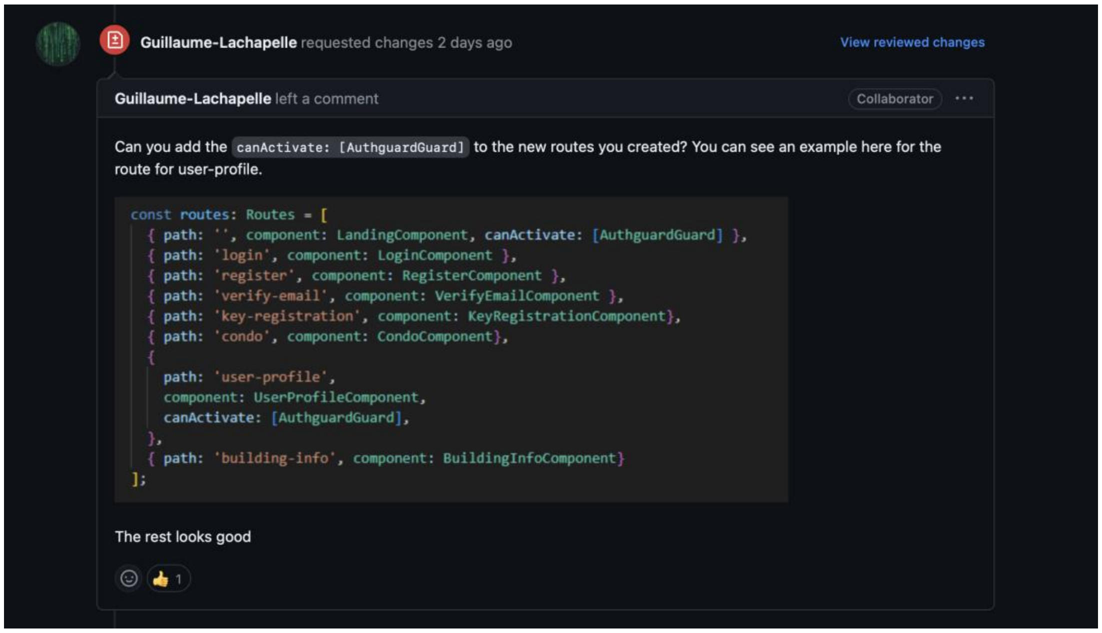
Code Management

○ Quality of source code reviews

In our project, we place a strong emphasis on the quality of our source code reviews to ensure that our codebase maintains a high standard. Each team member actively participates in peer reviews, providing constructive feedback on pull requests. Our reviews are thorough, focusing on code clarity, efficiency, and adherence to coding standards. We leverage tools like GitHub to facilitate the review process, with comments being clear, specific, and often accompanied by screenshots pinpointing areas for improvement. This ensures that all team members have a clear understanding of the feedback and can easily address any suggested changes.



Guillaume-Lachapelle requested changes 2 days ago

Guillaume-Lachapelle left a comment

Can you add the `canActivate: [AuthguardGuard]` to the new routes you created? You can see an example here for the route for user-profile.

```
const routes: Routes = [
  { path: '', component: LandingComponent, canActivate: [AuthguardGuard] },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'verify-email', component: VerifyEmailComponent },
  { path: 'key-registration', component: KeyRegistrationComponent},
  { path: 'condo', component: CondoComponent},
  {
    path: 'user-profile',
    component: UserProfileComponent,
    canActivate: [AuthguardGuard],
  },
  { path: 'building-info', component: BuildingInfoComponent}
];
```

The rest looks good

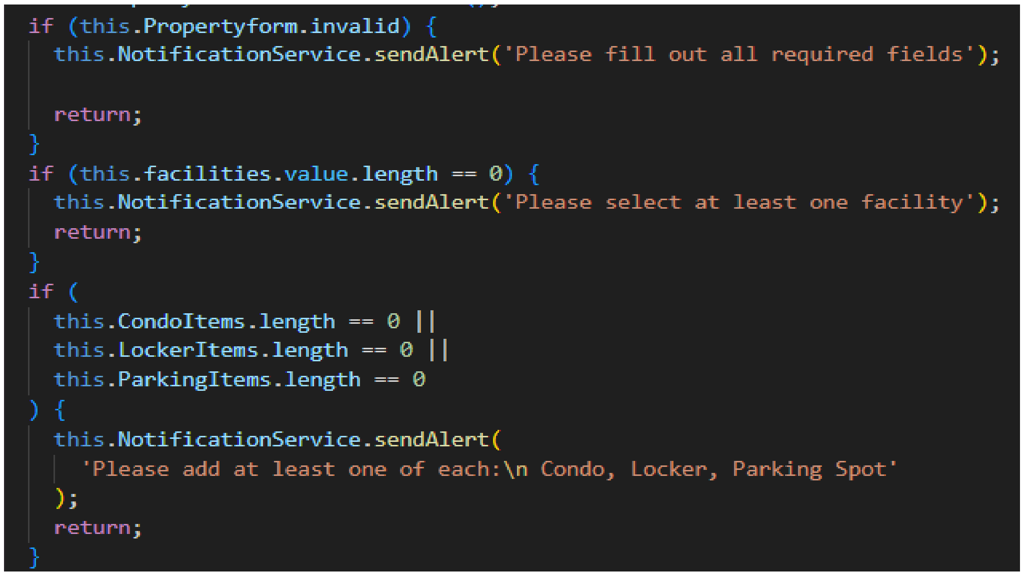
👍 1

Furthermore, we prioritize quick responses to comments, promptly addressing any concerns raised by team members. Whether it's minor adjustments or more critical issues, we take immediate action to refine our code and uphold its quality. We also hold meetings if necessary to discuss complex or unclear feedback, ensuring that everyone is on the same page and contributing to the continual improvement of our codebase. The screenshot above exemplifies our collaborative approach, with team members providing positive feedback while offering constructive suggestions for enhancement. These screenshots showcase real examples of our dedication to clear and specific communication, as team members highlight specific sections of code for improvement and offer valuable insights for optimization. Through these practices, we maintain a cohesive and high-quality codebase, driving the success of our project.

○ Correct use of design patterns

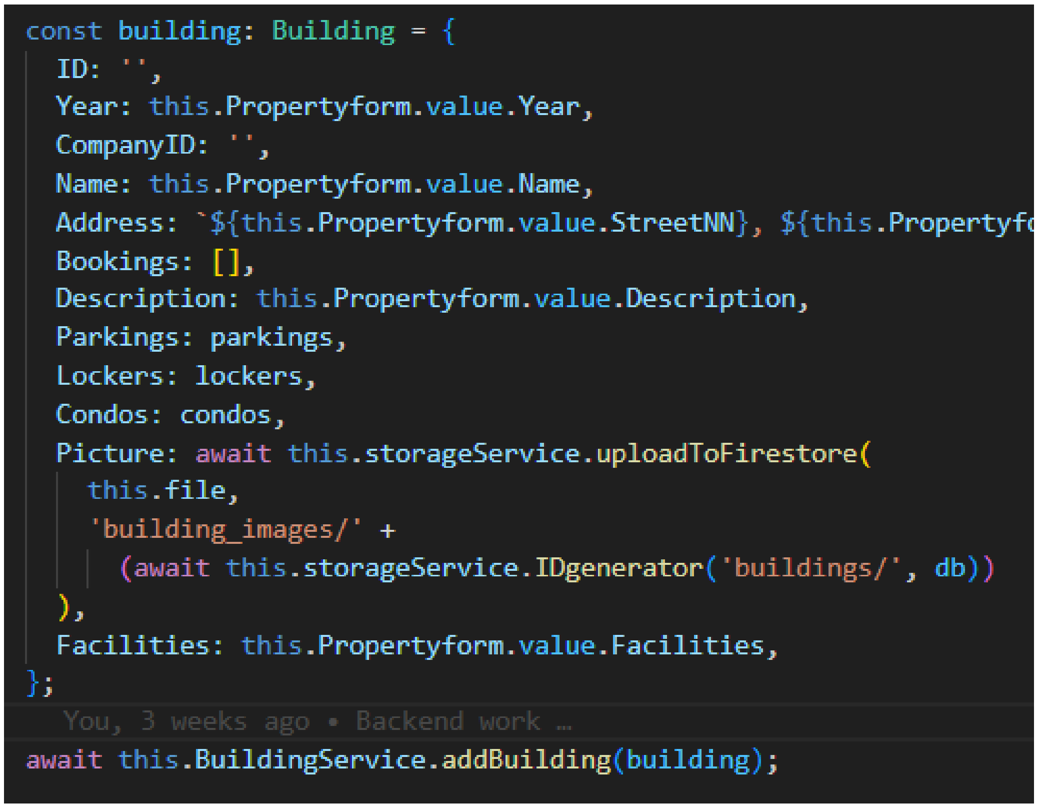
We followed the main design pattern principals

