

**unqork**



# **The 5 Key Steps to Building a Modern Enterprise Application: No-Code vs. Low-Code**

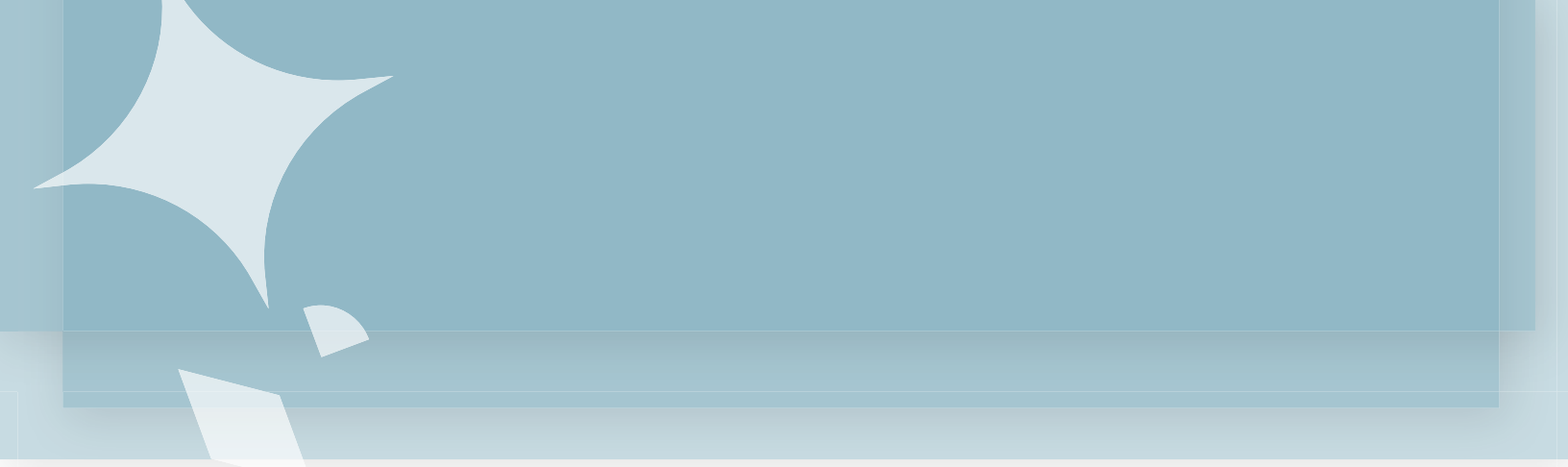
When it comes to enterprise development, the advantages of “no” over “low” are impossible to ignore.

## Contents

Introduction	3
The 5 Key Steps	5
Step 1: Capture User Data via a Multi-Step Form	8
Step 2: Define Business Logic	10
Step 3: Integrate With Third-Party Services and Data Sources	13
Step 4: Query & Process Data From Integrations	15
Step 5: Create APIs to Allow Systems to “Talk” to Your Application	17
Unqork: The World’s First Enterprise No-Code Application Platform	18

**TL:DR**

- “Low-code” and “no-code” sound similar, but are not the same
- Low-code helps engineers automate repeatable coding tasks; no-code eliminates the need to code
- No-code provides benefits at each part of the development process



**A**ccording to [Gartner](#), 65% of all applications created within the next five years will be built using low-code or no-code technologies. “Low-code” and “no-code” often end up grouped together; however, these similar-sounding terms are anything but interchangeable.

**Low-code** technologies were introduced in the early 2000s as a means to automate repeatable coding tasks. These tools added some efficiencies for the pre-mobile, proto-broadband era, but they don’t adequately address the mounting development challenges of today’s enterprise.

Modern companies are under increasing pressure to “go digital” to drive operational efficiencies and address users’ evolving expectations for digital service. However, they must do so within complex ecosystems while competing for a limited pool of experienced programmers—and as a result, development efficiencies across industries have taken a hit.

