








Streamline App Development with Gemini Code Assist



Learn about using generative AI in your applications, the foundation models provided by Google, and the challenges you may encounter when building generative AI applications.

Click the first lesson—or the **Start Course** button—to begin.

LESSONS

-  Getting Started
-  Introduction to Gemini Code Assist
-  Gemini Code Assist for New Developers
-  Gemini Code Assist for Experienced Developers
-  Summary

Getting Started

Generative AI is being used to power so many experiences these days. App developers want to build compelling applications that are powered with artificial intelligence (AI). Generative AI can also be used to make it easier to build applications and perform application development tasks. [Gemini Code Assist](#) offers AI-powered assistance to help developers build applications with higher velocity and better quality.

Designed for developers of all levels, this course will introduce you to the core features and functionalities of Gemini Code Assist, an AI-powered application development collaborator for Google Cloud. From intelligent code suggestions and auto-completion to real-time error detection and refactoring assistance, you'll discover how Gemini Code Assist can significantly enhance your productivity and code quality and save valuable time to focus on more productive and enjoyable tasks.

Course learning objectives



- Use Gemini Code Assist to explain code.
- Use Gemini Code Assist to make code more readable.
- Use Gemini Code Assist to add comments to code, in English or another language.
- Generate unit tests with Gemini Code Assist.

Introduction to Gemini Code Assist

How cloud users spend their time

Cloud development is difficult. Cloud users spend too much time trying to understand the cloud and how to use it. Developers can better spend their time focusing on creative tasks like creating innovative applications.

Cloud developers spend too much time on:



Developer onboarding: Learning a new complex codebase can lead to significant delays before developers become productive.



Context-switching: Leaving the integrated development environment (IDE) to search documentation pages can affect productivity and cause stress.



Learning cloud services: Google Cloud provides multiple options for services, like application hosting and databases. Choosing between services and learning the options available for each service can be time-consuming.



Finding best practices: Determining best practices for cloud products can be difficult.

Gemini for Google Cloud



The [Gemini for Google Cloud](#) portfolio offers generative AI-powered assistance to a wide range of Google Cloud users, including Gemini Code Assist for application developers. To provide an integrated assistance experience, Gemini for Google Cloud is also embedded in many Google Cloud products, including [Colab notebooks](#), [databases](#), and [BigQuery](#).

As an application developer, [Gemini Code Assist](#) can help with many of your daily app development tasks. Gemini Code Assist provides an AI-powered collaborator to help developers and developer teams build, deploy, and operate applications throughout the development cycle.

Gemini Code Assist



You can use Gemini Code Assist in your IDE for AI-powered coding assistance in many popular languages. Gemini Code Assist uses large language models (LLMs) that are developed by Google. The LLMs are [fine tuned with billions of lines of open source code](#), security data, and Google Cloud-specific documentation and sample code. You get [code completions](#) as you write your code, [generate full functions and code blocks](#) from comments, [generate unit tests](#), and get help with [debugging, understanding, and documenting your code](#).

Gemini Code Assist provides [contextualized responses to your prompts](#), including source citations regarding which documentation and code samples Gemini used to generate its responses. When generating code or offering code completion, Gemini provides citation and license information when it directly quotes at length from another source, such as existing source code.

Supported IDEs

Gemini Code Assist is available in many popular IDEs. To use Gemini Code Assist in an IDE, install the [Cloud Code](#) plugin.

Supported IDEs include:

- [VS Code](#) (within the Cloud Code plugin).
- [JetBrains IDEs](#), including IntelliJ IDEA, PyCharm, WebStorm, and GoLand (within the Cloud Code plugin).
- [Cloud Workstations](#) (built-in support).
- [Cloud Shell Editor](#) (built-in support).

Supported languages

Gemini Code Assist's large language model supports [over 30 languages for prompts](#).

Gemini Code Assist offers assistance with many [coding languages](#) and [code infrastructure interfaces](#), including:

Bash	C	C++	C#	Dart	Go
GoogleSQL	Java	JavaScript	Kotlin	Lua	Matlab
PHP	Python	R	Ruby	Rust	Scala
SQL	Swift	TypeScript	YAML	Google Cloud CLI	Kubernetes Resource Model

Using Gemini Code Assist

You've been introduced to Gemini Code Assist, the AI-powered collaborator for application developers. Gemini Code Assist is available for many popular IDEs, and can be used with many different languages.

To illustrate how helpful Gemini Code Assist can be throughout the software development lifecycle, let's meet some fictional app developers who work at Cymbal Eats, a fictional food delivery service company.

Charlie, new developer



Charlie was just hired at Cymbal Eats as an entry-level developer. Charlie has programming experience designing mobile applications, but he has not developed applications in Google Cloud. Charlie's first language is Italian.

Besides helping with everyday software development tasks, Charlie hopes Gemini Code Assist will help him acclimate to Cymbal Eat's code and architecture.

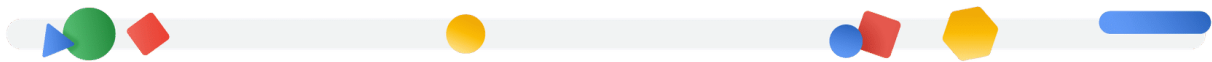
Kalani, lead developer



Kalani has been a lead developer at Cymbal Eats for the past three years. She is the lead developer for the company's backend codebase. She is also responsible for code quality and testing.

Kalani hopes Gemini Code Assist can help improve quality, add unit tests, and improve readability.

In the next two lessons, you'll see how app developers, like Charlie and Kalani, can use Gemini Code Assist to improve everyday application development tasks.



Review

Quiz yourself.

How does Gemini Code Assist work?

- ☐ A platform for sharing and collaborating on code.
- ☐ A service that helps debug and optimize cloud applications.
- ☐ A tool for generating code from natural language descriptions.
- ☐ A core component of Gemini for Google Cloud that provides AI-powered assistance to application developers.

SUBMIT

What is Gemini for Google Cloud?

- ☐ A cloud-based IDE for developing and deploying applications.
- ☐ A large language model trained on code and Google Cloud documentation.
- ☐ A suite of tools designed to enhance the Google Cloud experience with the power of generative AI.
- ☐ A chat interface for interacting with Google Cloud Services.

SUBMIT

Gemini Code Assist for New Developers

Charlie, the new developer



Charlie was just hired at Cymbal Eats as an **entry-level developer** living in Italy. He has programming experience designing mobile applications, but he **has not developed applications in Google Cloud**.

Getting started

It's day one of Charlie's new job at Cymbal Eats. Developers at Cymbal Eats use [Cloud Workstations](#) to develop their apps. Charlie is logged into a workstation, but he is struggling to understand how to start.

[Gemini Code Assist](#) is built into the Cloud Workstations integrated development environment. Gemini Code Assist provides **AI-powered code assistance and natural language chat** to get answers to coding questions, receive guidance on coding best practices, and explain Google Cloud products and features.

Gemini Code Assist is integrated with Cloud Code. To get a better idea about how Cloud Code will help him, **Charlie asks Gemini Code Assist:**



What is Cloud Code?

Gemini Code Assist answers that:

Cloud Code is a powerful extension for Visual Studio Code that makes it easier to develop, debug, and deploy applications on Google Cloud Platform. It's like having a Google Cloud expert right inside your code editor!

Gemini Code Assist also lists features of Cloud Code:

- **Simplified development workflow:** You can easily create, deploy, and manage your applications directly from VS Code.
- **Integrated debugging:** Debug your applications running on Google Cloud without leaving your code editor.
- **Code completion and intelligence:** Cloud Code provides intelligent code completion and suggestions, helping you write code faster and more accurately. It also offers documentation and code snippets for Google Cloud APIs and services.
- **Cloud-native development tools:** Cloud Code includes a variety of tools for working with cloud-native technologies, such as Kubernetes, Cloud Run, and Cloud Functions.
- **Support for multiple languages:** including Java, Python, Go, Node.js, and more.

Gemini Code Assist also lists Google Cloud services that can be utilized from Cloud Code, including Cloud Run, Google Kubernetes Engine (GKE), Cloud Storage, Cloud SQL, and Cloud Logging and Monitoring.

You may explore the full responses that were generated by Gemini Code Assist. Remember, Gemini's generative AI can provide different answers for the same prompt.

Gemini Code Assist full response (Cloud Code)

Cloud Code is a powerful extension for Visual Studio Code that makes it easier to develop, debug, and deploy applications on Google Cloud Platform. It's like having a Google Cloud expert right inside your code editor!

Here's what makes Cloud Code so awesome:

- **Simplified development workflow:** Cloud Code provides a streamlined experience for working with Google Cloud services. You can easily create, deploy, and manage your applications directly from VS Code.
- **Integrated debugging:** Debug your applications running on Google Cloud without leaving your code editor. Cloud Code provides a seamless debugging experience, allowing you to set breakpoints, inspect variables, and step through your code.
- **Code completion and intelligence:** Cloud Code provides intelligent code completion and suggestions, helping you write code faster and more accurately. It also offers documentation and code snippets for Google Cloud APIs and services.
- **Cloud-native development tools:** Cloud Code includes a variety of tools for working with cloud-native technologies, such as Kubernetes, Cloud Run, and Cloud Functions.
- **Support for multiple languages:** Cloud Code supports a wide range of programming languages, including Java, Python, Go, Node.js, and more.

Think of Cloud Code as your one-stop shop for developing and deploying applications on Google Cloud. It's a must-have tool for any developer who wants to take advantage of the power and flexibility of Google Cloud.

Here are some of the key features of Cloud Code:

- **Cloud Run and Cloud Functions:** Deploy and debug your applications on Cloud Run and Cloud Functions directly from VS Code.
- **Kubernetes:** Develop and deploy Kubernetes applications with ease.
- **Cloud Storage:** Access and manage your Cloud Storage buckets directly from VS Code.
- **Cloud SQL:** Connect to and manage your Cloud SQL databases.
- **Cloud Logging and Monitoring:** View logs and monitor your applications running on Google Cloud.

Cloud Code is a powerful and versatile tool that can help you streamline your development workflow and build amazing applications on Google Cloud.

