Google Summer of Code

Project Details

Organisation: Chromium

Project: Snoozing, Filtering and Searching for Issues in Issues Tab

Mentors: Wolfgang Beyer, Sigurd Schneider

Personal Details

Name: Rahat Muneeb

Email: irahatmuneeb@gmail.com

Contact: +447760373095

University: University of Bristol

GitHub: github.com/rahahahat

Table of Contents

Abstract

Project Description

The aim of this project is to augment the functionality of the Issues tab in Chrome DevTools by implementing Snooze, filter and search functionalities. These features will promote a better developer experience, clean the UI of the issues tab and improve the usability when the amount of added issues is high.

Main Goals of the Project

- **Snoozing Issues**

  The feature allows users to snooze certain types of issues, so they are not displayed in the Issues Tab at all. The functionality extends to un-snooze issues as well.

- **Filtering Issues**

  Through this feature users can change the visibility of issues for a debugging session by allowing them to filter different types of issues.

- **Searching for Issues**

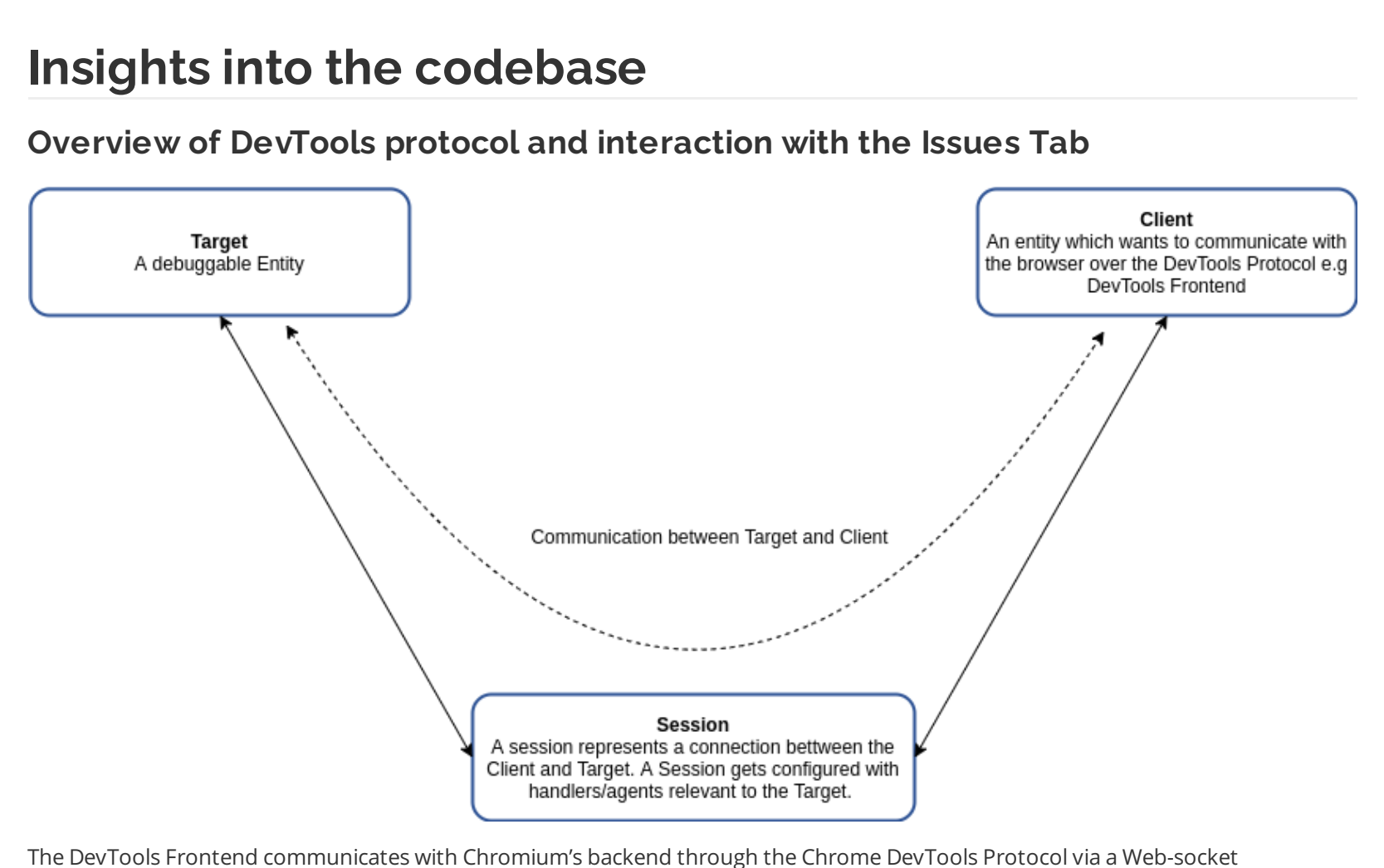
  This feature is similar to the search feature available in the console panel. This feature will allow users to search for particular types of issues.