Consider that between 1980 and 2000, developer productivity steadily improved thanks to new programming languages, coding strategies, and development tools such as low-code. But then, around 2010, the numbers began moving [in the opposite direction](#). The reason is that traditional development tools and methodologies have failed to keep up with the demands of today’s digital ecosystems, which is why a new approach became necessary.

No-code technologies eliminate the need to write or manage code, which means organizations can focus all their resources on addressing business challenges rather than dealing with syntax, bugs, and legacy code (some of which may be decades old).

<sup>1</sup>Consider last year’s strange rush on [COBOL-literate programmers](#) to update long-untouched government systems built on a mostly-forgotten language.

Unqork, the world's first enterprise no-code application platform, is specifically designed to streamline the development of sophisticated applications in complex, highly regulated sectors such as finance, insurance, healthcare, and government. [Global organizations](#) from various sectors have used Unqork to realize a number of benefits such as:

**3x**

Faster  
Development\*

**3x**

Cost Savings\*

**600x**

Fewer Bugs\*

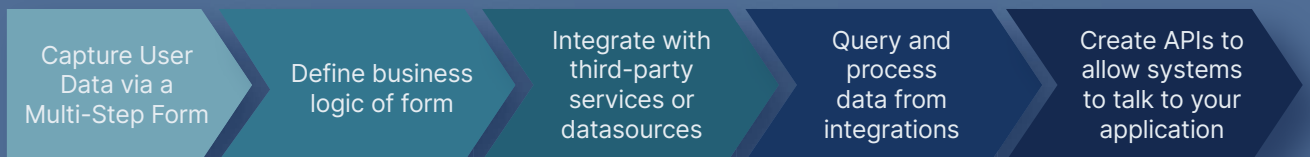
No-code technologies have been around for decades (WYSIWYG web builders from the 90s were technically no-code), but Unqork has expanded the concept using guidance from industry experts with decades of experience to build a robust no-code application platform for accelerating enterprise software development.

Here at Unqork, we strongly disagree with the decision of some industry-watchers to casually bundle both technologies together into a single "low-code/no-code" category. In this eBook, we will attempt to place some sunlight between these two technologies and demonstrate how Unqork is the obvious choice when it comes to building a modern enterprise application.

# The 5 Key Steps

Building with low-code requires trained programmers to write-out and manage hundreds of LOCs—often using a mix of standard programming languages (e.g., Java, Python, etc.) and a proprietary language unique to that low-code tool. No-code application platforms like Unqork provide a level of abstraction between developers (or “Creators” as we refer to them) and the underlying codebase. In Unqork, software is built by manipulating configurable visual components representing user-facing elements, back-

end logic, and third-party integrations via an intuitive visual UI. As a result, development is dramatically simplified and even non-IT staff can take part in the development process. The process of building a modern enterprise application can be separated into five key steps. When using a traditional code/low-code based approach, each step requires a great deal of time and resources from a trained engineer. While with no-code, organizations can tap into the power of a unified, visual platform to streamline and accelerate development.



**unqork**  
Pure No-Code



**Low-Code Platform**  
Requires Code





## Step 1

### Capture User Data via a Multi-Step Form

In order to kick-start a business process, users supply relevant information about who they are and what they want to accomplish. In a previous technological era, this would happen through in-person or phone engagements with a live agent or representative. This was an inherently expensive approach that wasn't always available (i.e., "please try your call again during regular business hours") or would lead to poor data quality as it relies on human intermediation from the agent. Fast forward to today and this information can be collected via a digital form online from the comfort and familiarity of a user's personal device. These forms are the foundation for any modern enterprise application, but they can be challenging to produce with a low-code platform.

