Vuelidate

model-based validation

Vuelidate

model-based validation

Why *model-based* ?
Why not *form* validation ?

The biggest difference you will notice is that the validations are completely decoupled from the template.

Instead of providing rules for different inputs inside a template, you declare those rules for your **data model**.

between Vuelidate and Vue "form" validation libraries

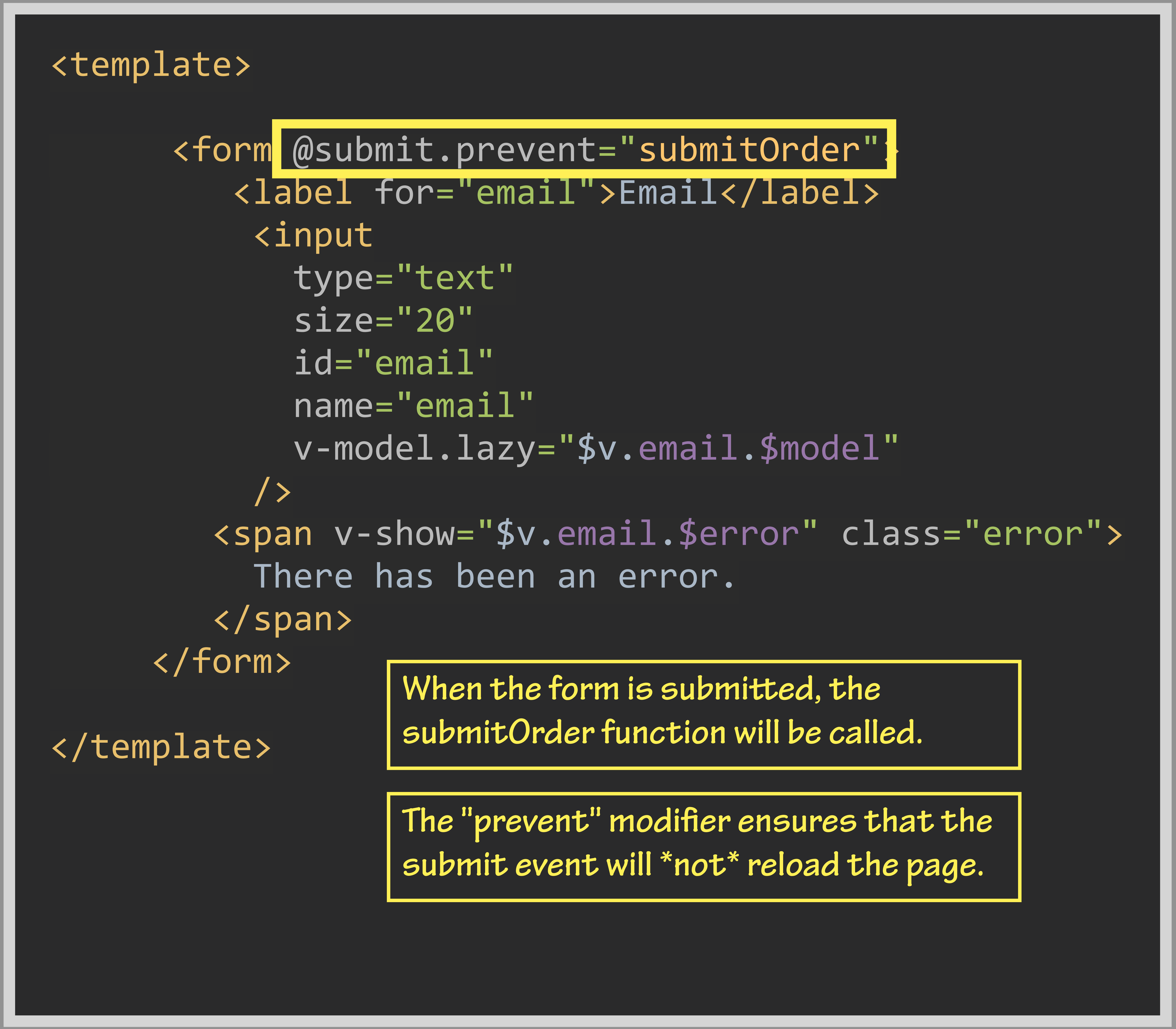
The biggest difference you will notice is that the validations are completely decoupled from the template.

