command.reply({ ephemeral: true, content: "Done!" });
  }
}
```

This function had 0% coverage as seen in the code snippet above and can be confirmed by built in Vitest v8 output:

```
-----------------------------------------------------------|---------|----------|---------|---------|-------------------
File                                                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------------------------------------------------|---------|----------|---------|---------|-------------------
  meta.ts                                                  |       0 |        0 |       0 |       0 | 1-49
```

The following commit on a forked repository includes the instrumented code to gather coverage measurements by tracking the branches hit while running test (and prints the output to the console) as well as the actual code (meta.spec.ts file) of the unit tests added to cover the function:
https://github.com/Yes-Theory-Fam/yesbot-ts/commit/c33ed45c14ea337336597cb10a75c7cbace3506d

Function 2: FeatureRequestPending (src\programs\feature-request-pending.ts)
Code snippet:

```
class FeatureRequestPending
  implements CommandHandler<DiscordEvent.THREAD_CREATE>
{
  async handle(channel: ThreadChannel, _unused: boolean): Promise<void> {
    const parent = channel.parent;
    if (!parent || parent.type !== ChannelType.GuildForum) return;

    const pendingTag = parent.availableTags.find(
      (t) => t.name.toLowerCase() === "pending"
    );
    if (!pendingTag) return;

    await channel.setAppliedTags([...channel.appliedTags, pendingTag.id]);
  }
}
```

This function had 0% coverage as well as seen in the code snippet above and can be confirmed by built in Vitest v8 output:

```
-------------------------------------------------------|---------|----------|---------|---------|-------------------
File                                                   | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------------------------------------|---------|----------|---------|---------|-------------------
 feature-request-pending.ts                            |       0 |        0 |       0 |       0 | 1-28
```

The following commit on a forked repository includes the instrumented code to gather coverage measurements by tracking the branches hit while running test (and prints the output to the console) as well as the actual code (meta.spec.ts file) of the unit tests added to cover the function:

https://github.com/Yes-Theory-Fam/yesbot-ts/commit/959fdb55a6c4526c87f052397a5e4a33d3fe3afe

Sebastian

Function 1: Resources

```
export const resourcesCoverage = [0,0,0]; Show usages  s-baal

@Command( options: { Show usages  Mohamed Sami +2
  event: DiscordEvent.SLASH_COMMAND,
  root: "resources",
  description: "Get the resources associated with the channel.",
})
export class Resources implements CommandHandler<DiscordEvent.SLASH_COMMAND> {
  async handle(interaction: ChatInputCommandInteraction): Promise<void> { Show usages  Mohamed Sami +2
    const channel = interaction.channel as TextChannel;

    switch (channel.name) {
      case ChatNames.CODING:
        await interaction.reply(RESOURCES_CODING);
        resourcesCoverage[0] = 1;
        break;

      case ChatNames.LEARNING_SPANISH:
        await interaction.reply(RESOURCES_SPANISH);
        resourcesCoverage[1] = 1;
        break;

      default:
        await interaction.reply( options: {
          content: "No resource exists for this channel.",
          ephemeral: true,
        });
        resourcesCoverage[2] = 1;
        break;
```

```
  afterEach( fn: () => {
      const resourceCoverageSum = resourcesCoverage.reduce((sum, x) => sum + x);
      console.log("Branch coverage on resource after test run: " +
          ((resourceCoverageSum / (resourcesCoverage.length) * 100).toPrecision( precision: 3) + "%"));
```

Results:

```
stdout | __tests__/programs/resources.spec.ts > resources.handle > reply with coding resources in the CODING cha
nnel
Branch coverage on resource after test run: 0.00%

stdout | __tests__/programs/resources.spec.ts > resources.handle > reply with Spanish resources in the LEARNING_
SPANISH channel
Branch coverage on resource after test run: 0.00%

stdout | __tests__/programs/resources.spec.ts > resources.handle > reply with default message in an unknown chan
nel
Branch coverage on resource after test run: 0.00%
```

Function 2: custom-message

```
export const customMessageMethodsCoverage = [0,0,0,0]  Show usages  s-baal

@Command( options: {  Show usages  s-baal +2 *
  event: DiscordEvent.MESSAGE,
  description:
    "This handler is for custom messages that do not have a specific trigger.",
})
export class CustomMessageMethods implements CommandHandler<DiscordEvent.MESSAGE> {
  async handle(message: Message): Promise<void> {  Show usages  s-baal +2 *
    const messageContent = message.content;

    if (messageContent.match( matcher: /^(yesbot).*(\?)$/gi)){
      await randomReply(message);
      customMessageMethodsCoverage[0] = 1;
    }

    if (messageContent.match( matcher: /yesbot i love you|yesbot i love u|i love you yesbot/i)){
      await sendLove(message);
      customMessageMethodsCoverage[1] = 1;
    }

    if (messageContent.match( matcher: /^F$/i)){
      await message.react( emoji: "🇫 ");
      customMessageMethodsCoverage[2] = 1;
    }

    if (messageContent.match( matcher: /(abooz|mod abuse)/i)){
      await message.react( emoji: "👀 ");
      customMessageMethodsCoverage[3] = 1;
```