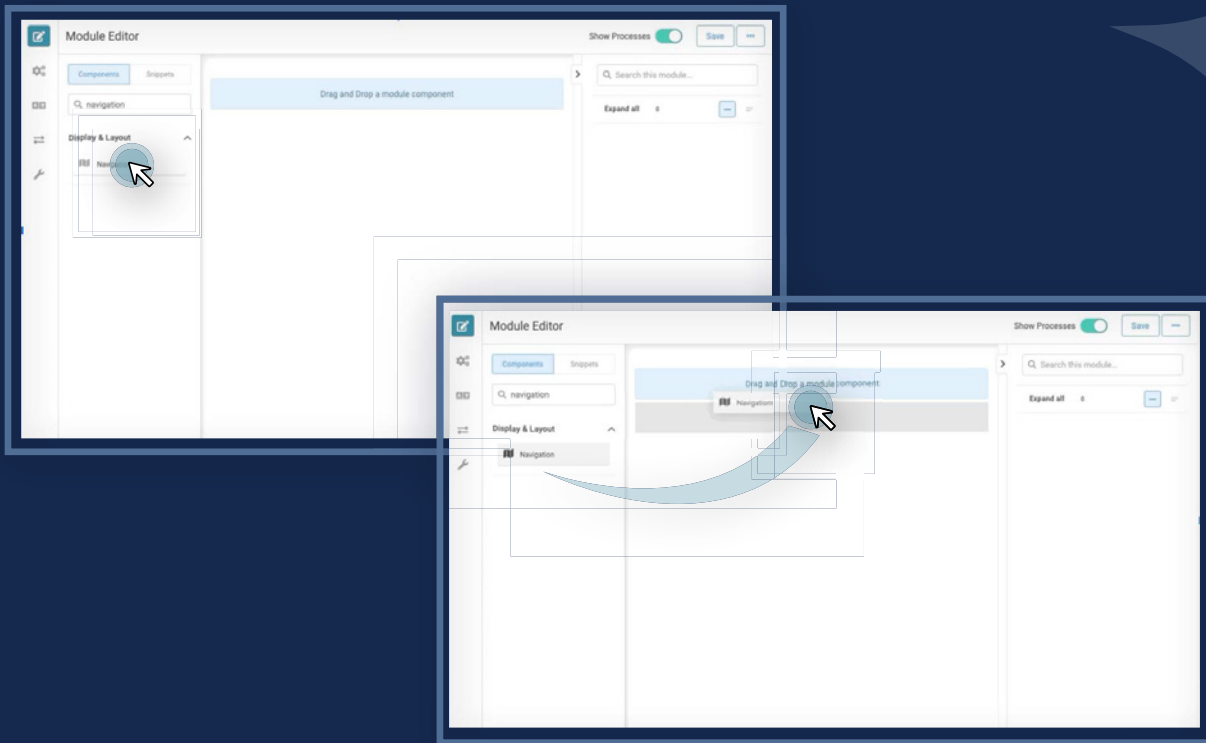
### Onboarding Wizard

First Name*	Department*
<input type="text" value="John"/>	<input type="text" value="HR"/>
Last Name*	Title*
<input type="text" value="Doe"/>	<input type="text" value="Manager"/>