Instead of providing rules for different inputs inside a template, you declare those rules for your **data model**.

Damien Dulisz, 2016
(co-creator of Vuelidate)

```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```

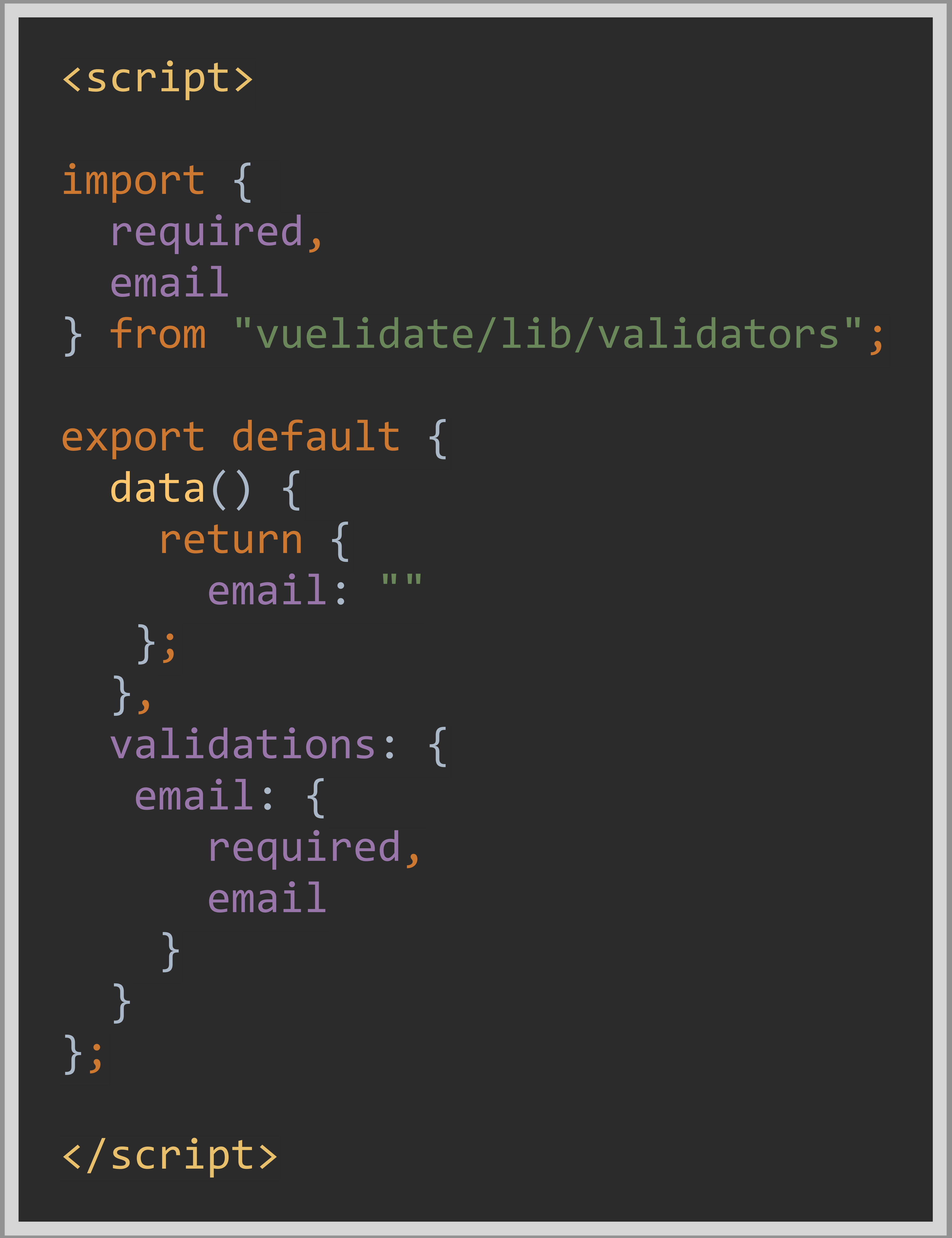
```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
   };
  },
  validations: {
   email: {
      required,
      email
    }
  }
};

</script>
```



```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```

When the form is submitted, the submitOrder function will be called.