Additional Project Ideas

- **Cross-linking between issues tab and other parts of DevTools**
- **Differentiate between third-party and first-party issues**
- **Port UI code to custom components using Lit-HTML**

Insights into the codebase

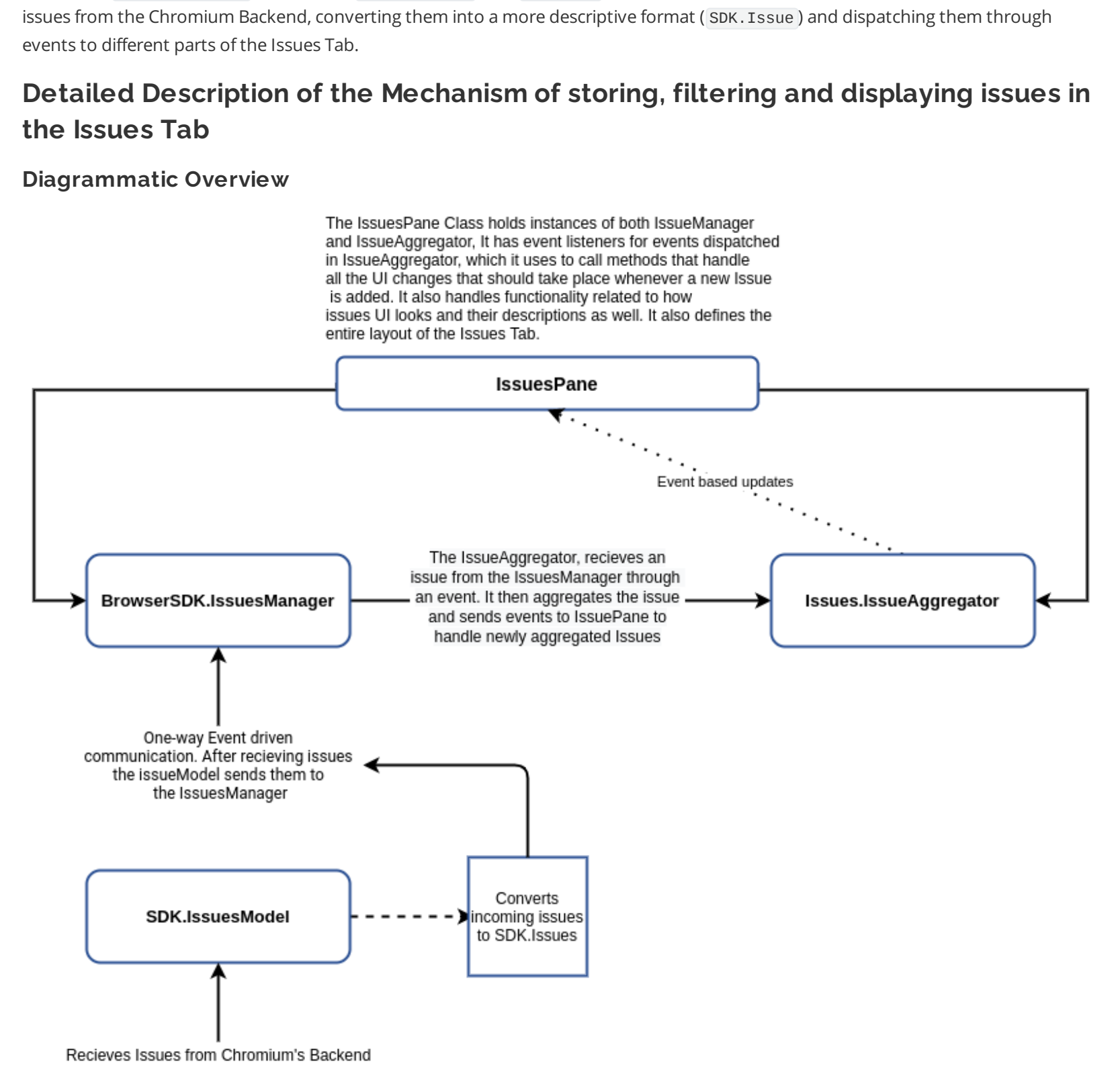
Overview of DevTools protocol and interaction with the Issues Tab



