spacehuddle

SpaceHuddle developer documentation

SpaceHuddle Version

Version: 0.2.0
Datum: 29.04.2024

SpaceHuddle API

SpaceHuddle uses a REST API for exchanging data between the different parts of the application (backend, moderator, client). The API's code is located in the `api` directory.

Requirements

The SpaceHuddle API requires the following setup:

- a web server: tested with Apache 2.4,
- an SQL database instance: tested with MariaDB 10.4,
- PHP 8.1,
- Composer dependency manager.

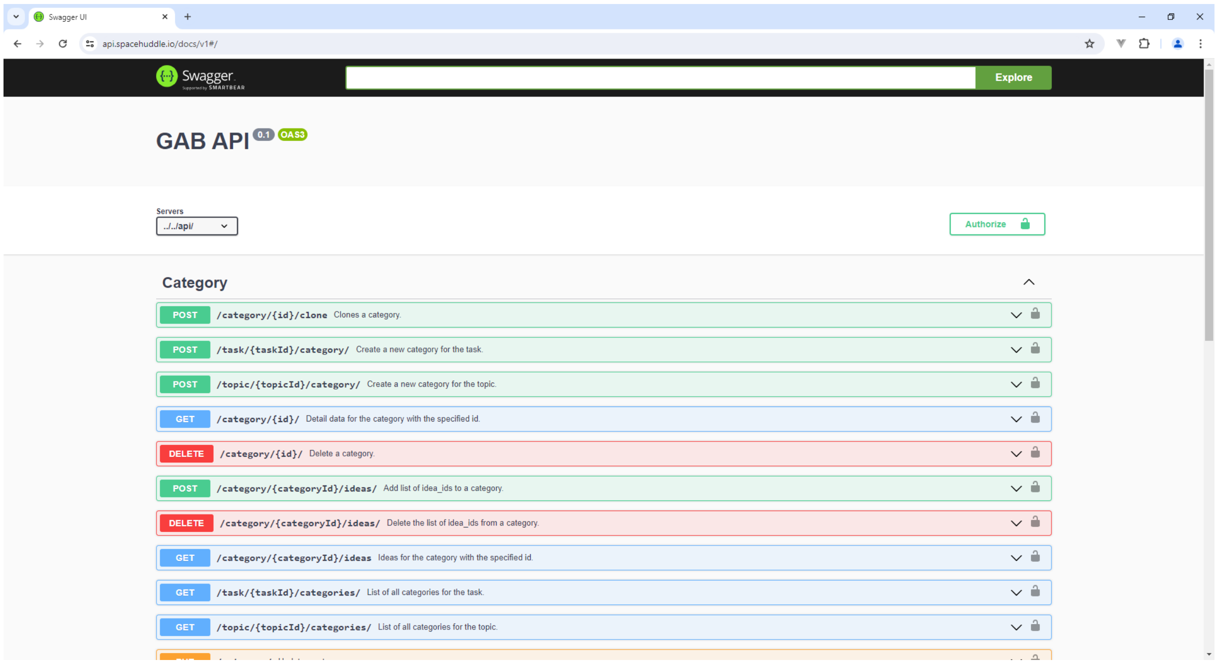
Installation and running of the API

1. Install a local web development environment that meets the requirements above. XAMPP works well, setups using Vagrant, Docker, or other virtual machines will work as well.
2. Check out the GitHub project on your webserver (for XAMPP for example in the directory `\xampp\htdocs\SpaceHuddle`).
3. If not included in your setup, install Composer following the instructions for your operating system.
4. Open a shell/terminal/command prompt, change to the `api` directory and install the dependencies by calling `composer install`.
5. Import `api/resources/schema.sql` into your database. This will create a database called "spacehuddle". Create a MySQL user with full permissions on the database. Enter your database credentials in `api/config/env.php` (see item 7).
6. Create a public and private key and copy them into the directory `api/resources/keys`
   - `openssl genrsa -out private.pem 2048`
   - `openssl rsa -in private.pem -outform PEM -pubout -out public.pem`
7. Copy `api/config/env.example.php` to `api/config/env.php` and adjust the properties
8. Start web server and database

API Documentation

API documentation and testing are done using Swagger. The documentation is located in `api/docs/v1`.

To run it, point your browser to http://{hostname}/{path}/api/docs/v1, e.g., http://localhost/api/docs/v1 or http://localhost/SpaceHuddle/api/docs/v1.