The "prevent" modifier ensures that the submit event will *not* reload the page.



```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
    };
  },
  validations: {
   email: {
      required,
      email
    }
  }
};

</script>
```

```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```

The for attribute indicates that the label is associated with the input whose ID is "email" (regardless of the name)

```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
    };
  },
  validations: {
   email: {
      required,
      email
    }
  }
};

</script>
```

```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-                                          or">
          There
        </span>
     </form>

</template>
```

The type attribute has a value of "text", meaning that this field expects text input.

Using a value of "email" here means the browser will try to validate the input.

We don't want that – we want control over how the input is validated and how we respond when the input is not valid.

```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
    };
  },
  validations: {
   email: {
      required,
      email
    }
  }
};

</script>
```

```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```

V-model sets up a two-way binding between the input field and the data.

The lazy modifier means the data changes only after the field loses focus instead of as changing after each keystroke.

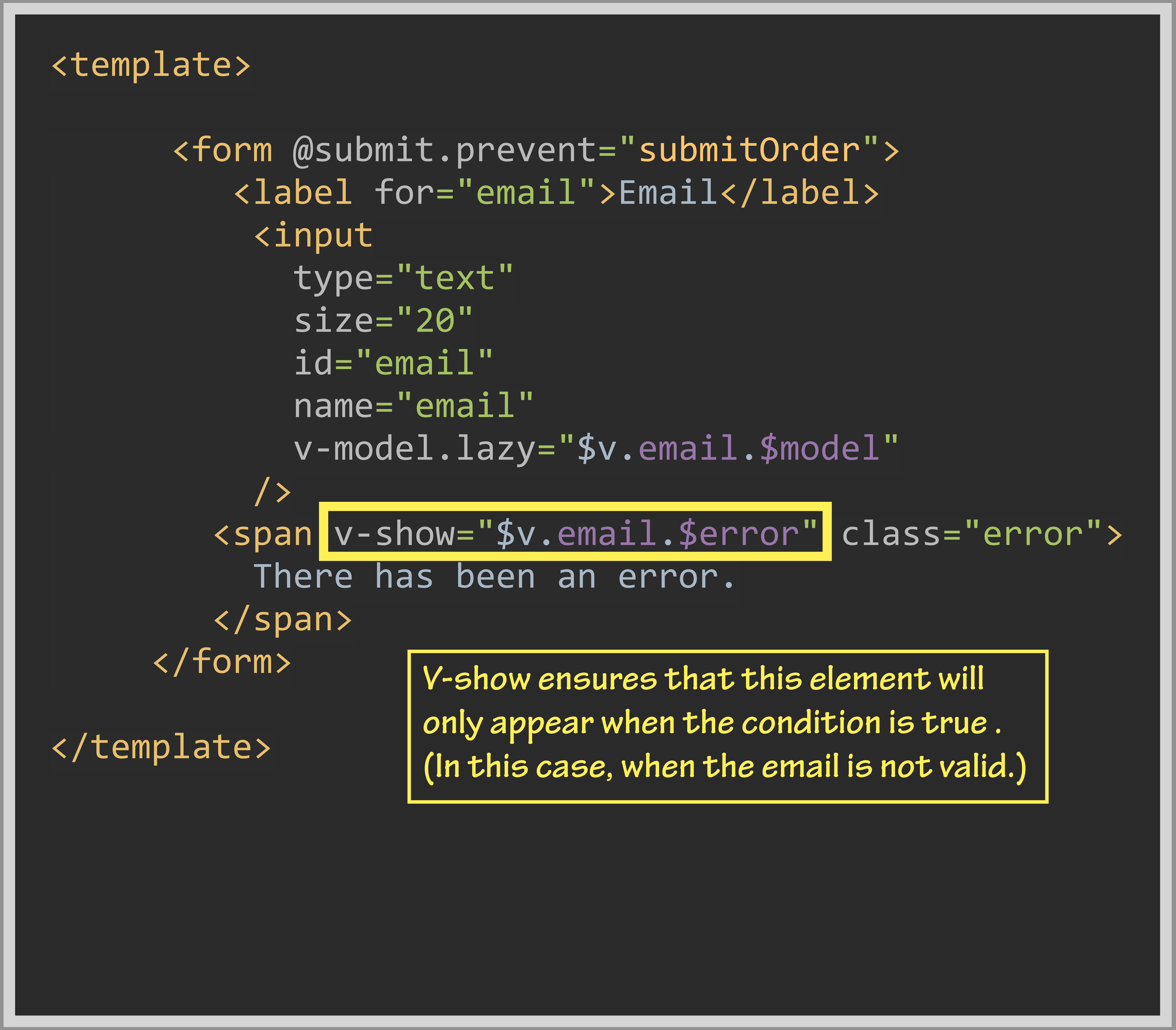
```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
   };
  },
  validations: {
   email: {
      required,
      email
    }
  }
};

</script>
```

```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```

$v.email.$model and the email data will always have the same value.

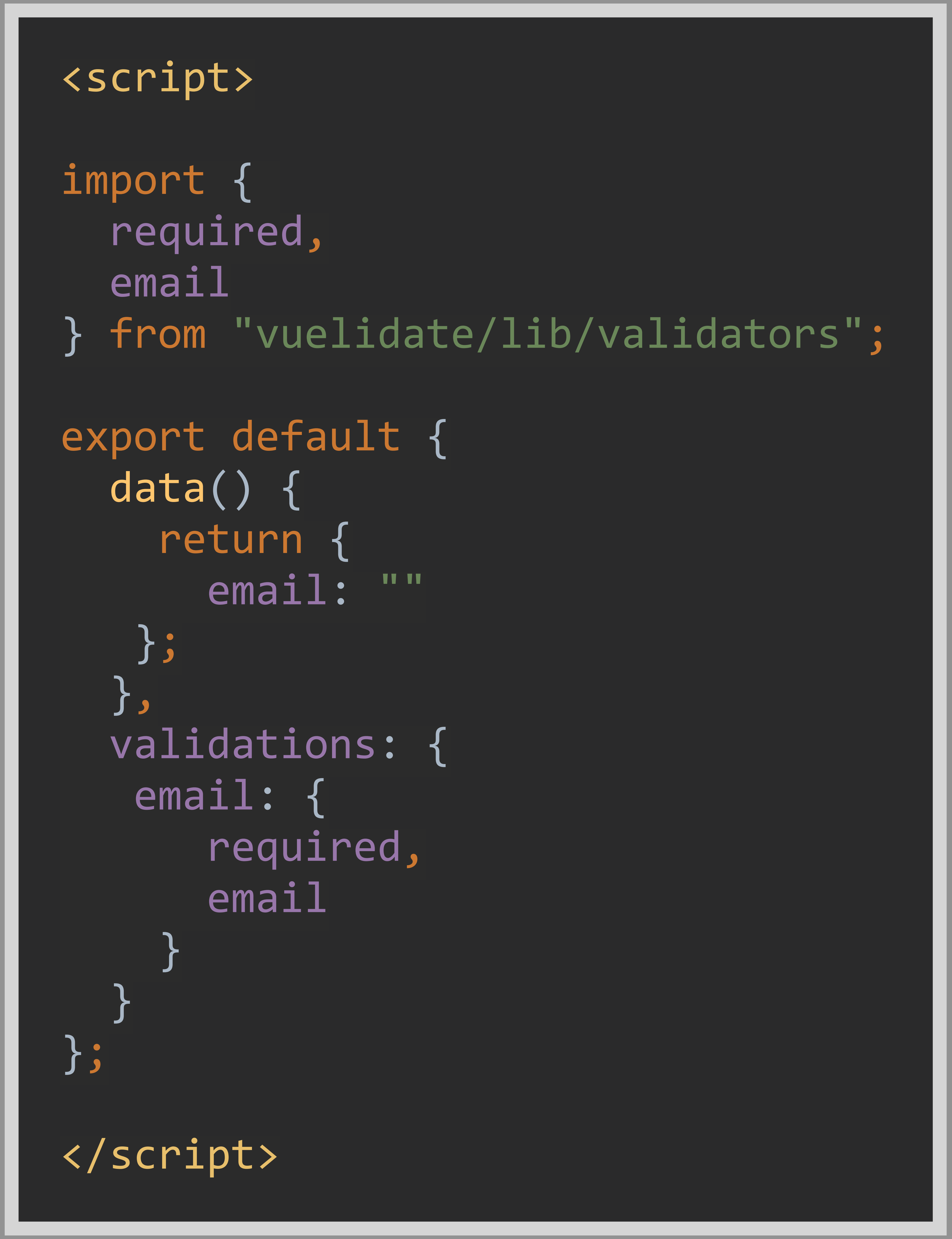
However, note that $v.email will <u>not exist</u> until email is listed in validations.