The DevTools Frontend communicates with Chromium's backend through the Chrome DevTools Protocol via a Web-socket connection.
A Client (e.g. DevTools Frontend) connects with a `Target`, once it is connected, a session object which gets configured with `Target` relevant agents/handlers is created, this enables the client to send commands over the DevTools Protocol. In context of the Issues Tab, The Model `SDK.IssueModel` extends the `SDK.SDKModel` (The `SDKModel` is initialised with a Target) and is responsible for receiving all issues from the Chromium Backend, converting them into a more descriptive format (`SDK.Issue`) and dispatching them through events to different parts of the Issues Tab.

Detailed Description of the Mechanism of storing, filtering and displaying issues in the Issues Tab

Diagrammatic Overview



Storing Issues

Following the diagram above, issues are received in `SDK.IssueModel` as instances of `InspectorIssue`. Upon receiving an `InspectorIssue` it gets converted into a particular type of issue class which extends `SDK.Issue` depending on its `code` through the `_createIssuesFromProtocolIssue` method. After the issue has been converted it then gets dispatched via the `IssueModel.Events.IssueAdded` event to the `IssueManager`.

code snippet:

```
issueAdded(issueAddedEvent) {
  const issues = this._createIssuesFromProtocolIssue(issueAddedEvent.issue);
  for (const issue of issues) {
    this.addIssue(issue);
  }
}
```

Upon receiving the issue in the `IssueManager` it gets added to a set of all issues and then filtered by the `issueFilter` method. If the issue passes the filter it gets added to the `filteredIssues` Map and is then dispatched through the `IssueManager.Events.IssueAdded` event to the `IssueAggregator`.

code snippet:

```
private onIssueAdded(event: Common.EventTarget.EventTargetEvent): void {
  const {issuesModel, issue} = event.data as {
    issuesModel: SDK.IssuesModel.IssuesModel,
    issue: SDK.Issue.Issue,
  };
  if (!issue.getDescription()) {
    return;
  }
  const primaryKey = issue.primaryKey();
  if (this.allIssues.has(primaryKey)) {
    return;
  }
  this.allIssues.set(primaryKey, issue);
  if (this.issueFilter(issue)) {
    this.filteredIssues.set(primaryKey, issue);
    this.issueCounts.set(issue.getKind(), 1 +
(this.issueCounts.get(issue.getKind()) || 0));
    this.dispatchEventToListeners(Events.IssueAdded, {issuesModel, issue});
  }
  this.dispatchEventToListeners(Events.IssuesCountUpdated);
}
```

In the `IssueAggregator`, after an issue has been received it gets aggregated into an instance of `AggregatedIssue` and stored in `AggregatedIssueByCode` based on its `issue.code`. After this is done the entire `AggregatedIssue` instance is sent via the `IssueAggregator.Events.AggregatedIssueUpdated` event to the `IssuesPane` class. Upon receiving this event the `IssuesPane` class then updates the UI of the Issues Tab.

code snippet:

```
private onIssueAdded(event: Common.EventTarget.EventTargetEvent): void {
  const {issue} = (event.data as {
    issuesModel: SDK.IssuesModel.IssuesModel,
    issue: SDK.Issue.Issue,
  });
  this.aggregateIssue(issue);
}
private aggregateIssue(issue: SDK.Issue.Issue): AggregatedIssue {
  let aggregatedIssue = this.aggregatedIssuesByCode.get(issue.code());
  if (!aggregatedIssue) {
    aggregatedIssue = new AggregatedIssue(issue.code());
    this.aggregatedIssuesByCode.set(issue.code(), aggregatedIssue);
  }
  aggregatedIssue.addInstance(issue);
  this.dispatchEventToListeners(Events.AggregatedIssueUpdated, aggregatedIssue);
  return aggregatedIssue;
}
```

Filtering Issues

Currently the only type filtering implemented in the Issues Tab is filtering for issues that contain third party cookies. This filtering is achieved through attaching a `changeListener` to a `Common.Setting` created specifically for this purpose in the `SDK.Issue.ts` file. The changes to this `Setting` are triggered by a checkbox present in the Issues Tab toolbar. After a change has been triggered (checked or unchecked), the entire `Setting` is updated with a new set of values and the `changeListener` calls the `updateFilteredIssues` method. This method then clears the entire `filteredIssues` Map and repopulates it depending upon the filter chosen by the updated `Setting`. After the map has been repopulated, `IssueManager.Events.FullUpdateRequired` and `IssueManager.Events.IssuesCountUpdated` events are dispatched which after causing multiple changes in listening classes lead to the issues rendered in the Issues Tab to change.