Creational Patterns: These deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. We created objects to reduce problems and complexity to the design. We did this by controlling the object's creation. For example:



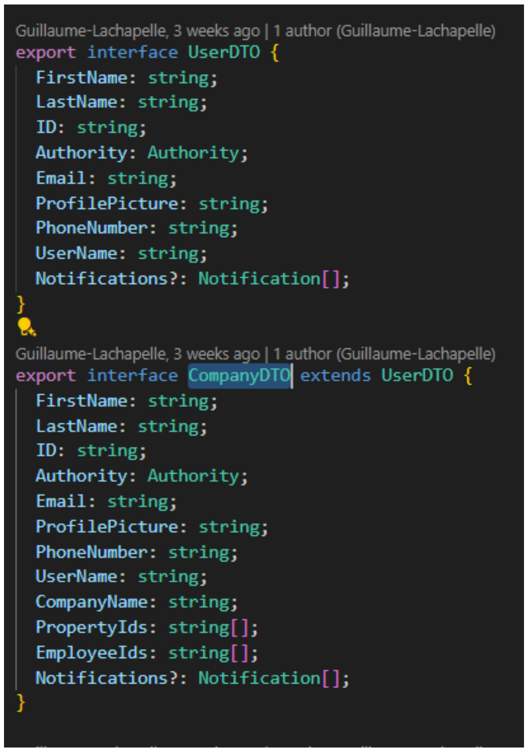
```
if (this.Propertyform.invalid) {
  this.NotificationService.sendAlert('Please fill out all required fields');

  return;
}
if (this.facilities.value.length == 0) {
  this.NotificationService.sendAlert('Please select at least one facility');
  return;
}
if (
  this.CondoItems.length == 0 ||
  this.LockerItems.length == 0 ||
  this.ParkingItems.length == 0
) {
  this.NotificationService.sendAlert(
    'Please add at least one of each:\n Condo, Locker, Parking Spot'
  );
  return;
}
```

In the example above the if statements prevent the building object to be created if certain of its possible attributes are not set.



```
const building: Building = {
  ID: '',
  Year: this.Propertyform.value.Year,
  CompanyID: '',
  Name: this.Propertyform.value.Name,
  Address: `${this.Propertyform.value.StreetNN}, ${this.Propertyfo
  Bookings: [],
  Description: this.Propertyform.value.Description,
  Parkings: parkings,
  Lockers: lockers,
  Condos: condos,
  Picture: await this.storageService.uploadToFirestore(
    this.file,
    'building_images/' +
      (await this.storageService.IDgenerator('buildings/', db))
  ),
  Facilities: this.Propertyform.value.Facilities,
};
   You, 3 weeks ago • Backend work …
await this.BuildingService.addBuilding(building);
```

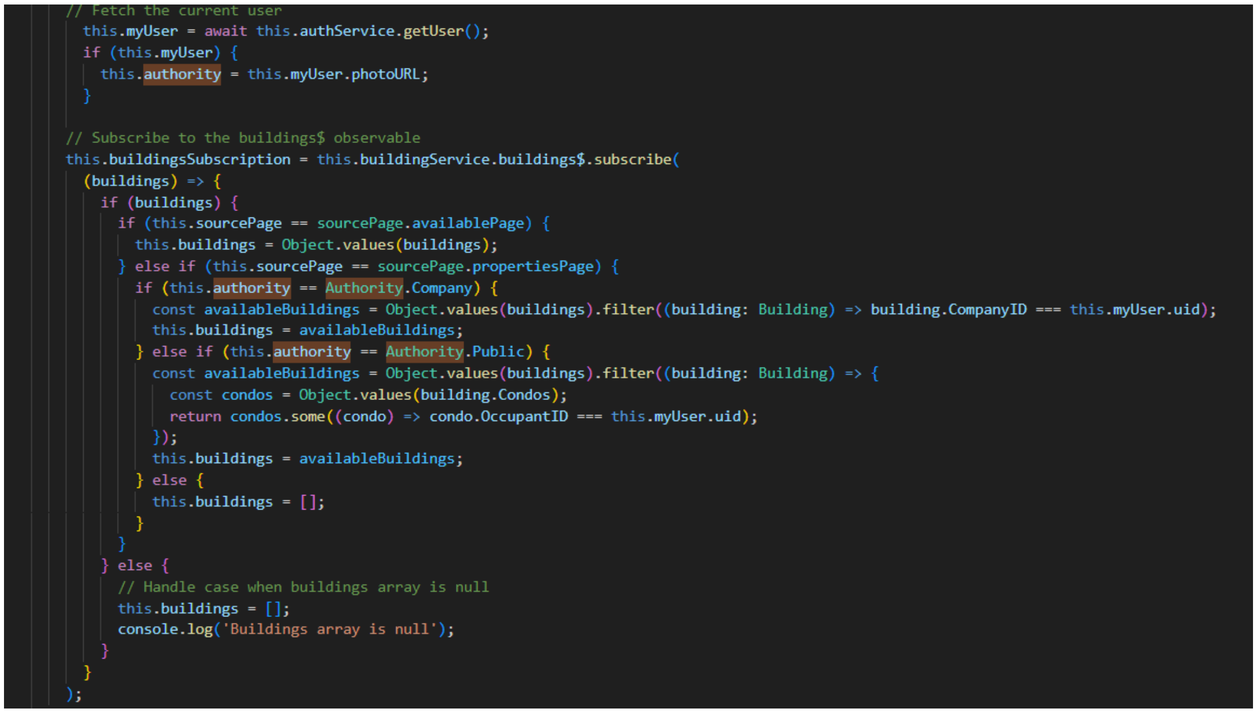
Structural Patterns: We use structural class patterns and use inheritance to compose interfaces and implementations.



```
Guillaume-Lachapelle, 3 weeks ago | 1 author (Guillaume-Lachapelle)
export interface UserDTO {
  FirstName: string;
  LastName: string;
  ID: string;
  Authority: Authority;
  Email: string;
  ProfilePicture: string;
  PhoneNumber: string;
  UserName: string;
  Notifications?: Notification[];
}

Guillaume-Lachapelle, 3 weeks ago | 1 author (Guillaume-Lachapelle)
export interface CompanyDTO extends UserDTO {
  FirstName: string;
  LastName: string;
  ID: string;
  Authority: Authority;
  Email: string;
  ProfilePicture: string;
  PhoneNumber: string;
  UserName: string;
  CompanyName: string;
  PropertyIds: string[];
  EmployeeIds: string[];
  Notifications?: Notification[];
}
```

In this example CompanyDTO inherits from UserDTO. This is essential to keep the code clean and to avoid redundancy. It also helps with creating similar child elements and create consistency between users.