```
  afterEach( fn: () => {
      const CoverageSum = customMessageMethodsCoverage.reduce((sum, x) => sum + x);
      console.log("Branch coverage on custom-message after test run: " +
          ((CoverageSum / (customMessageMethodsCoverage.length) * 100).toPrecision( precision: 4) + "%"));
```

Results:

```
stdout | __tests__/programs/custom-messages.spec.ts > CustomMessageMethods > reply with a random message when co
ntent matches yesbot and ends with a ?
Branch coverage on custom-message after test run: 0.000%

stdout | __tests__/programs/custom-messages.spec.ts > CustomMessageMethods > send love reply and react with hear
t when content matches love expressions
Branch coverage on custom-message after test run: 0.000%

stdout | __tests__/programs/custom-messages.spec.ts > CustomMessageMethods > react with 🇫 "F"
Branch coverage on custom-message after test run: 0.000%

stdout | __tests__/programs/custom-messages.spec.ts > CustomMessageMethods > react with 👀 when content matches
mod abuse phrases
Branch coverage on custom-message after test run: 0.000%

stdout | __tests__/programs/custom-messages.spec.ts > CustomMessageMethods > should not trigger any actions for
unrelated messages
Branch coverage on custom-message after test run: 0.000%
```

Improvement individual tests

Hugo

Test 1

I added my test in helper.spec.ts. The code can be found here: https://github.com/HugoVU/yesbot-ts/commit/024355bbfe8c517b56eb188cc54d8b2778ab9b86

As mentioned earlier, the function had 0% coverage. After adding all 4 branches we have 100% coverage for this specific function:

```
107  1x  export const ensureGuildMemberOrNull = (
108  4x    member: GuildMember | APIGuildMember | null,
109  4x    client: Client,
110  4x    guild: Guild | null
111  4x  ): GuildMember | null => {
112  4x    if (!member) return null;
113  3x
114  4x    if (member instanceof GuildMember) {
115  1x      return member;
116  1x    }
117  2x
118  4x    if (!guild) {
119  1x      throw new Error(
120  1x        "Could not instantiate GuildMember from raw data; missing guild from button interaction"
121  1x      );
122  1x    }
123  1x
124  1x    return Reflect.construct(GuildMember, [client, member, guild]) as GuildMember;
125  1x  };
```

Compared to earlier:

```
export const ensureGuildMemberOrNull = (
  member: GuildMember | APIGuildMember | null,
  client: Client,
  guild: Guild | null
): GuildMember | null => {
  if (!member) return null;

  if (member instanceof GuildMember) {
    return member;
  }

  if (!guild) {
    throw new Error(
      "Could not instantiate GuildMember from raw data; missing guild from button interaction"
    );
  }

  return Reflect.construct(GuildMember, [client, member, guild]) as GuildMember;
};
```

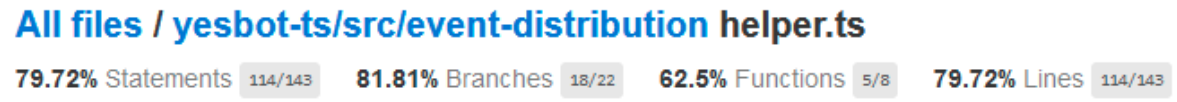
Measured with own branch tracking:

```
stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns null if member is null
Branch coverage on ensureGuildMemberOrNull after test run: 25.0%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns the member if it is a GuildMember
Branch coverage on ensureGuildMemberOrNull after test run: 50.0%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Throws an error if the member is not a GuildMem
ber and guild is null
Branch coverage on ensureGuildMemberOrNull after test run: 75.0%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns a GuildMember if the member is not a Gu
ildMember and guild is not null
Branch coverage on ensureGuildMemberOrNull after test run: 100%
```

Compared to earlier:

```
stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns
 null if member is null
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns
 the member if it is a GuildMember
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Throws
an error if the member is not a GuildMember and guild is null
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%

stdout | __tests__/event-distribution/helper.spec.ts > ensureGuildMemberOrNull > Returns
 a GuildMember if the member is not a GuildMember and guild is not null
Branch coverage on ensureGuildMemberOrNull after test run: 0.00%
```

And as a result we now cover 5 out of 8 functions with a branch coverage of 81,81% for the entire file:



Since there was no unit test for this function at all, the coverage has been significantly improved. A branch coverage of 81.81% has been achieved for the entire file, which is a solid improvement over the previous 75%.

Test 2

I added my unit tests in a new folder called programs. The code can be found here: https://github.com/HugoVU/yesbot-ts/commit/9a2d3bff7d690fa627d849ff6e157d6dec114bde

After my unit tests we cover 100% of the branches of the function:

```ts
const letterToEmoji = (letter: string) => {
  const unicodeOffset = 0x1f1e6; //Regional Indicator A
  const asciiOffset = "A".charCodeAt(0);

  if (letter === "B") return "🅱️";
  const letterIndex = letter.charCodeAt(0) - asciiOffset;
  const unicodeCodePoint = unicodeOffset + letterIndex;
  return String.fromCodePoint(unicodeCodePoint);
};

// Resolves single letters at the start of a line and returns their unicode version
const getLetterEmojis = (lines: string[]): string[] => {
  return lines
    .map((line) => line.toUpperCase().split(/\b/)[0])
    .filter((firstWord) => firstWord.match(/^[A-Z]$/))
    .map(letterToEmoji);
};

export const resolveEmojis = (lines: string[], bot: Client): EmojiResolvable[] => {
  const emojiEmojis = getEmojis(lines, bot);

  if (emojiEmojis && emojiEmojis.length > 0) {
    return emojiEmojis;
  }

  const letterEmojis = getLetterEmojis(lines);
  if (letterEmojis && letterEmojis.length > 0) {
    return letterEmojis;
  }

  return ["A", "B"].map(letterToEmoji);
};
```

Compared to earlier:

```ts
const letterToEmoji = (letter: string) => {
  const unicodeOffset = 0x1f1e6; //Regional Indicator A
  const asciiOffset = "A".charCodeAt(0);

  if (letter === "B") return "🅱";
  const letterIndex = letter.charCodeAt(0) - asciiOffset;
  const unicodeCodePoint = unicodeOffset + letterIndex;
  return String.fromCodePoint(unicodeCodePoint);
};

// Resolves single letters at the start of a line and returns their unicode version
const getLetterEmojis = (lines: string[]): string[] => {
  return lines
    .map((line) => line.toUpperCase().split(/\b/)[0])
    .filter((firstWord) => firstWord.match(/^[A-Z]$/))
    .map(letterToEmoji);
};

const resolveEmojis = (lines: string[], bot: Client): EmojiResolvable[] => {
  const emojiEmojis = getEmojis(lines, bot);

  if (emojiEmojis && emojiEmojis.length > 0) {
    return emojiEmojis;
  }

  const letterEmojis = getLetterEmojis(lines);
  if (letterEmojis && letterEmojis.length > 0) {
    return letterEmojis;
  }

  return ["A", "B"].map(letterToEmoji);
};
```

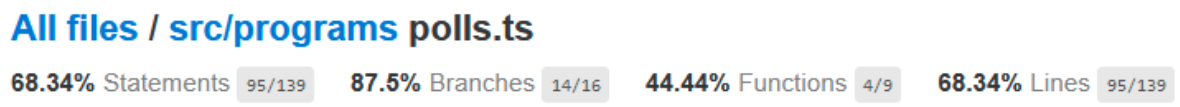
Measured with our own branch tracking:

```
stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array of emojis if they are present
Branch coverage on resolveEmojis after test run: 33.3%

stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array Emoji letters of the letters
Branch coverage on resolveEmojis after test run: 66.7%

stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array of default emojis if no emojis are present
Branch coverage on resolveEmojis after test run: 100%
```

Compared to earlier:

```
stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array of e
mojis if they are present
Branch coverage on resolveEmojis after test run: 0.00%

stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array Emoj
i letters of the letters
Branch coverage on resolveEmojis after test run: 0.00%

stdout | __tests__/programs/resolve-emojis.spec.ts > resolveEmojis > Returns an array of d
efault emojis if no emojis are present
Branch coverage on resolveEmojis after test run: 0.00%
```

After my implementations we have 87.5% branch coverage of the entire file and 44.44% of all functions are covered.



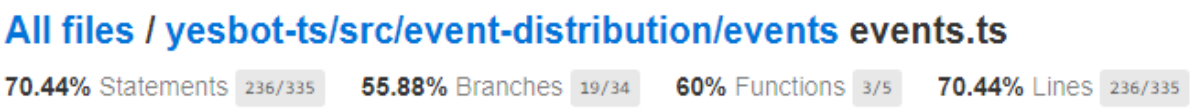
As you can see, resolveEmojis, getLetterEmojis, and letterToEmoji are all covered now by unit tests. Considering we had 0% coverage of the entire file, adding these tests ensured a higher branch coverage.

Mark

addEventHandler
The tests for this function are placed in events.spec.ts

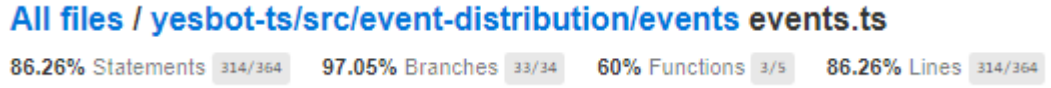
Github diffs for this function can be found:
Here, here and here

The original branch coverage can be seen here:



```ts
export const addEventHandler: AddEventHandlerFunction<EventHandlerOptions> = (
  options,
  ioc,
  tree
) => {
  switch (options.event) {
    case DiscordEvent.BUTTON_CLICKED:
      return addButtonClickedHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.BUTTON_CLICKED>
      );
    case DiscordEvent.CONTEXT_MENU_MESSAGE:
      return addContextMenuMessageHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.CONTEXT_MENU_MESSAGE>
      );
    case DiscordEvent.CONTEXT_MENU_USER:
      return addContextMenuUserHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.CONTEXT_MENU_USER>
      );
    case DiscordEvent.MEMBER_LEAVE:
      return addMemberLeaveHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.MEMBER_LEAVE>
      );
    case DiscordEvent.MEMBER_JOIN:
      return addMemberJoinHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.MEMBER_JOIN>
      );
    case DiscordEvent.MESSAGE:
      return addMessageHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.MESSAGE>
      );
    case DiscordEvent.REACTION_ADD:
    case DiscordEvent.REACTION_REMOVE:
      return addReactionHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<
          DiscordEvent.REACTION_ADD | DiscordEvent.REACTION_REMOVE
        >
      );
    case DiscordEvent.GUILD_MEMBER_UPDATE:
      return addGuildMemberUpdateHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.GUILD_MEMBER_UPDATE>
      );
    case DiscordEvent.READY:
      return addReadyHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.READY>
      );
    case DiscordEvent.SLASH_COMMAND:
      return addSlashCommandHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.SLASH_COMMAND>
      );
    case DiscordEvent.THREAD_CREATE:
      return addThreadCreateHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.THREAD_CREATE>
      );
    case DiscordEvent.TIMER:
      return addTimerHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.TIMER>
      );
    case DiscordEvent.VOICE_STATE_UPDATE:
      return addVoiceStateUpdateHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.VOICE_STATE_UPDATE>
      );
  }
};
```

The improved branch coverage can be seen here:



```ts
export const addEventHandler: AddEventHandlerFunction<EventHandlerOptions> = (
  options,
  ioc,
  tree
) => {
  switch (options.event) {
    case DiscordEvent.BUTTON_CLICKED:
      addEventHandlerCoverage[0] = 1;
      return addButtonClickedHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.BUTTON_CLICKED>
      );
    case DiscordEvent.CONTEXT_MENU_MESSAGE:
      addEventHandlerCoverage[1] = 1;
      return addContextMenuMessageHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.CONTEXT_MENU_MESSAGE>
      );
    case DiscordEvent.CONTEXT_MENU_USER:
      addEventHandlerCoverage[2] = 1;
      return addContextMenuUserHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.CONTEXT_MENU_USER>
      );
    case DiscordEvent.MEMBER_LEAVE:
      addEventHandlerCoverage[3] = 1;
      return addMemberLeaveHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.MEMBER_LEAVE>
      );
    case DiscordEvent.MEMBER_JOIN:
      addEventHandlerCoverage[4] = 1;
      return addMemberJoinHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.MEMBER_JOIN>
      );
    case DiscordEvent.MESSAGE:
      addEventHandlerCoverage[5] = 1;
      return addMessageHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.MESSAGE>
      );
    case DiscordEvent.REACTION_ADD:
    case DiscordEvent.REACTION_REMOVE:
      addEventHandlerCoverage[6] = 1;
      return addReactionHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<
          DiscordEvent.REACTION_ADD | DiscordEvent.REACTION_REMOVE
        >
      );
    case DiscordEvent.GUILD_MEMBER_UPDATE:
      addEventHandlerCoverage[7] = 1;
      return addGuildMemberUpdateHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.GUILD_MEMBER_UPDATE>
      );
    case DiscordEvent.READY:
      addEventHandlerCoverage[8] = 1;
      return addReadyHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.READY>
      );
    case DiscordEvent.SLASH_COMMAND:
      addEventHandlerCoverage[9] = 1;
      return addSlashCommandHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.SLASH_COMMAND>
      );
    case DiscordEvent.THREAD_CREATE:
      addEventHandlerCoverage[10] = 1;
      return addThreadCreateHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.THREAD_CREATE>
      );
    case DiscordEvent.TIMER:
      addEventHandlerCoverage[11] = 1;
      return addTimerHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.TIMER>
      );
    case DiscordEvent.VOICE_STATE_UPDATE:
      addEventHandlerCoverage[12] = 1;
      return addVoiceStateUpdateHandler(
        options,
        ioc,
        tree as StringIndexedHIOCTree<DiscordEvent.VOICE_STATE_UPDATE>
      );
  }
};
```

The branch coverage improved from 50% to 100% for this given function. This was done by adding test cases that reach the switch case statements that were previously left untested.

extractEventInfo
The tests for this function are in event.spec.ts

The GitHub diffs can be found: here

The original branch coverage can be seen here:

```
export const extractEventInfo: ExtractInfoFunction<DiscordEvent> = (
  event,
  ...args
) => {
  const getInfos = () => {
    switch (event) {
      case DiscordEvent.BUTTON_CLICKED:
        return extractButtonClickedInfo(args[0] as ButtonInteraction);
      case DiscordEvent.CONTEXT_MENU_MESSAGE:
        return extractContextMenuMessageInfo(
          args[0] as MessageContextMenuCommandInteraction
        );
      case DiscordEvent.CONTEXT_MENU_USER:
        return extractContextMenuUserInfo(
          args[0] as UserContextMenuCommandInteraction
        );
      case DiscordEvent.MEMBER_LEAVE:
        return extractMemberLeaveInfo(args[0] as MemberLeaveArgument);
      case DiscordEvent.MEMBER_JOIN:
        return extractMemberJoinInfo(args[0] as MemberJoinArgument);
      case DiscordEvent.MESSAGE:
        return extractMessageInfo(args[0] as Message);
      case DiscordEvent.REACTION_ADD:
      case DiscordEvent.REACTION_REMOVE:
        return extractReactionInfo(args[0] as MessageReaction, args[1] as User)
      case DiscordEvent.GUILD_MEMBER_UPDATE:
        return extractGuildMemberUpdateInfo(
          args[0] as GuildMemberUpdateArgument,
          args[1] as GuildMemberUpdateArgument
        );
      case DiscordEvent.READY:
        return extractReadyInfo(args[0] as Client);
      case DiscordEvent.THREAD_CREATE:
        return extractThreadCreateInfo(
          args[0] as ThreadChannel,
          args[1] as boolean
        );
      case DiscordEvent.TIMER:
        return extractTimerInfo(args[0] as Timer);
      case DiscordEvent.SLASH_COMMAND:
        return extractSlashCommandInfo(args[0] as ChatInputCommandInteraction);
      case DiscordEvent.VOICE_STATE_UPDATE:
        return extractVoiceStateUpdateInfo(
          args[0] as VoiceState,
          args[1] as VoiceState
        );
      default:
        throw new Error("Could not extract info for event " + event);
    }
  };
  const infos = getInfos();

  return Array.isArray(infos) ? infos : [infos];
};
```

The improved branch coverage can be seen here:

```
export const extractEventInfo: ExtractInfoFunction<DiscordEvent> = (
  event,
  ...args
) => {
  const getInfos = () => {
    switch (event) {
      case DiscordEvent.BUTTON_CLICKED:
        extractEventInfoCoverage[0] = 1;
        return extractButtonClickedInfo(args[0] as ButtonInteraction);
      case DiscordEvent.CONTEXT_MENU_MESSAGE:
        extractEventInfoCoverage[1] = 1;
        return extractContextMenuMessageInfo(
          args[0] as MessageContextMenuCommandInteraction
        );
      case DiscordEvent.CONTEXT_MENU_USER:
        extractEventInfoCoverage[2] = 1;
        return extractContextMenuUserInfo(
          args[0] as UserContextMenuCommandInteraction
        );
      case DiscordEvent.MEMBER_LEAVE:
        extractEventInfoCoverage[3] = 1;
        return extractMemberLeaveInfo(args[0] as MemberLeaveArgument);
      case DiscordEvent.MEMBER_JOIN:
        extractEventInfoCoverage[4] = 1;
        return extractMemberJoinInfo(args[0] as MemberJoinArgument);
      case DiscordEvent.MESSAGE:
        extractEventInfoCoverage[5] = 1;
        return extractMessageInfo(args[0] as Message);
      case DiscordEvent.REACTION_ADD:
      case DiscordEvent.REACTION_REMOVE:
        extractEventInfoCoverage[6] = 1;
        return extractReactionInfo(args[0] as MessageReaction, args[1] as Us
      case DiscordEvent.GUILD_MEMBER_UPDATE:
        extractEventInfoCoverage[7] = 1;
        return extractGuildMemberUpdateInfo(
          args[0] as GuildMemberUpdateArgument,
          args[1] as GuildMemberUpdateArgument
        );
      case DiscordEvent.READY:
        extractEventInfoCoverage[8] = 1;
        return extractReadyInfo(args[0] as Client);
      case DiscordEvent.THREAD_CREATE:
        extractEventInfoCoverage[9] = 1;
        return extractThreadCreateInfo(
          args[0] as ThreadChannel,
          args[1] as boolean
        );
      case DiscordEvent.TIMER:
        extractEventInfoCoverage[10] = 1;
        return extractTimerInfo(args[0] as Timer);
      case DiscordEvent.SLASH_COMMAND:
        extractEventInfoCoverage[11] = 1;
        return extractSlashCommandInfo(args[0] as ChatInputCommandInteractio
      case DiscordEvent.VOICE_STATE_UPDATE:
        extractEventInfoCoverage[12] = 1;
        return extractVoiceStateUpdateInfo(
          args[0] as VoiceState,
          args[1] as VoiceState
        );
      default:
        throw new Error("Could not extract info for event " + event);
    }
  };
  const infos = getInfos();