**According to a popular low-code platform's website, building a simple form like the one shown here requires a trained programmer to write 111 LOCs.** Note: the 111 LOCs do not include key enterprise features like field-level validation or a responsive UI—that functionality would require even more coding resources.

```
1 a!localVariables(  
2   local!employee:a!map( firstName:null,  
3   lastName:null, department:null, title:null,  
4   phoneNumber:null, startDate:null ),  
5   local!currentStep: 1,  
6   local!steps: {"Step 1", "Step 2",  
7   "Review"},  
8   a!formLayout(  
9     label: "Example: Onboarding Wizard",  
10    contents:{  
11      a!sectionLayout(  
12        contents:{  
13          a!milestoneField(  
14            steps: local!steps,  
15            active: local!currentStep  
16          )  
17        }  
18      ),  
19      a!sectionLayout(  
20        contents:{  
21          a!columnsLayout(  
22            columns:{  
23              a!columnLayout(  
24                contents:{  
25                  a!textField(  
26                    label: "First Name",  
27                    labelPosition: if(  
28                      local!currentStep = 3, "ADJACENT", "ABOVE"),  
29                    value: local!employee.  
30                    firstName,  
31                    saveInto:  
32                      local!employee.firstName,  
33                    readOnly:  
34                      local!currentStep = 3,  
35                    required:  
36                      not(local!currentStep = 3),  
37                    showWhen: or(  
38                      local!currentStep = {1,3} )  
39                    ),  
40                  a!textField(  
41                    label: "Last Name",  
42                    labelPosition: if(  
43                      local!currentStep = 3, "ADJACENT", "ABOVE"),  
44                    value: local!employee.  
45                    lastName,  
46                    saveInto:  
47                      local!employee.lastName,  
48                    readOnly:  
49                      local!currentStep = 3,  
50                    required:  
51                      not(local!currentStep = 3),  
52                    showWhen: or(  
53                      local!currentStep = {1,3} )  
54                    ),  
55                }  
56              )  
57            }  
58          )  
59        }  
60      )  
61    }  
62  )  
63 )
```

Using **Unqork**, the same form with advanced functionality detailed above can be built and deployed **in a matter of minutes**. Creators simply drag-and-drop components and visually configure each step in the multi-step navigation form.



**unqork** Logout

Progress: ✓ Name ✓ Address ● Date of Claim ○ Claim Description

### Enter details about your claim

Date of Claim

Save Draft and Exit Previous Next

**unqork** Logout

Progress: ✓ Name ✓ Address ✓ Date of Claim ○ Claim Description

### Complete the following details

Address

Save Draft and Exit Previous Next



## Step 2

### Define Business Logic

Now that the user has submitted information, organizations can begin building a high-quality journey in which users are presented with relevant (if not personalized) information and processes.

Regardless of which development tools are used, developers must anticipate all potential inputs and how the system should react to them. This underlying business logic is defined by a series of if/then scenarios. For example: **IF** the user selects radio button A, **THEN** show field B; or, **IF** the user types value X, **THEN** display pop-up Y. In a real-life scenario, this might result in behavior such as:

- Reflexively displaying a “Country of Citizenship” field after your end-user answers that they’re not a US citizen.
- Showing state-specific question labels, based on your end-user’s selected state of residence (e.g., if a user states they live in Louisiana, the form could ask them for their “Parish” as opposed to “county”).
- Adjusting a maximum loan amount based on your end-user’s calculated credit score.



**In a low-code system**, developers need to write out the code for each individual scenario and the resulting workflows (e.g., error and/or success messages, or automatic NIGO identification and remediation workflows to ensure high quality data capture).

```
/*  
 * This validation occurs at the form level  
 and is useful when the form or  
 * section's validation checks are non-field  
 specific.  
 */  
 validations: {  
   if(  
     and(isnull(local!phone),  
 isnull(local!email)),  
     a!validationMessage(  
       message: "You must enter either a  
 phone number or an email address!",  
       validateAfter: "SUBMIT"  
     ),  
   },  
 }
```



**Submission Error** ⚠

Phone Number\*

Email Address\*

You must enter either a phone number or an email address!

Submit

**Using Unqork**, all if/then logic has been abstracted into a purely visual configurable decision table where Creators can easily and quickly define:



**Inputs:**

This is what drives the action. Any component works here, including buttons or text fields.



**Decision logic:**

Here you'll configure the if/then rules connecting inputs to outputs.



**Outputs:**

This is what results from the decision logic. It could include user-facing messages or other dynamic changes in the UI.

The screenshot shows the unqork interface with a form titled "Complete the following fields". The form has two input fields: "First Name" with the value "Cammy" and "Last Name" which is empty. A red border highlights the "Last Name" field. A modal dialog box is displayed over the form, containing the text: "Oops - 1 error(s) have been found! Please check your module and fix fields outlined in red". There is an "OK" button and a close icon in the dialog.