GAB API 0.1 OAS3

Servers
../../api/

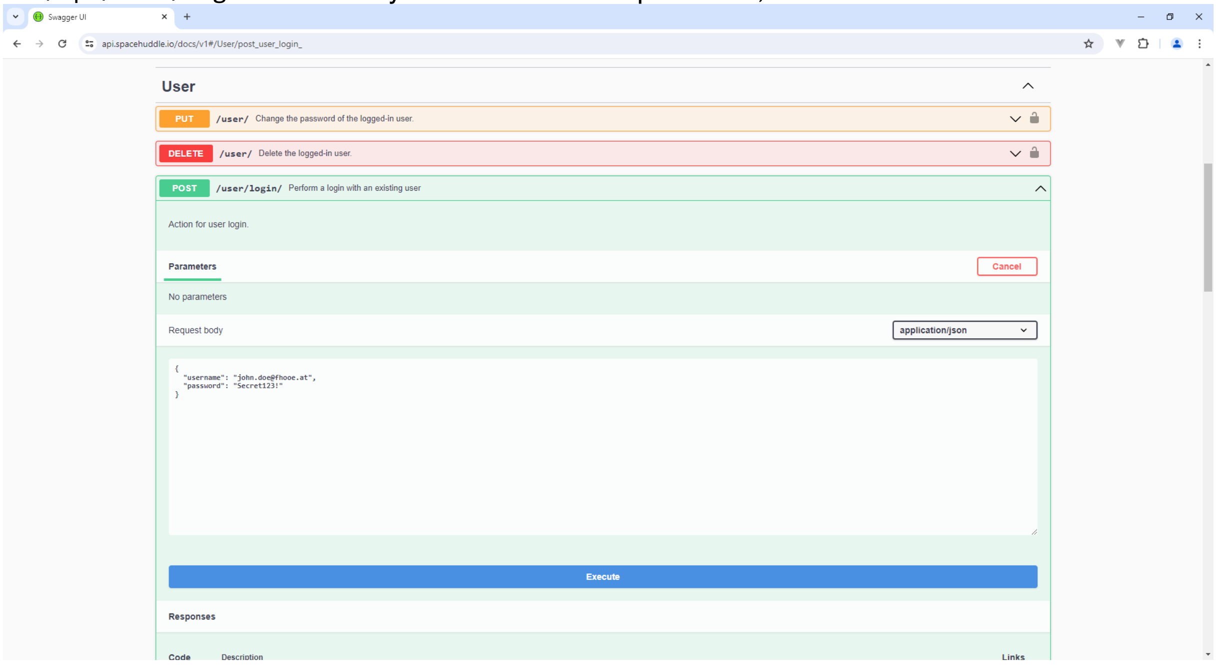
Category

POST	`/category/{id}/clone`	Clones a category.
POST	`/task/{taskId}/category/`	Create a new category for the task.
POST	`/topic/{topicId}/category/`	Create a new category for the topic.
GET	`/category/{id}/`	Detail data for the category with the specified id.
DELETE	`/category/{id}/`	Delete a category.
POST	`/category/{categoryId}/ideas/`	Add list of idea_ids to a category.
DELETE	`/category/{categoryId}/ideas/`	Delete the list of idea_ids from a category.
GET	`/category/{categoryId}/ideas`	Ideas for the category with the specified id.
GET	`/task/{taskId}/categories/`	List of all categories for the task.
GET	`/topic/{topicId}/categories/`	List of all categories for the topic.

To test the various API endpoints, select one from the list, adapt the proposed request body if necessary and press "Execute". You will see the server response below.

To test for the *moderator* or *facilitator*, use the following steps:

1. call `/api/user/register/` and enter any details for `username`, `password` and `password_confirmation`,
2. call`/api/user/login` and enter your username and password,



User

PUT	`/user/`	Change the password of the logged-in user.
DELETE	`/user/`	Delete the logged-in user.
POST	`/user/login/`	Perform a login with an existing user

Action for user login.

Parameters

No parameters

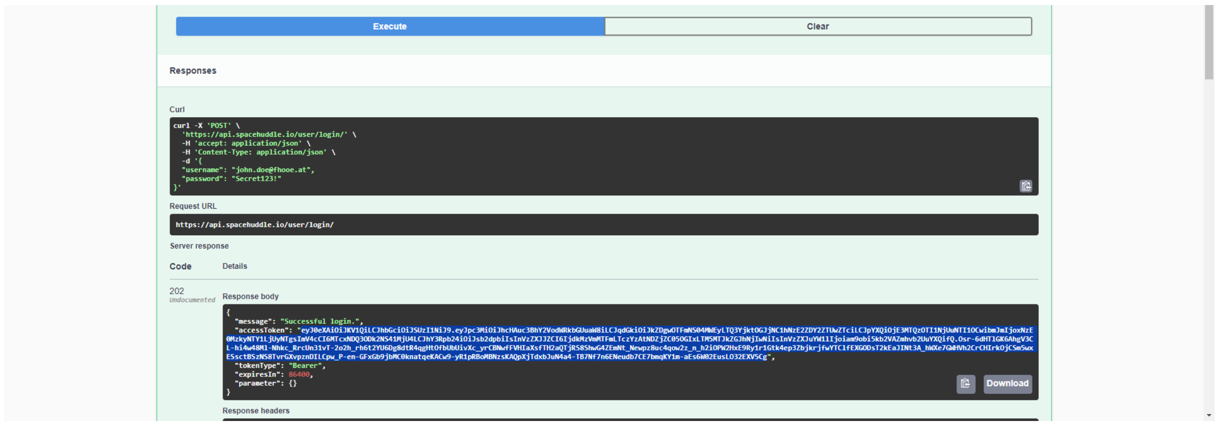
Request body: application/json

```
{
  "username": "john.doe@fhooe.at",
  "password": "Secret123!"
}
```