  return Array.isArray(infos) ? infos : [infos];
};
```

In total the branch coverage of this function increased from 50% to 100%. This was done by testing cases that were previously left untested. What was convenient is that these cases were the same as the previous function.

Tomas

Function 1: MetaCommand (src\programs\meta.ts):

Code snippet with added the instrumented code to gather coverage measurements by tracking the branches hit:

```
export class MetaCommand implements CommandHandler<DiscordEvent.CONTEXT_MENU_MES
  async handle(command: MessageContextMenuCommandInteraction): Promise<void> {
    const message = command.targetMessage;

    if (command.user === message.author) {
      metaCommandCoverage[0] = 1;
      throw new Error(MetaErrors.SELF_META);
    }

    const emojiByName = (name: string) =>
      message.guild?.emojis.cache.find((e) => e.name === name);
    const metaEmoji = emojiByName(metaEmojiName);

    const didBotReact = message.reactions.cache.some((reaction) => reaction.me);
    if (didBotReact) {
      metaCommandCoverage[1] = 1;
      throw new Error(MetaErrors.ALREADY_METAED);
    }

    await message.reply({
      content:
        "https://user-images.githubusercontent.com/17064122/122255708-ae6e9680-c
    });
    await message.react(metaEmoji ?? "✊");

    await command.reply({ ephemeral: true, content: "Done!" });

    metaCommandCoverage[2] = 1;
  }
}
```

The commit containing the updated code can be found in the following commit:

https://github.com/Yes-Theory-Fam/yesbot-ts/commit/c33ed45c14ea337336597cb10a75c7cbace3506d

The commit also contains the code (meta.spec.ts) of the actual tests performed on the function.
Code snippet of the unit tests performed:

```
it("Throws SELF_META error if user tries to meta itself", async () => {
  command.user = message.author;
  const handler = new MetaCommand();
  await expect(handler.handle(command)).rejects.toThrow("SELF_META");
});

it("Throws ALREADY_METAED error if the message is already metaed", async () => {
  message.reactions.cache.set("someKey", { me: true });
  const handler = new MetaCommand();
  await expect(handler.handle(command)).rejects.toThrow("ALREADY_METAED");
});

it("Reacts with default ✊ emoji if the specified emoji does not exist", async () => {
  const handler = new MetaCommand();
  await handler.handle(command);
  expect(message.react).toHaveBeenCalledWith("✊");
});
```

The tests improved the coverage of the function from 0% to ~90% as seen in the Vitest output bellow:

Before adding tests:

```
-------------------------------------------------------|---------|----------|---------|---------|-------------------
File                                                   | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------------------------------------|---------|----------|---------|---------|-------------------
 meta.ts                                               |       0 |        0 |       0 |       0 | 1-49
```

After adding tests:

```
-------------------------------------------------------|---------|----------|---------|---------|-------------------
File                                                   | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------------------------------------|---------|----------|---------|---------|-------------------
meta.ts                                                |     100 |    88.88 |     100 |     100 | 36
```

The branch coverage improved significantly because major branches of the code's logic (namaly - error handling ifs) were covered by the added tests.

Function 2: FeatureRequestPending (src\programs\feature-request-pending.ts)

Code snippet with added the instrumented code to gather coverage measurements by tracking the branches hit:

```
export class FeatureRequestPending
  implements CommandHandler<DiscordEvent.THREAD_CREATE>
{
  async handle(channel: ThreadChannel, _unused: boolean): Promise<void> {
    const parent = channel.parent;
    if (!parent || parent.type !== ChannelType.GuildForum){
      if (!parent) featureRequestPendingCoverage[0] = 1;
      else featureRequestPendingCoverage[1] = 1;
      return;
    }

    const pendingTag = parent.availableTags.find(
      (t) => t.name.toLowerCase() === "pending"
    );
    if (!pendingTag){
      featureRequestPendingCoverage[2] = 1;
      return;
    }