The screenshot shows the unqork interface with the same form. The "Last Name" field now contains the value "Albares". A modal dialog box is displayed over the form, containing the text: "Success!". There is an "OK" button and a close icon in the dialog.



## Step 3

### Integrate With Third-Party Services and Data Sources

Few modern enterprise systems exist in a vacuum. Any custom system must “play nice” with existing internal systems and data sources as well as external third-party services.

```
public class
SendAccountUsingRESTAPI {
1 private final String clientId =
'4GE.....ATL';
private final String clientSecret
= '892....465';
private final String username =
'abc@abc.com';
```

```
accounts[0].get('My_Formula_
Field_c');
// Option 2
System.debug('New value: ' +
results[0].getSObject().get('My_
Formula_Field_c');
```

```
Http h2 = new Http();
HttpRequest req1 = new
HttpRequest();
3
req1.
setHeader('Authorization','Bearer '
+ accessToken);
```

Write code to authenticate to the legacy system

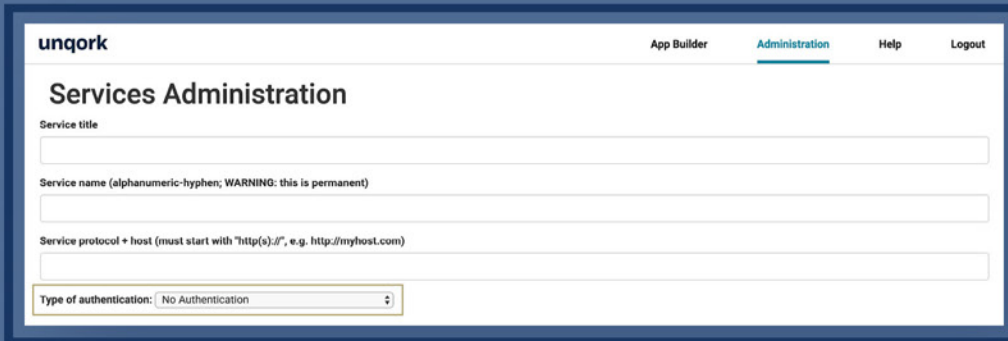
Write code to format data into supported structure

Write code to define how the legacy system will plug into your application

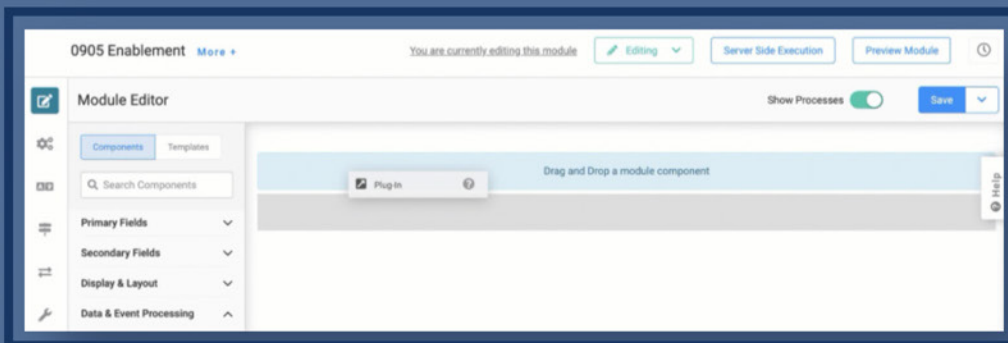
**Using a low-code system,** programmers must write out LOCs to authenticate legacy or external systems and then format data into supported structures. Most low-code systems require programmers to build and maintain a separate script for each separate activity with another system.