Responses

Code Description Links

3. copy the value for `access_token`, click the button "Authorize" in the upper right corner and enter the token in the field for `bearerAuth`,



Curl

```
curl -X 'POST' \
  'https://api.spacehuddle.io/user/login/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "john.doe@fhooe.at",
  "password": "Secret123!"
}'
```

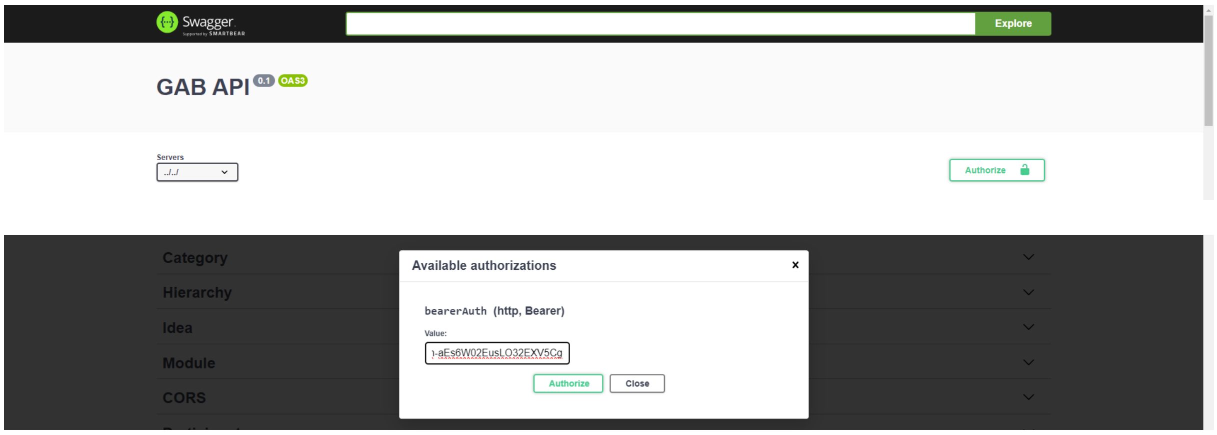
Request URL

```
https://api.spacehuddle.io/user/login/
```

Server response

Code	Details
202 *Undocumented*	Response body

```
{
  "message": "Successful login.",
  "accessToken": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOiJhcHAuc3BhY2VodWRkbGUuaW8iLCJqdGkiOiJkZDgwOTFmNS04MWEyLTQ3YjktOGJjNC1hNzE2ZDY2ZTUwZTciLCJpYXQiOjE3MTQzOTI1NjUuNTI1OCwibmJmIjoxNzE
0MzkyNTY1LjUyNTgsImV4cCI6MTcxNDQ3ODk2NS41MjU4LCJhY3Rpb24iOiJsb2dpbiIsInVzZXJJZCI6IjdkMzVmMTFmLTczYzAtNDZjZC05OGIxLTM5MTJkZGJhNjIwNiIsInVzZXJuYW1lIjoiam9obi5kb2VAZmhvb2UuYXQifQ.Osr-6dHTlGK6AhgV3C
L-hi4w48Ml-Nhkc_RrcUn31vT-2o2h_rh6t2YU6Dg8dtR4qgHtOfbUbUivXc_yrCBNwfFVHIaXsfTH2aQTjRS8ShwG4ZEmNt_Newpz8uc4qow2z_n_h2iOPW2HxE9Ry1r1Gtk4ep3ZbjkrjfwYTClfEXGODsT2kEaJINt3A_hWXe7GWHVh2CrCHIrkOjCSm5wx
ESsctBSzNS8TvrGXvpznDILCpw_P-en-GFxGb9jbMC0knatqeKACw9-yR1pRBoMBNzsKAQpXjTdxbJuN4a4-TB7Nf7n6ENeudb7CE7bmqKY1m-aEs6W02EusLO32EXV5Cg",
  "tokenType": "Bearer",
  "expiresIn": 86400,
  "parameter": {}
}
```