Charlie knows that Cymbal Eats uses both Cloud Run and Google Kubernetes Engine, but doesn't understand their benefits and tradeoffs. Maybe Gemini Code Assist can **help him understand the differences between Cloud Run and Google Kubernetes Engine?**



Compare Cloud Run and Google Kubernetes Engine

Gemini Code Assist breaks down the differences between Cloud Run and GKE and provides a summary of when you should choose each product (full response):

Cloud Run vs Google Kubernetes Engine (GKE): A Comparison

Both Cloud Run and Google Kubernetes Engine (GKE) are serverless platforms offered by Google Cloud, but they cater to different needs and offer distinct advantages. Here's a comparison:

Feature	Cloud Run	Google Kubernetes Engine (GKE)
Abstraction Level	Fully managed, higher abstraction. You focus on code, not infrastructure.	More control and customization. You manage Kubernetes clusters and deployments.
Scalability	Scales automatically from zero to handle traffic spikes. Ideal for sporadic or unpredictable workloads.	Highly scalable, but requires manual configuration or autoscaling setup.
Cost	Pay only for resources consumed during request execution. Can be very cost-effective for infrequent workloads.	Pay for cluster resources even when idle. More cost-effective for consistently high-traffic applications.
Deployment	Deploy containerized applications directly from source code or container images.	Deploy containerized applications using Kubernetes deployments and YAML configurations.
Learning Curve	Easier to learn and use, especially for developers new to containerization.	Steeper learning curve due to Kubernetes concepts and management.
Control & Flexibility	Less control over underlying infrastructure. Limited customization options.	More control over infrastructure, networking,

		and security. Highly customizable.
Use Cases	Microservices, APIs, event-driven applications, web apps with variable traffic.	Complex applications, machine learning workloads, stateful applications, large deployments.

In a nutshell:

- **Choose Cloud Run for:** Simplicity, rapid deployment, automatic scaling, cost-effectiveness for sporadic workloads.
- **Choose GKE for:** Maximum control, customization, complex applications, high-traffic and stateful workloads.

Ultimately, the best choice depends on your specific application requirements, technical expertise, and desired level of control.



Gemini Code Assist helps you understand your development environment and explains the products and services that make up Google Cloud.



Explaining code

Charlie has used Gemini Code Assist to understand the Cymbal Eats development environment and the Google Cloud services that are used for Cymbal Eats apps. Now, Charlie needs help understanding the Cymbal Eats application and codebase. Charlie will first be working on the customer frontend.

Charlie clones the [Cymbal Eats repository](https://github.com/GoogleCloudPlatform/cymbal-eats.git) (git clone https://github.com/GoogleCloudPlatform/cymbal-eats.git) into their workstation, opens the repo folder and opens the setup.sh file in the customer-ui folder:

customer-ui/setup.sh (input) —

```
source ../config-env.sh
export BASE_DIR=$PWD

curl -sL https://firebase.tools | upgrade=true bash

firebase login --no-localhost

firebase --non-interactive projects:addfirebase $PROJECT_ID

export APP_EXISTS=$(curl -H "Authorization: Bearer $(gcloud auth print-access-token)"
https://firebase.googleapis.com/v1beta1/projects/$PROJECT_ID/webApps | grep -o -w
$CUSTOMER_SERVICE_NAME | wc -w)

if [ "$APP_EXISTS" -eq "0" ]; then
firebase --non-interactive apps:create -P $PROJECT_ID WEB $CUSTOMER_SERVICE_NAME
fi

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.2/install.sh | bash

export NVM_DIR="/usr/local/nvm"
source $NVM_DIR/nvm.sh;
```

```
nvm install 16.4.0
npm install
npm install -g @quasar/cli
npm install -g envsub

if [[ -z "${ORDER_SERVICE_URL}" ]]; then
ORDER_SERVICE_URL=$(gcloud run services describe $ORDER_SERVICE_NAME \
--region=$REGION \
--format=json | jq \
--raw-output ".status.url")
export ORDER_SERVICE_URL
else
echo "Using pre-defined ORDER_SERVICE_URL=$ORDER_SERVICE_URL"
fi

if [[ -z "${INVENTORY_SERVICE_URL}" ]]; then
INVENTORY_SERVICE_URL=$(gcloud run services describe $INVENTORY_SERVICE_NAME \
--region=$REGION \
--format=json | jq \
--raw-output ".status.url")
export INVENTORY_SERVICE_URL
else
echo "Using pre-defined INVENTORY_SERVICE_URL=$INVENTORY_SERVICE_URL"
fi

if [[ -z "${MENU_SERVICE_URL}" ]]; then
MENU_SERVICE_URL=$(gcloud run services describe $MENU_SERVICE_NAME \
--region=$REGION \
--format=json | jq \
--raw-output ".status.url")
export MENU_SERVICE_URL
else
echo "Using pre-defined MENU_SERVICE_URL=$MENU_SERVICE_URL"
fi

export API_KEY=$(firebase --non-interactive apps:sdkconfig -P $PROJECT_ID --json | jq --
raw-output ".result.sdkConfig.apiKey")
export AUTH_DOMAIN=$(firebase --non-interactive apps:sdkconfig -P $PROJECT_ID --json |
jq --raw-output ".result.sdkConfig.authDomain")

envsub .env.tmpl .env

rm -rf cloud-run/public
```

```
quasar clean
quasar build

mkdir -p cloud-run/public
cp -r dist/spa/* cloud-run/public
cd cloud-run

gcloud run deploy $CUSTOMER_SERVICE_NAME \
--source . \
--platform managed \
--region $REGION \
--project=$PROJECT_ID \
--allow-unauthenticated \
--max-instances=2 \
--set-env-
vars=VUE_APP_PROJECT_ID=$PROJECT_ID,VUE_APP_API_KEY=$API_KEY,VUE_APP_AUTH_D
OMAIN=$AUTH_DOMAIN,VUE_APP_MENU_SERVICE_URL=$MENU_SERVICE_URL,VUE_APP_IN
VENTORY_SERVICE_URL=$INVENTORY_SERVICE_URL,VUE_APP_ORDER_SERVICE_URL=$ORDE
R_SERVICE_URL \
--quiet

printf "\nManual steps\n"
echo "Enable Google sign-in in the Firebase console:"
printf "https://console.firebase.google.com/u/0/project/$PROJECT_ID/authentication/providers"

printf "\n\nAdd Customer UI service domain to Authorized domains under
Authentication/Settings:"
printf
"\nhttps://console.firebase.google.com/u/0/project/$PROJECT_ID/authentication/settings\n"
```

Charlie then asks Gemini Code Assist to explain the setup.sh file in detail:



Explain the setup.sh file, the tools that are installed, and the Google Cloud services used

Gemini Code Assist automatically includes the context of the setup.sh file that is open in the IDE, and explains the setup script the way an expert colleague might. Gemini Code Assist **breaks down the parts of the setup script** and **explains the tools that are installed and the Google Cloud services that are used**. Gemini Code Assist also summarizes key points that are important for Charlie:

Key Points

- This script demonstrates a common approach to setting up a web application using Google Cloud services.
- It leverages Firebase for authentication and hosting, Cloud Run for serverless deployment, and other Google Cloud services for storage and backend integration.
- The script uses environment variables to configure the application, making it easy to adapt to different environments.

Gemini Code Assist full response (setup.sh)

Let's break down this setup.sh script and explore the tools and Google Cloud services it utilizes.

Script Overview

This script is designed to set up a customer-facing user interface (UI) for a web application. It leverages various tools and Google Cloud services to create

a robust and secure environment.

Tools Installed

1. **Firebase CLI:** The script starts by installing the Firebase CLI, a command-line tool for interacting with Firebase services. It's used for tasks like:
 - **Firebase Login:** Authenticating with your Firebase account.
 - **Project Creation:** Adding a Firebase project to your Google Cloud project.
 - **App Creation:** Creating a web app within your Firebase project.
 - **SDK Configuration:** Retrieving API keys and other configuration details for your web app.
2. **Node Version Manager (nvm):** This tool allows you to manage different versions of Node.js on your system. The script installs Node.js version 16.4.0, which is likely required for the UI's development dependencies.
3. **npm:** The Node Package Manager is used to install the following packages:
 - **@quasar/cli:** A framework for building user interfaces with Quasar, a popular framework for creating cross-platform web applications.
 - **envsub:** A tool for substituting environment variables into template files.

Google Cloud Services

1. **Firebase:** As mentioned earlier, Firebase is a platform for building web and mobile applications. It provides services like:
 - **Authentication:** Securely managing user logins and authorization.
 - **Hosting:** Deploying your web app's static content.
 - **Cloud Functions:** Running server-side code in response to events.

2. **Cloud Run:** This fully managed serverless platform is used to deploy the customer UI as a containerized application. It offers:
 - **Scalability:** Automatically scaling your application based on demand.
 - **Cost-Effectiveness:** Only paying for the resources you use.
 - **Ease of Deployment:** Simplifying the process of deploying and managing your application.
3. **Cloud Storage:** While not explicitly mentioned in the script, Cloud Storage is likely used to store static assets for the UI, such as images and CSS files.

Script Breakdown

1. Environment Setup:

- The script sources a configuration file ([config-env.sh](#)) that likely contains environment variables like project IDs, service names, and API keys.
- It sets the `BASE_DIR` variable to the current working directory.

2. Firebase Setup:

- It logs in to Firebase and adds a Firebase project to your Google Cloud project.
- It checks if a web app with the specified name (`CUSTOMER_SERVICE_NAME`) already exists in the Firebase project. If not, it creates one.

3. Node.js and Dependencies:

- It installs Node.js version 16.4.0 using `nvm`.
- It installs the necessary npm packages for building the UI.

4. Service URLs:

- It retrieves the URLs of other services (e.g., ORDER_SERVICE_URL, INVENTORY_SERVICE_URL, MENU_SERVICE_URL) that the customer UI might interact with. These services are likely deployed on Cloud Run.

5. **Firestore Configuration:**

- It retrieves the API key and authentication domain from Firestore.

6. **Environment Variables:**

- It uses envsubst to substitute environment variables into a template file (.env.tmpl) to create a .env file for the UI.

7. **UI Build and Deployment:**

- It cleans and builds the UI using Quasar.
- It copies the built UI files to a cloud-run/public directory.
- It deploys the UI to Cloud Run using gcloud run deploy.

