

Swagger / Joi



Full Stack Web Development



The most powerful schema description language and data validator for JavaScript

Get started with joi

- Joi Documentation : <https://joi.dev/>

Sandbox

- Joi “Sandbox” to experiment with schema validation behaviour

joi Sandbox v17.3.0

Schema:

```
1 //Insert your joi schema here
2 Joi.object({
3   username: Joi.string().alphanum().min(3).max(30).required(),
4   password: Joi.string().pattern(/^([a-zA-Z0-9]{3,30}$)/),
5   repeat_password: Joi.ref("password"),
6   access_token: [Joi.string(), Joi.number()],
7   birth_year: Joi.number().integer().min(1900).max(2013),
8   email: Joi.string().email({ minDomainSegments: 2, tlds: { allow: ["com", "net"] } })
9 }).with('username', 'birth_year').xor('password', 'access_token').with('password', 'repeat_password')
```

Data To Validate:

```
1 //Insert data to validate here
2 {
3   username: "abc",
4   password: "password",
5   repeat_password: "password",
6   birth_year: 1994
7 }
```

Validate

Result:

Validation Passed

Validated Object:


```
{
  username: "abc",
  password: "password",
  repeat_password: "password",
  birth_year: 1994
}
```

Joi in Playtime

db/joi-schema.js

```
import Joi from "joi";

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

 Playtime

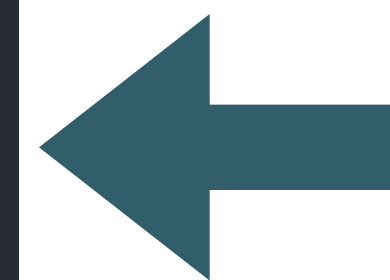
[Log in](#) [Sign up](#)

Sign up

Name

Email

Password



Define a schema to describe user signup information

Joi Schemas

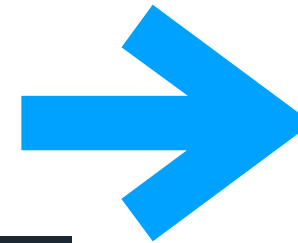
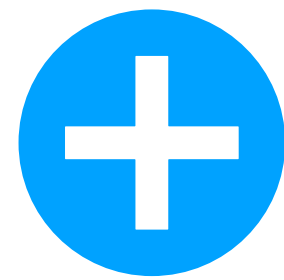
```
export const UserSpec = {  
  firstName: Joi.string().required(),  
  lastName: Joi.string().required(),  
  email: Joi.string().email().required(),  
  password: Joi.string().required(),  
};
```

```
export const UserSpec = Joi.object()  
  .keys({  
    firstName: Joi.string().example("Homer").required(),  
    lastName: Joi.string().example("Simpson").required(),  
    email: Joi.string().email().example("homer@simpson.com").required(),  
    password: Joi.string().example("secret").required(),  
  })  
  .label("UserDetails");  
  
export const UserArray = Joi.array().items(UserSpec).label("UserArray");
```

- Simple Joi specification for a User
- Enhanced Schema to include example data + Name (label)
- Specification defined as an array of another specification


```
export const UserSpec = Joi.object()
  .keys({
    firstName: Joi.string().example("Homer").required(),
    lastName: Joi.string().example("Simpson").required(),
    email: Joi.string().email().example("homer@simpson.com").required(),
    password: Joi.string().example("secret").required(),
  })
  .label("UserDetails");

