# **\*Note: This guidance was exported from Confluence (which we do not have access to) on 10/18/23. Ask Heather Rienks/Jeff Brauer for more information**

# **Form digitization development guide**

This guide is for developers and serves the purpose of clearly standardizing and explaining the process for digitizing forms using the forms-library. As a general rule, developers following this guide should not need to deviate from the instructions, but there is plenty of [platform documentation](https://depo-platform-documentation.scrollhelp.site/developer-docs/va-forms-library-overview) that can also be referenced when scenarios are not covered here. This guide contains a lot of code snippets. In some situations, these code snippets can be copied and pasted directly, while other times changes are necessary. For our internal team use, each section or step in the guide will correspond to an issue template in GitHub.

## **Before you begin**

Front end and back end developers will need [VA.gov](http://va.gov/) running locally. Front end developers should primarily [follow this guide](https://depo-platform-documentation.scrollhelp.site/developer-docs/setting-up-your-local-frontend-environment) and back end developers should primarily [follow this guide](https://depo-platform-documentation.scrollhelp.site/developer-docs/running-vets-api-locally). Front end developers do not *need* vets-api running locally to do development and back end developers do not *need* vets-website running locally to do development. However, to ultimately test pre-fill, save in progress, and submission functionality of the form, one developer will need to have both code bases running locally. *Note: When authenticating locally, the Docker setup on the backend yields the best results.*

## **Step-by-step guide**

### **Step 1 (front end): Create form directory and skeleton in vets-website**

This is the starting point for front end development.

1. Generate form skeleton using Yeoman generator

o  [Platform documentation for using the Yeoman generator](https://depo-platform-documentation.scrollhelp.site/developer-docs/va-gov-application-generator)

§ Run yarn new:app

§ The generator will ask a series of questions regarding specific details of the form. It is not critical that perfectly correct answers are given on generation as most of these fields can be changed later. However…

§ When prompted with “What folder in `src/applications/` should your app live in?”, the provided directory should be nested under /simple-forms, so /simple-forms/test-form for example

§ When prompted with “Which form template would you like to start with?”, the response should be BLANK: A form without any fields

§ As mentioned, it is possible to change nearly all provide values, but it is important to remember that some of these values live/are referenced in multiple locations, so all locations will need to be updated when making changes

§ Generally, generated values from the prompts are represented in config/form.js, containers/ConfirmationPage.jsx, containers/IntroductionPage.jsx, and manifest.json in vets-website and in src/applications/registry.json in content-build

2. Make tweaks to generated code as necessary

o Don’t make any changes to the Introduction and Confirmation pages as these will be addressed at a later point

o Check the files listed above to make sure all values are correct and make any updates as necessary

o Add the form to VA\_FORM\_IDS in src/platform/forms/constants.js, using the form id as the value.

o Add the form id from the platform constants to missingFromVetsJsonSchema in src/platform/forms/tests/forms.unit.spec.js so that tests can pass when creating a pull request for vets-website (Change this back after a JSON schema has been created in vets-json-schema, or leave in place for schema-less development)

3. Create vets-website pull request for platform approval

o There is a limit on the number of lines a pull request can have, so it’s best to go ahead and create a pull request for the generated code for the new form application in vets-website

4. Create content-build pull request for platform approval

o With the vets-website pull request, we also want to create a content-build pull request for the modified registry.json file

5. Once we are confident in the final URL we’ll also want to add it to the react routes in the devops repository. [See this PR that adds the URL to the react router in devops](https://github.com/department-of-veterans-affairs/devops/pull/12657) (Note: The Devops react routes changes get deployed weekly on Wednesday at 10am ET. You can request an off-cycle deployment if needed.) See appendix for more details on the impact of this change.

6. Merge the pull requests

### **Step 2 (front end): Create form schema in vets-json-schema (skip for schema-less)**

The schema is a somewhat confusing part of the form digitization process. While a schema in the vets-json-schema repository is required to deploy a digitized form to production, the schema doesn’t actually have to be used in the formConfig object and by extension the various uiSchema and schema objects. For our process, we **will** rely on the schema to build out our uiSchema and schema objects, so it’s important to structure the schema in a way that will ultimately be mirrored in the digitized form.