Behavioral Patterns: We created communication between objects, how objects interact and distribute responsibility.



```
// Fetch the current user
this.myUser = await this.authService.getUser();
if (this.myUser) {
  this.authority = this.myUser.photoURL;
}

// Subscribe to the buildings$ observable
this.buildingsSubscription = this.buildingService.buildings$.subscribe(
  (buildings) => {
    if (buildings) {
      if (this.sourcePage == sourcePage.availablePage) {
        this.buildings = Object.values(buildings);
      } else if (this.sourcePage == sourcePage.propertiesPage) {
        if (this.authority == Authority.Company) {
          const availableBuildings = Object.values(buildings).filter((building: Building) => building.CompanyID === this.myUser.uid);
          this.buildings = availableBuildings;
        } else if (this.authority == Authority.Public) {
          const availableBuildings = Object.values(buildings).filter((building: Building) => {
            const condos = Object.values(building.Condos);
            return condos.some((condo) => condo.OccupantID === this.myUser.uid);
          });
          this.buildings = availableBuildings;
        } else {
          this.buildings = [];
        }
      }
    } else {
      // Handle case when buildings array is null
      this.buildings = [];
      console.log('Buildings array is null');
    }
  }
);
```

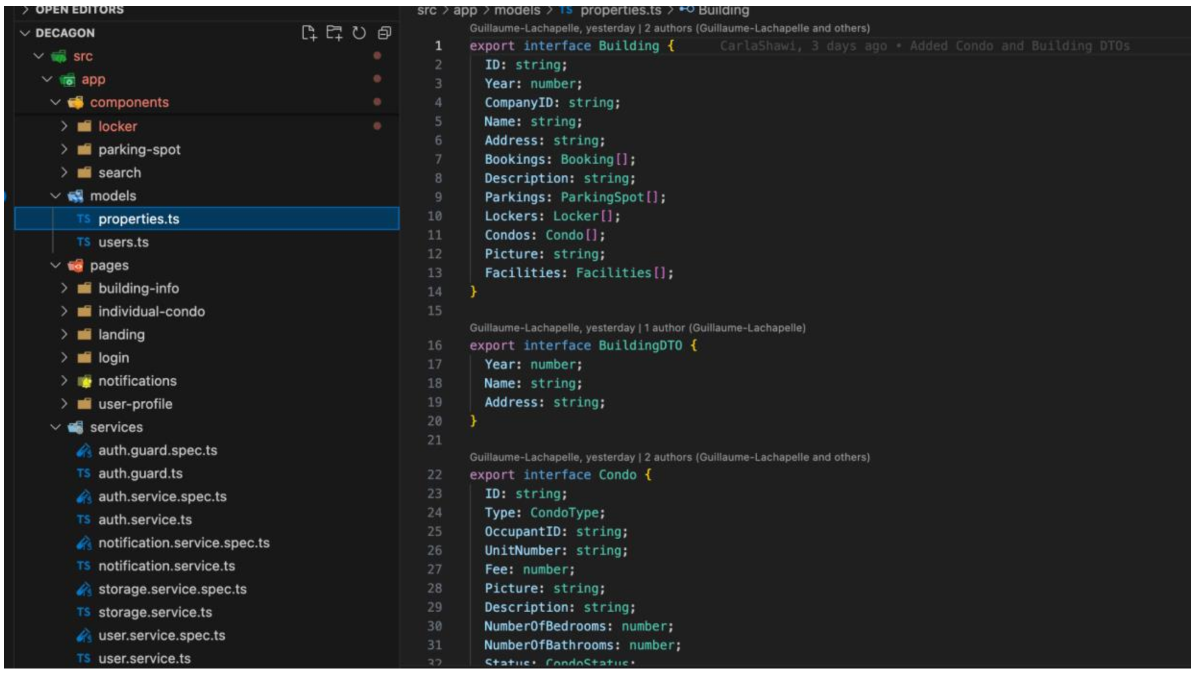
In this example we can see that the sourcePage and authority object direct the behavior of the building object.

- **Respect to code conventions**

We follow various types of code conventions in our project. Many of them are standard angular coding style conventions. File names are named such that words are all lowercase separated with dots and dashes. Class fields and methods are named using camel case. Class names, enums, interface names and interface fields are in pascal case. Every component and service is designed to be responsible for one thing only. There are never more than two classes in one file. Functions are designed to do one thing specifically and to not take up too many lines of code. Large functions are split into many smaller private helper functions or to use available services to complete their tasks. Smaller functions are easier to read and to test. Function names are also descriptive and their behavior is hinted by their name. Consistent indentation is used, usually a new line and indent for every block of code. Comments are occasionally written in code to describe complex code logic that may not be clear at first. Both async / await and .then() callbacks are used to wait for a promise to resolve, however, the async / await approach is preferred for blocks of code that deal with many promises so that code appears linearly at the same indentation, rather than using many nested .then() callbacks which would make code harder to read.

- **Design quality (number of classes/packages, size, coupling, cohesion)**

In our project, we prioritize design quality by adhering to specific metrics and principles to ensure a structured, maintainable, and efficient codebase. One key aspect we focus on is the number of classes/packages, where we aim to strike a balance between having enough classes to encapsulate functionality and avoiding excessive complexity. Our code layout screenshot demonstrates this, with each folder representing a distinct component, page, or model, contributing to a well-organized and modularized structure. By quantifying the number of classes and packages, we ensure that our codebase remains manageable and scalable.

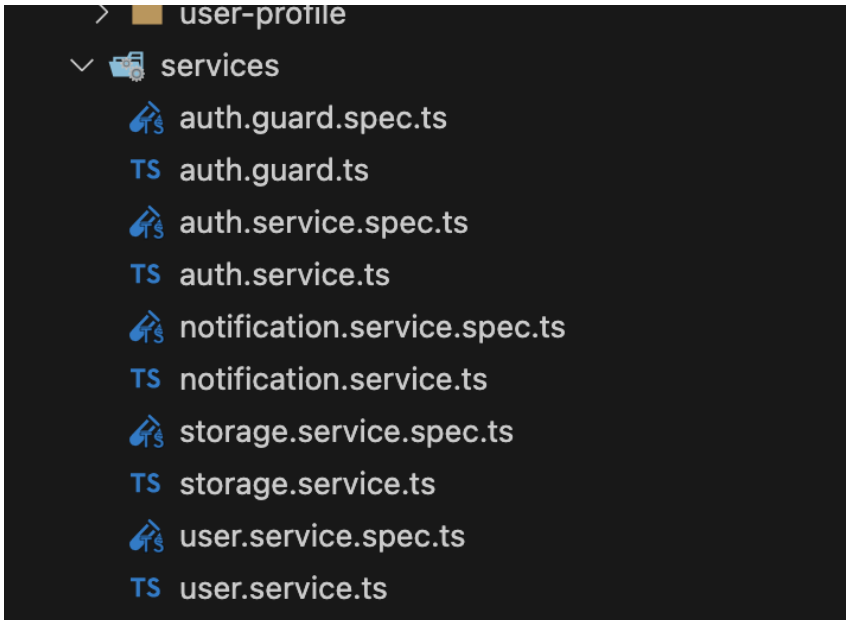


```
> OPEN EDITORS
v DECAGON
  v src
    v app
      v components
        > locker
        > parking-spot
        > search
      v models
          properties.ts
          users.ts
      v pages
        > building-info
        > individual-condo
        > landing
        > login
        > notifications
        > user-profile
      v services
          auth.guard.spec.ts
          auth.guard.ts
          auth.service.spec.ts
          auth.service.ts
          notification.service.spec.ts
          notification.service.ts
          storage.service.spec.ts
          storage.service.ts
          user.service.spec.ts
          user.service.ts
```

```
src > app > models > properties.ts > Building
Guillaume-Lachapelle, yesterday | 2 authors (Guillaume-Lachapelle and others)
export interface Building {      CarlaShawi, 3 days ago • Added Condo and Building DTOs
  ID: string;
  Year: number;
  CompanyID: string;
  Name: string;
  Address: string;
  Bookings: Booking[];
  Description: string;
  Parkings: ParkingSpot[];
  Lockers: Locker[];
  Condos: Condo[];
  Picture: string;
  Facilities: Facilities[];
}

Guillaume-Lachapelle, yesterday | 1 author (Guillaume-Lachapelle)
export interface BuildingDTO {
  Year: number;
  Name: string;
  Address: string;
}

Guillaume-Lachapelle, yesterday | 2 authors (Guillaume-Lachapelle and others)
export interface Condo {
  ID: string;
  Type: CondoType;
  OccupantID: string;
  UnitNumber: string;
  Fee: number;
  Picture: string;
  Description: string;
  NumberOfBedrooms: number;
  NumberOfBathrooms: number;
  Status: CondoStatus;
```

Furthermore, we emphasize size, coupling, and cohesion as additional metrics for design quality. Size refers to the length and complexity of individual classes or components, where we aim to keep them concise and focused on a single responsibility. This approach is reflected in our code layout, where each file contains clear and specific functionality, contributing to maintainable and understandable code.

Coupling and cohesion are crucial aspects of our design approach. Coupling measures the interdependence between classes or components, and we strive to keep it low by decoupling different parts of our system. Our use of services in separate TypeScript files, as demonstrated in the provided screenshot, exemplifies this principle. Each service encapsulates related functionality, minimizing dependencies and promoting modularization.