export const UserArray = Joi.array().items(UserSpec).label("UserArray");
```



```
export const userApi = {
  find: {
    auth: false,
    handler: async function(request, h) {
      try {
        const users = await db.userStore.getAllUsers();
        return users;
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
    tags: ["api"],
    description: "Get all userApi",
    notes: "Returns details of all userApi",
    response: { schema: UserArray }
  },
};
```

- Swagger documentation generated

Models									
UserDetails ▾ { <table> <tr> <td>firstName*</td><td>string example: Homer</td></tr> <tr> <td>lastName*</td><td>string example: Simpson</td></tr> <tr> <td>email*</td><td>string example: homer@simpson.com</td></tr> <tr> <td>password*</td><td>string example: secret</td></tr> </table> }		firstName*	string example: Homer	lastName*	string example: Simpson	email*	string example: homer@simpson.com	password*	string example: secret
firstName*	string example: Homer								
lastName*	string example: Simpson								
email*	string example: homer@simpson.com								
password*	string example: secret								
UserArray ▾ [UserArray ▾ { <table> <tr> <td>firstName*</td><td>string example: Homer</td></tr> <tr> <td>lastName*</td><td>string example: Simpson</td></tr> <tr> <td>email*</td><td>string example: homer@simpson.com</td></tr> <tr> <td>password*</td><td>string example: secret</td></tr> </table> }]		firstName*	string example: Homer	lastName*	string example: Simpson	email*	string example: homer@simpson.com	password*	string example: secret
firstName*	string example: Homer								
lastName*	string example: Simpson								
email*	string example: homer@simpson.com								
password*	string example: secret								

Schema Validation

```
export const IdSpec = Joi.alternatives().try(Joi.string(), Joi.object()).description("a valid ID");

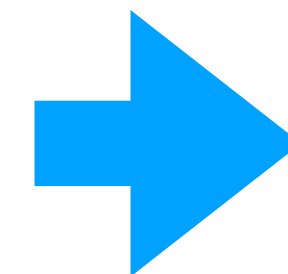
export const UserSpec = Joi.object()
  .keys({
    firstName: Joi.string().example("Homer").required(),
    lastName: Joi.string().example("Simpson").required(),
    email: Joi.string().email().example("homer@simpson.com").required(),
    password: Joi.string().example("secret").required(),
    _id: IdSpec,
    __v: Joi.number(),
  })
  .label("UserDetails");
```

- Response validation will fail, unless `_id` & `__v` attributes declared in schema

```
response: { schema: UserArray, failAction: validationError },
```

```
export function validationError(request, h, error) {
  console.log(error.message);
}
```

```
"[0]._id" is not allowed
```



Models

UserDetails ▾ {

<code>firstName*</code>	<code>string</code> <i>example: Homer</i>
<code>lastName*</code>	<code>string</code> <i>example: Simpson</i>
<code>email*</code>	<code>string</code> <i>example: homer@simpson.com</i>
<code>password*</code>	<code>string</code> <i>example: secret</i>
<code>_id</code>	<code>string</code>
<code>__v</code>	<code>number</code>

- Define schema for valid IDs

```
export const IdSpec = Joi.alternatives().try(Joi.string(), Joi.object()).description("a valid ID");

export const UserSpec = Joi.object()
  .keys({
    firstName: Joi.string().example("Homer").required(),
    lastName: Joi.string().example("Simpson").required(),
    email: Joi.string().email().example("homer@simpson.com").required(),
    password: Joi.string().example("secret").required(),
    _id: IdSpec,
    __v: Joi.number(),
  })
  .label("UserDetails");
```

- Reuse the IdSpec schema in UserSec

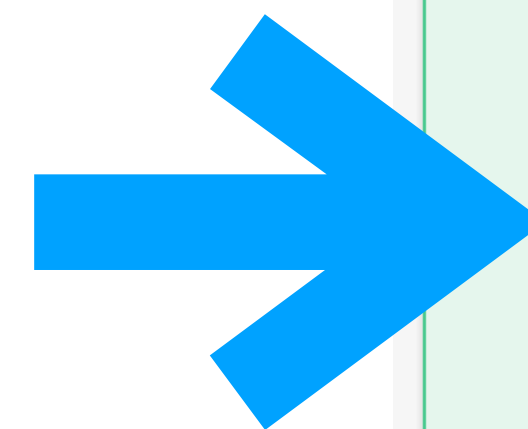
Schema
Reuse