1. Identify all properties and their types from the PDF

o All fields in the PDF that are going to be digitized and be made available to the user on VA.gov need to be identified

o The schema will contain all of these fields as properties and will be frequently imported into vets-website, like when adding the fields to pages in the formConfig object

o The actual property name used for the field isn’t that critical of a decision because the data will ultimately be sent to vets-api where it is parsed and mapped to a PDF

§ The more important part of this will be communicating with the back end developer to know which properties map to which fields in the PDF

o As a general rule, each “section” in the PDF should correspond to a top-level property in the generated JSON schema and the fields in that section should be nested properties

§ For example, if we have a “Veteran” section in the PDF with fields like first name, middle name, last name, social security number, and so on, the properties of the JSON schema that ultimately gets generated should look something like this:

§ "properties": {

§ "veteran": {

§ "type": "object",

§ "additionalProperties": false,

§ "required": [

§ "fullName",

§ "ssnOrTin"

§ ],

§ "properties": {

§ "fullName": {

§ "$ref": "#/definitions/fullName"

§ },

§ "ssnOrTin": {

§ "$ref": "#/definitions/ssn"

§ }

§ }

§ }

},

§ It may not always be possible to follow this pattern exactly, like when fields aren’t clearly in a specific section, so make note of these exceptions when creating the constants.js file in **Step 3**

2. Where possible, utilize existing definitions for properties

o  [Location of all common definitions](https://github.com/department-of-veterans-affairs/vets-json-schema/blob/master/src/common/definitions.js)

3. Build JSON schema following the guide in the vets-json-schema repository

o  [GitHub documentation for creating a form schema](https://github.com/department-of-veterans-affairs/va.gov-team/blob/master/platform/engineering/frontend/vets-website/creating-form-schema.md)

4. Write unit tests for the schema

1. <https://github.com/department-of-veterans-affairs/va.gov-team/blob/master/platform/engineering/frontend/vets-website/creating-form-schema.md#testing>

5. Update package.json version number based on [semantic versioning](https://semver.org/).

6. Create vets-json-schema pull request for platform approval

7. Merge the pull request

### **Step 3 (front end): Build front end of the form using the forms-library**

**BEFORE YOU START**: Examine the mockups *in detail*, and call-out anything that doesn’t look like we can technically do with our current forms-library, uiSchema, schema, or fields. If anything’s deemed required by Designer but involves significant hacks around our current features, sync with fellow Engineers before committing to implement.

This step could potentially be the longest development time depending on the size of the form, but following these sub-steps should make it a consistent and straightforward process. After this step is complete, the majority of the front end work on the actual form is complete.

**NOTE**: As you complete each page, screenshare it with Designer for review/comment on that page, so that any issues can be caught and addressed early.

[Form 26-4555](https://github.com/department-of-veterans-affairs/vets-website/tree/main/src/applications/simple-forms/26-4555) is an example of a form that uses vets-json-schema

[Form 21-10210](https://github.com/department-of-veterans-affairs/vets-website/tree/main/src/applications/simple-forms/21-10210) is an example of a form that does not (schema-less pattern)

1. Reference an existing form’s introduction page, finish the generated Introduction page

o The page is located in the generated app directory under containers/IntroductionPage.jsx

o Instead of using the page-file as-is, import and use our shared [IntroductionPageView.jsx](https://github.com/department-of-veterans-affairs/vets-website/blob/main/src/applications/simple-forms/shared/components/IntroductionPageView.jsx) wrapper. [See [21-10210’s IntroductionPage.jsx](https://github.com/department-of-veterans-affairs/vets-website/blob/main/src/applications/simple-forms/21-10210/containers/IntroductionPage.jsx) for example]

2. Reference an existing form’s confirmation page, finish the generated Confirmation page

o The page is located in the generated app directory under containers/ConfirmationPage.jsx

3. Using the JSON schema and design mockups, build out the chapters and pages structure

o  [Platform documentation](https://depo-platform-documentation.scrollhelp.site/developer-docs/va-forms-library-form-config-options#VAFormsLibrary-FormConfigOptions-chapters)

o This work will all be done in the generated config/form.js file by modifying the chapters property in the generated formConfig object

o Generally, each “section” in the PDF corresponds to a “chapter” in the formConfig object, but the design mockups should reflect any differences

o Each named chapter object will have a title property and a pages object, which should contain at least one nested named page object with a path and title property

§ Again, follow the mockups for chapter and page titles and page distribution/structure