Available authorizations

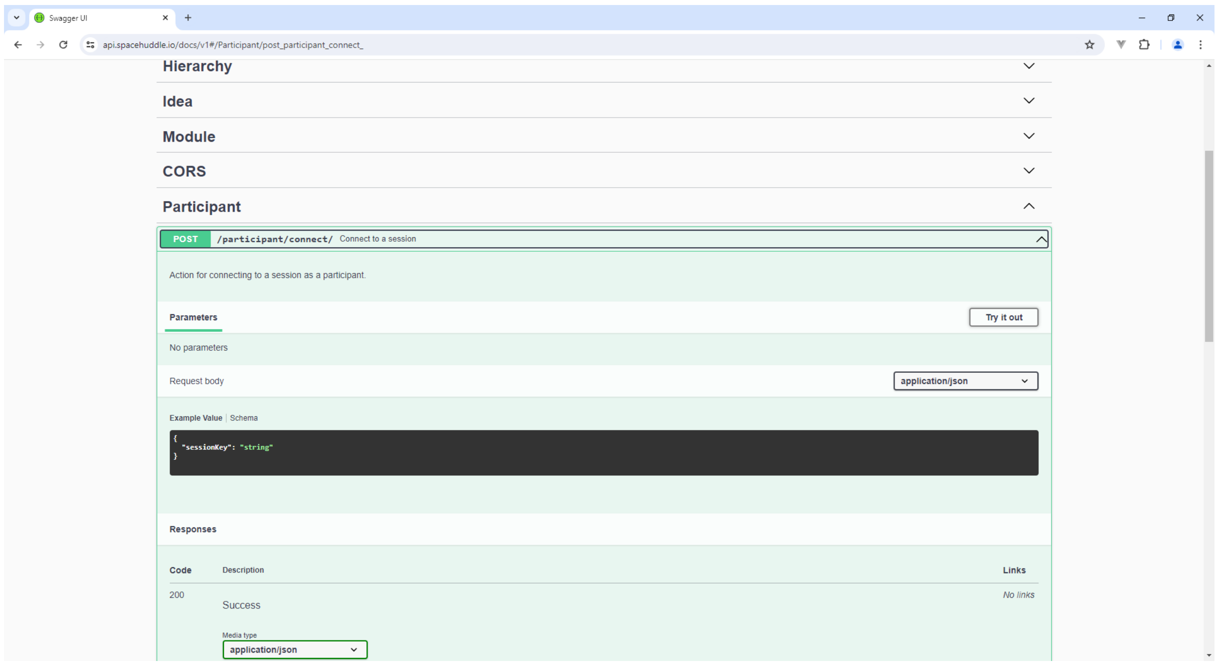
bearerAuth (http, Bearer)

Value: ı-aEs6W02EusLO32EXV5Cg

4. execute any arbitrary REST call for the moderator tool.

To test for a *participant*, use the following steps:

1. call `api/session` to create a new session with given details and copy the received value for `connection_key`,
2. call `/api/participant/connect/` and enter the key in `session_key`,


Swagger UI
api.spacehuddle.io/docs/v1#/Participant/post_participant_connect_
Hierarchy
Idea
Module
CORS
Participant
POST

`/participant/connect/` Connect to a session

Action for connecting to a session as a participant.

No parameters

Request body: application/json

```
{
  "sessionKey": "string"
}
```

Responses

Code	Description	Links
200	Success Media type: application/json	*No links*

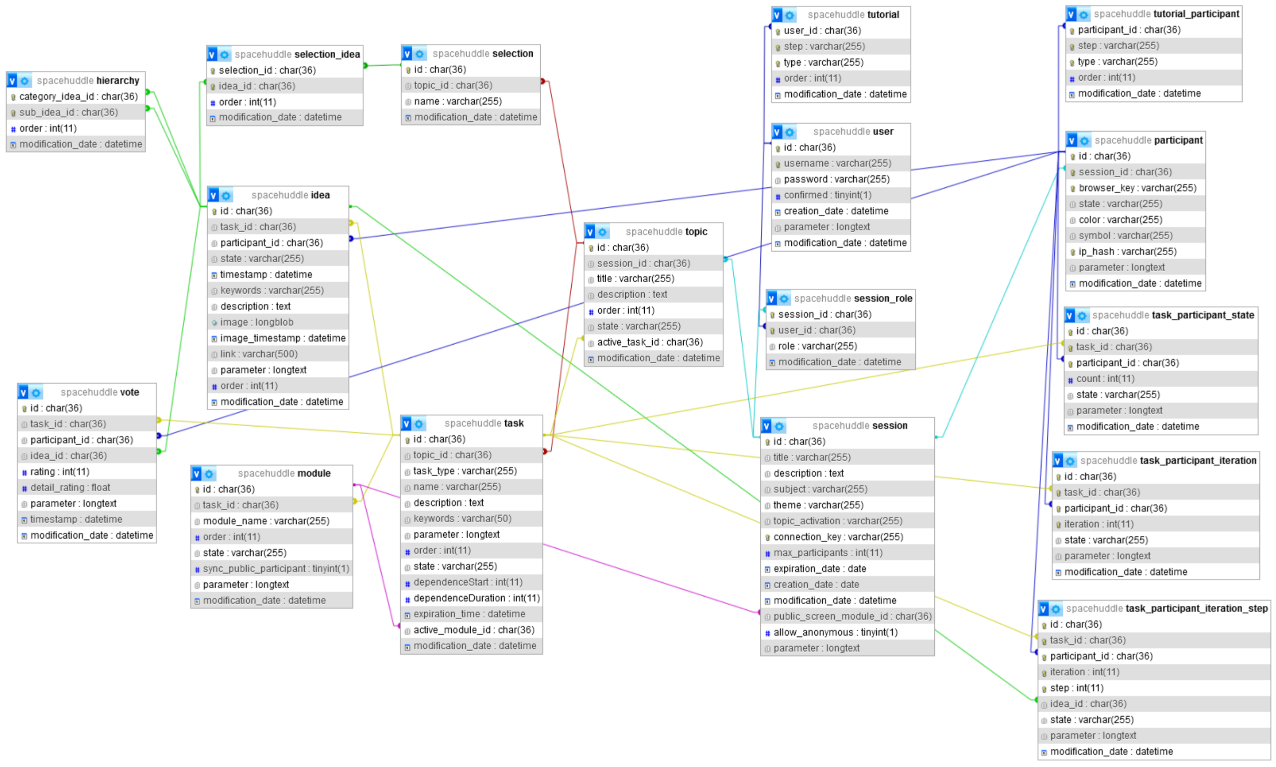
3. copy the value for `access_token`, click the button "Authorize" in the upper right corner and enter the token in the field for `bearerAuth`,
4. execute any arbitrary REST call for the client tool.