code snippet:

```
if (!this.showThirdPartySettingsChangeListener) {
    const showThirdPartyIssuesSetting =
SDK.Issue.getShowThirdPartyIssuesSetting();
    this.showThirdPartySettingsChangeListener = showThirdPartyIssuesSetting
      .addChangeListener(() => { this.updateFilteredIssues(); });
}
private updateFilteredIssues(): void {
  this.filteredIssues.clear();
  this.issueCounts.clear();
  for (const [key, issue] of this.allIssues) {
    if (this.issueFilter(issue)) {
      this.filteredIssues.set(key, issue);
      this.issueCounts.set(issue.getKind(),
    1 + (this.issueCounts.get(issue.getKind()) ?? 0));
    }
  }
  this.dispatchEventToListeners(Events.FullUpdateRequired);
  this.dispatchEventToListeners(Events.IssuesCountUpdated);
}
```

Displaying Issues

The entire mechanism of displaying issues is handled by the `IssuesPane` class. The `IssuesPane` class holds instances of `IssueManager` and `IssueAggregator` which it uses to display issues. The `IssuesPane` class extends the `UI.Widget.VBox` and registers itself as a `shadowRoot` component. It handles the creation of the toolbars through the `createToolbars()` method. The actual visual representation of what the UI of different types of issues looks like is contained in the `IssuesView` class. The `IssuePane` contains event listeners for `IssueAggregatorEvents.AggregatedIssueUpdated` and `IssueAggregatorEvents.FullUpdateRequired`, which call the `issueUpdate` and `fullUpdate` methods of the `IssuePane` class respectively. The `issueUpdate` method ultimately leads to the execution of the `updateIssueView` method. This method is responsible for creating the issue description from pre-supplied markdown descriptions and constructing an `IssueView` with the same. After this is done the `IssueView` gets appended to `issueTree` (extends the `UI.TreeOutline.TreeOutline` class) depending upon its category and is then shown in the Issues Tab. The `fullUpdate` method clears the current instances of `IssueViews` and `categoryViews`, and then calls the `updateIssueView` method on every aggregated issue present in the `IssueAggregator` instance.

code snippet:

```
private async updateIssueView(issue: AggregatedIssue): Promise<void> {
  let issueView = this.issueViews.get(issue.code());
  if (!issueView) {
    const description = issue.getDescription();
    if (!description) {
      console.warn('Could not find description for issue code:',
issue.code());
      return;
    }
    const markdownDescription = await
    createIssueDescriptionFromMarkdown(description);
    issueView = new IssueView(this, issue, markdownDescription);
    this.issueViews.set(issue.code(), issueView);
    const parent = this.getIssueViewParent(issue);
    parent.appendChild(issueView, (a, b) => {
      if (a instanceof IssueView && b instanceof IssueView) {
        return a.getIssueTitle().localeCompare(b.getIssueTitle());
      }
      console.error('The issues tree should only contain IssueView objects as
direct children');
      return 0;
    });
  }
  issueView.update();
  this.updateCounts();
}
```

```
private fullUpdate(): void {
  this.clearViews(this.categoryViews);
  this.clearViews(this.issueViews);
  if (this.aggregator) {
    for (const issue of this.aggregator.aggregatedIssues()) {
      this.scheduleIssueViewUpdate(issue);
    }
  }
  this.updateCounts();
}
```

Note

Due to the refactor proposed in (http://crbug.com/1192078) the process of transforming an `InspectorIssue` into a specific `SDK.Issue` will be handled by the `IssueManager` class rather than the `IssuesModel` class. This refactor could lead to some additional changes as well. However, the underlying mechanism of how issues are receiving, stored and displayed will mostly remain the same.

Prototypes of Project Bugs

I have been experimenting with DevTools codebase and working on prototypes of the project bugs. My approach for making the prototypes for the project bugs has been inspired by examples of filtering spread throughout the codebase for e.g `ConsoleFilter`, `ConsoleViewFilter` filtering `third-party` issues in the Issues Tab. Below I have added a link to a video which showcases these prototype features.