§ Skip writing the uiSchema and schema properties for now

o Example:

o formConfig = {

o ...

o chapters: {

o sponsorChapter: {

o title: 'Sponsor Information',

o pages: {

o sponsorInformation: {

o path: 'sponsor/information',

o title: 'Sponsor Information',

o },

o sponsorAddress: {

o path: 'sponsor/address',

o title: 'Sponsor Address',

o },

o },

o },

o applicantChapter: {

o title: 'Applicant Information',

o pages: {

o applicantInformation: {

o path: 'applicant/information',

o title: 'Applicant Information',

o },

o },

o },

o },

};

4. Build out the page definitions

o Create necessary directory and files

§ Create a pages directory at the root of the generated application directory

§ This directory will contain files with the uiSchema and schema objects for each page

§ For each page in the design mockups, create a <pageName>.js file under the created pages directory. The pageName should match the page property key in the formConfig object

5. (If using vets-json-schema) Create a definitions directory at the root of the generated application directory

o Create a constants.js file inside the created definitions directory

o constants.js

§ Example:

§ export const sponsorFields = {

§ parentObject: 'veteran',

§ fullName: 'fullName',

§ ssn: 'ssnOrTin',

§ ...

§ };

§

§ export const applicantFields = {

§ parentObject: 'applicant',

§ fullName: 'fullName',

§ ssn: 'ssnOrTin',

§ ...

};

§ For each top-level property in the previously created JSON schema, create an exported constant that contains the top-level property and all nested properties (like the example above)

§ The parentObject value should match the property name of the top-level property previously defined in the JSON schema

§ The keys of all the nested properties can be anything, but the values must match the property names previously defined in the JSON schema

§ If the previously created JSON schema has top-level properties that don’t have nested properties, they can all be contained in a exported constant like this:

§ export const otherFields = {

§ agreedToTerms: 'agreedToTerms',

§ claim: 'vaClaimNumber',

};

6. Create uiSchema and schema

o uiSchema

§  [Platform documentation](https://depo-platform-documentation.scrollhelp.site/developer-docs/va-forms-library-about-schema-and-uischema#VAFormsLibrary-AboutschemaanduiSchema-uiSchemaobject) on the uiSchema object

§ For each created <pageName>.js file under the created pages directory, use the design mockups to define the uiSchema for each visible field

§ In platform/forms-system/src/js/definitions, there are a number of standardized uiSchema and schema objects for various field types; these should be used wherever possible

§ In addition to utilizing the pre-defined platform uiSchema objects, this is where the previously created constants.js file is imported and utilized (for schema pattern only)

§ Example:

§ import fullSchema from 'vets-json-schema/dist/10-10D-schema.json';

§ import \* as address from 'platform/forms-system/src/js/definitions/address';

§ import { sponsorFields } from '../../definitions/constants';

§

§ export default {

§ uiSchema: {

§ [sponsorFields.parentObject]: {

§ [sponsorFields.address]: address.uiSchema(),

§ },

§ },

};

To summarize the example:

§ Import the generated fullSchema from the vets-json-schema repo

§ Import the created <section>Fields object from constants.js

§ Import all necessary uiSchema objects from the platform/forms-system/src/js/definitions directory

§ Create a uiSchema object that has the <section>Fields.parentObject at the top-level

§ Nested under the parentObject, add all other <section>Fields.<fieldName> properties that belong on the page (based on the design mockup)

§ For each <fieldName>, use an imported uiSchema or write a custom implementation if no importable definition exists

§ Export the uiSchema object

o schema

§  [Platform documentation](https://depo-platform-documentation.scrollhelp.site/developer-docs/va-forms-library-about-schema-and-uischema#VAFormsLibrary-AboutschemaanduiSchema-schemaobject) on the schema object

§ For each created <pageName>.js file under the created pages directory, use the previously created uiSchema objects to define the matching schema

§ Example:

§ import fullSchema from 'vets-json-schema/dist/10-10D-schema.json';

§ import { sponsorFields } from '../../definitions/constants';

§ import { intersection, pick } from 'lodash';

§

§ const { required, properties } = fullSchema.properties[sponsorFields.parentObject];

§ const pageFields = [sponsorFields.address];

§

§ export default {

§ schema: {

§ type: 'object',

§ properties: {

§ [sponsorFields.parentObject]: {

§ type: 'object',

§ required: intersection(required, pageFields),

§ properties: pick(properties, pageFields),

§ },

§ },

§ },

};