```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
    };
  },
  validations: {
    email: {
      required,
      email
    }
  }
};

</script>
```



```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```

V-show ensures that this element will only appear when the condition is true . (In this case, when the email is not valid.)

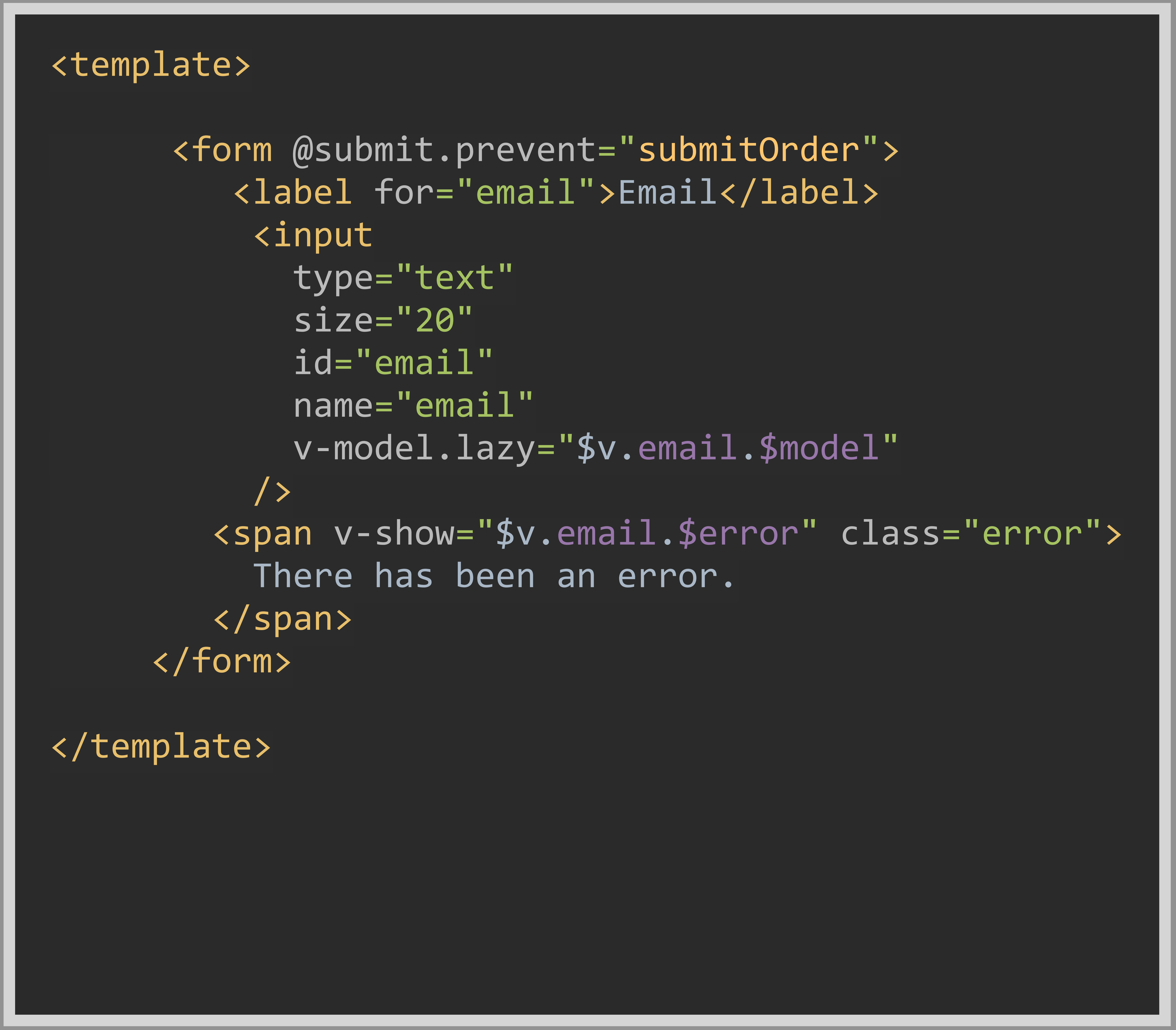


```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
   };
  },
  validations: {
   email: {
      required,
      email
    }
  }
};

</script>
```

