A Modular Approach To The App Inventor Projects View

Modernising and modularising the App Inventor Projects View by introducing shortcuts for project actions and integrating markup UI declarations with standardised design systems

MIT App Inventor
Google Summer of Code 2021

Vishwas Adiga
vishwasadiga@gmail.com

Full application at gsoc-21.vishwasadiga.me

Abstract

The App Inventor front-end, a subset of the larger appinventor-sources monorepo, is a complex and dynamic code-base written almost entirely in GWT (Google Web Toolkit). Alongside the Blockly editor sources, it handles everything that the end user can see and interact with — from dragging and dropping components to creating and deleting projects. The intricate symphony of a multitude of listeners and handlers enables App Inventor to provide a functional online development environment and hence democratise mobile app development.

However, as the project has grown, it has become increasingly more complex to add new features to the front-end. With every widget dependent on every other in some form or the other, an ideal separation of concerns has become difficult to ascertain. In addition, the introduction of each new feature has brought along with it a slightly different widget design than the standard, eventually racking up several design inconsistencies and workflow redundancies.

This project proposes to revamp the Projects View of the App Inventor interface and introduce shortcuts for essential project actions. The aim is to have a modern, modular, and fully functional user interface that leverages the latest advances in markup UI declarations, design systems, and accessibility to deliver an improved look and feel to App Inventor users. By making use of strong design pillars, the proposal intends to streamline app development workflows and ensure an enjoyable and accessible experience to all stakeholders and users.

Project Description and Objectives

The Projects View is a major component of the App Inventor front-end that handles all actions related to user projects. It is also the first view that the user sees when visiting App Inventor, and is thus effectively an entry point to the development experience. Hence, it is imperative to provide a usable, equitable, and efficient interface that sets the tone for what the user can expect as they dive further into the Online Development Environment (ODE).

One of the primary goals of the project is to modularise the Projects View and have it be a standalone bundle that can be used independently. This would enable easier debugging and addition of new features. Furthermore, a modular system would introduce stronger separation of concerns in the code-base and allow new external contributors to get started quicker than ever before. By leveraging GWT's Lazy Loading and module splitting features, the project aims to improve load times and reduce the bundle size of each module. A similar approach can then be taken for other parts of the App Inventor front-end to further modularise the code-base.

Another objective of the project is to ease access to project actions (e.g., create & delete projects, export AIA and APK/AAB) by placing them closer to each project item in the projects list. By extension, a user should also be able to batch these actions, thus affecting multiple projects at once. These changes will streamline project workflows, especially benefiting stakeholders like educators who often deal with multiple projects at once. Project imports are also slated to be improved, with abilities to auto-rename files with conflicting names.

Finally, this project aims to overhaul the Projects View and bring it up to modern accessibility and design standards. A fresher look for the interface would go a long way in providing an enjoyable development experience to users. The findings and work done in this project can also be extrapolated to the entire App Inventor front-end, thus ensuring a uniform user interface, thereby easing the learning curve.

Design Pillars

Design pillars are fundamental rules that set the tone for how the project should proceed and be implemented. They provide a filter for ideas and a tenet to weigh design decisions against. A few design pillars that would help set the tone for future discussion are

- **Accessibility and clarity**
  All proposed ideas and interfaces must be accessible to individuals with disabilities.
- **Control**
  The user must always be in control of their experience with App Inventor.
- **Standardisation**
  Widgets must be standard; both among other widgets of the same kind and among similar widgets across the internet.
- **Brand**
  Ideas and interfaces must reflect and promote the ethos of the App Inventor project.

Implementation

The first step to implementing the new Projects View would be to gather feedback on how users currently make use of the projects panel. Input from the Community and other major stakeholders would pave the way for an initial understanding of the fundamental changes required. Wireframes and low-fidelity mockups can then be created to solicit further feedback.

With the overall layout of the new Projects View ready, work can begin on implementing the interface in GWT. UiBinder[1] would be the preferred medium of implementation, considering its close semblance to JSX syntax given App Inventor's plan to transition to a React-like markup in the long run. UiBinder would also greatly simplify the DOM and ease inspection and UI testing.