To summarize the example:

§ Import the generated fullSchema from the vets-json-schema repo

§ Import the created <section>Fields object from constants.js

§ Import intersection and pick from lodash

§ Create constants for required and properties from the imported fullSchema using the parentObject property

§ Create a schema object that models the example above, with the <section>Fields.parentObject at the top-level under the properties key and with the nested required and properties keys utilizing intersection and pick respectively

§ Export the schema object

o Full <pageName>.js example:

o import fullSchema from 'vets-json-schema/dist/10-10D-schema.json';

o import \* as address from 'platform/forms-system/src/js/definitions/address';

o import { sponsorFields } from '../../definitions/constants';

o import { intersection, pick } from 'lodash';

o

o const { required, properties } = fullSchema.properties[sponsorFields.parentObject];

o const pageFields = [sponsorFields.address];

o

o export default {

o uiSchema: {

o [sponsorFields.parentObject]: {

o [sponsorFields.address]: address.uiSchema(),

o },

o },

o schema: {

o type: 'object',

o properties: {

o [sponsorFields.parentObject]: {

o type: 'object',

o required: intersection(required, pageFields),

o properties: pick(properties, pageFields),

o },

o },

o },

};

7. In config/form.js, import the created pages and fullSchema and add them to the formConfig object

o Import the fullSchema

§ Example: import fullSchema from 'vets-json-schema/dist/10-10D-schema.json';

o Import each created page from the pages directory

§ Example: import sponsorInformation from '../pages/sponsorInformation';

o Add each imported page to its place in the formConfig object

§ Example:

§ formConfig = {

§ ...

§ chapters: {

§ ...

§ sponsorChapter: {

§ title: 'Sponsor Information',

§ pages: {

§ sponsorInformation: {

§ path: 'sponsor/information',

§ title: 'Sponsor Information',

§ uiSchema: sponsorInformation.uiSchema,

§ schema: sponsorInformation.schema,

§ },

§ },

§ },

§ },

};

o In the formConfig object, add/change the defaultDefinitions object to use the definitions from the imported fullSchema

§ Example line to add to the formConfig object: defaultDefinitions: fullSchema.definitions,

o Full formConfig object example showing the changes explained in this section:

o import fullSchema from '../10-10D-schema.json';

o

o import sponsorInformation from '../pages/sponsorInformation';

o import sponsorAddress from '../pages/sponsorAddress';

o import applicantInformation from '../pages/applicantInformation';

o

o const formConfig = {

o ...

o defaultDefinitions: fullSchema.definitions,

o chapters: {

o sponsorChapter: {

o title: 'Sponsor Information',

o pages: {

o sponsorInformation: {

o path: 'sponsor/information',

o title: 'Sponsor Information',

o uiSchema: sponsorInformation.uiSchema,

o schema: sponsorInformation.schema,

o },

o sponsorAddress: {

o path: 'sponsor/address',

o title: 'Sponsor Address',

o uiSchema: sponsorAddress.uiSchema,

o schema: sponsorAddress.schema,

o },

o },

o },

o applicantChapter: {

o title: 'Applicant Information',

o pages: {

o applicantInformation: {

o path: 'applicant',

o title: 'Applicant Information',

o uiSchema: applicantInformation.uiSchema,

o schema: applicantInformation.schema,

o },

o },

o },

o },

};

8. Add some additional configuration to formConfig

1. Add the standard footer and get help info

2. Add the submitUrl to submit wherever is needed (typically the simple forms api)

3. Add the preSubmitInfo configuration that adds the signature to the end of the form

4. Add the shared submit transformer that adds the formId to the submitted payload (if needed create a transformer specific to your form and include that. See the [26-4555 submit transformer](https://github.com/department-of-veterans-affairs/vets-website/blob/main/src/applications/simple-forms/26-4555/config/submit-transformer.js) as an example)

5. import footerContent from 'platform/forms/components/FormFooter';

6. import getHelp from '../../shared/components/GetFormHelp';

7. import transformForSubmit from '../../shared/config/submit-transformer';

8. ...

9. const formConfig = {

10. submitUrl: `${environment.API\_URL}/simple\_forms\_api/v1/simple\_forms`,

11. ...