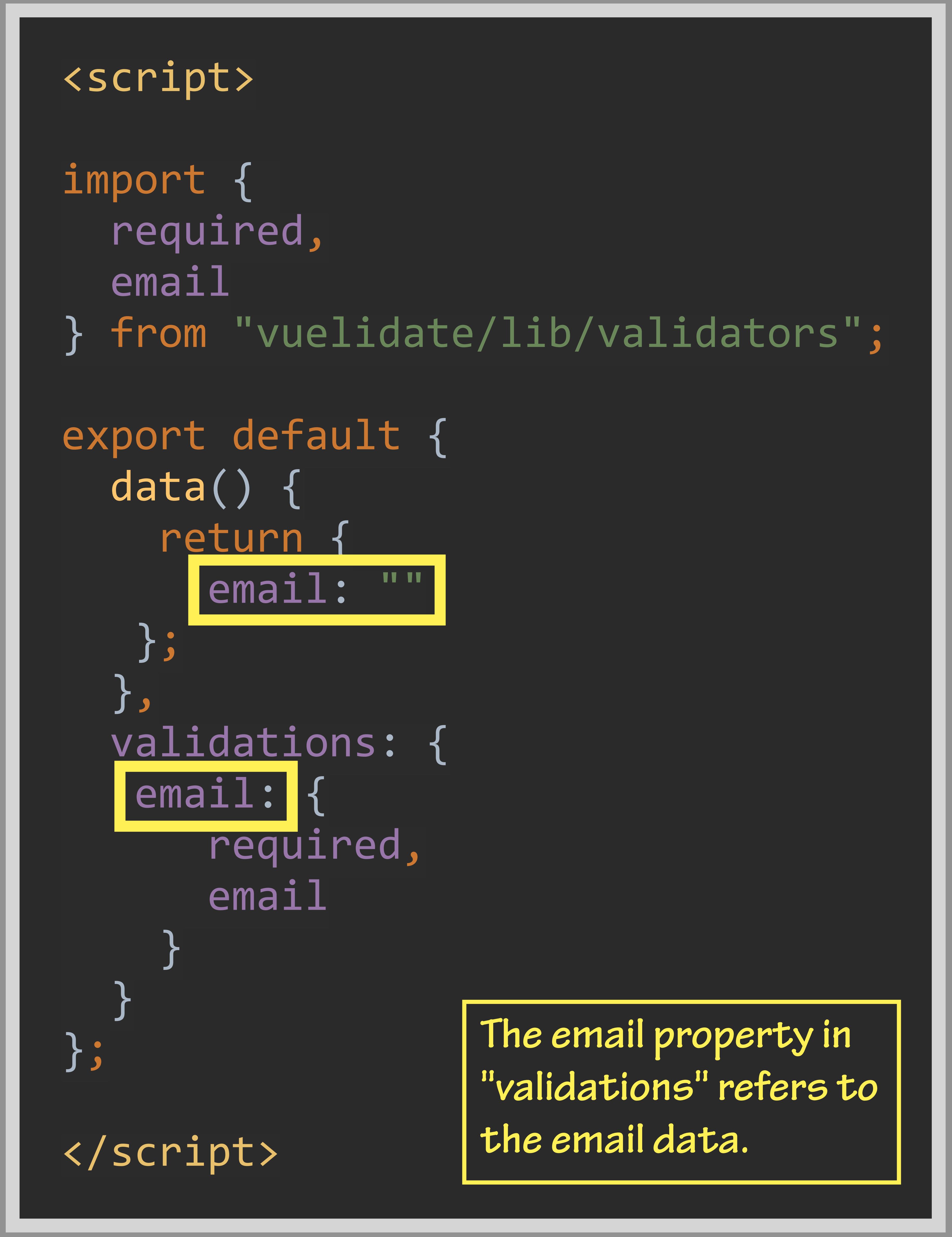
```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```