Technology stack

The SpaceHuddle-API is built on the following technologies. Visit the websites to learn more about their use.

- PHP 8.1: programming language `https://www.php.net/`
- Slim: micro framework for PHP `https://www.slimframework.com/`
- CakePHP: database query framework `https://cakephp.org/`
- Swagger-PHP: REST API documentation https://zircote.github.io/swagger-php/

Database model



User administration

- user: User data for registered moderators and co-moderators.
- participant: Participants taking part in the session.
- session_role: Role in a session as moderators or co-moderators.

Tutorial

- tutorial: Which explanatory text has already been read by the user.
- tutorial_participant: Which explanatory text has already been read by the participant.

Tracking

- task_participant_state: How often was the task performed by the participant? Was the task completed?
- task_participant_iteration: How did the individual run go? Was the round won?
- task_participant_iteration_step: Which steps were completed within the round?

Session

- session: A session defines a bundle of topics on tasks for an event which are to be worked on by a joint group of participants.

Topic

- topic: A session can consist of several topics. Topics organize a session to ensure a better overview.

Task

- task: A topic can consist of several tasks.
- selection: A selection is a special task that makes a choice from existing inputs (ideas).

Module

- module: A task must consist of a main module and any number of add-ons. The main module defines which task is to be performed (brainstorming, categorization, evaluation, selection). The add ons define whether a special visualization or game components are desired.

Idea

- idea: Tasks generate ideas (brainstorming) or use ideas as input for further processing in a (selection, categorization or evaluation).
- selection_idea: Defines a list of existing ideas for a selection.
- hierarchy: Categorizes the existing ideas.
- Vote: Evaluates the existing ideas.

SpaceHuddle frontend

Installation

1. Adjust the properties in the `frontend/.env` file or create your own .env.local or .env.production version of the file. Create your individual MapTiler-key on https://www.maptiler.com/ and set it in .env file.
2. Download and install Node `https://nodejs.org/en/download/`. Use a Node version that is lower or equal to 19.2.0
3. Install dependencies with: `npm install`

Compiles and hot-reloads for development

You can use either `npm start` or `npm run serve`.

Compiles and minifies for production

```
npm run build
```

Run your unit tests

```
npm run test:unit
```

Lints and fixes files

```
npm run lint
```

Technology stack

SpaceHuddle is built on the following technologies. Visit the websites to learn more about their use.

- Typescript: programming language `https://www.typescriptlang.org/`
- Sass: css styling language `https://sass-lang.com/`
- ESLint: code linter `https://eslint.org/`
- VUE3: JavaScript Framework `https://vuejs.org/guide/introduction.html`
- Vue Class Component: class-style syntax `https://class-component.vuejs.org/`
- Element Plus: component library `https://element-plus.org/en-US/component/button.html`
- Bulma: responsive web interfaces `https://bulma.io/`
- Font Awesome: icon library `https://fontawesome.com/`
- i18n: translation module `https://www.npmjs.com/package/i18n`
- Axios: backend access `https://axios-http.com/`
- Chart.js: chart components `https://www.chartjs.org/`
- Matter-js: physics engine `https://brm.io/matter-js/`
- MapLibre: map engine `https://maplibre.org/`
- MapTiler: map styles `https://www.maptiler.com/`
- OSRM: open source routing machine `http://project-osrm.org/`
- PixiJS: 2d webgl renderer `https://pixijs.com/`
- Turf: geospatial analysis `http://turfjs.org/`