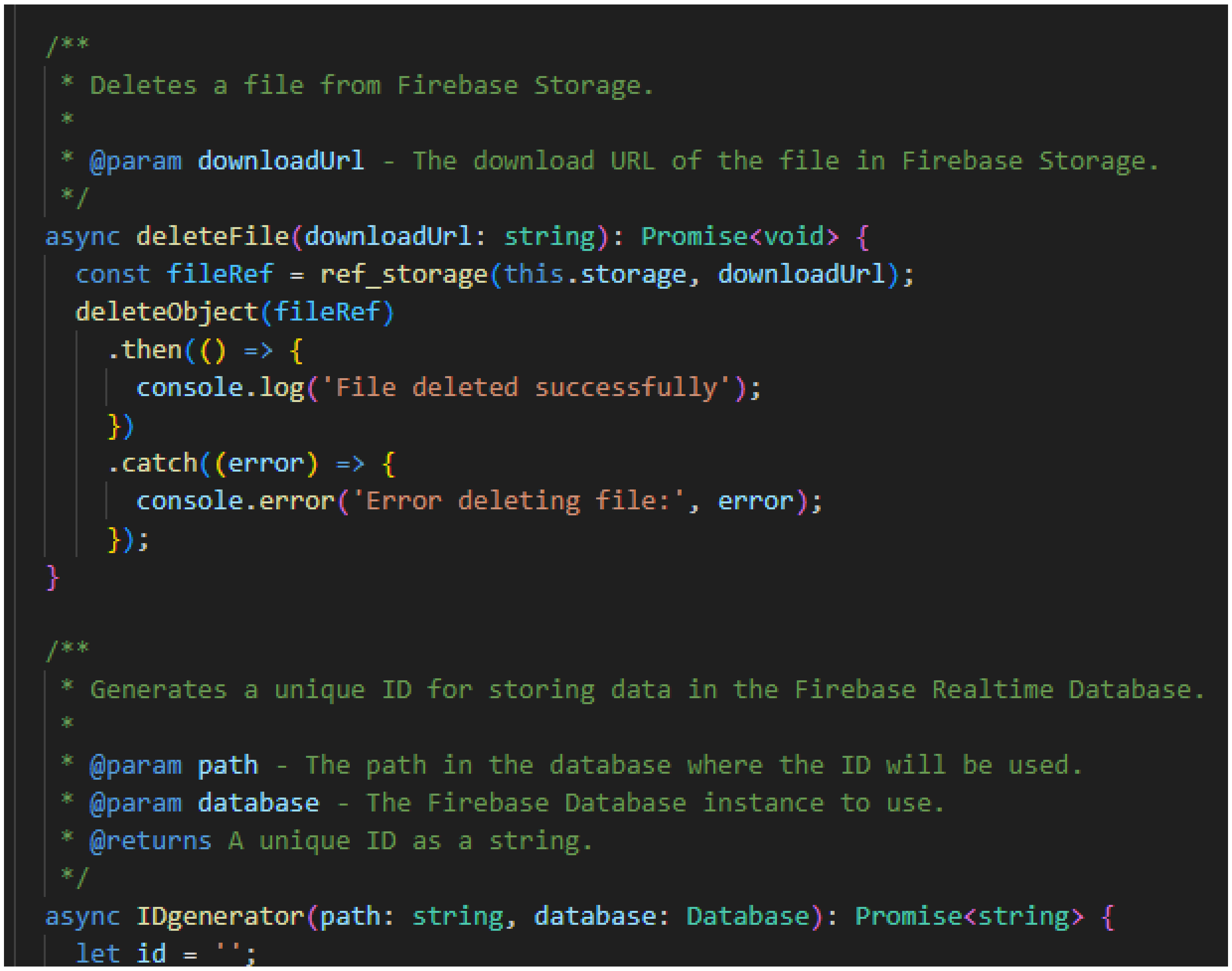


```
  > user-profile
v services
    auth.guard.spec.ts
    auth.guard.ts
    auth.service.spec.ts
    auth.service.ts
    notification.service.spec.ts
    notification.service.ts
    storage.service.spec.ts
    storage.service.ts
    user.service.spec.ts
    user.service.ts
```

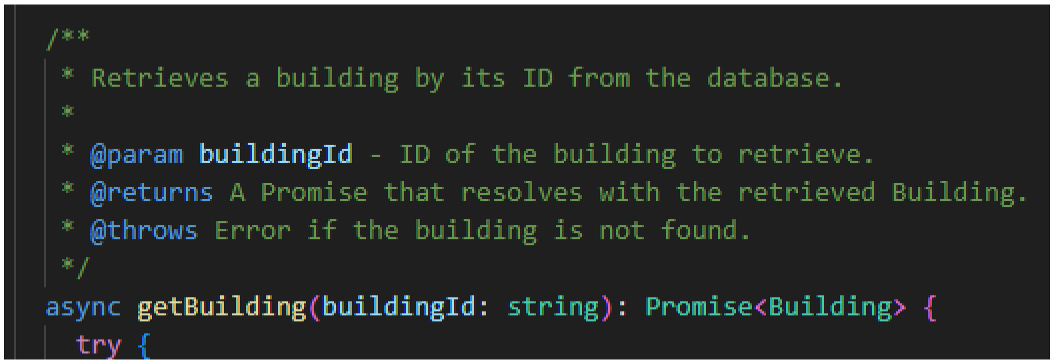
Similarly, we prioritize high cohesion, ensuring that components or classes have a clear and consistent purpose. By separating concerns and organizing code into logical units, we enhance maintainability and readability. Our folder structure and file organization reflect this commitment to cohesion, with distinct directories for different aspects of our application, such as services, models, and pages.

○ **Quality of source code documentation**

Quality of source code documentation is a critical aspect of our project's development process. The main goal of comments in our code is to have a better understanding, improve maintainability, higher code quality, and have higher collaboration within the team. To do so, we have made sure to document the main functions, especially in our Angular services. Our source code documentation follows the standard Angular TypeScript documentation template with a small explanation of what the function does, its parameters, and what it returns.

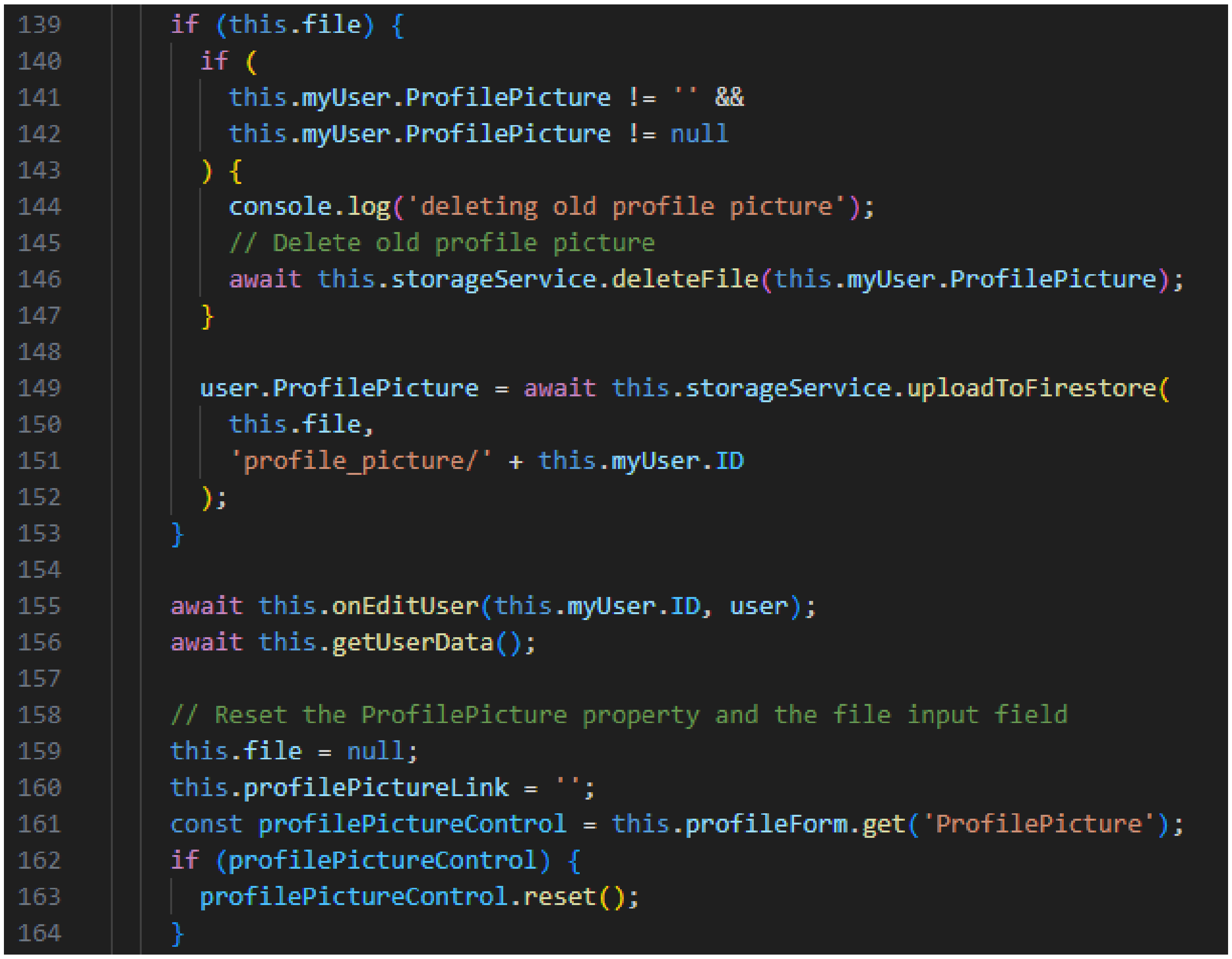


```
/**
 * Deletes a file from Firebase Storage.
 *
 * @param downloadUrl - The download URL of the file in Firebase Storage.
 */
async deleteFile(downloadUrl: string): Promise<void> {
  const fileRef = ref_storage(this.storage, downloadUrl);
  deleteObject(fileRef)
    .then(() => {
      console.log('File deleted successfully');
    })
    .catch((error) => {
      console.error('Error deleting file:', error);
    });
}

/**
 * Generates a unique ID for storing data in the Firebase Realtime Database.
 *
 * @param path - The path in the database where the ID will be used.
 * @param database - The Firebase Database instance to use.
 * @returns A unique ID as a string.
 */
async IDgenerator(path: string, database: Database): Promise<string> {
  let id = '';
```



```
/**
 * Retrieves a building by its ID from the database.
 *
 * @param buildingId - ID of the building to retrieve.
 * @returns A Promise that resolves with the retrieved Building.
 * @throws Error if the building is not found.
 */
async getBuilding(buildingId: string): Promise<Building> {
  try {
```