8. **Manual Steps:**

- The script provides instructions for enabling Google sign-in in the Firestore console and adding the customer UI service domain to the authorized domains list.

Key Points

- This script demonstrates a common approach to setting up a web application using Google Cloud services.
- It leverages Firestore for authentication and hosting, Cloud Run for serverless deployment, and other Google Cloud services for storage and backend integration.
- The script uses environment variables to configure the application, making it easy to adapt to different environments.

Let me know if you have any other questions about this script or Google Cloud services.

Gemini Code Assist provided a great overview of the setup script and the tools and services. Charlie has no prior knowledge of [Vue](#), a framework used by Cymbal Eats to build its web user interface. Charlie opens a .vue page in the customer UI:

customer-ui/src/pages/OrderStatusPage.vue (input)

```
<!--
Copyright 2023 Google LLC
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<template>
<q-layout view="hhh lpr fff">
<q-header elevated>
<Toolbar/>
</q-header>
<q-page-container>
<div class="q-gutter-md q-pa-md">
<div class="row items-center">
<div class="col-4">
Order number
</div>
<div class="col-4">
<q-input
```

```

filled
v-model="orderNumber"
:readonly="true"
/>
</div>
</div>
<div class="row items-center">
<div class="col-4">
Order status
</div>
<div class="col-4">
<q-input
filled
v-model="status"
:readonly="true"
/>
</div>
</div>
<br>
<br>
<OrderView
:items="orderItems"
:allowdelete="false"
/>
</div>
</q-page-container>
</q-layout>
</template>
<script setup>
import { ref, onMounted } from 'vue';
import { useStore } from 'vuex';
import { useRouter, useRoute } from 'vue-router';
import { doc, onSnapshot } from 'firebase/firestore';
import * as Firebase from '../utils/Firebase.js';
import OrderView from '../components/OrderView.vue';
import Toolbar from '../components/Toolbar.vue';
const store = useStore();
const router = useRouter();
const route = useRoute();
const orderNumber = ref(route.params.orderNumber);
const orderItems = ref([]);
const status = ref("");
onMounted(async () => {

```

```
const db = await Firebase.getDb();
onSnapshot(doc(db, 'orders', orderNumber.value), (doc) => {
  const order = doc.data();
  orderItems.value = order.orderItems;
  status.value = order.status;
});
})
</script>
```

Charlie wants to understand how the order status page works. **He can select specific code within a code file and ask Gemini Code Assist "Explain this code to me."** However, Charlie wants more information than just an explanation of the code, so he prompts Gemini Code Assist:



Explain OrderStatusPage.vue and explain how Vue works

Gemini Code Assist **explains the structure, key components, and data and logic** of the Vue order status [single-file component](#). Gemini Code Assist also explains how Vue.js works:

How Vue.js Works

Vue.js is a JavaScript framework that makes building dynamic user interfaces (UIs) easier. Here's how it powers OrderStatusPage.vue:

1. **Data Binding:** Vue.js automatically updates the UI whenever the data in the `orderNumber`, `orderItems`, and `status` variables changes. This means that as the order progresses and its data is updated in Firestore, the UI will reflect those changes in real-time.
2. **Component-Based Architecture:** Vue.js encourages breaking down UIs into reusable components. `OrderStatusPage.vue` is a component itself, and it likely uses other components like `OrderView` and `Toolbar`. This modular approach makes development and maintenance easier.
3. **Reactivity:** Vue.js uses a system called "reactivity" to track changes in data. When data changes, Vue.js automatically updates the affected parts of the UI, ensuring that the UI always reflects the current state of the application.
4. **Lifecycle Hooks:** Vue.js provides lifecycle hooks like `onMounted` that allow you to execute code at specific points in a component's lifecycle. This is useful for tasks like fetching data or setting up event listeners.

Charlie can continue to use Gemini Code Assist to explore and understand the code and technologies used in the codebase.

Gemini Code Assist full response (Vue)

Let's break down `OrderStatusPage.vue` and explore how Vue.js brings it to life.

OrderStatusPage.vue: A Customer's Order Journey

This component is designed to display the status of a customer's order. It's a key part of a food ordering application, providing a clear and informative view

of the order's progress.

Structure

- **Template:** This section defines the visual layout of the page using Quasar, a popular UI framework for Vue.js. It includes:
 - **Header:** A toolbar (likely containing branding or navigation).
 - **Main Content:**
 - Order details: Displays the order number and status.
 - Order items: A list of the items included in the order.
- **Script Setup:** This section contains the logic and data that power the component.

Key Components

- **q-layout:** Provides the overall structure of the page, including header, content, and footer areas.
- **q-header:** Displays the header section.
- **q-page-container:** Contains the main content of the page.
- **q-input:** Used to display the order number and status, making them read-only.
- **OrderView:** A custom component (likely defined elsewhere) that displays the order items.
- **Toolbar:** Another custom component (likely defined elsewhere) for the header.

Data and Logic

- **orderNumber:** A reactive variable that stores the order number, obtained from the URL parameters.

- **orderItems:** A reactive array that holds the items in the order.
- **status:** A reactive variable that stores the current status of the order.
- **onMounted:** A lifecycle hook that runs when the component is mounted. It fetches the order data from a Firestore database using onSnapshot.
- **Firebase.getDb():** A function (likely defined in utils/Firebase.js) that connects to the Firestore database.
- **doc(db, 'orders', orderNumber.value):** A Firestore reference to the specific order document.
- **onSnapshot:** A Firestore listener that updates the orderItems and status variables whenever the order document changes.

How Vue.js Works

Vue.js is a JavaScript framework that makes building dynamic user interfaces (UIs) easier. Here's how it powers OrderStatusPage.vue:

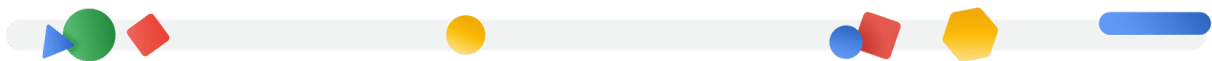
1. **Data Binding:** Vue.js automatically updates the UI whenever the data in the orderNumber, orderItems, and status variables changes. This means that as the order progresses and its data is updated in Firestore, the UI will reflect those changes in real-time.
2. **Component-Based Architecture:** Vue.js encourages breaking down UIs into reusable components. OrderStatusPage.vue is a component itself, and it likely uses other components like OrderView and Toolbar. This modular approach makes development and maintenance easier.
3. **Reactivity:** Vue.js uses a system called "reactivity" to track changes in data. When data changes, Vue.js automatically updates the affected parts of the UI, ensuring that the UI always reflects the current state of the application.
4. **Lifecycle Hooks:** Vue.js provides lifecycle hooks like onMounted that allow you to execute code at specific points in a component's lifecycle. This is useful for tasks like fetching data or setting up event listeners.

In Summary

OrderStatusPage.vue is a well-structured component that leverages Vue.js's features to provide a dynamic and informative view of an order's status. It demonstrates how Vue.js can be used to build interactive and data-driven web applications.



Gemini Code Assist explains code and
configuration files,
helping you understand new programming
languages,
code, tools, and frameworks.



Commenting code

Another way Gemini Code Assist can help Charlie understand their new codebase is by adding comments to code. Charlie looks at the top-level JavaScript file for the order service:

order-service/index.js (input)

```

/**
 * Copyright 2023 Google LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
const express = require('express');
const cors = require('cors');
const app = express();
const axios = require('axios');
app.use(cors());
app.use(express.json());
const port = process.env.PORT || 8080;
app.listen(port, () => {
  console.log(` API listening on port ${port}`);
});
const {Firestore} = require('@google-cloud/firestore');
const {PubSub} = require('@google-cloud/pubsub');
const db = new Firestore();
const pubsub = new PubSub();

const admin = require('firebase-admin');
admin.initializeApp();

const TOPIC_NAME = 'order-topic';

const inventoryServer = axios.create({
  baseURL: process.env.INVENTORY_SERVICE_URL,
  headers: {
    get: {
      "Content-Type": 'application/json'
    }
  }
})

```



```

app.get('/order', async (req, res) => {
  try {
    const orderColl = await db.collection(`orders`).get();
    const orders = orderColl.docs.map(d => d.data());
    res.json({status: 'success', data: orders});
  } catch (ex) {
    console.error(ex);
    res.status(500).json({error: ex.toString()});
  }
})

app.get('/orders/customer', async (req, res) => {
  try {
    if (req.headers['authorization'] != null && req.headers['authorization']
    != "undefined") {
      var userId = await getUserIdFromAuthHeader(req.headers['authorization']);
      const orderColl = await db.collection(`orders`).where('userId', '==',
      userId).get();
      const orders = orderColl.docs.map(d => d.data());
      res.json({status: 'success', data: orders});
    } else {
      console.error('Unauthorized');
      res.status(401).json({error: 'Unauthorized'});
    }
  } catch (ex) {
    console.error(ex);
    res.status(500).json({error: ex.toString()});
  }
})

app.get('/order/:orderNumber', async (req, res) => {
  try {
    const orderNumber = req.params.orderNumber;
    const orderDoc = await db.doc(`orders/${orderNumber}`).get();
    if (orderDoc.exists) {
      res.json({status: 'success', data: orderDoc.data()});
    } else {
      res.status(404).json({error: `Order "${orderNumber}" not found`});
    }
  } catch (ex) {
    console.error(ex);
    res.status(500).json({error: ex.toString()});
  }
})

```

```

    })

    async function getUserIdFromAuthHeader(authHeader) {
    if (authHeader) {
    const token = await admin.auth().verifyIdToken(authHeader);
    return token.uid;
    }
    }

    app.post('/order', async (req, res) => {
    try {
    var userId = null;
    if (req.headers['authorization'] != null && req.headers['authorization']
    != "undefined") {
    userId = await getUserIdFromAuthHeader(req.headers['authorization']);
    }

    if (!await inventoryAvailable(req.body.orderItems)) {
    throw 'Incorrect Order Quantity or Item';
    }

    if (userId) {
    req.body.userId = userId;
    }

    const orderNumber = await createOrderRecord(req.body);
    await subtractFromInventory(req.body.orderItems);
    res.json({orderNumber: orderNumber});
    const data = req.body;
    if (userId) {
    data.userId = userId;
    }

    data.orderNumber = orderNumber;

    publishMessage(data);
    } catch (ex) {
    console.error(ex);
    res.status(500).json({error: ex.toString()});
    }
    })

    app.delete('/order/:orderNumber', async (req, res) => {
    try {