```
export const IdSpec = Joi.alternatives().try(Joi.string(), Joi.object()).description("a valid ID");

export const UserSpec = Joi.object()
  .keys({
    firstName: Joi.string().example("Homer").required(),
    lastName: Joi.string().example("Simpson").required(),
    email: Joi.string().email().example("homer@simpson.com").required(),
    password: Joi.string().example("secret").required(),
    _id: IdSpec,
    __v: Joi.number(),
  })
  .label("UserDetails");
```

Schema Validation

- Single UserSpec schema defined for input & output
- Correct for output - but not appropriate for input



POST
/api/users
Create a User

Returns the newly created user

Parameters

Name	Description
body object (body)	<div> <div>Example Value</div> <div>Model</div> </div> <pre>{ "firstName": "Homer", "lastName": "Simpson", "email": "homer@simpson.com", "password": "secret", "_id": "string", "__v": 0 }</pre>

Parameter content type

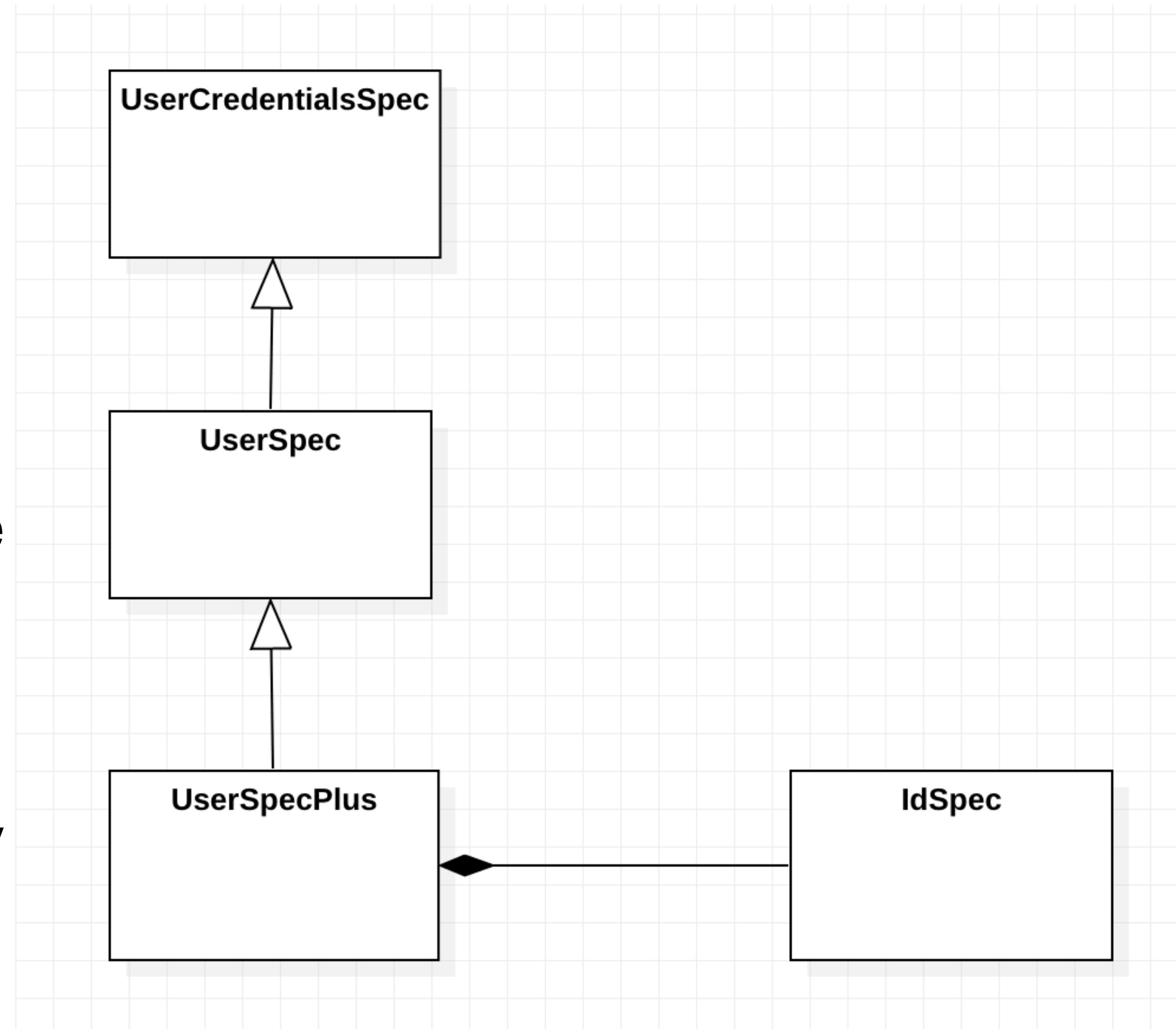
application/json

Schema Inheritance

username, password

+ firstname, last name

+ _id, __v



Schema Inheritance