The most important functions to document are those inside the different Angular services since those functions are used by every team member inside the components they are developing. Though it is essential to have consistent documentation through the app, we realized that this wasn’t the case initially, which is why we decided to create a documentation task to add this source code documentation throughout the app. This task was successfully completed during Sprint 3. We decided not to add this documentation to every single function in every component to not clutter the source code. This is why we only added this type of thorough documentation inside the services and authentication guards. For the rest of the components, we still decided to add some comments, but not as detailed, to make the code more readable.

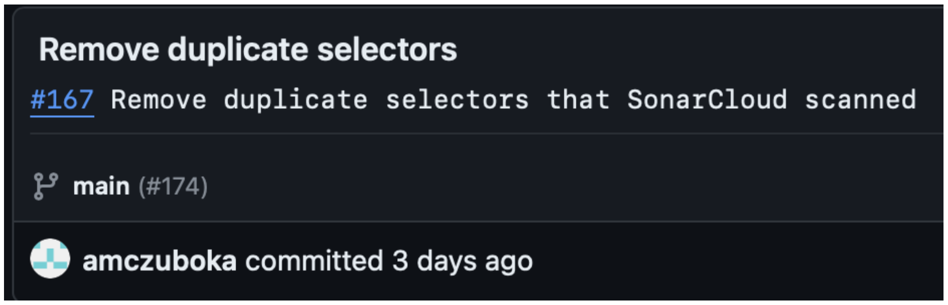


```
if (this.file) {
  if (
    this.myUser.ProfilePicture != '' &&
    this.myUser.ProfilePicture != null
  ) {
    console.log('deleting old profile picture');
    // Delete old profile picture
    await this.storageService.deleteFile(this.myUser.ProfilePicture);
  }

  user.ProfilePicture = await this.storageService.uploadToFirestore(
    this.file,
    'profile_picture/' + this.myUser.ID
  );
}

await this.onEditUser(this.myUser.ID, user);
await this.getUserData();

// Reset the ProfilePicture property and the file input field
this.file = null;
this.profilePictureLink = '';
const profilePictureControl = this.profileForm.get('ProfilePicture');
if (profilePictureControl) {
  profilePictureControl.reset();
}
```

○ **Refactoring activity documented in commit messages**

It is important to refactor the repository's code. This allows us to keep our code better maintained and more readable. Since our team implemented SonarCloud as our code scanning solution, this tool provides us with measures such as issues, security hotspots and the percentage of code duplication. Furthermore, apart from code smells brought up by SonarCloud, we also perform other code refactoring throughout development, for example, refactoring code by extracting functions.

Below are two commit messages that are linked to a refactoring task that fixes issues revealed with SonarCloud

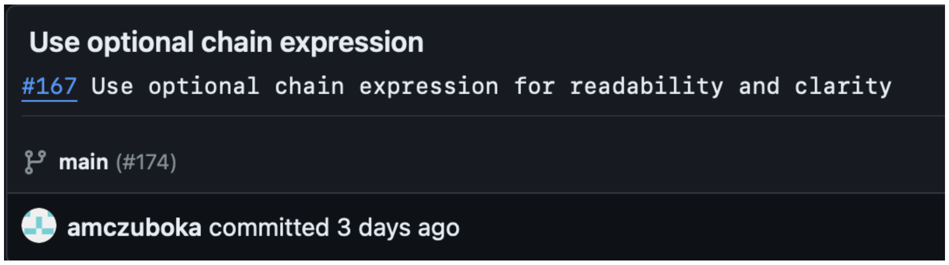


Remove duplicate selectors

```
#167 Remove duplicate selectors that SonarCloud scanned
```

main (#174)

amczuboka committed 3 days ago



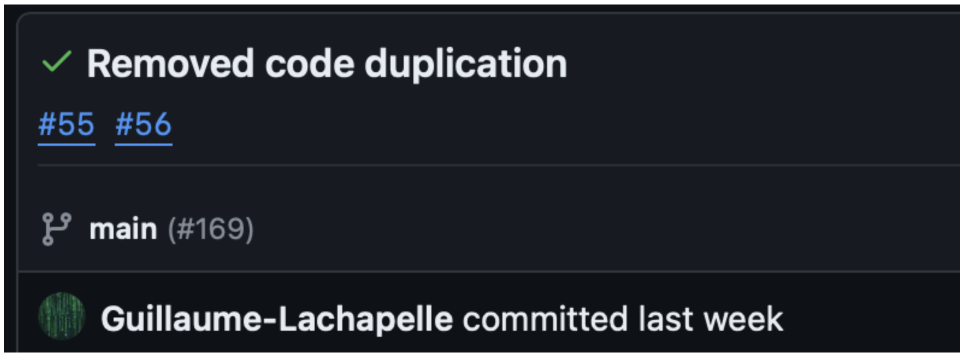
Use optional chain expression

```
#167 Use optional chain expression for readability and clarity
```

main (#174)

amczuboka committed 3 days ago

Below is an example of a commit that removes code duplication during the development of a feature