```

```

const orderDoc = db.doc(`orders/${req.params.orderNumber}`);
await orderDoc.delete();
res.json({status: 'success'});
} catch (ex) {
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})

app.patch('/order/:orderNumber', async (req, res) => {
try {
const orderDoc = db.doc(`orders/${req.params.orderNumber}`);
await orderDoc.update(req.body);
res.json({status: 'success'});
} catch (ex) {
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})

async function inventoryAvailable(orderItems) {
const inventory = await inventoryServer.get("/getAvailableInventory");
const inventoryDict = {};
for (item in inventory.data) {
inventoryDict[parseInt(
inventory.data[item].ItemID)] = inventory.data[item].Inventory;
}
for (ol in orderItems) {
var orderItem = orderItems[ol];
if (!(orderItem.id in inventoryDict) || (inventoryDict[orderItem.id]
< orderItem.quantity)) {
return false;
}
}
return true;
}

async function createOrderRecord(requestBody) {
const orderNumber = getNewOrderNumber();
const orderDoc = db.doc(`orders/${orderNumber}`);
await orderDoc.set({
orderNumber: orderNumber,
userId: requestBody.userId ? requestBody.userId : "",
name: requestBody.name,
email: requestBody.email,

```

```
address: requestBody.address,
city: requestBody.city,
state: requestBody.state,
zip: requestBody.zip,
orderItems: requestBody.orderItems,
status: 'New',
statusUpdatedAt: new Date(),
placedAt: new Date()
});
return orderNumber;
}
```

```
async function subtractFromInventory(orderItems) {
  await inventoryServer.post("/updateInventoryItem",
    orderItems.map(x => ({
      itemID: x.id,
      inventoryChange: -x.quantity
    })))
};
}
```

```
function getNewOrderNumber() {
  return Math.round(10000 + Math.random() * 90000);
}
```

```
async function publishMessage(data) {
  try {
    const dataBuffer = Buffer.from(JSON.stringify(data))
    const messageId = await pubsub
      .topic(TOPIC_NAME)
      .publishMessage({data: dataBuffer});
    console.log(` Message ${messageId} published.`);
  } catch (error) {
    console.error(` Received error while publishing: ${error.message}`);
  }
}

app.post('/order/points', async (req, res) => {
  try {

    const orderNumber = req.body.message.attributes.orderNumber;
    const rewardPoints = parseInt(req.body.message.attributes.rewardPoints);
    const totalAmount = parseFloat(
      parseFloat(req.body.message.attributes.totalAmount).toFixed(2));
```

```

const updateRec = {
  'rewardPoints': rewardPoints,
  'totalAmount': totalAmount
};

const orderDoc = db.doc(`orders/${orderNumber}`);
if (orderDoc) {
  await orderDoc.update(updateRec);
  res.json({status: 'success'});
} else {
  console.log("Order not found. orderNumber: ", orderNumber);
}
} catch (ex) {
  console.error(ex);
  res.status(500).json({error: ex.toString()});
}
})

```

Charlie wants to understand how the order service works. Unfortunately, the order service code is not commented. **He can select code within a code file and ask Gemini Code Assist "Comment this code."** However, he wants comments throughout the file, so he asks Gemini Code Assist:



Add comments to this file

Gemini Code Assist comments the entire file. For example, this function that had zero comments, now looks like this:

```

// GET endpoint to retrieve all orders
app.get('/order', async (req, res) => {
  try {
    // Get all documents from the 'orders' collection
    const orderColl = await db.collection('orders').get();
    // Map the documents to their data
    const orders = orderColl.docs.map(d => d.data());
    // Send a success response with the orders data
    res.json({status: 'success', data: orders});
  } catch (ex) {
    // Log the error and send an error response
    console.error(ex);
    res.status(500).json({error: ex.toString()});
  }
})

```

Gemini Code Assist full response (English comments)

```

/**
 * Copyright 2023 Google LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

```

```
const express = require('express');
const cors = require('cors');
const app = express();
const axios = require('axios');
// Enable CORS for all origins
app.use(cors());
// Parse incoming JSON requests
app.use(express.json());
const port = process.env.PORT || 8080;
// Start the server and listen on the specified port
app.listen(port, () => {
  console.log(` API listening on port ${port}`);
});
// Import Firestore and Pub/Sub clients
const {Firestore} = require('@google-cloud/firestore');
const {PubSub} = require('@google-cloud/pubsub');
// Initialize Firestore and Pub/Sub instances
const db = new Firestore();
const pubsub = new PubSub();

// Initialize Firebase Admin SDK
const admin = require('firebase-admin');
admin.initializeApp();

// Define the name of the Pub/Sub topic
const TOPIC_NAME = 'order-topic';

// Create an Axios instance for making requests to the inventory service
const inventoryServer = axios.create({
  baseURL: process.env.INVENTORY_SERVICE_URL,
  headers: {
    get: {
      "Content-Type": 'application/json'
    }
  }
});

// GET endpoint to retrieve all orders
app.get('/order', async (req, res) => {
  try {
    // Get all documents from the 'orders' collection
    const orderColl = await db.collection('orders').get();
    // Map the documents to their data
```

```

const orders = orderColl.docs.map(d => d.data());
// Send a success response with the orders data
res.json({status: 'success', data: orders});
} catch (ex) {
// Log the error and send an error response
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})

// GET endpoint to retrieve orders for a specific customer
app.get('/orders/customer', async (req, res) => {
try {
// Check if an authorization header is present
if (req.headers['authorization'] != null && req.headers['authorization']
!= "undefined") {
// Get the user ID from the authorization header
var userId = await getUserIdFromAuthHeader(req.headers['authorization']);
// Get all documents from the 'orders' collection where the userId matches
const orderColl = await db.collection(`orders`).where('userId', '==',
userId).get();
// Map the documents to their data
const orders = orderColl.docs.map(d => d.data());
// Send a success response with the orders data
res.json({status: 'success', data: orders});
} else {
// Log an unauthorized error and send an unauthorized response
console.error('Unauthorized');
res.status(401).json({error: 'Unauthorized'});
}
} catch (ex) {
// Log the error and send an error response
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})

// GET endpoint to retrieve a specific order by order number
app.get('/order/:orderNumber', async (req, res) => {
try {
// Get the order number from the request parameters
const orderNumber = req.params.orderNumber;
// Get the document for the specified order number
const orderDoc = await db.doc(`orders/${orderNumber}`).get();

```



```

// Check if the document exists
if (orderDoc.exists) {
// Send a success response with the order data
res.json({status: 'success', data: orderDoc.data()});
} else {
// Send a not found response
res.status(404).json({error: `Order "${orderNumber}" not found`});
}
} catch (ex) {
// Log the error and send an error response
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})

// Helper function to get the user ID from the authorization header
async function getUserIdFromAuthHeader(authHeader) {
// Check if the authorization header is present
if (authHeader) {
// Verify the ID token and return the user ID
const token = await admin.auth().verifyIdToken(authHeader);
return token.uid;
}
}

// POST endpoint to create a new order
app.post('/order', async (req, res) => {
try {
// Initialize the user ID to null
var userId = null;
// Check if an authorization header is present
if (req.headers['authorization'] != null && req.headers['authorization']
!= "undefined") {
// Get the user ID from the authorization header
userId = await getUserIdFromAuthHeader(req.headers['authorization']);
}

// Check if the inventory is available for the requested order items
if (!await inventoryAvailable(req.body.orderItems)) {
// Throw an error if the inventory is not available
throw 'Incorrect Order Quantity or Item';
}

// Set the user ID in the request body if it's available

```

```
if (userId) {
  req.body.userId = userId;
}
// Create a new order record in Firestore
const orderNumber = await createOrderRecord(req.body);
// Subtract the ordered items from the inventory
await subtractFromInventory(req.body.orderItems);
// Send a success response with the order number
res.json({orderNumber: orderNumber});
// Create a copy of the request body for publishing to Pub/Sub
const data = req.body;
// Set the user ID in the data if it's available
if (userId) {
  data.userId = userId;
}

// Set the order number in the data
data.orderNumber = orderNumber;

// Publish the order data to the Pub/Sub topic
publishMessage(data);
} catch (ex) {
  // Log the error and send an error response
  console.error(ex);
  res.status(500).json({error: ex.toString()});
}
})

// DELETE endpoint to delete an order by order number
app.delete('/order/:orderNumber', async (req, res) => {
  try {
    // Get the order document from Firestore
    const orderDoc = db.doc(`orders/${req.params.orderNumber}`);
    // Delete the order document
    await orderDoc.delete();
    // Send a success response
    res.json({status: 'success'});
  } catch (ex) {
    // Log the error and send an error response
    console.error(ex);
    res.status(500).json({error: ex.toString()});
  }
})
```

```

// PATCH endpoint to update an order by order number
app.patch('/order/:orderNumber', async (req, res) => {
  try {
    // Get the order document from Firestore
    const orderDoc = db.doc(`orders/${req.params.orderNumber}`);
    // Update the order document with the request body
    await orderDoc.update(req.body);
    // Send a success response
    res.json({status: 'success'});
  } catch (ex) {
    // Log the error and send an error response
    console.error(ex);
    res.status(500).json({error: ex.toString()});
  }
})

// Helper function to check if the inventory is available for the order items
async function inventoryAvailable(orderItems) {
  // Get the available inventory from the inventory service
  const inventory = await inventoryServer.get("/getAvailableInventory");
  // Create a dictionary to store the inventory by item ID
  const inventoryDict = {};
  // Iterate over the inventory data and populate the dictionary
  for (item in inventory.data) {
    inventoryDict[parseInt(
      inventory.data[item].ItemID)] = inventory.data[item].Inventory;
  }
  // Iterate over the order items
  for (ol in orderItems) {
    var orderItem = orderItems[ol];
    // Check if the item ID is in the inventory dictionary and if the available
    // inventory is sufficient
    if (!(orderItem.id in inventoryDict) || (inventoryDict[orderItem.id]
      < orderItem.quantity)) {
      // Return false if the inventory is not available
      return false;
    }
  }
  // Return true if the inventory is available
  return true;
}

// Helper function to create a new order record in Firestore
async function createOrderRecord(requestBody) {