Develop your own modules

1. Navigate to the folder `frontend/src/modules` and choose one of the following subdirectory depending on the module type to be developed.
    i. information: information phase preceding the brainstorming (e.g. inspirational material, explaining the initial situation, evaluating the initial state (quiz, survey)).
    ii. brainstorming: idea collection
    iii. categorisation: structuring ideas
    iv. selection: restrict ideas for further use
    v. voting: evaluate ideas
    vi. playing: ice breaker games
    vii. common: General module overlapping visualization components for the public screen.
2. Create your own module subdirectory in the desired type folder.
3. Configure module
    i. Create `config.json` file within your module folder
    ii. Set the properties required for your module in the json file
        - `icon`: name of the fontawesome icon to be assigned to the module (`https://fontawesome.com/`)
        - `iconPrefix`: optional if the icon category is not fas

    - `type`: choose one of the two options `main` or `addOn`. `main` modules stand alone. `addOn`s extend any `main` module of the same type.
    - `input`: Input indicates whether the module uses other modules as an input source. Choose one of the three option `yes`, `no`, `optional`.
    - `syncPublicParticipant`: Indicates whether the flow of the client module can be controlled by the moderator. Choose one of the two option `true`, `false`.
    - `fallback`: optional if a module extends another module
4. Set up multilingualism for module
    i. Create a `locals` folder within your module folder.
    ii. Add a `[language abbreviation].json` to the `locales` folder for all available languages.
    iii. Structure of the language files

```
{
   "description": {
      "title": "...",
      "description": "..."
   },
   "publicScreen": {
      "...": "...",
      "...": "..."
   },
   "participant": {
      "...": "...",
      "...": "..."
   },
   "moderatorContent": {
      "...": "...",
      "...": "..."
   },
   "moderatorConfig": {
      "...": "...",
      "...": "..."
   },
   "statistic": {
      "...": "...",
      "...": "..."
   }
}
```

    iv. The sections `publicScreen`, `participant`, `moderatorContent`, `moderatorConfig`, `statistic` are optional and should only help to structure the code.
    v. Replace the `"..."` information with your own content.
    vi. The translation text can be embedded in the vue code as follows.

```
$t('module.!moduletype!.!modulename!.!outputType!.!translationKey!')
```

    - `!moduletype!`: Specifies the name of the module type folder (`selection`, `categorisation`, `brainstorming`, `information`, `voting`, `playing`, `common`)
    - `!modulename!`: Specifies the name of the module folder
    - `!outputType!`: Specifies the view name (`publicScreen`, `participant`, `moderatorContent`, `moderatorConfig`, `statistic`)
    - `!translationKey!`: Specifies the translation key
5. Develop your module.
    i. Create a `output` folder within your module folder.
    ii. Create a `ModeratorContent.vue` file in the `output` folder if you need a moderator view in your module that differs from the default view. In the following example, replace the information between ! and !, and expand the functionality according to individual needs.

```
<template>
!html section!
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop } from 'vue-property-decorator';
import { IModeratorContent } from '@/types/ui/IModeratorContent';

@Options({
  components: {},
})
export default class ModeratorContent extends Vue implements
IModeratorContent {
  @Prop() readonly taskId!: string;
}
</script>

<style lang="scss" scoped>
!scss section!
</style>
```

iii. Create a `ModeratorConfig.vue` file in the `output` folder if you need individual adjustable configuration parameters for the moderator in your module. In the following example, replace the information between ! and !, and expand the functionality according to individual needs.

```
<template>
   <el-form-item

:label="$t('module.!moduletype!.!modulename!.moderatorConfig.!parametername!'
)"
   :prop="`${rulePropPath}.!parametername!`"
   :rules="[defaultFormRules.ruleRequired]"
   >
    <el-input-number
      v-model="modelValue.!parametername!"

:placeholder="$t('module.!moduletype!.!modulename!.moderatorConfig.!parameter
nameExample!')"
    />
  </el-form-item>
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop, Watch } from 'vue-property-decorator';
import * as moduleService from '@/services/module-service';
import { Module } from '@/types/api/Module';
import { ValidationRuleDefinition, defaultFormRules } from
'@/utils/formRules';
import * as cashService from '@/services/cash-service';

@Options({
  components: {},
  emits: ['update'],
})

/* eslint-disable @typescript-eslint/no-explicit-any*/
export default class ModeratorConfig extends Vue {
  defaultFormRules: ValidationRuleDefinition = defaultFormRules;
  @Prop() readonly rulePropPath!: string;

  @Prop() readonly moduleId!: string;
  @Prop() readonly taskId!: string;
  @Prop() readonly topicId!: string;
  @Prop({ default: {} }) modelValue!: any;
  @Prop({ default: {} }) formData!: any;
  @Prop({ default: {} }) taskType!: any;

  module: Module | null = null;

  @Watch('modelValue', { immediate: true })
  async onModelValueChanged(): Promise<void> {
    if (this.modelValue && !this.modelValue.!parametername!) {
      this.modelValue.!parametername! = !parameternameDefaultValue!;
    }
  }

  @Watch('moduleId', { immediate: true })
  async onModuleIdChanged(): Promise<void> {
    if (this.moduleId) {
      moduleService.registerGetModuleById(
        this.moduleId,
        this.updateModule,
        EndpointAuthorisationType.MODERATOR,
        60 * 60
      );
    }
  }

  updateModule(module: Module): void {
    this.module = module;
  }

  deregisterAll(): void {
    cashService.deregisterAllGet(this.updateModule);
  }

  unmounted(): void {
    this.deregisterAll();
  }
}
</script>
```