Prototypes

- **Snoozing of Issues.**
- **Searching and Filtering of Issues.**

Note: I have added some mock Issues from the `IssuesModel` class to help me test and demonstrate these prototypes

Link: https://www.youtube.com/watch?v=4rtLk7pcXHs

Implementation Details

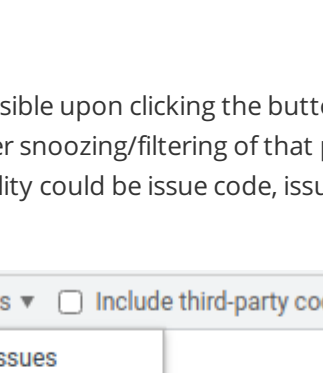
The main goals of the project can be achieved in three parts

UI Components

All the UI components can be appended to the Issues Tab toolbar. The UI components needed for the the main goals of this project are:

- `UI.Toolbar.ToolbarButton`
  - This component is the button that will be used to display a menu with multiple items. Through an event listener, listening for click events, this button can make the menu visible.

  UI example:

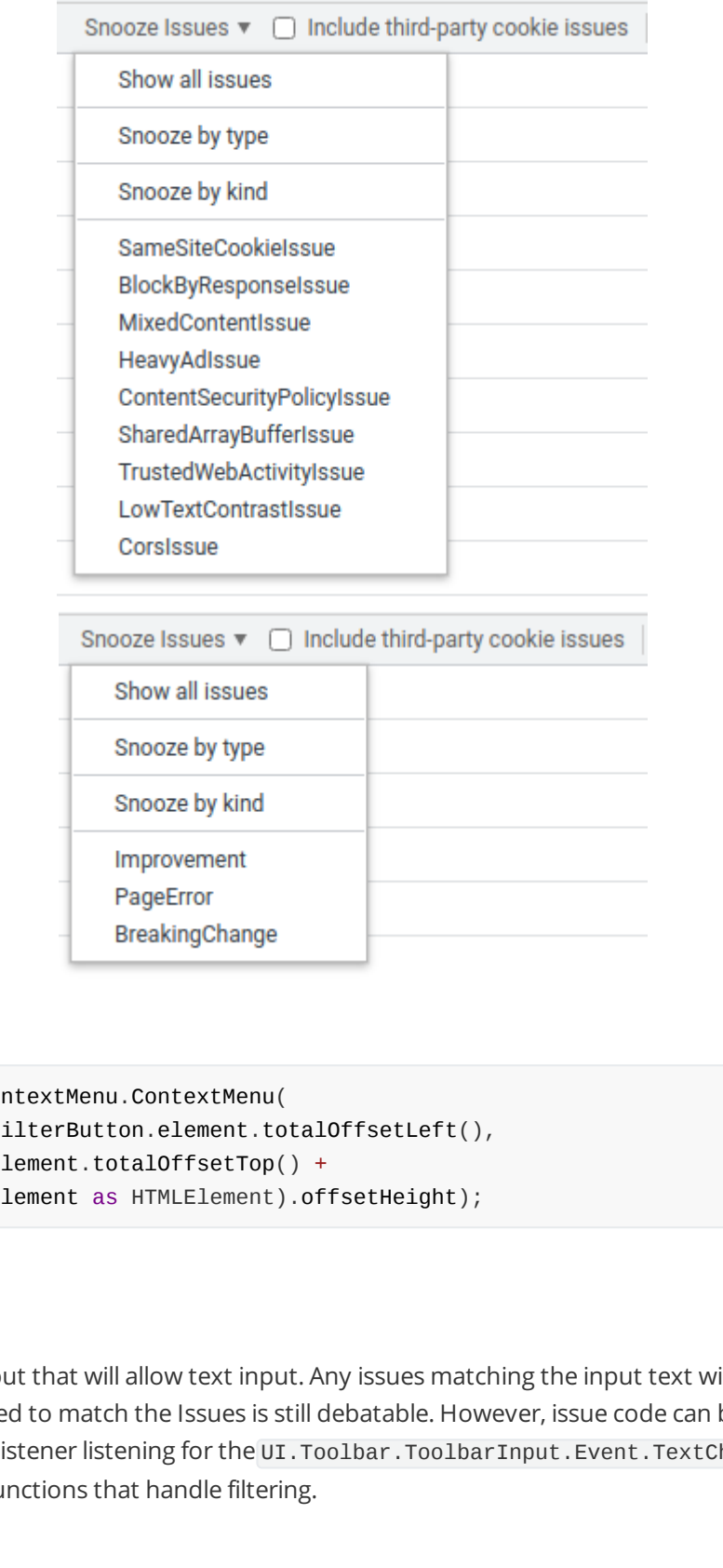
  

  code snippet:

  ```
  new UI.Toolbar.ToolbarButton('Snooze Issues');
  ```

- `UI.ContextMenu.ContextMenu`
  - This component is the menu which will become visible upon clicking the button. The menu can be populated with multiple checkboxes, each of which could be used to trigger snoozing/filtering of that particular item. The property of issues that could be used for the snoozing/filtering functionality could be issue code, issue kind or both.