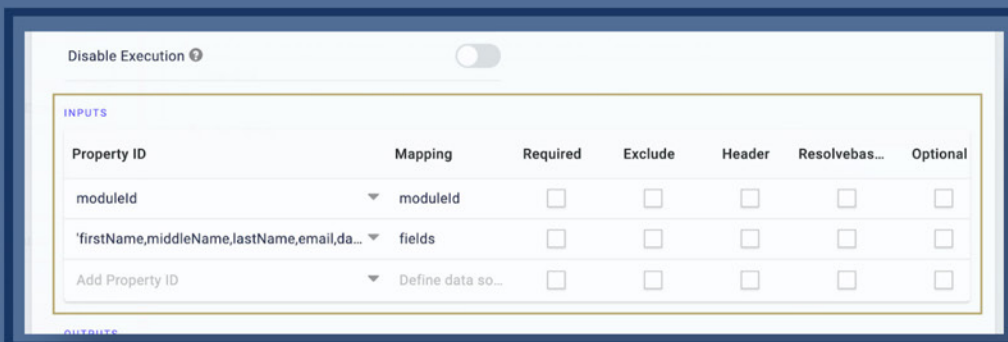
Using Unqork, Creators can quickly set-up authentications, API inputs, and outputs using a visual UI. Once a Creator authenticates an integration, it becomes a reusable component that can be repeatedly placed in a single application or into multiple applications within the ecosystem.



Set up authentication once in the platform administration visual UI



Drag and drop integration component onto the designer canvas



Property ID	Mapping	Required	Exclude	Header	Resolvebas...	Optional
moduleid	moduleid	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
firstName,middleName,lastName,email,da...	fields	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Add Property ID	Define data so...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Add API inputs and outputs into a simple data table



User no longer has to code the integration endpoints and code how data should be sent back and forth



## Step 4

### Query & Process Data From Integrations

Enterprise applications rely on data to function. However, data by itself doesn't result in functionality. Data processing does. Data processing is the collection and manipulation of data to create a desired result.

Using a **traditional code/low-code** approach, programmers are required to precisely define where the data is being pulled, what function you wish to perform with that data, and then where the transformed data should be sent to.

```
Account A = new Account(Name='xxx');
insert A;
Account B;

// A simple bind
B = [SELECT Id FROM Account WHERE Id =
:A.Id];

// A bind with arithmetic
B = [SELECT Id FROM Account
WHERE Name = :('x' + 'xx')];

String s = 'XXX';

// A bind with expressions
B = [SELECT Id FROM Account
WHERE Name = :'XXXX'.substring(0,3)];

// A bind with INCLUDES clause
B = [SELECT Id FROM Account WHERE :A.TYPE
INCLUDES ('Customer - Direct; Customer -
Channel')];

// A bind with an expression that is itself
a query result
B = [SELECT Id FROM Account
WHERE Name = :[SELECT Name FROM
Account
WHERE Id = :A.Id].
Name];

Contact C = new Contact(LastName='xxx',
AccountId=A.Id);
insert new Contact[] {C, new
Contact(LastName='yyy',
accountId=A.id)};

// Binds in both the parent and aggregate
queries
B = [SELECT Id, (SELECT Id FROM Contacts
WHERE Id = :C.Id)
FROM Account
WHERE Id = :A.Id];

// One contact returned
```

**Using Unqork,** Creators need only configure a pre-made visual component to define inputs, outputs, and data transformations. Our visual data workflow empowers Creators to process data in almost-infinite ways such as:

- ✓ Ingesting data.
- ✓ Unwinding complex data structures.
- ✓ Filtering data to obtain specific items.
- ✓ Viewing data at different points within the workflow.
- ✓ Appending data items to create new structures.
- ✓ Outputting data.