iv. Create a `participant.vue` file in the `output` folder if you need a participant view for your module that differs from the default view. In the following example, replace the information between ! and !, and expand the functionality according to individual needs.

```
<template>
  <ParticipantModuleDefaultContainer :task-id="taskId" :module="moduleName">
    !html section!
  </ParticipantModuleDefaultContainer>
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop, Watch } from 'vue-property-decorator';
import ParticipantModuleDefaultContainer from
'@/components/participant/organisms/layout/ParticipantModuleDefaultContainer.
vue';
import * as moduleService from '@/services/module-service';
import { Module } from '@/types/api/Module';
import EndpointAuthorisationType from
'@/types/enum/EndpointAuthorisationType';
import * as cashService from '@/services/cash-service';

@Options({
  components: {
    ParticipantModuleDefaultContainer,
  },
})
export default class Participant extends Vue {
  @Prop() readonly taskId!: string;
  @Prop() readonly moduleId!: string;
  @Prop({ default: false }) readonly useFullSize!: boolean;
  @Prop({ default: '' }) readonly backgroundClass!: string;
  module: Module | null = null;

  get moduleName(): string {
    if (this.module) return this.module.name;
    return '';
  }

  @Watch('moduleId', { immediate: true })
  onModuleIdChanged(): void {
    if (this.moduleId) {
      moduleService.registerGetModuleById(
        this.moduleId,
        this.updateModule,
        EndpointAuthorisationType.PARTICIPANT,
        60 * 60
      );
    }
  }

  updateModule(module: Module): void {
    this.module = module;
  }

  deregisterAll(): void {
    cashService.deregisterAllGet(this.updateModule);
  }

  unmounted(): void {
    this.deregisterAll();
  }
}
</script>

<style lang="scss" scoped>
!scss section!
</style>
```

v. Create a `PublicScreen.vue` file in the `output` folder if you need an individual public screen for your module that differs from the default view. In the following example, replace the information between ! and !, and expand the functionality according to individual needs.

```
<template>
!html section!
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop } from 'vue-property-decorator';
import EndpointAuthorisationType from
'@/types/enum/EndpointAuthorisationType';

@Options({
  components: {},
})
export default class PublicScreen extends Vue {
  @Prop() readonly taskId!: string;
  @Prop({ default: EndpointAuthorisationType.MODERATOR })
  authHeaderTyp!: EndpointAuthorisationType;
}
</script>

<style lang="scss" scoped>
!scss section!
</style>
```

vi. Create a `ModuleStatistic.vue` file in the `output` folder if you need an individual statistic for your module that differs from the default view. In the following example, replace the information between ! and !, and expand the functionality according to individual needs.

```
<template>
  <div></div>
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop } from 'vue-property-decorator';

@Options({
  components: {},
})

/* eslint-disable @typescript-eslint/no-explicit-any*/
export default class ModuleStatistic extends Vue {
  @Prop() readonly taskId!: string;
}
</script>

<style lang="scss" scoped>
!scss section!
</style>
```

vii. Develop additional components and types required for the module within the module folder. Spacehuddle uses atomic design to structure the files. Therefore, depending on what is required, structure the subdirectories into `types`, `organisms`, `molecules` and `atoms`. All these folders are optional.

