

JOI Schemas



Full Stack Web Development

More Joi Schemas

joi-schemas.js

```
export const UserSpec = {
  firstName: Joi.string().required(),
  lastName: Joi.string().required(),
  email: Joi.string().email().required(),
  password: Joi.string().required(),
};
```

▼ models

- > json
- > mem
- > mongo
- JS db.js
- JS joi-schemas.js

Sign up

Name

Email

Password

There was a problem...

- "firstName" is not allowed to be empty
- "lastName" is not allowed to be empty
- "email" is not allowed to be empty
- "password" is not allowed to be empty

```
export const UserCredentialsSpec = {
  email: Joi.string().email().required(),
  password: Joi.string().required(),
};
```

Log in

Email

Password

There was a problem...

- "email" is not allowed to be empty
- "password" is not allowed to be empty

accounts-controller.js

```
...
import { UserSpec, UserCredentialsSpec } from "../models/joi-schemas.js";
...

login: {
  auth: false,
  validate: {
    payload: UserCredentialsSpec,
    options: { abortEarly: false },
    failAction: function (request, h, error) {
      return h.view("login-view", { title: "Log in error", errors: error.details }).takeover().code(400);
    },
  },
},
handler: async function (request, h) {
  const { email, password } = request.payload;
  const user = await db.userStore.getUserByEmail(email);
  if (!user || user.password !== password) {
    return h.redirect("/");
  }
  request.cookieAuth.set({ id: user._id });
  return h.redirect("/dashboard");
},
},
```

```
export const PlaylistSpec = {
  title: Joi.string().required(),
};
```

dashboard-controller.js

The addPlaylist action:

```
...
import { PlaylistSpec } from "../models/joi-schemas.js";
...

addPlaylist: {
  validate: {
    payload: PlaylistSpec,
    options: { abortEarly: false },
    failAction: function (request, h, error) {
      return h.view("dashboard-view", { title: "Add Playlist error", errors: error.details }).takeover().code(400);
    },
  },
  handler: async function (request, h) {
    const loggedInUser = request.auth.credentials;
    const newPlayList = {
      userid: loggedInUser._id,
      title: request.payload.title,
    };
    await db.playlistStore.addPlaylist(newPlayList);
    return h.redirect("/dashboard");
  },
},
```

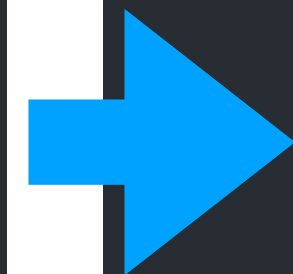
Playlist Title

Add Playlist

There was a problem...

- "title" is not allowed to be empty

```
export const TrackSpec = {
  title: Joi.string().required(),
  artist: Joi.string().required(),
  duration: Joi.number().allow("").optional(),
};
```



playlist-controller.js

And the addTrack action:

```
...
import { TrackSpec } from "../models/joi-schemas.js";
...

addTrack: {
  validate: {
    payload: TrackSpec,
    options: { abortEarly: false },
    failAction: function (request, h, error) {
      return h.view("playlist-view", { title: "Add track error", errors: error.details }).takeover().code(400);
    },
  },
  handler: async function (request, h) {
    const playlist = await db.playlistStore.getPlaylistById(request.params.id);
    const newTrack = {
      title: request.payload.title,
      artist: request.payload.artist,
      duration: Number(request.payload.duration),
    };
    await db.trackStore.addTrack(playlist._id, newTrack);
    return h.redirect(`/playlist/${playlist._id}`);
  },
},
```

Track	Artist	Duration
<div>Enter Track Details:</div> <div><input type="text" value="Enter Title"/></div> <div><input type="text" value="Enter Artist"/></div> <div><input type="text" value="Enter duration"/></div> <div><button>Add Track</button></div>		
<div>There was a problem...</div> <ul style="list-style-type: none">"title" is not allowed to be empty"artist" is not allowed to be empty		



[API](#) [Resources](#) [Policies](#) [Modules](#) [Sandbox](#)

The most powerful schema description language and data validator for JavaScript

Get started with joi

- Joi's biggest advantage is its usability. It's easy to use, easy to extend, and it has the full power of JavaScript.
- The downside is that if you want to reuse your validation logic on the frontend and the backend, your only choice of language on the backend is node.



JSON Schema

[Specification](#) [Learn](#) [Implementations](#) [Blog](#) [Join our Slack](#)

JSON Schema is a vocabulary that allows you to **annotate** and **validate** JSON documents.

Benefits

- Describes your existing data format(s).
- Provides clear human- and machine- readable documentation.
- Validates data which is useful for:
 - Automated testing.
 - Ensuring quality of client submitted data.

- The big win you get from JSON Schema is that it's cross platform. JSON Schema validator implementations exist in every major programming language. Write once, validate anywhere.
- The downside is that because it's cross platform, it's also somewhat limited in what it can do. It's purposely kept simple enough so that it's not too difficult to implement in any programming language.

<https://stackoverflow.com/questions/54228539/why-is-joi-more-popular-than-ajv>

JOI Schemas



Full Stack Web Development