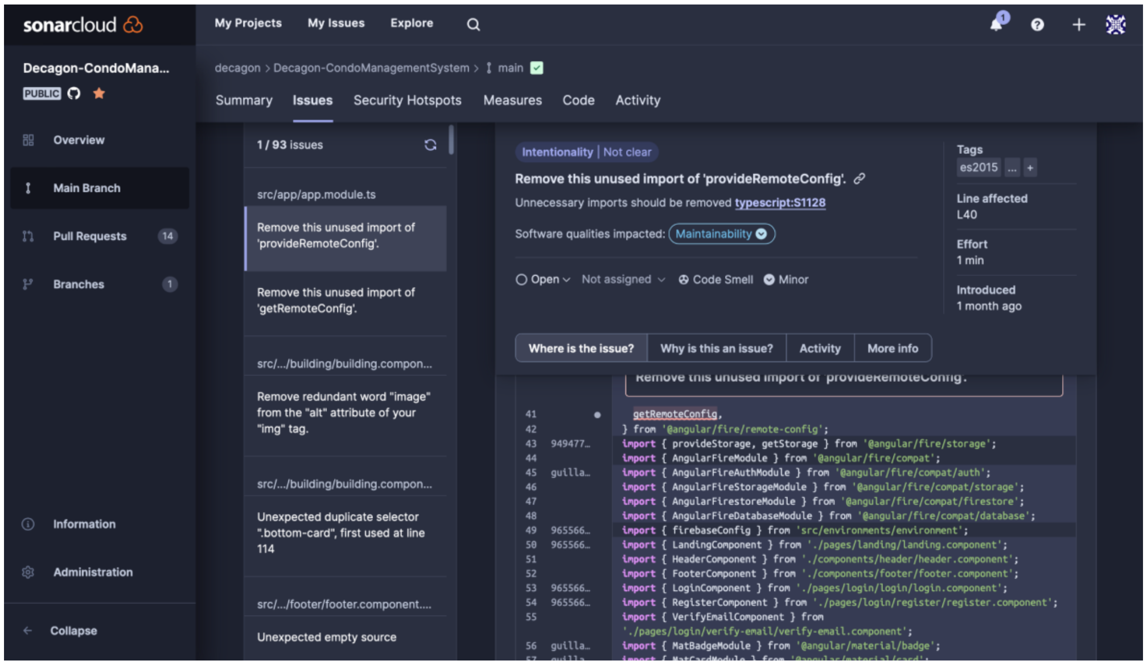
✓ **Removed code duplication**

#55 #56

main (#169)

Guillaume-Lachapelle committed last week

Below is an image of SonarCloud before merging in a few new branches. There were 93 issues



src/app/app.module.ts

Remove this unused import of 'provideRemoteConfig'.

Remove this unused import of 'getRemoteConfig'.

src/.../building/building.compon...

Remove redundant word "image" from the "alt" attribute of your "img" tag.

src/.../building/building.compon...

Unexpected duplicate selector ".bottom-card", first used at line 114

src/.../footer/footer.component....

Unexpected empty source

Intentionality | Not clear

Remove this unused import of 'provideRemoteConfig'.

Unnecessary imports should be removed typescript:S1128

Software qualities impacted: Maintainability

Open Not assigned Code Smell Minor

Tags: es2015

Line affected: L40

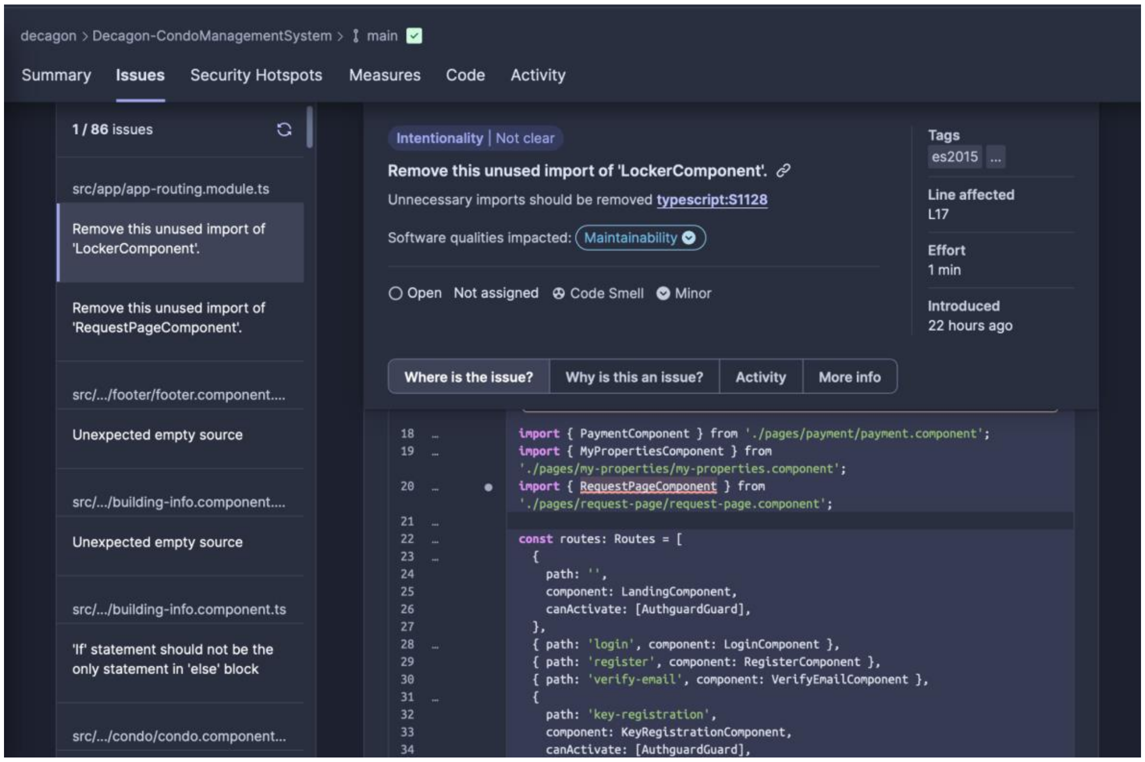
Effort: 1 min

Introduced: 1 month ago

Remove this unused import of 'provideRemoteConfig'.

```
  getRemoteConfig,
} from '@angular/fire/remote-config';
import { provideStorage, getStorage } from '@angular/fire/storage';
import { AngularFireModule } from '@angular/fire/compat';
import { AngularFireAuthModule } from '@angular/fire/compat/auth';
import { AngularFireStorageModule } from '@angular/fire/compat/storage';
import { AngularFirestoreModule } from '@angular/fire/compat/firestore';
import { AngularFireDatabaseModule } from '@angular/fire/compat/database';
import { firebaseConfig } from 'src/environments/environment';
import { LandingComponent } from './pages/landing/landing.component';
import { HeaderComponent } from './components/header/header.component';
import { FooterComponent } from './components/footer/footer.component';
import { LoginComponent } from './pages/login/login.component';
import { RegisterComponent } from './pages/login/register/register.component';
import { VerifyEmailComponent } from
'./pages/login/verify-email/verify-email.component';
import { MatBadgeModule } from '@angular/material/badge';
```

Below is an image of SonarCloud after fixing a few issues and merging in a few new branches. Now there are 86 issues



src/app/app-routing.module.ts

Remove this unused import of 'LockerComponent'.

Remove this unused import of 'RequestPageComponent'.

src/.../footer/footer.component....

Unexpected empty source

src/.../building-info.component....

Unexpected empty source

src/.../building-info.component.ts

'If' statement should not be the only statement in 'else' block

src/.../condo/condo.component...

Intentionality | Not clear

Remove this unused import of 'LockerComponent'.

Unnecessary imports should be removed typescript:S1128

Software qualities impacted: Maintainability

Open Not assigned Code Smell Minor

Tags: es2015

Line affected: L17

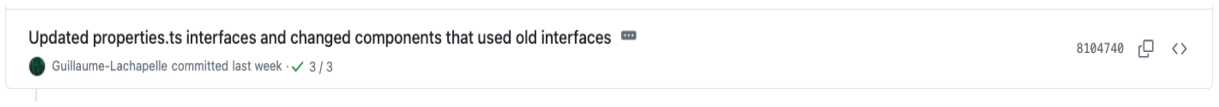
Effort: 1 min

Introduced: 22 hours ago

```
import { PaymentComponent } from './pages/payment/payment.component';
import { MyPropertiesComponent } from
'./pages/my-properties/my-properties.component';
import { RequestPageComponent } from
'./pages/request-page/request-page.component';

const routes: Routes = [
  {
    path: '',
    component: LandingComponent,
    canActivate: [AuthguardGuard],
  },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'verify-email', component: VerifyEmailComponent },
  {
    path: 'key-registration',
    component: KeyRegistrationComponent,
    canActivate: [AuthguardGuard],
```

○ **Quality/detail of commit messages**

The quality and detail of commit messages are important to describe and communicate the changes made to all the teammates. A quality and detailed commit message is descriptive and concise as follows:



Updated properties.ts interfaces and changed components that used old interfaces

Guillaume-Lachapelle committed last week · ✓ 3 / 3

8104740

Detailed commit messages provide context about what changes were made and reference to which issue it relates, which allows developers to trace the changes and understand them easily.