    await channel.setAppliedTags([...channel.appliedTags, pendingTag.id]);
  }
}
```

The commit containing the updated code can be found in the following commit:

https://github.com/Yes-Theory-Fam/yesbot-ts/commit/b79380c067e156aa6442d21ac1a6f9c0ffcf8fda

The commit also contains the code (feature-request-pending.spec.ts) of the actual tests performed on the function.

Code snippet of the unit tests performed:

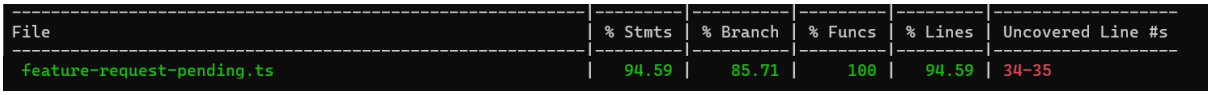
```
it("Function returns early if parent object is undefined", async () => {
  mockChannel.parent = undefined;
  await handler.handle(mockChannel, true);
  expect(mockChannel.setAppliedTags).not.toHaveBeenCalled();
});

it("Function returns early if parent is not of type GuildForum", async () => {
  mockChannel.parent.type = ChannelType.Text;
  await handler.handle(mockChannel, true);
  expect(mockChannel.setAppliedTags).not.toHaveBeenCalled();
});

it("should return early if 'pending' tag is not found", async () => {
  mockChannel.parent.availableTags = [
    { id: "tag1", name: "not pending" },
  ];

  await handler.handle(mockChannel, true);
  expect(mockChannel.setAppliedTags).not.toHaveBeenCalled();
});
```

The tests improved the coverage of the function from 0% to ~85% as seen in the vitest output bellow:

Before adding tests:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
feature-request-pending.ts	0	0	0	0	1-28

After adding tests:



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
feature-request-pending.ts	94.59	85.71	100	94.59	34-35

The branch coverage improved significantly because major branches of the code's logic (namely - early returns from the function) were covered by the added tests.

Sebastian

New test for function1

```
      import { ChatInputCommandInteraction, TextChannel } from 'discord.js';
      import {resourcesCoverage,RESOURCES_SPANISH,RESOURCES_CODING} from
"../../src/programs/resources.js";
      import {Resources} from "../../src/programs/resources.js";
      import {afterEach} from "vitest";
      import {ChatNames} from "../../src/collections/chat-names.js";

      const createMockInteraction = (channelName: string) => {
         const mockReply = vi.fn();
         const channel = { name: channelName } as TextChannel;
         const interaction = {channel, reply: mockReply} as unknown as
ChatInputCommandInteraction;

         return { interaction, replyMock: mockReply };
      };
      describe("resources.handle", () => {
         afterEach(() => {
             const resourceCoverageSum = resourcesCoverage.reduce((sum, x)
=> sum + x);
             console.log("Branch coverage on resource after test run: " +
                 ((resourceCoverageSum / (resourcesCoverage.length) *
100).toPrecision(3) + "%"));
         });

         it('reply with coding resources in the CODING channel', async () =>
{
             const { interaction, replyMock } =
createMockInteraction(ChatNames.CODING);
             const resourcesHandler = new Resources();

             await resourcesHandler.handle(interaction);

             expect(replyMock).toHaveBeenCalledWith(RESOURCES_CODING);
         });

         it('reply with Spanish resources in the LEARNING_SPANISH channel',
async () => {
             const { interaction, replyMock } =
createMockInteraction(ChatNames.LEARNING_SPANISH);
             const resourcesHandler = new Resources();

             await resourcesHandler.handle(interaction);

             expect(replyMock).toHaveBeenCalledWith(RESOURCES_SPANISH);
         });

         it('reply with default message in an unknown channel', async () =>
{
```

```
            const { interaction, replyMock } =
createMockInteraction('unknown-channel');
            const resourcesHandler = new Resources();

            await resourcesHandler.handle(interaction);

            expect(replyMock).toHaveBeenCalledWith({
                content: 'No resource exists for this channel.',
                ephemeral: true,
            });
        });
    });
```

```
File                                    | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
```

Old Test results:

```
resources.ts                            |       0 |        0 |       0 |       0 | 1-69
```

New Test results:

```
resources.ts                            |     100 |      100 |     100 |     100 |
```

Coverage was improved because I added tests tested all branches and covered all lines of the function

New test for function 2:

```
import { Message } from 'discord.js';
import { CustomMessageMethods, customMessageMethodsCoverage} from
"../../src/programs/custom-messages.js";

const createMessage = (content:String) => ({
   content,
   reply: vi.fn().mockResolvedValue(undefined),
   react: vi.fn().mockResolvedValue(undefined),
   }as unknown as Message);

describe('CustomMessageMethods', () => {

   afterEach(() => {
       const CoverageSum = customMessageMethodsCoverage.reduce((sum, x) =>
sum + x);
       console.log("Branch coverage on custom-message after test run: " +
((CoverageSum / (customMessageMethodsCoverage.length) * 100).toPrecision(4)
+ "%"));
   });

   it('reply with a random message when content matches yesbot and ends with
a ?', async () => {
       const message = createMessage('yesbot, are you there?');
       const customMessageHandler = new CustomMessageMethods()
       await customMessageHandler.handle(message);