```
export const IdSpec = Joi.alternatives().try(Joi.string(), Joi.object()).description("a valid ID");

export const UserCredentialsSpec = Joi.object()
  .keys({
    email: Joi.string().email().example("homer@simpson.com").required(),
    password: Joi.string().example("secret").required(),
  })
  .label("UserCredentials");

export const UserSpec = UserCredentialsSpec.keys({
  firstName: Joi.string().example("Homer").required(),
  lastName: Joi.string().example("Simpson").required(),
}).label("UserDetails");

export const UserSpecPlus = UserSpec.keys({
  _id: IdSpec,
  __v: Joi.number(),
}).label("UserDetailsPlus");

export const UserArray = Joi.array().items(UserSpecPlus).label("UserArray");
```

Schema Inheritance

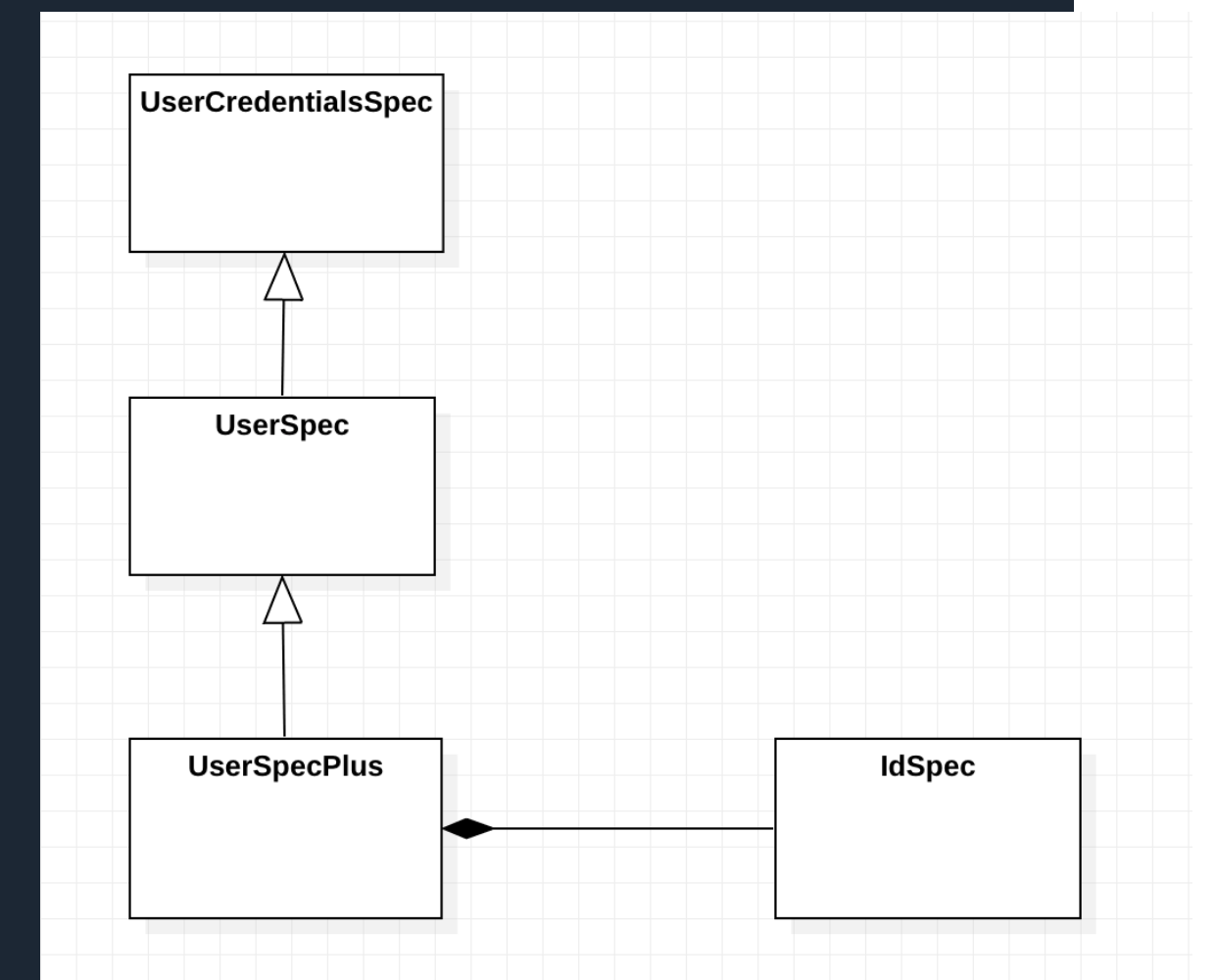
```
export const IdSpec = Joi.alternatives().try(Joi.string(), Joi.object()).description("a valid ID");