```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
    };
  },
  validations: {
   email: {
      required,
      email
    }
```

Required and email are validators.
They are used to validate that:
  (1) the email field is not empty, and
  (2) the email value looks like an email.

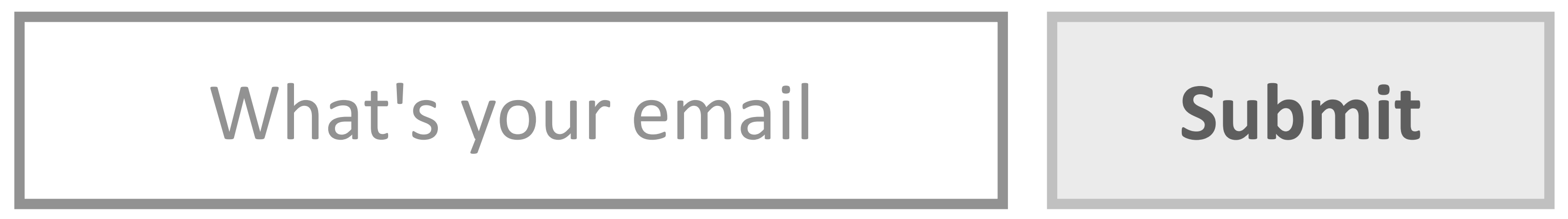


```
<template>

      <form @submit.prevent="submitOrder">
         <label for="email">Email</label>
          <input
            type="text"
            size="20"
            id="email"
            name="email"
            v-model.lazy="$v.email.$model"
          />
        <span v-show="$v.email.$error" class="error">
          There has been an error.
        </span>
     </form>

</template>
```



```
<script>

import {
  required,
  email
} from "vuelidate/lib/validators";

export default {
  data() {
    return {
      email: ""
    };
  },
  validations: {
   email: {
      required,
      email
    }
  }
};

</script>
```

The email property in "validations" refers to the email data.



```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

When you open the form in your browser, you see an input field followed by a submit button. If you open the Vue DevTools and look in the computed properties, you will see a $v object. This is called the $v model, and it represents the current state of validation.

On the left you can see the current state of validation for the email data.

```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

Has the email data been touched? For example, have you clicked on the email field to start modifying a value. If so, the data is dirty and these flags would be set to true.

If you are using the "lazy" modifier with your v-model, the flags will remain true until you tab out of the field.

If any model data that Vuelidate looks at has been touched, $v.anyDirty (not shown here) will be true.

```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

Does the email data have an error? A data property has an error if is has been touched ($dirty) and it is not valid ($invalid) and if the result is not pending (!$pending)

```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

Is the email data invalid? (based on validators)

```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

The value of "email" in the components data

```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

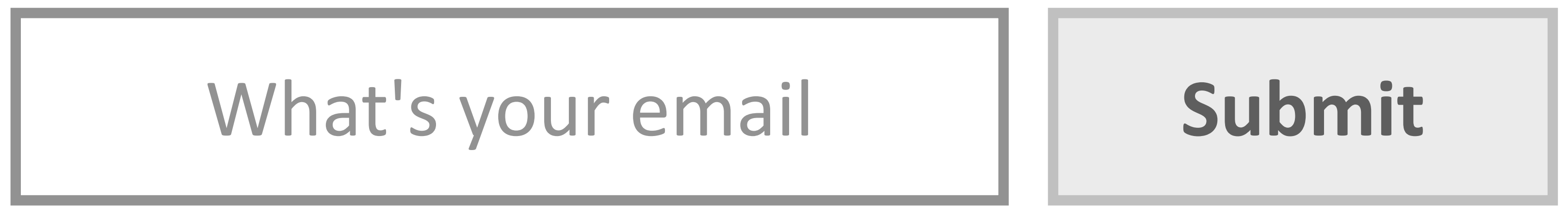
Contains types and params of all the current validators

```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

true if the model is still waiting for a result (used with asynchronous operations)