```

```

// Generate a new order number
const orderNumber = getNewOrderNumber();
// Get the order document from Firestore
const orderDoc = db.doc(`orders/${orderNumber}`);
// Set the order data in the document
await orderDoc.set({
  orderNumber: orderNumber,
  userId: requestBody.userId ? requestBody.userId : "",
  name: requestBody.name,
  email: requestBody.email,
  address: requestBody.address,
  city: requestBody.city,
  state: requestBody.state,
  zip: requestBody.zip,
  orderItems: requestBody.orderItems,
  status: 'New',
  statusUpdatedAt: new Date(),
  placedAt: new Date()
});
// Return the order number
return orderNumber;
}

// Helper function to subtract the ordered items from the inventory
async function subtractFromInventory(orderItems) {
  // Send a POST request to the inventory service to update the inventory
  await inventoryServer.post("/updateInventoryItem",
    orderItems.map(x => ({
      itemID: x.id,
      inventoryChange: -x.quantity
    })))
  );
}

// Helper function to generate a new order number
function getNewOrderNumber() {
  // Generate a random number between 10000 and 99999
  return Math.round(10000 + Math.random() * 90000);
}

// Helper function to publish a message to the Pub/Sub topic
async function publishMessage(data) {
  try {
    // Convert the data to a JSON string and create a buffer

```

```

const dataBuffer = Buffer.from(JSON.stringify(data))
// Publish the message to the Pub/Sub topic
const messageId = await pubsub
.topic(TOPIC_NAME)
.publishMessage({data: dataBuffer});
// Log the message ID
console.log(`Message ${messageId} published.`);
} catch (error) {
// Log any errors that occur during publishing
console.error(`Received error while publishing: ${error.message}`);
}
}

// POST endpoint to update order with reward points and total amount
app.post('/order/points', async (req, res) => {

  try {
    // Get the order number and reward points from the message attributes
    const orderNumber = req.body.message.attributes.orderNumber;
    const rewardPoints = parseInt(req.body.message.attributes.rewardPoints);
    const totalAmount = parseFloat(
      parseFloat(req.body.message.attributes.totalAmount).toFixed(2));

    // Create an object with the updated reward points and total amount
    const updateRec = {
      'rewardPoints': rewardPoints,
      'totalAmount': totalAmount
    };

    // Get the order document from Firestore
    const orderDoc = db.doc(`orders/${orderNumber}`);
    // Check if the order document exists
    if (orderDoc) {
      // Update the order document with the updated data
      await orderDoc.update(updateRec);
      // Send a success response
      res.json({status: 'success'});
    } else {
      // Log a message if the order document is not found
      console.log("Order not found. orderNumber: ", orderNumber);
    }
  } catch (ex) {
    // Log the error and send an error response
    console.error(ex);
  }
}

```

```
res.status(500).json({error: ex.toString()});
}
})
```

Charlie's first language is not English, so he would like comments in Italian for better understanding. He asks Gemini Code Assist:



Translate comments into Italian, but don't update text that is not in comments

All comments are now returned in Italian, with code strings untouched. The same function now looks like this:

```
// Endpoint GET per recuperare tutti gli ordini
app.get('/order', async (req, res) => {
  try {
    // Ottieni tutti i documenti dalla collezione 'orders'
    const orderColl = await db.collection(`orders`).get();
    // Mappa i documenti ai loro dati
    const orders = orderColl.docs.map(d => d.data());
    // Invia una risposta di successo con i dati degli ordini
    res.json({status: 'success', data: orders});
```

```
} catch (ex) {  
  // Registra l'errore e invia una risposta di errore  
  console.error(ex);  
  res.status(500).json({error: ex.toString()});  
}  
})
```

Charlie can also get similar results by prompting in Italian:



Traduci i commenti in italiano, ma non aggiornare il testo che non è nei commenti

Any prompt can be specified in a supported language, and the response will be returned in the same language. **Gemini Code Assist supports over 30 prompting languages.**

Gemini Code Assist full response (Italian comments)

```
/**  
 * Copyright 2023 Google LLC  
 *  
 * Licensed under the Apache License, Version 2.0 (the "License");  
 * you may not use this file except in compliance with the License.  
 * You may obtain a copy of the License at  
 *  
 * http://www.apache.org/licenses/LICENSE-2.0  
 *  
 * Unless required by applicable law or agreed to in writing, software  
 * distributed under the License is distributed on an "AS IS" BASIS,  
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

* See the License for the specific language governing permissions and
* limitations under the License.

*/

```
const express = require('express');
const cors = require('cors');
const app = express();
const axios = require('axios');
// Abilita CORS per tutte le origini
app.use(cors());
// Analizza le richieste JSON in arrivo
app.use(express.json());
const port = process.env.PORT || 8080;
// Avvia il server e ascolta sulla porta specificata
app.listen(port, () => {
  console.log(`API listening on port ${port}`);
});
// Importa i client Firestore e Pub/Sub
const {Firestore} = require('@google-cloud/firestore');
const {PubSub} = require('@google-cloud/pubsub');
// Inizializza le istanze Firestore e Pub/Sub
const db = new Firestore();
const pubsub = new PubSub();
// Inizializza il Firebase Admin SDK
const admin = require('firebase-admin');
admin.initializeApp();
// Definisci il nome dell'argomento Pub/Sub
const TOPIC_NAME = 'order-topic';
// Crea un'istanza Axios per effettuare richieste al servizio inventario
const inventoryServer = axios.create({
  baseURL: process.env.INVENTORY_SERVICE_URL,
  headers: {
    get: {
      "Content-Type": 'application/json'
    }
  }
});
// Endpoint GET per recuperare tutti gli ordini
app.get('/order', async (req, res) => {
  try {
    // Ottieni tutti i documenti dalla collezione 'orders'
    const orderColl = await db.collection('orders').get();
    // Mappa i documenti ai loro dati
    const orders = orderColl.docs.map(d => d.data());
```



```

// Invia una risposta di successo con i dati degli ordini
res.json({status: 'success', data: orders});
} catch (ex) {
// Registra l'errore e invia una risposta di errore
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})

// Endpoint GET per recuperare gli ordini per un cliente specifico
app.get('/orders/customer', async (req, res) => {
try {
// Controlla se è presente un'intestazione di autorizzazione
if (req.headers['authorization'] != null && req.headers['authorization'] != "undefined") {
// Ottieni l'ID utente dall'intestazione di autorizzazione
var userId = await getUserIdFromAuthHeader(req.headers['authorization']);
// Ottieni tutti i documenti dalla collezione 'orders' in cui l'userId corrisponde
const orderColl = await db.collection('orders').where('userId', '==', userId).get();
// Mappa i documenti ai loro dati
const orders = orderColl.docs.map(d => d.data());
// Invia una risposta di successo con i dati degli ordini
res.json({status: 'success', data: orders});
} else {
// Registra un errore di autorizzazione e invia una risposta di autorizzazione non riuscita
console.error('Unauthorized');
res.status(401).json({error: 'Unauthorized'});
}
} catch (ex) {
// Registra l'errore e invia una risposta di errore
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})

// Endpoint GET per recuperare un ordine specifico in base al numero di ordine
app.get('/order/:orderNumber', async (req, res) => {
try {
// Ottieni il numero di ordine dai parametri della richiesta
const orderNumber = req.params.orderNumber;
// Ottieni il documento per il numero di ordine specificato

```

```

const orderDoc = await db.doc(`orders/${orderId}`).get();
// Controlla se il documento esiste
if (orderDoc.exists) {
// Invia una risposta di successo con i dati dell'ordine
res.json({status: 'success', data: orderDoc.data()});
} else {
// Invia una risposta di non trovato
res.status(404).json({error: `Order "${orderId}" not found`});
}
} catch (ex)
// Registra l'errore e invia una risposta di errore
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})
// Funzione di supporto per ottenere l'ID utente dall'intestazione di autorizzazione
async function getUserIdFromAuthHeader(authHeader) {
// Controlla se è presente l'intestazione di autorizzazione
if (authHeader) {
// Verifica il token ID e restituisci l'ID utente
const token = await admin.auth().verifyIdToken(authHeader);
return token.uid;
}
}
// Endpoint POST per creare un nuovo ordine
app.post('/order', async (req, res) => {
try {
// Inizializza l'ID utente a null
var userId = null;
// Controlla se è presente un'intestazione di autorizzazione
if (req.headers['authorization'] != null && req.headers['authorization'] != "undefined") {
// Ottieni l'ID utente dall'intestazione di autorizzazione
userId = await getUserIdFromAuthHeader(req.headers['authorization']);
}
// Controlla se l'inventario è disponibile per gli articoli dell'ordine richiesti
if (!await inventoryAvailable(req.body.orderItems)) {

```

```

// Genera un errore se l'inventario non è disponibile
throw 'Incorrect Order Quantity or Item';
}

// Imposta l'ID utente nel corpo della richiesta se è disponibile
if (userId) {
  req.body.userId = userId;
}

// Crea un nuovo record di ordine in Firestore
const orderNumber = await createOrderRecord(req.body);

// Sottrai gli articoli ordinati dall'inventario
await subtractFromInventory(req.body.orderItems);

// Invia una risposta di successo con il numero di ordine
res.json({orderNumber: orderNumber});

// Crea una copia del corpo della richiesta per la pubblicazione su Pub/Sub
const data = req.body;

// Imposta l'ID utente nei dati se è disponibile
if (userId) {
  data.userId = userId;
}

// Imposta il numero di ordine nei dati
data.orderNumber = orderNumber;

// Pubblica i dati dell'ordine nell'argomento Pub/Sub
publishMessage(data);
} catch (ex) {
  // Registra l'errore e invia una risposta di errore
  console.error(ex);
  res.status(500).json({error: ex.toString()});
}
})

// Endpoint DELETE per eliminare un ordine in base al numero di ordine
app.delete('/order/:orderNumber', async (req, res) => {
  try {
    // Ottieni il documento dell'ordine da Firestore
    const orderDoc = db.doc(`orders/${req.params.orderNumber}`);