For example, a project list item can be written in markup as:

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
    xmlns:g='urn:import:com.google.gwt.user.client.ui'>
  <g:HorizontalPanel>
    <!-- Name -->
    <g:Label>MyProject</g:Label>
    <!-- Date created -->
    <g:Label>Apr 8, 2021, 1:59:20 AM</g:Label>
    <!-- Date modified -->
    <g:Label>Apr 8, 2021, 2:14:57 AM</g:Label>
    <!-- Shortcuts -->
    <g:Button ui:field='downloadBtn'>Download .AIA</g:Button>
    <g:Button>Export .APK</g:Button>
  </g:HorizontalPanel>
</ui:UiBinder>
```

[1] http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html

These widgets can then be bound to their respective events in the class that builds the UI, as demonstrated:

```
public class ProjectsList extends Composite {
  // create reference to button as declared in XML
  // this will automatically be linked to the button
  // object when the UI is built.
  @UiField Button downloadBtn;

  @UiHandler("downloadBtn")
  private void handleClick(ClickEvent e) {
    // fetch .AIA using ProjectService and trigger link
  }
}
```

This approach to handling events greatly simplifies the front-end and keeps the code DRY.

In conjunction with a design system or an optional CSS framework, it would also be possible to implement scoped styles[2], thus eliminating the need to maintain a single resource like Ya.css.

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
    xmlns:g='urn:import:com.google.gwt.user.client.ui'>
  <ui:style field='liStyles' src="ProjectListItem.css">
  <g:HorizontalPanel>
    <!-- Name -->
    <g:Label
        class='{liStyles.projectLabel}'>
      MyProject
    </g:Label>
  </g:HorizontalPanel>
</ui:UiBinder>
```

[2] http://www.gwtproject.org/javadoc/latest/com/google/gwt/resources/client/CssResource.html

`ProjectListItem.css` can thus be its own CSS file that can be managed independently of other stylesheets.

Alternatively, a CSS framework like TailwindCSS[3] can be used alongside scoped stylesheets to leverage standardised style declarations, as shown:

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
    xmlns:g='urn:import:com.google.gwt.user.client.ui'>
  <g:HorizontalPanel>
    <!-- Name -->
    <g:Label class='text-lg font-bold'>
      MyProject
    </g:Label>
  </g:HorizontalPanel>
</ui:UiBinder>
```

With these requirements considered, it is reasonable to envision the source code laid out in the following file structure:

- `appengine/src/.../client/views/`
  - This directory will house all modular views, including the proposed Projects View.
  - Having the new view be placed high in the package tree would be beneficial in ensuring complete separation between modules, especially in the long run

- `appengine/src/.../client/views/projects`
  - This is where all files pertaining to the modular Projects View will reside.
  - It should be noted that none of the files in this package should be used elsewhere except the composition class that is used to instantiate the UI lazily.

- `appengine/src/.../client/views/projects/ProjectsView.ui.xml`
  - This file will contain UI declarations in markup.

- `appengine/src/.../client/views/projects/ProjectsView.java`
  - This file will accompany the XML declarations and manipulate views that have been declared.
  - Event handlers, fetch operations, and other business logic will go here.

[3] https://tailwindcss.com/

- `appengine/src/.../client/views/projects/styles.css`
- `appengine/war/static/projects/styles.css`
  - The styles pertaining to the proposed module is to go in either of these files.
  - While GWT recommends the former, it may be more appropriate to go with the latter given App Inventor's pre-existing stylesheet file structure.

From a technical perspective, work will need to be done on integrating project folders into the interface in a way that is intuitive and accessible. In addition, optimisations and memoisations will have to be made when providing shortcuts for project actions. For example, a request to export an APK should return a pre-built binary if no changes were made to the project since the last build[4]. The timestamp of the last save or an equivalent function that takes the AIA as a parameter is ideal to check for any changes in the project.

Some of the changes needed to make this work are:

`ServerLayout.java`
A new server endpoint will have to be added that will enable handling multiple APK/AAB exports

```
public static final String DOWNLOAD_SELECTED_PROJECTS_OUTPUT =
"selected-projects-output";
```

`FileExporter.java`
A new method is required that will iterate over selected projects and return a zipped file. This will work similar to `FileExporter::exportSelectedProjectsSourceZip`

```
ProjectSourceZip exportSelectedProjectsOutputFile(String
userId, List<Long> projectIds, @Nullable String target)
      throws IOException;
```

`DownloadServlet.java`
A subroutine that will invoke the newly defined `FileExporter` method if multiple APK exports are requested needs to be added

[4] https://community.appinventor.mit.edu/t/shortcuts-for-project-actions-project/5643/2

UserProject.java A new field (and corresponding setter/getters) will have to be added that stores the timestamp of the last build. It may be more appropriate to have separate timestamps for different build types (perhaps in an extensible manner using a Map) that can accommodate future build targets like iOS
`private long lastBuildDate;`

In the “export selected” button handler in the new Projects View, a check will have to be added that compares the last modified and last built timestamps of each project, and requests a build if the built binary is stale.