export const UserCredentialsSpec = Joi.object()
  .keys({
    email: Joi.string().email().example("homer@simpson.com").required(),
    password: Joi.string().example("secret").required(),
  })
  .label("UserCredentials");

export const UserSpec = UserCredentialsSpec.keys({
  firstName: Joi.string().example("Homer").required(),
  lastName: Joi.string().example("Simpson").required(),
}).label("UserDetails");

export const UserSpecPlus = UserSpec.keys({
  _id: IdSpec,
  __v: Joi.number(),
}).label("UserDetailsPlus");

export const UserArray = Joi.array().items(UserSpecPlus).label("UserArray");
```




```

create: {
  auth: false,
  handler: async function (request, h) {
    try {
      const user = await db.userStore.addUser(request.payload);
      if (user) {
        return h.response(user).code(201);
      }
      return Boom.badImplementation("error creating user");
    } catch (err) {
      return Boom.serverUnavailable("Database Error");
    }
  },
  tags: ["api"],
  description: "Create a User",
  notes: "Returns the newly created user",
  validate: { payload: UserSpec, failAction: validationError },
  response: { schema: UserSpecPlus, failAction: validationError },
},

```

- Validate: UserSpec
- Response: UserSpecPlus

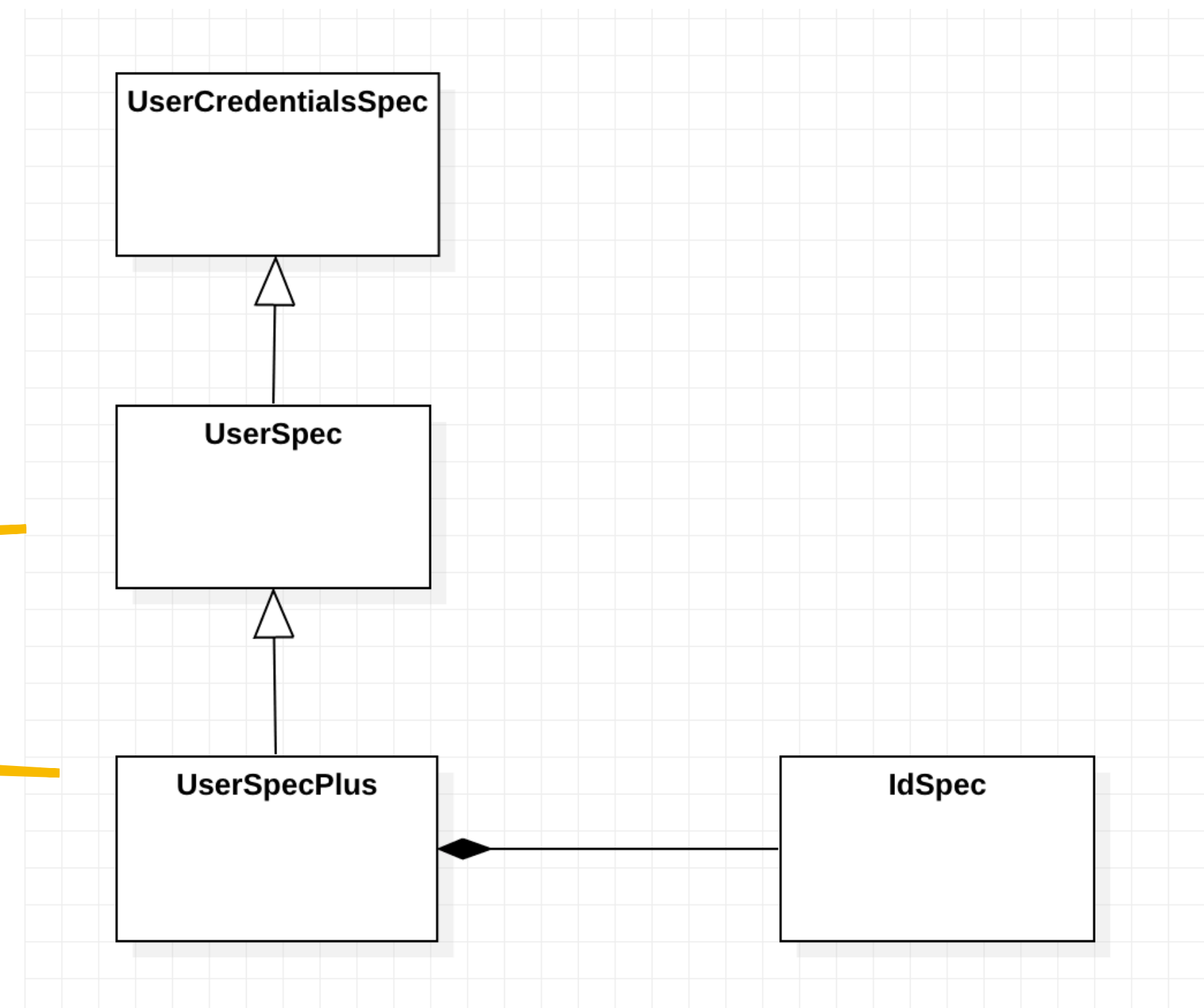
Refined Endpoints

```

create: {
  auth: false,