  UI Examples:

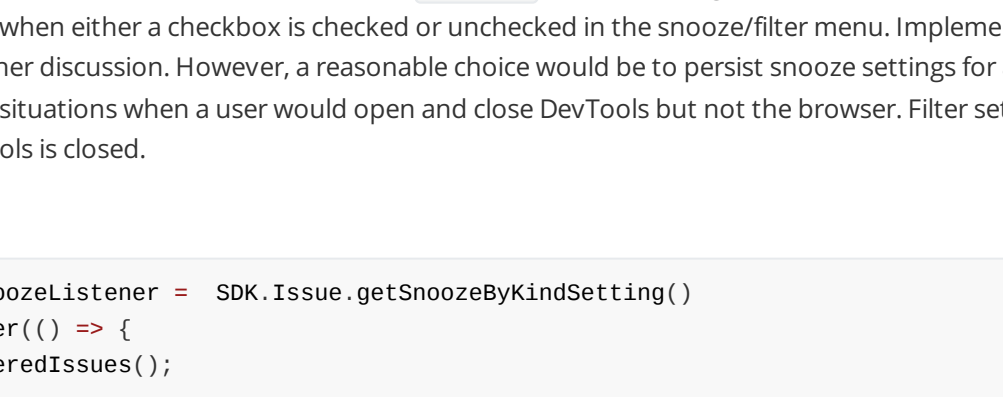
  

  code snippet:

  ```
  const menu = new UI.ContextMenu.ContextMenu(
      event, true, this.filterButton.element.totalOffsetLeft(),
      this.filterButton.element.totalOffsetTop() +
          (this.filterButton.element as HTMLElement).offsetHeight);
  ```

- `UI.Toolbar.ToolbarInput`
  - This component will be the input that will allow text input. Any issues matching the input text will be shown in the Issues Tab. The property of Issues used to match the Issues is still debatable. However, issue code can be a reliable parameter. This UI element will have an Event-listener listening for the `UI.Toolbar.ToolbarInput.Event.TextChanged` event. This listener will be responsible for calling functions that handle filtering.

  UI example:

  code snippet:

  ```
  new UI.Toolbar.ToolbarInput('Filter/Search Issues', '', 0.2, 1, 'Issues');
  ```

Settings

Inspired by the mechanism of filtering implemented in include `third-party cookie issues checkbox`, `consoleViewFilter` and `consoleFilter` classes. To implement the Snooze and filter functionalities we would need to declare multiple `Common.Settings.Setting(s)` and then depending upon whether issues are snoozed or searched, switch between multiple `Settings` and use them to achieve the intended effect. All `Settings` will have change listeners attached to them, and in this case a change will be triggered when either a checkbox is checked or unchecked in the snooze/filter menu. Implementation Details for `Setting(s)` needs further discussion. However, a reasonable choice would be to persist snooze settings for an entire browser session as it would be helpful in situations when a user would open and close DevTools but not the browser. Filter settings in the search bar will be reset once DevTools is closed.

code snippet:

```
this.issueByKindSnoozeListener =  SDK.Issue.getSnoozeByKindSetting()
  .addChangeListener(() => {
    this.updateFilteredIssues();
});
```

In case of searching for Issues, we would need a mechanism that would reset the search settings every time DevTools is closed or opened ,or opt for a different method to do so. To achieve the intended effect we could do the following:

- The following comment is present in the `IssueFilter` method in `BrowserSDK.IssuesManager`

  > The settings change listener can't be set up in IssuesManager's constructor. At that time, the settings storage is not initialized yet, so the setting can't be created.

- This comment supports our approach and enables us to reset the search `Setting` whenever the first batch issues pass through the `issueFilter` Method in `BrowserSDK.IssueManager`.

  code snippet:

  ```
  if (!this.showThirdPartySettingsChangeListener) {
      // issueFilter uses the 'showThirdPartyIssues' setting. Clients of
      // IssuesManager need a full update when the setting changes to get an up-to-date issues list.
      // The settings change listener can't be set up in IssuesManager's constructor.
      // At  that time, the settings storage is not initialized yet, so the setting can't
      // be  created.
      const showThirdPartyIssuesSetting = SDK.Issue.getShowThirdPartyIssuesSetting();
      this.showThirdPartySettingsChangeListener =   showThirdPartyIssuesSetting
      .addChangeListener(() => {
        this.updateFilteredIssues();
      });
    }
    if (!this.searchChangeListener) {
      const searchSettings = SDK.Issue.getIssueSearchFilterSetting();
      this.searchChangeListener = searchSettings.addChangeListener(() => {
        this.settingSearch();
      });
      searchSettings.set("") // This line resets search settings.
    }
  ```

  This way search settings will be reset whenever DevTools is opened.