    // Elimina il documento dell'ordine
    await orderDoc.delete();
  }
}

```

```

// Invia una risposta di successo
res.json({status: 'success'});
} catch (ex) {
// Registra l'errore e invia una risposta di errore
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})
// Endpoint PATCH per aggiornare un ordine in base al numero di ordine
app.patch('/order/:orderNumber', async (req, res) => {
try {
// Ottieni il documento dell'ordine da Firestore
const orderDoc = db.doc(`orders/${req.params.orderNumber}`);
// Aggiorna il documento dell'ordine con il corpo della richiesta
await orderDoc.update(req.body);
// Invia una risposta di successo
res.json({status: 'success'});
} catch (ex) {
// Registra l'errore e invia una risposta di errore
console.error(ex);
res.status(500).json({error: ex.toString()});
}
})
// Funzione di supporto per controllare se l'inventario è disponibile per gli articoli dell'ordine
async function inventoryAvailable(orderItems) {
// Ottieni l'inventario disponibile dal servizio inventario
const inventory = await inventoryServer.get("/getAvailableInventory");
// Crea un dizionario per memorizzare l'inventario in base all'ID articolo
const inventoryDict = {};
// Iterare sui dati dell'inventario e popolare il dizionario
for (item in inventory.data) {
inventoryDict[parseInt(
inventory.data[item].ItemID)] = inventory.data[item].Inventory;
}
// Iterare sugli articoli dell'ordine
for (oi in orderItems) {

```

```

var orderItem = orderItems[ol];
// Controlla se l'ID articolo è nel dizionario dell'inventario e se l'inventario disponibile è
sufficiente
if (!(orderItem.id in inventoryDict) || (inventoryDict[orderItem.id] < orderItem.quantity)) {
// Restituisci false se l'inventario non è disponibile
return false;
}
}
// Restituisci true se l'inventario è disponibile
return true;
}
// Funzione di supporto per creare un nuovo record di ordine in Firestore
async function createOrderRecord(requestBody) {
// Genera un nuovo numero di ordine
const orderNumber = getNewOrderNumber();
// Ottieni il documento dell'ordine da Firestore
const orderDoc = db.doc(`orders/${orderNumber}`);
// Imposta i dati dell'ordine nel documento
await orderDoc.set({
orderNumber: orderNumber,
userId: requestBody.userId ? requestBody.userId : "",
name: requestBody.name,
email: requestBody.email,
address: requestBody.address,
city: requestBody.city,
state: requestBody.state,
zip: requestBody.zip,
orderItems: requestBody.orderItems,
status: 'New',
statusUpdatedAt: new Date(),
placedAt: new Date()
});
// Restituisci il numero di ordine
return orderNumber;
}
// Funzione di supporto per sottrarre gli articoli ordinati dall'inventario
async function subtractFromInventory(orderItems) {

```

```

// Invia una richiesta POST al servizio inventario per aggiornare l'inventario
await inventoryServer.post("/updateInventoryItem",
orderItems.map(x => ({
itemID: x.id,
inventoryChange: -x.quantity
})))
);
}

// Funzione di supporto per generare un nuovo numero di ordine
function getNewOrderNumber() {
// Genera un numero casuale tra 10000 e 99999
return Math.round(10000 + Math.random() * 90000);
}

// Funzione di supporto per pubblicare un messaggio nell'argomento Pub/Sub
async function publishMessage(data) {
try {
// Converti i dati in una stringa JSON e crea un buffer
const dataBuffer = Buffer.from(JSON.stringify(data))
// Pubblica il messaggio nell'argomento Pub/Sub
const messageId = await pubsub
.topic(TOPIC_NAME)
.publishMessage({data: dataBuffer});
// Registra l'ID messaggio
console.log(` Message ${messageId} published.`);
} catch (error) {
// Registra eventuali errori che si verificano durante la pubblicazione
console.error(` Received error while publishing: ${error.message}`);
}
}

// Endpoint POST per aggiornare l'ordine con punti premio e importo totale
app.post('/order/points', async (req, res) => {
try {
// Ottieni il numero di ordine e i punti premio dagli attributi del messaggio
const orderNumber = req.body.message.attributes.orderNumber;
const rewardPoints = parseInt(req.body.message.attributes.rewardPoints);
const totalAmount = parseFloat(
parseFloat(req.body.message.attributes.totalAmount).toFixed(2));

```

```
// Crea un oggetto con i punti premio e l'importo totale aggiornati
const updateRec = {
  'rewardPoints': rewardPoints,
  'totalAmount': totalAmount
};

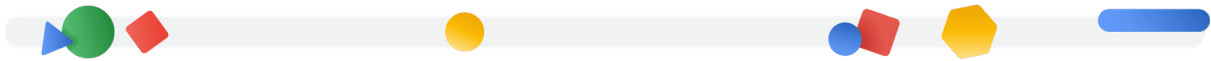
// Ottieni il documento dell'ordine da Firestore
const orderDoc = db.doc(`orders/${orderNumber}`);

// Controlla se il documento dell'ordine esiste
if (orderDoc) {
  // Aggiorna il documento dell'ordine con i dati aggiornati
  await orderDoc.update(updateRec);
  // Invia una risposta di successo
  res.json({status: 'success'});
} else {
  // Registra un messaggio se il documento dell'ordine non viene trovato
  console.log("Order not found. orderNumber: ", orderNumber);
}

} catch (ex) {
  // Registra l'errore e invia una risposta di errore
  console.error(ex);
  res.status(500).json({error: ex.toString()});
}
})
```



Gemini Code Assist translates code into different languages, and prompts can be written in those languages.



Creating new code

Gemini Code Assist is very helpful in understanding existing code, but it also can help you create new code.

For example, Charlie can use this prompt in the side panel to add a new function to an open file:



Add a function to create a Cloud Storage bucket

Prompts can also be added to code files. Contextual smart actions in the editor can automate tasks, like generating code, transforming code, or creating unit tests.

Charlie can add this prompt to an open code file:



Add a function to create a Cloud Storage bucket

Then press Control+Enter (for Windows or Linux) or Control+Return(for macOS). Gemini Code Assist will generate code as ghost text.

```
5  const {Storage} = require('@google-cloud/storage');
6
7  add a function to create a Cloud Storage bucket
   const bucketName = 'cymbal-eats-test';
   const storage = new Storage();
   const bucket = storage.bucket(bucketName);

   async function createBucket() {
     try {
       await bucket.create();
       console.log(`Bucket ${bucketName} created.`);
     } catch (err) {
       console.error(`Error creating bucket ${bucketName}:`, err);
     }
   }

   createBucket();
```

8

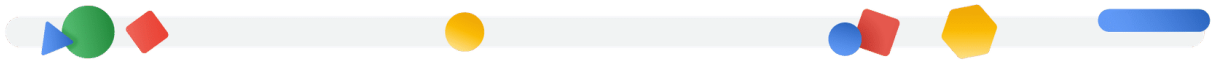
The generated function will be created using the programming language of the code in the file. Gemini Code Assist will attempt to match the patterns found in the file.

Gemini Code Assist can also generate code suggestions while you code, allowing you to accept or reject code suggestions, and accelerating your coding process.

With generative AI, you should always verify and test the code that Gemini Code Assist or any other large language model creates.



Gemini Code Assist completes code as you write, and generates whole code files, code blocks, or functions on demand.



Review

For a new developer like Charlie, which of the following is a primary benefit of using Gemini Code Assist?

- ☐ Automatically generates complete code projects without any need for developer input.
- ☐ Replaces the need for human code reviews and testing.
- ☐ Provides AI-powered code assistance and natural language chat to answer coding questions and offer guidance.
- ☐ Handles all deployment and infrastructure setup on Google Cloud.

SUBMIT

Gemini Code Assist for Experienced Developers

Kalani, the experienced developer



Kalani has been a **lead developer** at Cymbal Eats for the past three years. She is the primary developer for the company's backend code base. She is also **responsible for code quality and testing**.

Improving code

Kalani is responsible for code quality at Cymbal Eats, so one of her common tasks is improving code written by other developers. She turns to Gemini Code Assist to help clean up an existing code file:

cloud-functions/thumbnail/index.js (input)

```
// Copyright 2021 Google LLC
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
// https://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
'use strict';

const imageMagick = require('imagemagick');
const Promise = require("bluebird");
const path = require('path');
const functions = require('@google-cloud/functions-framework');
const vision = require('@google-cloud/vision');
const {Storage} = require('@google-cloud/storage');
const axios = require('axios');
var fs = require('fs');

functions.cloudEvent('process-thumbnails', async (cloudEvent) => {
  console.log(`Event ID: ${cloudEvent.id}`);
  console.log(`Event Type: ${cloudEvent.type}`);
  const file = cloudEvent.data;

  try {
    console.log(`Received thumbnail request for file ${file.name} from bucket ${file.bucket}`);
```

```
const storage = new Storage();
const bucket = storage.bucket(file.bucket);
const thumbBucket = storage.bucket(process.env.BUCKET_THUMBNAILS);

const client = new vision.ImageAnnotatorClient();
const visionRequest = {
  image: { source: { imageUri: `gs://${file.bucket}/${file.name}` } },
  features: [
    { type: 'LABEL_DETECTION' },
  ]
};

// We launch the vision call first so we can process the thumbnail while we wait for the
response.
const visionPromise = client.annotateImage(visionRequest);

if (!fs.existsSync("/tmp/original")){
  fs.mkdirSync("/tmp/original");
}
if (!fs.existsSync("/tmp/thumbnail")){
  fs.mkdirSync("/tmp/thumbnail");
}

const originalFile = `/tmp/original/${file.name}`;
const thumbFile = `/tmp/thumbnail/${file.name}`
await bucket.file(file.name).download({
  destination: originalFile
});

const originalImageUrl = await bucket.file(file.name).publicUrl()

console.log(` Downloaded picture into ${originalFile}`);

const itemID = parseInt(path.parse(file.name).name);

if (isNaN(itemID)){
  return;
}

const resizeCrop = Promise.promisify(imageMagick.crop);
await resizeCrop({
  srcPath: originalFile,
  dstPath: thumbFile,
  width: 400,
  height: 400
```