The screenshot displays the 'Data Workflow Component' interface. At the top, there are tabs for 'Data Workflow Options' and 'Permissions'. Below these are input fields for 'Label' (containing 'dwfPopulateAddress') and 'Property Name' (also containing 'dwfPopulateAddress'). A 'Go to Help Page' link is visible in the top right. The main workspace contains a workflow diagram. It starts with an 'Input' node labeled 'addressField', followed by a 'Get' node labeled 'Address Components', and a 'Console' node labeled 'AddressComp'. These three nodes are connected to an 'Unwind' node labeled 'unwind'. The 'Unwind' node is connected to a vertical list of output nodes: 'Output State', 'Output County', 'Output City', 'Output Zipcode', 'Output Street Number', 'Output Street Name', 'Output Unit', and 'Output Country'. Below the main workspace, there are 'Console' and 'Console Unwind' nodes. At the bottom, there is a 'Trigger Type' dropdown menu, and checkboxes for 'Disabled', 'Persistent', and 'Debug'. A 'Show Table' button is located in the bottom right corner.



## Step 5

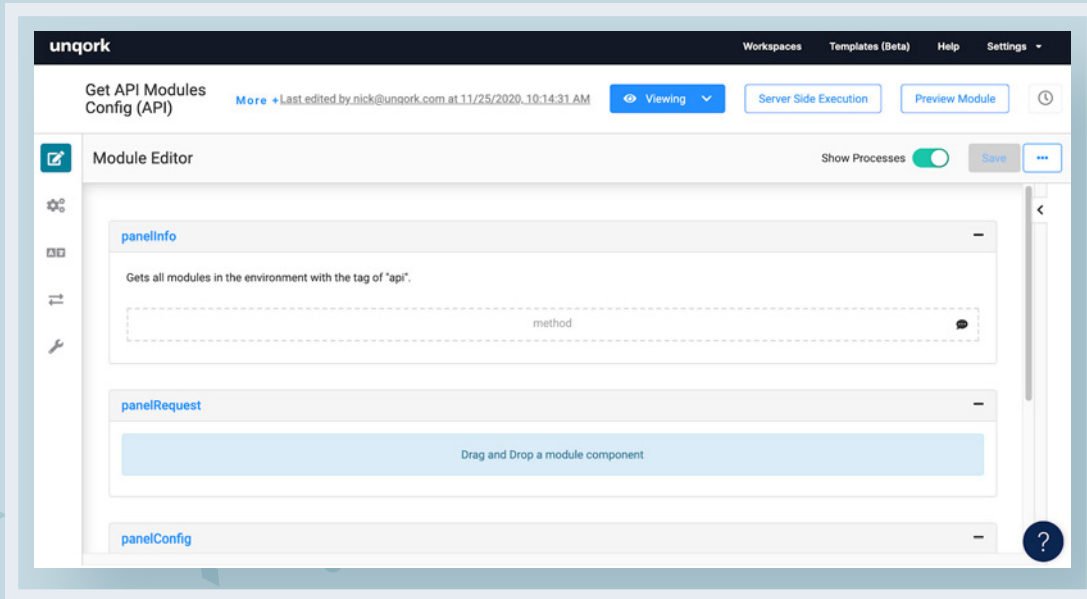
### Create APIs to Allow Systems to “Talk” to Your Application

To build a truly valuable application, you’ll want to make the functionality available to other systems. “Headless” architecture makes a solution’s logic and functionality available via APIs so other systems can become the new front-end for your functionality.

To achieve this using **low-code**, users would need to write code to define the API that exposes the data you want to send (i.e., build an API gateway) and write code to define what part of data you want to send and what format you need to send it in. This process needs to be repeated every time there is a new process in which information is sent to an external application.

```
1  import java.io.IOException;
2  import java.io.OutputStreamWriter;
3  import java.net.HttpURLConnection;
4  import java.net.URL;
5  import java.net.URLEncoder;
6  import java.util.Scanner;
7  import org.json.JSONObject;
8
9  public class RestApiClient {
10
11      public static void main(String[] args) throws IOException{
12
13          Scanner scanner = new Scanner(System.in);
14
15          System.out.println("Welcome to the Person Info Command Line Editor.");
16          System.out.println("(PICLER for short.)");
17          System.out.println("Do you want to get or set a person's info?");
18          System.out.println("(Type 'get' or 'set' now.)");
19          String getOrSet = scanner.nextLine();
20          if("get".equalsIgnoreCase(getOrSet)){
21              System.out.println("Whose info do you want to get?");
22              System.out.println("(Type a person's name now.)");
23              String name = scanner.nextLine();
24
25              String jsonString = getPersonData(name);
26              JSONObject jsonObject = new JSONObject(jsonString);
27
28              int birthYear = jsonObject.getInt("birthYear");
29
30              String about = jsonObject.getString("about");
31              System.out.println("About: " + about);
32          }
33      }
34  }
```