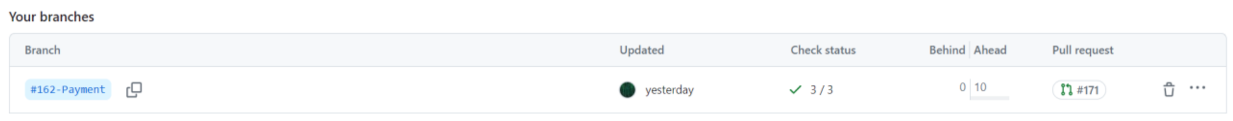
Good commit messages help the code review process. Reviewers can instantly understand the intent of modifications, making it easy to provide feedback and ensure the review is complete and effective.

When a bug is detected, a well-documented commit history can help pinpoint when it happened.

Therefore, a good quality commit message is a good practice as it has benefits for the project and development team on the run. It guarantees that the codebase is maintainable, clear, and easy to explore, which contributes to the project's overall success and sustainability. Despite the clear benefits of the detailed commit messages, we are lacking in consistency in applying this practice across our project. Moving forward, the team will address this issue by adding more descriptive commit messages to follow good software development practices.

○ **Use of feature branches**

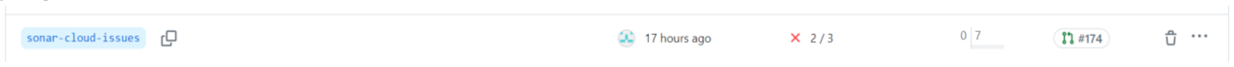
For more control over our work, all created branches have a specific purpose, usually related to a task to be done. It is important not to mix work within a branch since that would create code that is hard to follow. Moreover, it might cause duplication of work due to miscommunication. Another good point in narrowing branches to just one feature is that it makes the review process simple as well as the merging more seamless. It is also much better when problems arise. When debugging is required, the issue can be deduced to be surrounding the feature components. For example, a good branch would look like this:



Your branches

Branch	Updated	Check status	Behind	Ahead	Pull request
#162-Payment	yesterday	✓ 3 / 3	0	10	#171

Here, the issue number is present as well as a good description of the branch purpose at hand which is about the payment page. Another example is creating a separate branch for when bugs show up unexpectedly. This assures that the issue is fixed separately from important features in progress.


sonar-cloud-issues 17 hours ago ✗ 2 / 3 0 | 7 #174

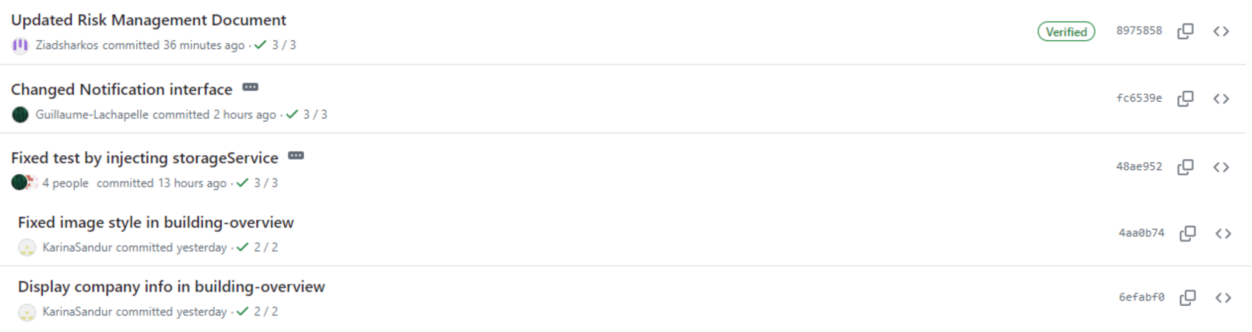
○ **Atomic commits**

Atomic commits are key to ensure efficient code management. An atomic commit is a commit that focuses on one specific change in the code that can easily be summarized. Atomic commits allow developers to easily track changes in the code when performing code reviews. Having atomic commits also allows developers to more easily identify and fix bugs in code by pinpointing exactly when and what change caused a problem. Atomic commits, unlike commits with many changes, can easily be reversed.

Therefore, our team made an effort to have atomic commits throughout the entire project to have a clean workflow.

Here are some examples of atomic commits in our project:

Recent Commits:


Updated Risk Management Document
Ziadsharkos committed 36 minutes ago · ✓ 3 / 3 Verified 8975858

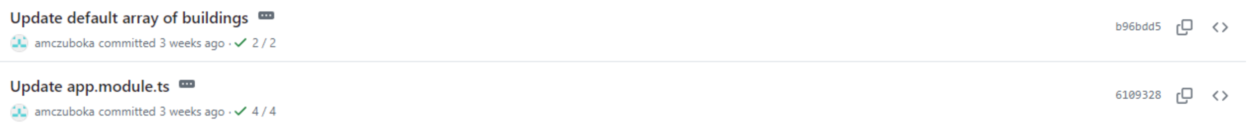
Changed Notification interface
Guillaume-Lachapelle committed 2 hours ago · ✓ 3 / 3 fc6539e

Fixed test by injecting storageService
4 people committed 13 hours ago · ✓ 3 / 3 48ae952

Fixed image style in building-overview
KarinaSandur committed yesterday · ✓ 2 / 2 4aa0b74

Display company info in building-overview
KarinaSandur committed yesterday · ✓ 2 / 2 6efabf0

Earlier Commits:


Update default array of buildings
amczuboka committed 3 weeks ago · ✓ 2 / 2 b96bdd5

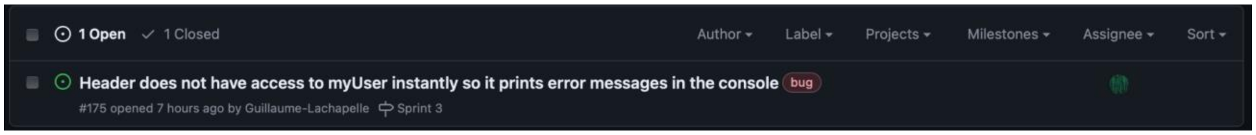
Update app.module.ts
amczuboka committed 3 weeks ago · ✓ 4 / 4 6109328

○ **Bug reporting**

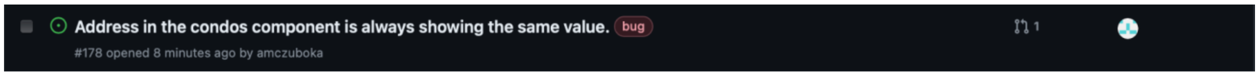
Bug reporting is essential in code management. It contributes to quality assurance by maintaining and improving the software. By tracking and fixing bugs, the team ensures that users get an intended user experience. Reporting is an important part of resolving bugs since it allows for communication across all teams and provides details on how to reproduce them.

Furthermore, when a history of multiple bugs are reported, developers can potentially find trends or patterns which can help solve future bugs more rapidly. Usually, the team uncovers bugs during testing, PR reviews and sometimes during development looking at the console.

Below are examples of open logged bugs.

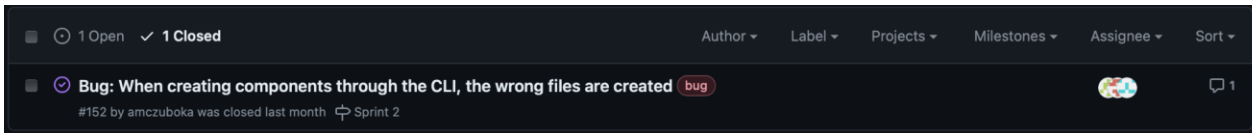


Header does not have access to myUser instantly so it prints error messages in the console bug
#175 opened 7 hours ago by Guillaume-Lachapelle Sprint 3



Address in the condos component is always showing the same value. bug
#178 opened 8 minutes ago by amczuboka

Below is an example of a closed logged bug.