```
if(userProject.getDateModified() > userProject.getDateBuilt())
{
  // call BuildCommand build chain
  // further discussion required on making this async/calling
  // multiple builds at once

  // a progress dialog can also be shown at this stage
  // to provide clarity and context to users
} else {
  // skip building for this project (i.e., do nothing)
}
```

This exercise would also provide ample opportunity to implement auto-renaming of projects with conflicting names[5], thus greatly aiding educators and other users who often deal with multiple files with possibly same names. An initial implementation would involve tweaking the ProjectUploadWizard to intelligently handle conflicting names. Similar changes will have to be made to `HTML5DragDrop.java` and its corresponding JavaScript file to mimic the same behaviour when importing projects via DnD.

[5] https://github.com/mit-cml/appinventor-sources/issues/2453

UserProject.java

```
while(!TextValidators.checkNewProjectName(filename)) {
  filename += UNIQUE_IFIER;
}
```

In addition, project memos[6] can be integrated into the new projects list, thus enabling users to add app-wide notes to their work. This can be implemented by adding a new ProjectProperty[7] that holds the memo string.

Because the new view is going to be independent of other modules, it can be separated into its own bundle and loaded lazily as and when required. This would also set the groundwork for further modularisation of the interface. GWT's lazy loading[8] and module splitting capabilities can be leveraged for this purpose.

Finally, an option would have to be implemented, preferably in the settings drop-down, to let users opt-in to the new interface. This would give stakeholders the flexibility to switch per their own convenience and pace. This would ideally be in the form of a new UserSetting that would hold a Simple True/False flag indicating the user's preference.

TopToolbar.java::createSettingsMenu

Add a new setting menu item. This setting can be used for all UI modularisations that may be made in the future. Alternatively, separate settings could be provided for each view that is refactored

```
if (Ode.getUserNewInterface()) {//as fetched from a UserSetting
  settingsItems.add(
      new DropDownItem(WIDGET_NAME_NEW_INTERFACE,
          MESSAGES.switchToOldInterface(),
      new SetOldInterfaceAction()));
} else {
  settingsItems.add(
      new DropDownItem(WIDGET_NAME_NEW_INTERFACE,
```

[6] https://github.com/mit-cml/appinventor-sources/issues/1172
[7] https://github.com/mit-cml/appinventor/appengine/.../shared/settings/SettingsConstants.java
[8] http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html#Lazy

```
          MESSAGES.switchToNewInterface(),
      new SetNewInterfaceAction()));
}
```

Building on the design pillar to provide control to the user, it is reasonable to have a flag that lets owners and operators of App Inventor's forks decide whether they would like to provide multiple APK exports from the projects list. Technical constraints could hinder some systems from handling a surge in buildserver requests, and the option to disable the feature would help greatly in easing server traffic. A sample implementation would look like:

`AppInventorFeatures.java`
A new flag that allows for enabling/disabling of multiple project exports from the new Projects View

```
public static boolean allowMultipleProjectExports() {
  return true;
}
```

Deliverables

The project aims to deliver a modular, new Projects View that integrates shortcuts for most used actions and a standard, modern design. This new interface will be implemented as an opt-in alongside the current interface[9] to ensure interoperability with existing App Inventor curricula and learning resources.

If time permits, the project aims to expand the process of modularisation and standardisation of the interface to other parts of the front-end. This would be accompanied by applying the same design changes to all of the UI, thus ensuring a standard experience across views.

9 https://community.appinventor.mit.edu/t/question-regarding-the-shortcuts-for-project-actions-proposal/29867/2

Timeline

May 17, 2021 - June 7, 2021	Wireframing, making mock-ups, and developing design systems in consultation with the Community
June 8, 2021 - June 21, 2021	Research best implementation pathway, conduct technological feasibility studies and finalise look and feel of new interface
June 22, 2021 - July 26, 2021	Implementation of the new Projects View
July 27, 2021 - August 8, 2021	Testing based on Community feedback and refinements
August 9, 2021 - August 15, 2021	Buffer week to account for last minute changes and the unexpected
August 16, 2021	Final submission