- Since `BrowserSDK.IssueManager` extends `Common.ObjectWrapper`, we can add an event listener listening for a custom event (`UpdateSearch`). This way can pass an instance of `BrowserSDK.IssueManager` to our Filter class and dispatch the `UpdateSearch` event when text is put in the search bar. This way no `Setting` is initialized.

  code snippet:

  ```
  this.filterInput.addEventListener(UI.Toolbar.ToolbarInput.Event.TextChanged, () => {
      this.issueManager.dispatchEventToListeners(
      BrowserSDK.IssuesManager.Events.UpdateSearch,
      {text: this.filterInput.value()});
  });
  ```

I have tested both these approaches in the prototype and both achieve the intended effect. However, which approach is better suited for implementation needs further discussion.

Filters

Different filter functions need to be implemented for different types of parameters that will be used for filtering or snoozing issues in Issues tab. The filter functions will be switched through UI elements present in the `IssuePane` Toolbar. The parameters (`Issue code` or `Issue kind`) used to filter or snooze issues can be changed by using button triggered menus, these menus will be associated to `Common.Setting.Setting(s)`. Most of the functionality related to filtering issues will be implemented in the `IssueManager` class, as it stores all issues and already has functionality for filtering `third-party cookie issues`. Functionalities like: context switching between multiple filter parameters and instantiating and configuring UI components could be handled by declaring new classes like `IssueFilter` and `IssueFilterView`. We could also refactor code responsible for filtering `third-party cookie issues` into the above mentioned classes, so that all code related to filtering and snoozing is contained in a set of classes. This will allow future improvements to be much smoother and easier.

code snippet:

```
private filterByKind(issue: SDK.Issue.Issue): boolean {
  const filterSetting = SDK.Issue.getSnoozeByKindSetting();
  const filters = filterSetting.get();
  const snooze = !filters[issue.getKind()];
  return snooze && true;
}
```

Timeline

- The timeline is tentative and provides a rough sketch of my planned work. I will try to keep progress ,at the very least, in consonance with the below described timeline. The timeline has been planned keeping in mind the maximum amount of time any task could take and the worst case scenario of dealing with many bugs and feature specific problems. However, most tasks will be done before the proposed temporal boundaries. In which case I would love to implement additional projects ideas and any other features that could be implemented.
- I have no other commitments during the summer and will be able to dedicate as much time needed, except for my summer exams (May 24 - June 14) which overlap with the last 2 weeks community bonding period and first week of the coding period. My availability during these weeks will be lower than normal weeks.
- According to the workload, each week will be roughly divided into research, planning, coding , documenting and testing features. I will also discuss documentation details with my mentors during the community bonding period, and maintain detailed documentation of all work done. This will help reduce a lot of confusion and complexity down the road, as well as provide a reference for future tasks.