Bug: When creating components through the CLI, the wrong files are created bug
#152 by amczuboka was closed last month Sprint 2

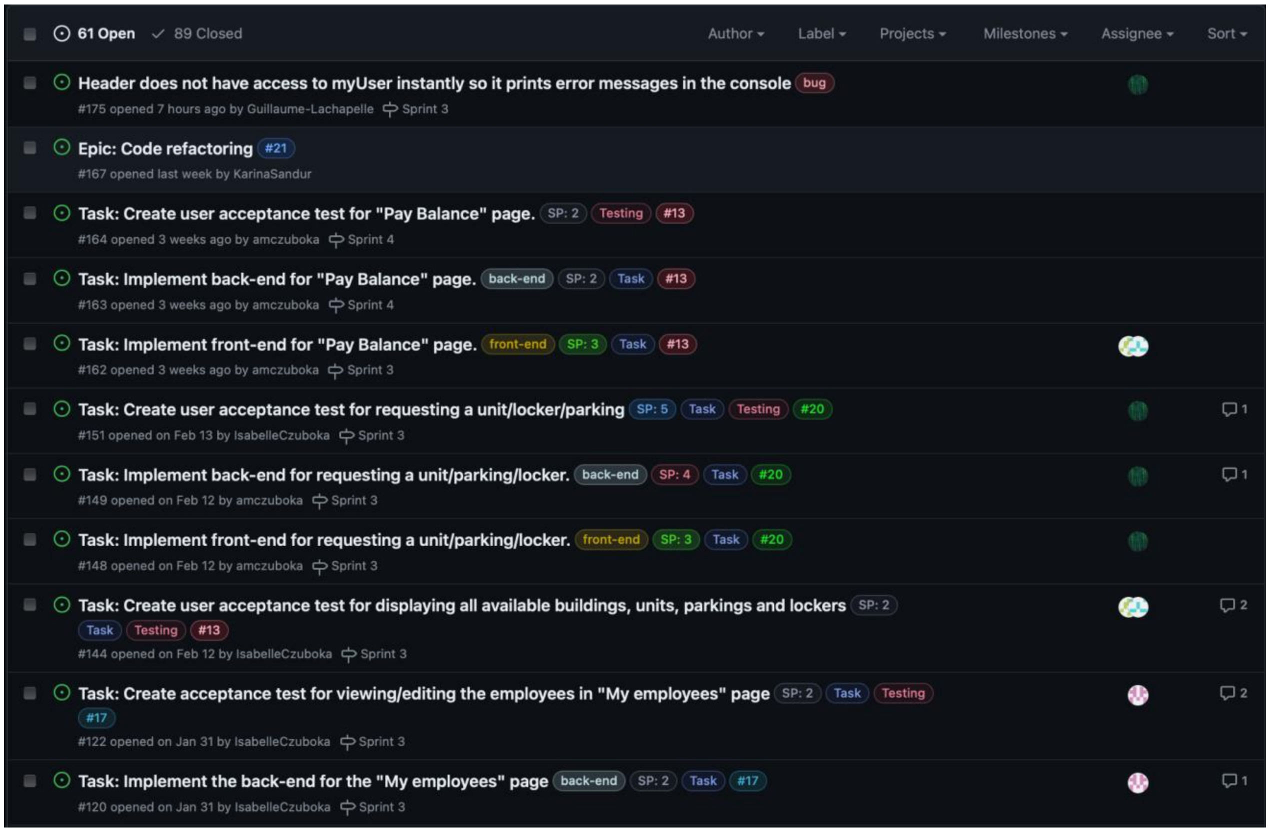
○ **Use of issue labels for tracking and filtering**

The use of issue labeling in our GitHub repository is crucial for the organization. It allows tracking and filtering of issues depending on several features. These features could be front-end, back-end, testing, documentation, bug, story points, and user story numbers. We even label them depending on if the issue is an Epic, User Story or Task.

On top of our created labels, GitHub already provides a way to filter by author, label, projects, milestones, assignee, sort with options and if the issue is open or closed. The most commonly used labels in our day-to-day tasks are the assignee, milestones and front-end and back-end labels/filters.

Labels simplify and enable a better workflow by allowing the team members to identify their tasks and prioritize them. Intuitively, these labels also help with the team's communication and collaboration by presenting our project logically and starting discussions on the project's workflow and priorities.

Below is an image of the use of labeling our issues



61 Open ✓ 89 Closed — Author ▾ Label ▾ Projects ▾ Milestones ▾ Assignee ▾ Sort ▾

Header does not have access to myUser instantly so it prints error messages in the console `bug`
#175 opened 7 hours ago by Guillaume-Lachapelle · Sprint 3

Epic: Code refactoring `#21`
#167 opened last week by KarinaSandur

Task: Create user acceptance test for "Pay Balance" page. `SP: 2` `Testing` `#13`
#164 opened 3 weeks ago by amczuboka · Sprint 4

Task: Implement back-end for "Pay Balance" page. `back-end` `SP: 2` `Task` `#13`
#163 opened 3 weeks ago by amczuboka · Sprint 4

Task: Implement front-end for "Pay Balance" page. `front-end` `SP: 3` `Task` `#13`
#162 opened 3 weeks ago by amczuboka · Sprint 3

Task: Create user acceptance test for requesting a unit/locker/parking `SP: 5` `Task` `Testing` `#20` — 1
#151 opened on Feb 13 by IsabelleCzuboka · Sprint 3

Task: Implement back-end for requesting a unit/parking/locker. `back-end` `SP: 4` `Task` `#20` — 1
#149 opened on Feb 12 by amczuboka · Sprint 3

Task: Implement front-end for requesting a unit/parking/locker. `front-end` `SP: 3` `Task` `#20`
#148 opened on Feb 12 by amczuboka · Sprint 3

Task: Create user acceptance test for displaying all available buildings, units, parkings and lockers `SP: 2` `Task` `Testing` `#13` — 2
#144 opened on Feb 12 by IsabelleCzuboka · Sprint 3

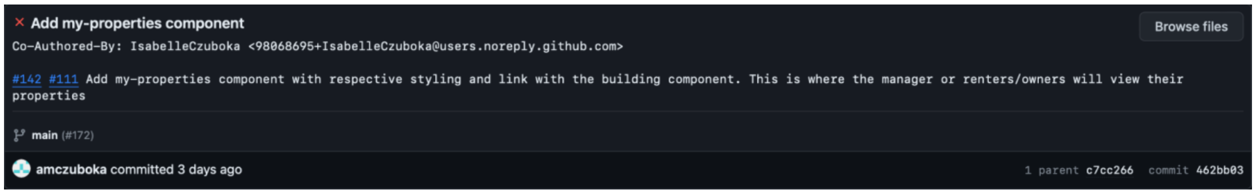
Task: Create acceptance test for viewing/editing the employees in "My employees" page `SP: 2` `Task` `Testing` `#17` — 2
#122 opened on Jan 31 by IsabelleCzuboka · Sprint 3

Task: Implement the back-end for the "My employees" page `back-end` `SP: 2` `Task` `#17` — 1
#120 opened on Jan 31 by IsabelleCzuboka · Sprint 3

○ **Links between commits and bug reports/features**

Links between commits and bug reports or features are very important when it comes to traceability. When looking at the main branch commits, it's easy to link every commit to a specific feature or bug and know which context changes were made. To add on, adding links to the tasks are important for managers of the team to get a quick look at what has been accomplished so far and assess the progress based on these commits. Furthermore, links to features/bugs allow for better organization and can be used in documentation later on for onboarding purposes.

Below is an example of a link to tasks for a feature for a commit



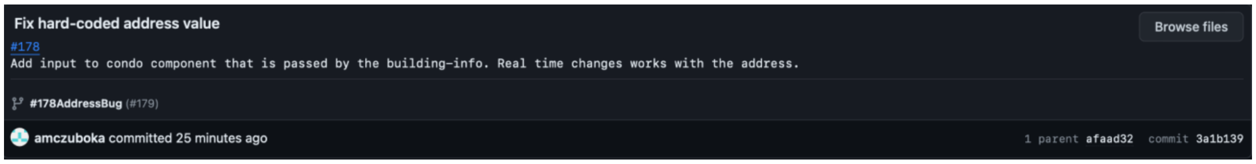
✕ Add my-properties component — Browse files

```
Co-Authored-By: IsabelleCzuboka <98068695+IsabelleCzuboka@users.noreply.github.com>

#142 #111 Add my-properties component with respective styling and link with the building component. This is where the manager or renters/owners will view their
properties
```

main (#172)

amczuboka committed 3 days ago — 1 parent c7cc266 commit 462bb03

Below is an example of a link to a bug task for a commit



Fix hard-coded address value

#178

Add input to condo component that is passed by the building-info. Real time changes works with the address.

#178AddressBug (#179)

amczuboka committed 25 minutes ago

1 parent afaad32 commit 3a1b139