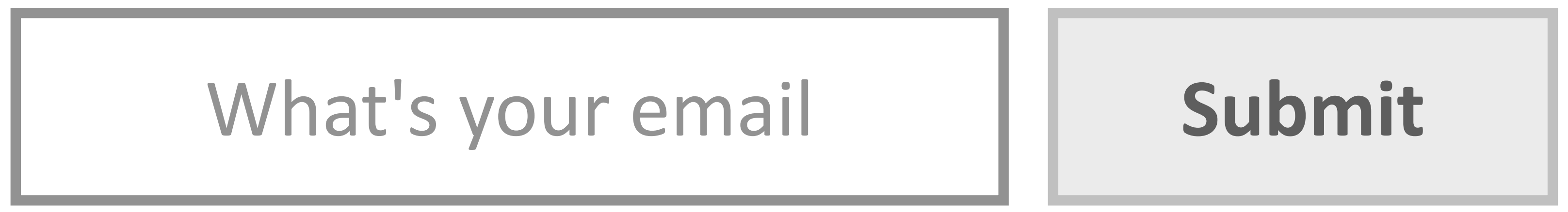
```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

true if the value passes the email validator test (if there is no value to test, the will report true by default)

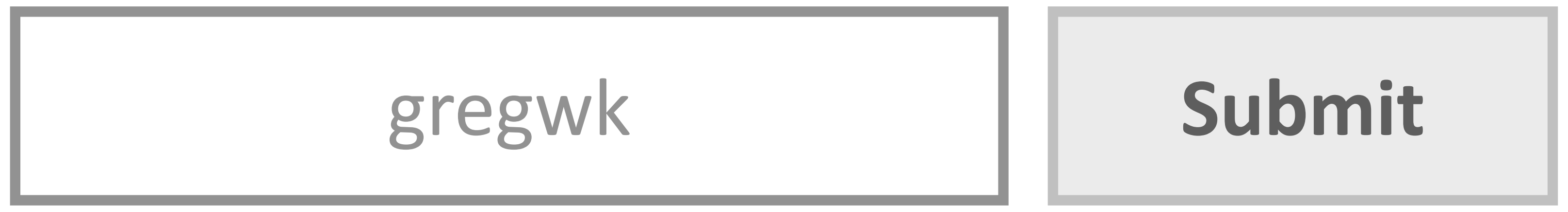


```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

true if the value is not null or empty



```
email: Object
  $anyDirty: false
  $anyError: false
  $dirty: false
  $error: false
  $invalid: true
  $model: null
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: false
```

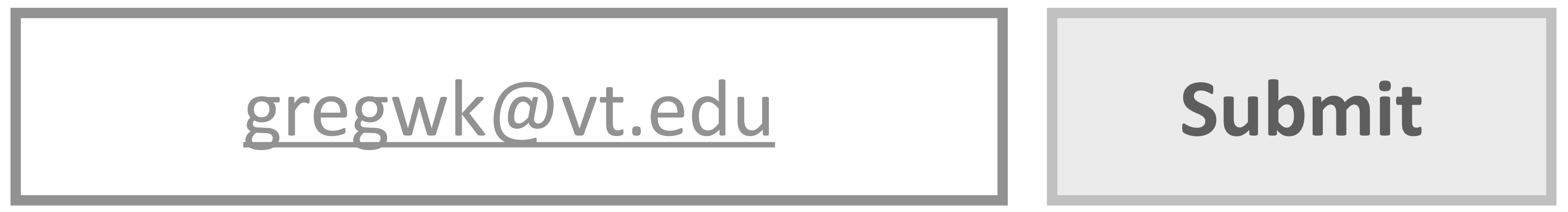

gregwk

Submit

```
email: Object
  $anyDirty: true
  $anyError: true
  $dirty: true
  $error: true
  $invalid: true
  $model: "gregwk"
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: false
  required: true
```

Consider how the values change when we enter a partial (but invalid) email in the field.

- The email data has been touched (is dirty)
- The email value is non-empty, so it passes the required validator test
- The email value fails the email validator test, therefore it is invalid, and since it has been touched (is dirty) that means the email data has an error


gregwk@vt.edu

```
email: Object
  $anyDirty: true
  $anyError: false
  $dirty: true
  $error: false
  $invalid: false
  $model: "gregwk@vt.edu"
  $params: Object
    email: Object
      type: "email"
    required: Object
      type: "required"
  $pending: false
  email: true
  required: true
```

When a valid email is entered, no validators are false, and there are no errors.