```
});  
console.log(` Created local thumbnail in ${thumbFile}`);  
  
const thumbnaillImage = await thumbBucket.upload(thumbFile);  
const thumbnaillImageUrl = thumbnaillImage[0].publicUrl();  
console.log(` Uploaded thumbnail to Cloud Storage bucket  
${process.env.BUCKET_THUMBNAILS}`);  
const visionResponse = await visionPromise;  
console.log(` Raw vision output for: ${file.name}: ${JSON.stringify(visionResponse[0])}`);  
let status = "Failed"  
let labels = "";  
for (const label of visionResponse[0].labelAnnotations){  
  status = label.description == "Food" ? "Ready" : status  
  labels = labels.concat(label.description, ", ");  
}  
console.log(` \nVision API labels: ${labels}\n`);  
console.log(` Menu Item status will be set to: ${status}\n`);  
  
const menuServer = axios.create({  
  baseURL: process.env.MENU_SERVICE_URL,  
  headers :{  
    get: {  
      "Content-Type": 'application/json'  
    }  
  }  
});  
  
const item = await menuServer.get(`/menu/${itemID}`);  
// Send update call to menu service  
const request = await menuServer.put(`/menu/${itemID}`, {  
  itemImageUrl: originalImageUrl,  
  itemName: item.data.itemName,  
  itemPrice: item.data.itemPrice,  
  itemThumbnailURL: thumbnaillImageUrl,  
  spiceLevel: item.data.spiceLevel,  
  status: status,  
  tagLine: item.data.tagLine  
})  
} catch (err) {  
  console.log(` Error: processing the thumbnail: ${err}`);  
}  
});
```

A different developer wrote the process-thumbnails function, and Kalani would like Gemini Code Assist's help cleaning up the code. **First, Kalani asks Gemini Code Assist to explain the code:**



Explain index.js

Gemini Code Assist summarizes the code:

This JavaScript code defines a Google Cloud Function that processes image thumbnails and integrates with Google Cloud Vision API and a menu service.

Gemini Code Assist also divides the functionality in several sections, explains each section, and describes the steps in the workflow and the code's purpose:

Gemini Code Assist full response (explanation)

Explanation:

This JavaScript code defines a Google Cloud Function that processes image thumbnails and integrates with Google Cloud Vision API and a menu service. Here's a breakdown:

1. **Initialization:**

- Imports necessary libraries for image processing, Cloud Functions, Vision API, Cloud Storage, and HTTP requests.
- Defines a Cloud Function triggered by a Cloud Event named 'process-thumbnails'.

2. **Event Handling:**

- Logs event details (ID and type).
- Extracts file information (name, bucket) from the Cloud Event data.

3. **Image Processing:**

- Initializes Google Cloud Storage client and retrieves the relevant buckets.
- Downloads the original image from Cloud Storage to a temporary location.
- Generates a thumbnail image using ImageMagick's crop function.
- Uploads the thumbnail image to a designated Cloud Storage bucket.

4. **Vision API Integration:**

- Initializes Google Cloud Vision API client.
- Sends an asynchronous request to the Vision API for label detection on the original image.

5. **Menu Service Integration:**

- Extracts the item ID from the file name.
- Retrieves existing menu item details from the menu service using its API.
- Updates the menu item with the original and thumbnail image URLs.

6. **Status Update:**

- Processes the Vision API response to determine the menu item's status based on detected labels.
- Updates the menu item's status in the menu service accordingly.

7. **Error Handling:**

- Includes a try...catch block to handle any errors during the process.

Workflow:

1. A Cloud Event triggers the 'process-thumbnails' function when a new image is uploaded to a specific Cloud Storage bucket.
2. The function downloads the image, generates a thumbnail, and uploads it to another bucket.
3. Concurrently, it sends the image to the Vision API for label detection.
4. Based on the detected labels, the function determines the menu item's status (e.g., "Ready" if "Food" label is present).
5. The function updates the menu item in the menu service with the image URLs and the determined status.

Purpose:

This Cloud Function automates the process of:

- Generating thumbnails for uploaded images.
- Using AI (Vision API) to analyze image content and categorize menu items.
- Updating a menu service with image information and status based on AI analysis.

The explanation helps Kalani quickly understand the code. However, she notices that there is too much code in the base function, which can make the code confusing.

Kalani asks Gemini Code Assist to improve the readability of the code, and specifies some changes that should be made:



Make index.js more readable. Extract functions and constants where appropriate, and add detailed comments.

Gemini Code Assist simplifies the code by moving five functional parts of the process-thumbnails function into separate functions. Comments are added, and the thumbnail dimension constants are extracted. The code is more readable than it was before the request.

Gemini Code Assist full response (readability updates)

```
// Copyright 2021 Google LLC
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
// https://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
'use strict';
```

```

const imageMagick = require('imagemagick');
const Promise = require("bluebird");
const path = require('path');
const functions = require('@google-cloud/functions-framework');
const vision = require('@google-cloud/vision');
const {Storage} = require('@google-cloud/storage');
const axios = require('axios');
const fs = require('fs');

// Constants for thumbnail dimensions
const THUMBNAIL_WIDTH = 400;
const THUMBNAIL_HEIGHT = 400;

// Function to extract item ID from a file name
const extractItemId = (fileName) => {
  const itemId = parseInt(path.parse(fileName).name);
  return isNaN(itemId) ? null : itemId;
}

// Function to create a thumbnail from an image file
const createThumbnail = async (originalFile, thumbFile) => {
  await Promise.promisify(imageMagick.crop)({
    srcPath: originalFile,
    dstPath: thumbFile,
    width: THUMBNAIL_WIDTH,
    height: THUMBNAIL_HEIGHT
  });
  console.log(` Created local thumbnail in ${thumbFile}`);
};

// Function to analyze an image using Google Vision API
const analyzeImage = async (bucketName, fileName) => {
  const client = new vision.ImageAnnotatorClient();
  const visionRequest = {
    image: { source: { imageUri: `gs://${bucketName}/${fileName}` } },
    features: [{ type: 'LABEL_DETECTION' }],
  };
  const visionResponse = await client.annotateImage(visionRequest);
  console.log(` Raw vision output for: ${fileName}: ${JSON.stringify(visionResponse[0])}`);
  return visionResponse[0].labelAnnotations;
};

// Function to determine menu item status based on Vision API labels
const determineMenuItemStatus = (labels) => {

```

```

let status = "Failed";
let labelsString = "";
for (const label of labels) {
  // If the label is "Food", set the status to "Ready"
  status = label.description === "Food" ? "Ready" : status;
  labelsString = labelsString.concat(label.description, ", ");
}
console.log(`\nVision API labels: ${labelsString}\n`);
console.log(` Menu Item status will be set to: ${status}\n`);
return status;
};

// Function to update menu item details in the menu service
const updateMenuItem = async (menuServer, itemId, originalImageUrl, thumbnailImageUrl,
status) => {
  // Get the current menu item details
  const item = await menuServer.get(`/menu/${itemId}`);
  // Update the menu item details
  await menuServer.put(`/menu/${itemId}`, {
    imageUrl: originalImageUrl,
    itemName: item.data.itemName,
    itemPrice: item.data.itemPrice,
    itemThumbnailURL: thumbnailImageUrl,
    spiceLevel: item.data.spiceLevel,
    status: status,
    tagLine: item.data.tagLine
  });
};

// Cloud Function to process thumbnail requests
functions.cloudEvent('process-thumbnails', async (cloudEvent) => {
  console.log(` Event ID: ${cloudEvent.id}`);
  console.log(` Event Type: ${cloudEvent.type}`);
  const file = cloudEvent.data;
  try {
    console.log(` Received thumbnail request for file ${file.name} from bucket ${file.bucket}`);
    // Initialize Google Cloud Storage
    const storage = new Storage();
    const bucket = storage.bucket(file.bucket);
    const thumbBucket = storage.bucket(process.env.BUCKET_THUMBAILS);
    // Extract item ID from file name
    const itemId = extractItemId(file.name);
    if (!itemId) {
      console.log(` Invalid file name: ${file.name}`);
    }
    return;
  }

```

```

}
// Download the original image
const originalFile = `/tmp/original/${file.name}`;
const thumbFile = `/tmp/thumbnail/${file.name}`;
fs.mkdirSync(path.dirname(originalFile), { recursive: true });
fs.mkdirSync(path.dirname(thumbFile), { recursive: true });
await bucket.file(file.name).download({ destination: originalFile });
console.log(`Downloaded picture into ${originalFile}`);
// Create a thumbnail
await createThumbnail(originalFile, thumbFile);
// Upload the thumbnail to Cloud Storage
const thumbnailImage = await thumbBucket.upload(thumbFile);
const thumbnailImageUrl = thumbnailImage[0].publicUrl();
console.log(`Uploaded thumbnail to Cloud Storage bucket
${process.env.BUCKET_THUMBNAILS}`);
// Analyze the image using Google Vision API
const labels = await analyzeImage(file.bucket, file.name);
// Determine menu item status based on Vision API labels
const status = determineMenuItemStatus(labels);
// Update menu item details in the menu service
const menuServer = axios.create({
  baseURL: process.env.MENU_SERVICE_URL,
  headers: {
    get: {
      "Content-Type": 'application/json'
    }
  }
});
await updateMenuItem(menuServer, itemId, bucket.file(file.name).publicUrl(),
thumbnailImageUrl, status);
} catch (err) {
  console.log(`Error: processing the thumbnail: ${err}`);
}
});

```

Kalani is an experienced developer, so she included specific instructions for Gemini Code Assist to extract functions and constants, and add comments. **She can also ask Gemini Code Assist to evaluate the code.**

When she asks additional questions in the Gemini Code Assist chat window, **the chat history is an input to the large language model**. Before testing this prompt, Kalani uses the trash can icon to reset chat so that the chat history does not affect the response.

With the original index.js file open, **Kalani asks Gemini Code Assist:**



Are there any issues with this code?

Gemini Code Assist explains the issues it sees in the code, including issues that Kalani did not see. The response includes an edited file to address some of the issues, and an explanation of the changes made to the edited file.

Gemini Code Assist calls out six different categories of issues, and explains why they might be problematic:

Yes, there are a few potential issues with the code:

1. **Error Handling:** The code catches errors broadly in the `catch (err)` block. While this prevents the function from crashing, it might mask potential issues. It's better to have more specific error handling, especially for network requests and file system operations.
2. **File System Operations:** The code uses synchronous file system operations like `fs.existsSync()` and `fs.mkdirSync()`. In a serverless

environment, it's generally recommended to use asynchronous operations for better performance.

3. **Resource Cleanup:** The code downloads the original image and creates a thumbnail in the `/tmp` directory. It doesn't clean up these temporary files after processing. This can lead to storage issues over time. Consider deleting the files after they are no longer needed.
4. **Status Logic:** The code sets the `status` to "Ready" only if the Vision API returns a label "Food". This might be too restrictive. Consider adding more sophisticated logic to determine the status based on the labels or using a different approach altogether.
5. **Hardcoded Values:** The code uses hardcoded values for thumbnail dimensions (400x400). Consider making these configurable through environment variables or function arguments.
6. **Security:** The code directly uses data from the `cloudEvent` object without validation. It's crucial to validate the input data, especially when interacting with external services or file systems.

By addressing these issues, you can improve the code's reliability, performance, and security.

Gemini Code Assist also provides example code to fix some of the issues:

Gemini Code Assist (file edit examples)

// Error Handling

// Instead of:

```
if (!fs.existsSync("/tmp/original")){  
  fs.mkdirSync("/tmp/original");  
}
```

// Use:

```
const exists = await fs.promises.access('/tmp/original').catch(() => false);  
if (!exists) {  
  await fs.promises.mkdir('/tmp/original');  
}
```

// File System Operations

// Instead of:

```
if (!fs.existsSync("/tmp/original")){  
  fs.mkdirSync("/tmp/original");  
}
```

// Use:

```
const exists = await fs.promises.access('/tmp/original').catch(() => false);  
if (!exists) {  
  await fs.promises.mkdir('/tmp/original');  
}
```

// Resource Cleanup

// After uploading the thumbnail:

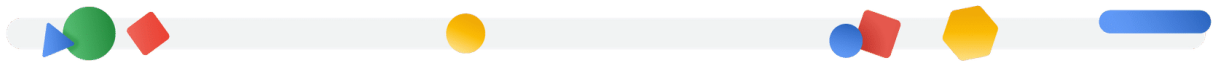
```
await fs.promises.unlink(originalFile);  
await fs.promises.unlink(thumbFile);
```

Kalani can review this feedback and fix the code issues before they are reported by customers.

Kalani can also ask Gemini Code Assist to help in other ways, such as improving the UI of an application and making the code more efficient.



Gemini Code Assist helps you transform
your code
in many ways, including improving
readability,
code structure, and code efficiency.



Debugging and fixing code

Gemini Code Assist can also help debug and fix broken code. A coworker has given Kalani a code snippet for a sorting algorithm that isn't working:

mySort.js (input)

```
const mySort = (arr) => {  
  if (arr.length <= 1) {  
    return arr;  
  }  
}
```

```
const splitIndex = Math.floor(arr.length / 2);
const splitValue = arr[splitIndex];
const left = [];
const right = [];

for (let i = 0; i < arr.length; i++) {
  if (i === splitIndex) continue;
  if (arr[i] < splitValue) {
    left.push(arr[i]);
  } else {
    right.push(arr[i]);
  }
}

return [...mySort(left), ...mySort(right)];
};

// Example usage:
const unsortedArray = [5, 2, 9, 3, 7, 6, 1, 4, 8];
const sortedArray = mySort(unsortedArray);
console.log(sortedArray); // Expected output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

module.exports = { mySort };
```

Kalani can probably debug the code herself, but she decides to let Gemini Code Assist try.

Kalani asks Gemini Code Assist:



What's wrong with this code?

Gemini Code Assist quickly detects the flaw in the code and provides the solution with the fix explanation added as comments:

The provided code implements a sorting algorithm, but it has a critical flaw: **it doesn't include the `splitValue` back into the sorted result.** This results in an incomplete sort where the `splitValue` is omitted in each recursive step.

Here's why this happens and how to fix it:

Problem:

- The code correctly divides the array into `left` and `right` portions based on the `splitValue`.
- However, after recursively sorting `left` and `right`, it directly combines them (`[...mySort(left), ...mySort(right)]`) without reintroducing the `splitValue`.

Solution:

The `splitValue` should be inserted between the sorted `left` and `right` arrays during the merge step. Here's the corrected code:

```
const mySort = (arr) => {
  if (arr.length <= 1) {
    return arr;
  }

  const splitIndex = Math.floor(arr.length / 2);
  const splitValue = arr[splitIndex];
  const left = [];
  const right = [];

  for (let i = 0; i < arr.length; i++) {
    if (i === splitIndex) continue; // Skip the splitValue itself
    if (arr[i] < splitValue) {
      left.push(arr[i]);
    } else {
      right.push(arr[i]);
    }
  }

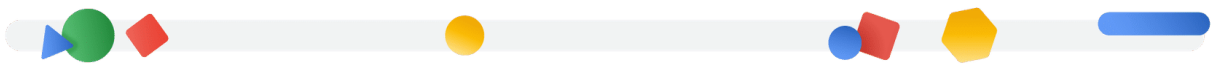
  // Correctly merge with splitValue
  return [...mySort(left), splitValue, ...mySort(right)];
};

// Example usage:
const unsortedArray = [5, 2, 9, 3, 7, 6, 1, 4, 8];
const sortedArray = mySort(unsortedArray);
console.log(sortedArray); // Expected output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

module.exports = { mySort };
```



Gemini Code Assist helps you debug and fix your code.



Generating unit tests

Testing code can help you maintain code quality. Kalani wants to create unit tests for code wherever possible. The sort code, which was broken but now is fixed, seems like a good candidate:

fixed mySort.js (input) —

```
const mySort = (arr) => {  
  if (arr.length <= 1) {  
    return arr;  
  }  
  
  const splitIndex = Math.floor(arr.length / 2);  
  const splitValue = arr[splitIndex];  
  const left = [];  
  const right = [];  
  for (let i = 0; i < arr.length; i++) {  
    if (i === splitIndex) continue; // Skip the splitValue itself  
    if (arr[i] < splitValue) {  
      left.push(arr[i]);  
    }  
  }  
}
```

```
} else {  
  right.push(arr[i]);  
}  
}  
  
// Correctly merge with splitValue  
return [...mySort(left), splitValue, ...mySort(right)];  
};  
  
// Example usage:  
const unsortedArray = [5, 2, 9, 3, 7, 6, 1, 4, 8];  
const sortedArray = mySort(unsortedArray);  
console.log(sortedArray); // Expected output: [1, 2, 3, 4, 5, 6, 7, 8, 9]  
  
module.exports = { mySort };
```

Kalani uses Gemini Code Assist to generate unit tests for the fixed code.

Kalani asks Gemini Code Assist:



Generate unit tests for this code. Include instructions to run the unit tests.

Gemini Code Assist creates a test file, mySort.test.js:

```
const { mySort } = require('./mySort');  
describe('mySort', () => {
```

```
it('should return an empty array when given an empty array', () => {  
  expect(mySort([])).toEqual([]);  
});
```

```
it('should return the array as is when given an array with one element',  
  () => {  
    expect(mySort([1])).toEqual([1]);  
  });
```

```
it('should sort an unsorted array in ascending order', () => {  
  expect(mySort([5, 2, 9, 3, 7, 6, 1, 4, 8])).toEqual([1, 2, 3, 4, 5, 6, 7, 8, 9]);  
});
```

```
it('should handle an already sorted array', () => {  
  expect(mySort([1, 2, 3, 4, 5, 6, 7, 8, 9])).toEqual([1, 2, 3, 4, 5, 6, 7, 8, 9]);  
});
```

```
it('should handle an array with duplicate values', () => {  
  expect(mySort([3, 2, 1, 2, 3])).toEqual([1, 2, 2, 3, 3]);  
});  
});
```

The tests created by Gemini Code Assist include edge cases (empty array, one element), starting with sorted and unsorted arrays, and sorting an array with duplicate values.

As instructed, Gemini Code Assist also returns instructions for running the tests:

To run the tests:

1. **Make sure you have Node.js and npm installed.** You can check by running `node -v` and `npm -v` in your terminal. If you don't have them, download and install from <https://nodejs.org/>.
2. **Save the test code above** as a `.js` file in the same directory as your `mySort.js` file. For example, you can save it as `mySort.test.js`.
3. **Open your terminal** and navigate to the directory where you saved the files.
4. **Run the tests** using the command `npm test mySort.test.js`. This will execute the tests and show you the results.

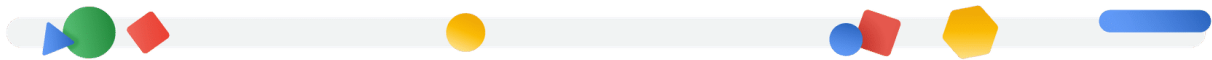
Explanation:

- The tests use the Jest testing framework. If you don't have Jest installed, you can install it globally using `npm install -g jest`.
- Each `it()` block describes a specific test case.
- `expect(mySort(...)).toEqual(...)` asserts that the output of your `mySort` function matches the expected output.

These tests cover various scenarios to ensure your sorting algorithm works correctly.



Gemini Code Assist helps you create tests
for your code
and provides instructions for performing
the tests.



Review

For an experienced developer like Kalani, which of the following is a primary benefit of using Gemini Code Assist?

- ☐ Enhances code quality and readability by suggesting improvements and refactoring.
- ☐ Assists in debugging and fixing code issues, saving time and effort.
- ☐ Generates unit tests to aid in code quality assurance.

☐

All of the above.

SUBMIT

Summary

In this course, you learned how Gemini Code Assist can make it easier to build applications and perform application development tasks. You learned how Gemini Code Assist can be used by both new and experienced developers to perform tasks like creating, explaining, and refactoring code, explaining programming languages, frameworks, and Google Cloud services, and generating unit tests for your code.

Module recap:

- ✓ Use Gemini Code Assist to explain code.
- ✓ Use Gemini Code Assist to make code more readable.
- ✓ Use Gemini Code Assist to add comments to code, in English or another language.
- ✓ Generate unit tests with Gemini Code Assist.