Weeks	Tasks
April 13 - May 17 Pre-GSOC Period	I will be focusing all my efforts on solving all the starter bugs mentioned in the project description as well as work on other bugs including the refactoring issues (if allowed) mentioned in http://crbug.com/1192078. I will also spend this time trying to gain more insights about the code base.
May 17 - June 7 Community Bonding	During the community bonding period, I will spend time working with my mentors to frame a detailed project road-map as well as dive deep into the parts of the code base to gain insights that will enable me to work not just on the project bugs but also on the additional project bugs.
June 7 - June 14 Week 1	My summer exams are from May 24 - June 14, so I will not be as available as I will be in upcoming weeks. • Demonstrate my prototype code solution to my mentors and gain feedback regarding the same. • Come up with a detailed architecture of the solution with my mentors.
June 14 - June 21 Week 2	• Setup the folder structure and configure the module setup for the new architecture and classes. • Detail the class specific implementation and implement an initial layout for the new classes.
June 21 - June 28 Week 3	• Implement all the UI components needed for the solution • Discuss `Common.Setting(s)` configuration and implementation with mentors.
June 28 - July 5 Week 4	• Implement project specific functionalities of the Snooze feature in the new classes and connect them with UI component • Extensive manual testing of the snooze feature. • Writing Unit and interactions tests for the newly implemented feature.
July 5 - July 12 Week 5	• Implement project specific functionalities of the filter/ search feature and connect them with UI • Extensive manual testing of search feature. • Writing Unit and interactions tests for the newly implemented feature.
July 12 - 16 Week 6	**Evaluations**
July 19 - July 26 Week 7	• Discuss evaluations with my mentors, gain feedback regarding the newly implemented features and improve their implementation. • Working on Porting Issues Tab UI components to Lit-HTML.
July 26 - August 2 Week 8	• Discussing further steps with mentors. • Dive deep into DevTools protocol details and discuss ways of differentiating between third-party and first party issues with mentors • Implement related functionality, integrate with already existing logic and write tests.
August 2 - August 9 Week 9	• Explore the code base and the DevTools application to study the logic of existing cross-linking between various sections/panels of DevTools Frontend codebase. • Discuss with mentors and implement cross-linking between multiple parts of DevTools with Issues.Tab.
August 9 - August 16 Week 10 (Last Week)	• Work on bug fixes, code refactor and testing. • Improvements based on feedback from mentors and community.. • Work on project documentation and finalising the project.
August 16 - 30	**Final Evaluations**

About Me

I am a second year international student, studying computer science and electronics at the University of Bristol. I derive my passion for computer science from the fact that it enables me to collect and express my ideas. It is a discipline where I can draw satisfaction from the process of creating something just as much as the final result. Since the past couple of months I have invested time in open-source and have become increasingly passionate about it. What fascinates me most about open sourcing is the opportunity it provides for anyone to work to a professional standard at the forefront of technology. I believe that in the current environment, from code practices to architectural designs, open-sourcing provides everyone with a platform to experience the highest echelon of software development, It gives everyone a chance to learn and interact with industry professionals and contribute to projects that impact the world. I also believe that as software developers it is our responsibility to chart ways for upcoming developers and contribute our time and effort towards the betterment of the community, open sourcing provides a brilliant way to achieve that, helping others along the way.

Why am I interested in Chromium and this project

The impact chromium has over the internet and people's daily lives is no small feat at all. Chromium has changed the way people interact with the internet, It has revolutionised the entire experience of browsing the internet and paved ways for numerous technologies that have revolutionised the Web. Contributing to such an impactful project and witnessing the positive impact of my contributions excites my passion and drives me to work harder. With the current trend of working with high performance frameworks like Angular.js, React.js and Vue.js, working on Chrome DevTools has made me appreciate the power of pure JavaScript. It exposed me to architectural patterns and good code practices that have helped me strengthen my programming skills and code quality. The kind of exposure and work experience I would gather working on this project will help me improve multi-fold as a computer scientist as well. Speaking from a developer's point of view this feature will reduce all extra noise from the issues tab and help developers concentrate and eliminate one issue at a time. This could massively increase productivity as well as help reduce a lot of confusion and complexity. Not only this but it also promotes a cleaner UI and would enable developers to snooze issues that aren't directly caused by the consequences of the code they are working on e.g. Third-party issues or search for Issues they specifically want to target.

Projects and Work Experience

Software Developer / Hatless Studios

During my time at Hatless I have worked on numerous projects, some of which include GoTLearning (Growth over time), which is a Silicon Valley startup that aims to track the intellectual growth of children throughout their high school through thorough analysis of their performance and actively monitored conversations with their peers and teachers on the platform regarding curricular and extracurricular activities.

Team Lead / CSRN

CSRN is a network of student-led consultancies from world leading universities working together to offer management consultancy services to social impact and charitable organisations impacted by crises. I am a member and lead of the core development team, working on automating their response process, by developing a web application which will serve as a platform that connects established student or independent consultancies with charitable organisations in need of support. The platform is in production and is currently being used by 25+ consultancies worldwide.

Contributions

Contributions to Chrome DevTools

- https://chromium-review.googlesource.com/c/devtools/devtools-frontend/+/2764413
- https://chromium-review.googlesource.com/c/devtools/devtools-frontend/+/2784966

I will continue to contribute to the DevTools code past the application deadline and keep working on the starter bugs.

Contributions to other Open-source projects

- https://github.com/WikiEducationFoundation/WikiEduDashboard/pull/4264
- https://github.com/WikiEducationFoundation/WikiEduDashboard/pull/4218
- https://github.com/WikiEducationFoundation/WikiEduDashboard/pull/4284