12. preSubmitInfo: {

13. statementOfTruth: {

14. body:

15. 'insert body',

16. messageAriaDescribedby:

17. 'insert aria message',

18. fullNamePath: <<logic here to determine who should sign the form>>

19. },

20. },

21. ...

22. transformForSubmit,

23. ...

24. footerContent,

25. getHelp,

};

9. Create vets-website pull request for internal approval

10. Merge pull request

### **Step 4 (back end): Generate the back end form model and mapping in the simple\_forms\_api module**

This is the starting point for back end development. See the note on byebug at the end of this document for help troubleshooting.

1. Download the PDF for the form from [va.gov/find-forms](http://va.gov/find-forms)

2. Move that PDF into the modules/simple\_forms\_api/templates folder

3. From the top level of vets-api execute the command:  
 rails simple\_forms\_api:generate\['path to pdf file'\]

*Note: escaping the square brackets is for* zsh*. Omit the backslashes for* bash *or other terminals.* This generates a model and mapping file. The key here is the mapping file, which will determine the PDF recognized field names and map your input field values to them, by name.  
 Note that the pdf\_filler.rb service identifies the mapping file name by the form\_number passed from the front end. Make sure to update the file name as necessary to match. The model and mapping files will be named the same as the PDF file upon generation.

Here is a [demo video for generating the model files](https://dsva.slack.com/archives/C044AGZFG2W/p1669145165640939) from the creator, Charley Stran (need access to DSVA slack)  
 *Note: the generator has been tweaked since this video. The overall demo stands.*

### **Step 5 (back end): Identify and implement form data to PDF mappings in the back end**

This section is the first to use work previously completed by the front end.

1. Add the form to the FORM\_NUMBER\_MAP hash, where the key is the form id passed from the frontend (typically hyphen separated and uppercase letters) and the value is the corresponding snake case representation of the form in vets-api (typically underscore separated and lowercase letters). The value should match the naming convention for all files that contain the form id in their name.

2. For the form data, identify how each property maps to a fillable field in the PDF. This requires a joint effort between front and back to determine schema (Step 2.)

1. In the form\_mappings/\*.json.erb file for the given form, the right side of the object is where you should update the mappings. *Note: the keys get translated from camel case to snake case in the API call, so treat them as snake case in the back end.* Example of getting the first name:  
 .json

2. {

3. veterans: [

4. {

5. full\_name: {

6. first: "Jane",

7. middle\_initial: "M",

8. last: "Doe"

9. }

10. }

11. ]

}

.json.erb

<%= data['veterans'][0]&.dig('full\_name', 'first') %>

3. Add a sample JSON payload in the back end for tests - in the spec/fixtures folder of the simple\_forms\_api module

4. For non-fillable fields in the PDF, either regenerate the PDF to make them fillable or stamp the data using an x, y coordinate

1. See this [guide for filling fields via X,Y coordinates](https://vfs.atlassian.net/l/cp/oXDMmTru)

5. Add tests for the new model in the spec/ folder of the module

1. in the pdf\_filler\_spec add tests for filling the pdf

2. test\_pdf\_fill 'form\_number\_here'

3. # and/or

test\_pdf\_fill 'form\_number\_here', 'test\_payload\_file\_here'

and to test the .json.erb is valid

test\_json\_valid 'json\_erb\_file\_name'

4. in uploads\_spec add tests for calling the submit endpoint

test\_submit\_request 'test\_payload\_file\_here'

6. If needed, follow Step 6 below. Otherwise create vets-api pull request for internal and platform approval

7. Merge pull request

Note that at any point while testing locally you can find generated PDFs in tmp/<pdf name>.pdf at the root of your local vets-api clone.

### **(As necessary) Step 6 (back end): Configure additional attachments or considerations for the form that submits to Benefits Intake API**

The simple\_forms\_api module supports sending the filled out PDF to the [Lighthouse Benefits Intake API](https://developer.va.gov/explore/benefits/docs/benefits?version=current) The module submission endpoint is: /simple\_forms\_api/v1/simple\_forms

1. If the form expects uploaded files, add attachments to the request (6/26/23 yet to be done)

2. If the form expects multiple form sheets for array-like data, generate and fill the appropriate amount, or consider limiting the front end to one forms-worth at a time

See notes at the end of this document for more information on how the backend interacts with the Benefits Intake API.

### **Step 7 (collaborative): Add authenticated experience support to the form**

Working together, the front end and back end developers should add save in progress and pre-fill functionality to the digitized form.