In **Unqork** API creation is a completely visual experience. All API gateways are pre-built & pre-exposed, which means any application built in Unqork can run “headless” at the flip of a switch.





# unqork

## The World's First Enterprise No-Code Application Platform

The enterprise leaders of tomorrow will be the firms who can digitize their processes most thoroughly and adapt their infrastructure most rapidly around shifting business challenges. With no-code, firms are empowered to build scalable, secure, complex, compliant, custom applications with unprecedented speed and flexibility.

That's why many of the most innovative players are partnering with Unqork, the first enterprise no-code development platform specifically designed for the world's most complex and regulated industries. Our platform represents an entirely new paradigm that optimizes every aspect of enterprise development through:



**A unified SaaS platform:** Unqork is a completely unified SaaS platform, which means it provides all the components and capabilities related to crucial areas like **compliance** (up-to-date regulatory and enterprise rules engines for FATCA, CRS, UK CDOT, Dodd-Frank, EMIR, and MiFID II, etc.), **security** (native encryption both in transit and rest, custom RBAC capabilities, and crowd-sourced penetration tests), and **application management** (SDLC governance, application versioning, and module management)<sup>5</sup>.



**A visual UI:** Applications are built via an intuitive, visual User Interface (UI) featuring drag-and-drop components representing user-facing elements, backend processes, data transformations, third-party integrations, and a growing library of industry-specific templates.



**Enterprise-grade standards:** While there are several business-area-specific or consumer-level no-code systems on the market, Unqork is the only no-code platform designed specifically to build complex, scalable, enterprise-ready applications, which is why it's already being used by some of the world's leading organizations.

<sup>5</sup>While Unqork is a SaaS platform, our customers operate in single-tenant environments, which means there is never a mixing of client data between Unqork customers. Unqork is cloud-agnostic, so customers can avoid cloud vendor lock-in and deploy

Unqork allows enterprises to shift all their focus to addressing business challenges instead of technical ones. The platform takes on the “heavy lifting” and frees organizations to invest their resources building operational efficiencies and perfecting the client experience. This streamlined approach helps organizations achieve:

- **Accelerated speed-to-market:** No-code automates many high-volume development tasks so new applications can be built and deployed much faster. In many cases, applications that would take months or years to reach the market can be built in a matter of weeks, or even days.
- **The elimination of legacy code:** Code becomes legacy nearly instantly. With no-code, organizations only need to be concerned with building business logic, even if there is a technical change, the platform handles all that on the backend.
- **Ease of updates and maintenance:** Large enterprises can spend up to 75% of total IT budget maintaining existing systems. One of the reasons is the complexity of making a change in one area requires changes throughout the process. A no-code platform automates many of these cascading tasks and therefore reduces the complexity of making changes.
- **Business agility:** Whether it is a pandemic or disruptions of a smaller scale, no-code can provide organizations with a way to address events quickly.

Curious about how no-code can be applied within your organization? Get in touch to [schedule a demonstration](#) from one of our no-code experts.

---

# unqork

## Enterprise application development, reimagined

Unqork is a no-code application platform that helps large enterprises build complex custom software faster, with higher quality, and lower costs than conventional approaches.

[Request a Demo](#)

[Learn More](#)