viii. The implementation of the access to the backend interfaces can be found in the folder `frontend/src/services`. To illustrate their use, here is an example implementation for querying all ideas of a task.

```
<template>
!html section!
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop } from 'vue-property-decorator';
import { IModeratorContent } from '@/types/ui/IModeratorContent';
import * as ideaService from '@/services/idea-service';
import { Idea } from '@/types/api/Idea.ts';
import IdeaSortOrder from '@/types/enum/IdeaSortOrder';
import EndpointAuthorisationType from
'@/types/enum/EndpointAuthorisationType';
import * as cashService from '@/services/cash-service';

@Options({
  components: {},
})
export default class ModeratorContent extends Vue implements
IModeratorContent {
  @Prop() readonly taskId!: string;
  ideas: Idea[] = [];
  activeIdea!: Idea;

  @Watch('taskId', { immediate: true })
  onTaskIdChanged(): void {
    ideaService.registerGetIdeasForTask(
      this.taskId,
      IdeaSortOrder.TIMESTAMP,
      null,
      this.updateIdeas,
      EndpointAuthorisationType.MODERATOR,
      2 * 60
    );
  }

  updateIdeas(ideas: Idea[]): void {
    this.ideas = ideas;
  }

  deregisterAll(): void {
    cashService.deregisterAllGet(this.updateIdeas);
  }

  unmounted(): void {
    this.deregisterAll();
  }

  async save(): Promise<void> {
    if (this.activeIdea.id) {
       await ideaService
          .putIdea(this.activeIdea, EndpointAuthorisationType.MODERATOR)
          .then((queryResult) => {
             //todo
          });
    } else if (this.taskId) {
       await ideaService
       .postIdea(this.taskId, this.activeIdea,
EndpointAuthorisationType.MODERATOR)
       .then((queryResult) => {
          if (queryResult) {
           this.activeIdea = {};
           this.ideas.push(queryResult);
         }
       });
    }
  }
}
</script>

<style lang="scss" scoped>
!scss section!
</style>
```

- `GET` is used to read data. Get calls are implemented by registering them at the client-side cash to prevent multiple loading of data by different components. An update interval can be specified by which the data is reloaded from the backend. If no regular update is to take place, the value is to be set to 24 * 60 * 60 seconds = 1 day. It is important to disable the registration when leaving the page, otherwise the update will continue.
- `POST` is used for the initial insertion of data.
- `PUT` for changing already inserted data.
- `DELETE` for deleting data.

ix. Under `frontend/src/components` custom VUE components such as entity cards (e.g. `IdeaCard`), entity change dialogs (e.g. `IdeaSettings`) or layout components (e.g. `ParticipantModuleDefaultContainer`) can be found.

```
<template>
  <ParticipantModuleDefaultContainer :task-id="taskId" :module="moduleName">
    ...
   <div class="media" v-for="idea in ideas" :key="idea.id">
      <IdeaCard
          :idea="idea"
          :is-editable="false"
          class="public-idea"
          :show-state="false"
        />
   </div>
    ...
  </ParticipantModuleDefaultContainer>
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop, Watch } from 'vue-property-decorator';
import { Module } from '@/types/api/Module';
import EndpointAuthorisationType from
'@/types/enum/EndpointAuthorisationType';
import ParticipantModuleDefaultContainer from
'@/components/participant/organisms/layout/ParticipantModuleDefaultContainer.
vue';
import IdeaCard from '@/components/moderator/organisms/cards/IdeaCard.vue';

@Options({
  components: {
      ParticipantModuleDefaultContainer,
      IdeaCard
  },
})
export default class Participant extends Vue {
  @Prop() readonly taskId!: string;
  @Prop() readonly moduleId!: string;
  @Prop({ default: false }) readonly useFullSize!: boolean;
  @Prop({ default: '' }) readonly backgroundClass!: string;
  module: Module | null = null;
  ideas: Idea[] = [];

  get moduleName(): string {
    if (this.module) return this.module.name;
    return '';
  }

  ...
}
</script>

<style lang="scss" scoped>
.public-idea {
   max-width: 20rem;
}
...
</style>
```

If these components are to be used in the module in a way that differs from the predefined implementation, we kindly request that you implement an individual development of the components in the module folder. The components can be copied as a template for this purpose.

x. If data should be updated automatically from the backend, this can be solved by registering at client side cash. It is important to disable the registration when leaving the page, otherwise the update will continue.

```
<template>
!html section!
</template>

<script lang="ts">
import { Options, Vue } from 'vue-class-component';
import { Prop } from 'vue-property-decorator';
import EndpointAuthorisationType from
'@/types/enum/EndpointAuthorisationType';
import * as cashService from '@/services/cash-service';

@Options({
  components: {},
})
export default class PublicScreen extends Vue {
  @Prop() readonly taskId!: string;
  @Prop({ default: EndpointAuthorisationType.MODERATOR })
  authHeaderTyp!: EndpointAuthorisationType;

  @Watch('taskId', { immediate: true })
  onTaskIdChanged(): void {
    ideaService.registerGetIdeasForTask(
      this.taskId,
      IdeaSortOrder.TIMESTAMP,
      null,
      this.updateIdeas,
      EndpointAuthorisationType.MODERATOR,
      2 * 60
    );
  }

  updateIdeas(ideas: Idea[]): void {
    this.ideas = ideas;
  }

  deregisterAll(): void {
    cashService.deregisterAllGet(this.updateIdeas);
  }

  unmounted(): void {
    this.deregisterAll();
  }
}
</script>

<style lang="scss" scoped>
!scss section!
</style>
```