       expect(message.reply).toHaveBeenCalled();
   });
```

```
   it('send love reply and react with heart when content matches love
expressions', async () => {
       const message = createMessage('yesbot i love you');
       const resourceHandler = new CustomMessageMethods()

       await resourceHandler.handle(message);

       expect(message.reply).toHaveBeenCalled();
       expect(message.react).toHaveBeenCalledWith('😍');
   });

   it('react with 🇫 "F"', async () => {
       const message = createMessage('F');
       const resourceHandler = new CustomMessageMethods()

       await resourceHandler.handle(message);

       expect(message.react).toHaveBeenCalledWith('🇫');
   });

   it('react with 👀 when content matches mod abuse phrases', async () => {
       const message = createMessage('mod abuse');
       const customMessageHandler = new CustomMessageMethods();
       await customMessageHandler.handle(message);

       expect(message.react).toHaveBeenCalledWith('👀');
   });

   it('should not trigger any actions for unrelated messages', async () => {
       const message = createMessage('hello');

       const customMessageHandler = new CustomMessageMethods();
       await customMessageHandler.handle(message);

       expect(message.reply).not.toHaveBeenCalled();
       expect(message.react).not.toHaveBeenCalled();
   });
});
```

Old test results;

```
custom-messages.ts                    |       0 |        0 |       0 |       0 | 1-83
```

New test results:

```
custom-messages.ts                    |     100 |      100 |     100 |     100 |
```

Coverage was improved because I added tests, tested all branches and covered all lines of the function.

Overall

As previously mentioned the entire project had an initial branch coverage of 39.22%, and 12.75% of the functions. Lastly, a 22.92% statement coverage:

All files

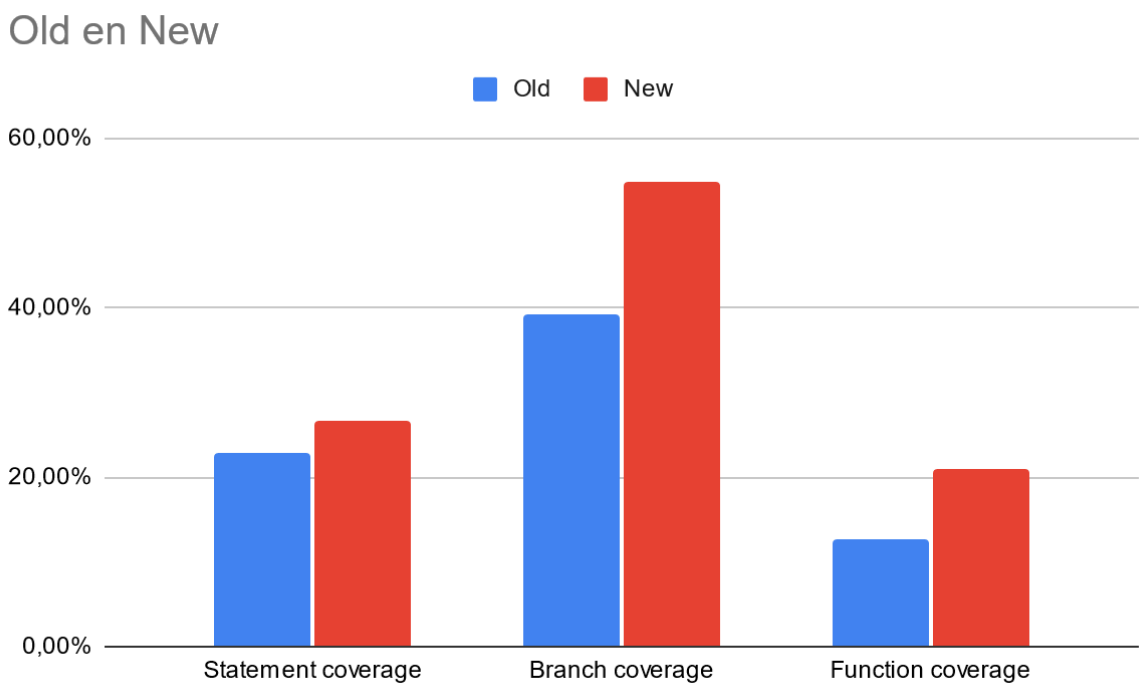
22.92% Statements 3454/15066 **39.22%** Branches 111/283 **12.75%** Functions 31/243 **22.92%** Lines 3454/15066

After merging everything together we get:

All files

26.69% Statements 4042/15140 **54.97%** Branches 199/362 **20.94%** Functions 53/253 **26.69%** Lines 4042/15140

This concludes 588 extra statements covered resulting in a 3.77% improvement in statement coverage, 88 branches extra resulting in a 15.75% boost in branch coverage and lastly 22 extra functions covered resulting in an 8.19% increase in function coverage.
Visualized:



The work was evenly distributed throughout the assignment. Each member worked on their own branch on 2 specific functions. Each member reviewed