1. Follow the [Save In Progress documentation](https://depo-platform-documentation.scrollhelp.site/developer-docs/va-forms-library-how-to-set-up-save-in-progress-si) *Note: the documentation indicates you need prefill to be configured. We found it works regardless.*

2. Create vets-website pull request for Save in Progress work for internal approval

3. Merge pull request

(Optional) Add prefill

1. Follow the [documentation for prefill](https://depo-platform-documentation.scrollhelp.site/developer-docs/va-forms-library-how-to-work-with-pre-fill), taking the following notes into consideration

1. Identify all fields that can be pre-filled in the form by cross-referencing the available data in the back end

1. The vets-api config/form\_profile\_mappings/{formfile}.yml file can be tweaked to include as much info as available.  
 The structure of this info comes from app/models/form\_profile.rb. You can reference the first section of this file to get more information on its proper pathing.  
 ex:  
 app/model/form\_profile.rb

2. ...

3. class FormIdentityInformation

4. include Virtus.model

5.

6. attribute :full\_name, FormFullName

...

corresponding config/form\_profile\_mappings/{formfile}.yml

fullName: [identity\_information, full\_name]

FormIdentityInformation -> identity\_information  
 :full\_name -> full\_name

2. Add Prefill support for all pre-fillable fields to the front end by creating a [prefill-transformer](https://github.com/department-of-veterans-affairs/vets-website/blob/form/1010d/src/applications/1010d/config/prefill-transformer.js)

3. The name you give the form in app/model/form\_profile.rb is just to easily identify the form. The value assigned to that form name (the form number) must match the key added to the FORM\_ID\_TO\_CLASS hash. Similarly the value in that hash pair must match the class name given to your new model file.

4. In the config/settings.yml file, simply use the name you gave the form above and set prefill to true.

5. The returnUrl in the form profile model must point to the desired path the website should go to after clicking on 'Start form' in the UI

2. Create vets-website pull request for internal approval

1. Note: the SiP changes overlap with a few teams' folders.

1. For the platform-design-system-team, raise a support request for the "design system/forms library"

2. Ping someone from benefits-team-1-frontend

3. Tag vsa-authd-experience-frontend in a comment, and followup with someone on the team if necessary

3. Create vets-api pull request for internal and platform approval

4. Merge pull requests

There is an [additional dev markdown file outlining the prefill setup](https://github.com/department-of-veterans-affairs/va.gov-team/blob/master/teams/vsa/teams/ebenefits/engineering/prefill_setup.md). It has more detail if needed.

The back end tests (I believe) will fail unless you add the form json to the [vets-json-schema](https://github.com/department-of-veterans-affairs/vets-json-schema) repo. If following the schema-less pattern, omit the tests.

### **Step 8 (front end): Design and UI/UX catch-all for the form in the front end**

This step represents all the “finishing touches” to the fields in the digitized form. In theory, not much should need to be changed here if the mockups were mimicked properly.

If the IA review wants us to change the URL keep in mind we need to make changes in vets-website, content-build, and devops.

1. Work with design/content to finalize all UI/UX decisions throughout the form

2. If necessary, create vets-website pull request for internal approval

3. Merge pull request

### **Step 9 (front end): Complete 508/Accessibility review for the form**

The accessibility review process is somewhat tedious, especially if there are many issues. However, it is required in order to ship a digitized form to production.

1. Follow all required steps in the 508/Accessibility testing process to ensure the form meets all requirements

2. Makes changes as necessary from the results of the test

3. If necessary, create vets-website pull request for internal approval

4. Merge pull request

### **Step 10 (collaborative): Write front end and back end tests for the form**

At this point, the digitized form is complete and we’re ready to start heading towards a production release.

1. Write unit tests in vets-website

2. Write end-to-end tests in vets-website

3. Write all necessary tests in vets-api if not covered in Step 4

4. Create vets-website pull request for internal approval

5. Create vets-api pull request for internal and platform approval

6. Merge pull requests

### **Step 11 (collaborative): Verify everything works in Staging**

For our team’s scope, we should be able to do this testing internally. Any issues identified during this step should be addressed immediately.