  handler: async function (request, h) {
    try {
      const user = await db.userStore.addUser(request.payload);
      if (user) {
        return h.response(user).code(201);
      }
      return Boom.badImplementation("error creating user");
    } catch (err) {
      return Boom.serverUnavailable("Database Error");
    }
  },
  tags: ["api"],
  description: "Create a User",
  notes: "Returns the newly created user",
  validate: { payload: UserSpec, failAction: validationError },
  response: { schema: UserSpecPlus, failAction: validationError },
},

```

- Validate: UserSpec
- Response: UserSpecPlus



Refined Endpoints

GET

/api/users

Get all userApi

▼

POST

/api/users

Create a User

▲

Returns the newly created user

Parameters

Try it out

Name	Description
body object (body)	<div>Example Value Model</div> <div><pre>{ "email": "homer@simpson.com", "password": "secret", "firstName": "Homer", "lastName": "Simpson"} </pre></div> <div>Parameter content type</div> <div>application/json ▼</div>

Responses

Response content type

application/json ▼

Code	Description
200	<div>Successful</div> <div>Example Value Model</div> <div><pre>{ "email": "homer@simpson.com", "password": "secret", "firstName": "Homer", "lastName": "Simpson", "_id": "string", "__v": 0}</pre></div>

Swagger / Joi



Full Stack Web Development