1. Do user testing to verify that all aspects of the digitized form work in the Staging environment

2. If issues are identified, create vets-website and/or vets-api pull requests to address the problems

3. Update tests in vets-website and/or vets-api to catch the identified issues

4. Reminder: if you haven’t, add the final URL to the react routes in the devops repository. [See this PR that adds the URL to the react router in devops](https://github.com/department-of-veterans-affairs/devops/pull/12657) (Note: The Devops react routes changes get deployed weekly on Wednesday at 10am ET. You can request an off-cycle deployment if needed.)

5. Merge pull requests

6. Repeat testing

### **Step 12 (collaborative): Slow-rollout the form to production**

The final step in the form digitization process.

[Add flipper functionality by adding static pages](https://github.com/department-of-veterans-affairs/vets-website/pull/24531/files) and provide a link to the static pages folder to the Drupal sitewide team to implement.

[Add flipper in the backend](https://github.com/department-of-veterans-affairs/vets-api/pull/12248/files)

· Make sure someone on the team has a valid [http://ID.me](http://id.me/) login and [add their email to the flipper admin list](https://github.com/department-of-veterans-affairs/vets-api/pull/12260)

·  [For more info on using the flippers](https://department-of-veterans-affairs.github.io/veteran-facing-services-tools/platform/tools/feature-toggles/#flipper-ui)

1. Implement feature toggles using Flipper to slow-rollout the form to production

2. Update definition in content-build to enable the form route in prod

1. Specifically, update it such that "vagovprod": true,

3. Create vets-website, vets-api, and content-build pull requests for internal and platform approval

4. Merge pull requests

Once launched

1. Release the form at some team-defined percentage. Be sure to update both the conditional toggles with the defined value.

2. Observe Datadog (see notes below) traces and logs. Identify and address issues if they come up in production

3. Once a few forms are submitted either disabled the form once again to verify processing, or release the form further as per team’s plan

Once live at 100%

1. Remove feature toggles to make the form fully available in production

2. Create vets-website and vets-api pull requests for internal and platform approval

3. Merge pull requests

## **Appendix**

### **Postman API collection for backend development**

We have individual guides for setting up Benefits Intake API and Benefits Claims API.

This collection is merely a set of sample form submissions and two benefits intake API calls for retrieving info about a submitted form.

If any of this stops working please check for updates to our vets-api module, or for benefits intake the [Benefits Intake API Lighthouse Docs](https://developer.va.gov/explore/benefits/docs/benefits?version=current) for updated schemas, and you can download the full collection from the same page.

### **Devops repository react routes**

Until you add the form URL to [react\_routes.yml](https://github.com/department-of-veterans-affairs/devops/blob/master/ansible/deployment/config/revproxy-vagov/vars/react_routes.yml) the authenticated experience will not redirect properly. To mitigate this, after logging in remove /introduction from the URL and it should redirect properly. You can also login in one tab and navigate to the form in another after logging in.

### 

### **How to use byebug for back end troubleshooting**

byebug works like a break point within the code. You can view variables at this location in the execution to see how it's going.

1. Type the word byebug on its own line where you want to pause execution

2. Start the api (locally is quickest, with rails server or rails s)

3. Execute the code (submit via UI, postman, or similar API tool) and the terminal where it's running will turn into an interactive environment.

1. Type a variable name (ex: form\_number) to see its value.

2. Type logic statements in anticipation of future execution to see if it works as expected

3. Debug as you please

4. Type c and hit enter to continue code execution as normal

### **Local auth in vets-api**

If you see auth issues when submitting forms via Postman or vets-website locally, add this line to the top of the uploads\_controller, with the other skip actions.

skip\_before\_action :verify\_authenticity\_token

We aren’t allowed to add this to the codebase because of security rules, but it often helps locally.

Additionally if you are trying to login as a test user locally the vets-api full docker setup often works best. You can run that with a single command:  
 make up

### **Benefits Intake API integration**

The [Benefits Intake API](https://developer.va.gov/explore/benefits/docs/benefits?version=current), owned by the Lighthouse team, is the downstream receiver of our completed PDFs. The have the ability to store, resubmit, and check the status of files. They have a daily and monthly submission report. Talk to #vfs-evss-service-migration in DSVA slack to get access. Our prod API key is used by all of [va.gov](http://va.gov/). The consumer name is va\_gov\_benefits\_team. Unfortunately that means we cannot know for sure if errors referenced in that report are from our form or someone else’s. Work with the Lighthouse team for followup. They have told us they already address every error.

We got the prod API key using the form in the Lighthouse documentation. At this time they said it will never need to be rotated. The key was put in an AWS secret by the platform team, via request. Additionally we had to add that secret to the deployment environment by making a change in the [devops repository](https://github.com/department-of-veterans-affairs/devops/pull/13052/files) and the [vsp-manifests repository](https://github.com/department-of-veterans-affairs/vsp-infra-application-manifests/pull/1932/files). Follow the [platform documentation](https://depo-platform-documentation.scrollhelp.site/developer-docs/vets-api-on-eks#VetsAPIonEKS-SecretValues) for latest guidance should we ever need to make changes.

Anyone can request a sandbox key to use locally. You can download the OpenAPI spec from the Lighthouse documentation and import it to [Postman](https://www.postman.com/) or a similar tool for testing. You can configure your local vets-api to submit to the sandbox environment by adding this to your settings.local.yml

forms\_api\_benefits\_intake:

api\_key: <sandbox key>

url: https://sandbox-api.va.gov/services/vba\_documents/v1/

You can then either interact with it using your setup, or follow this guide for a simplified experience downloading PDFs from the sandbox.

The nice thing about this is you can also retrieve PDFs that were submitted in the staging environment for review, which improves our end-to-end testing capability.

### **Monitoring in Datadog**

[Platform guide for getting access to Datadog](https://depo-platform-documentation.scrollhelp.site/developer-docs/get-access-to-datadog)

In our vets-api controller we’ve added a tag to the top level trace using:  
 Datadog::Tracing.active\_trace&.set\_tag('form\_id', params[:form\_number])

You can add the column view by selecting “Options” and typing @form\_id

Additionally you can filter the trace view by form type, to get a better idea of number of forms and errors by form type. In the “Search for” query box add the property and the desired form number:

@form\_id:21-4142

[Here is a link to our production trace view](https://vagov.ddog-gov.com/apm/traces?query=%40_top_level%3A1%20env%3Aeks-prod%20service%3A%22vets-api%22%20resource_name%3A%22SimpleFormsApi%3A%3AV1%3A%3AUploadsController%23submit%22%20&cols=service%2Cresource_name%2C%40duration%2C%40http.method%2C%40http.status_code%2Cstatus%2C%40_duration.by_service%2Ccomplete_trace%2Canalytics_enabled%2C%40http.url%2C%40error.file&historicalData=true&messageDisplay=inline&query_translation_version=v0&sort=desc&spanType=service-entry&spanViewType=errors&start=1686681746711&end=1687286546711&paused=false)

We also added a log for submissions

Rails.logger.info("Simple forms api - sent to benefits intake: #{params[:form\_number]}, status: #{status}, uuid #{confirmation\_number}")

This logs the form number and confirmation number so we can get status updates for individual forms if need be.

[Here is a link to our production log view](https://vagov.ddog-gov.com/logs?query=env%3Aeks-prod%20%22Simple%20forms%20api%22%20service%3A%22vets-api%22%20&cols=host%2Cservice&index=%2A&messageDisplay=inline&stream_sort=desc&viz=stream&from_ts=1682694785433&to_ts=1682709185433&live=true)

### **Monitoring in Sentry**

To filter issues in Sentry select “platform-api” and “production” in the top dropdowns. To filter for just our form add transaction:SimpleFormsApi::V1::UploadsController#submit to the query, next to is:unresolved  
 To view the associated log click the blue text indicating the generic issue grouping

At the top you can click to find all events associated

Then within those events you can find the specific ones that were logged by your api.  
  
 To remove logs that reveal PII open the individual event and select the trash icon. Click the prompt to delete the event.

## **Help and feedback**

·  [Suggest content changes to this page.](https://github.com/department-of-veterans-affairs/va.gov-team/issues/new?assignees=vbellissimo%2C+ssimmorins%2C+amygoldman&labels=documentation-support%2C+pw-footer-feedback&template=platform-website-footer-feedback.md&title=)

·  [Submit new Platform Website content.](https://depo-platform-documentation.scrollhelp.site/support/how-to-provide-and-suggest-new-content-for-the-pla)

·  [Get help](https://depo-platform-documentation.scrollhelp.site/support/getting-help-from-the-platform-in-slack) from the Platform Support Team in Slack.

·  [Submit a feature idea](https://depo-platform-documentation.scrollhelp.site/support/submitting-a-feature-request) to the Platform.