Ref. Ares(2023)4540465 - 30/06/2023

D2.5. EDUBOX DIY toolkit architecture

Co-funded by the European Union

This Project has received funding from the European Union's Creative Europe Media programme under grant agreement: CREA-CROSS-2021-INNOVLAB-Project 101059958

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or [name of the granting authority]. Neither the European Union nor the granting authority can be held responsible for them.

D2.5. EDUBOX DIY toolkit architecture

1. General introduction

EDUbox is a learning experience that combines storytelling, pedagogy, and technology. In every EDUbox we integrate interaction to stimulate learning-by-doing. During the last 5 years we have already developed several digital tools & apps. In this project it is our ambition to make some of these digital tools re-usable. These tools are also part of the EDUbox DIY toolkit.

To easily allow the consortium partners and newly attracted partners to create their own EDUboxes, a DIY toolkit will be developed. The DIY toolkit will consist of all templates, software packages and manuals needed to easily allow others to kickstart the creation of their own EDUbox.

In this deliverable we will only focus on the digital tools and how we will make them re-usable.

During the project we foresee the following activities:

Activity 1: developing a structure and common software architecture for the EDUbox DIY toolkit.

Activity 2: Next, the plan is to package and open source all software components of the EDUbox. Those software packages will be published on an appropriate open-source software distribution platform.

In this deliverable we thus focus on activity 1 where we give an overview of the common software architecture for the different digital tools. This will serve as the basis for the development of the tools and the back end.

First, we list all the tools we want to work on. We give a short description of the purpose of the tools.

Next, we provide more information about the common technical architecture. We highlight the commonalities in the setup.

In the following chapter, we go more in depth into the different tools and give information about the setup, game-flow, data-model, and the content management architecture.

Lastly, we explain how we want to distribute the code packages to other organisations.

2. Short description EDUbox tools

a. Co-op game

The co-op game has its origin in and educational context where collaborative problem solving (CPS) is researched and analysed. In this game, these CPS principles are applied and gamified. The creators can create different thematical contexts (eg a spacecraft going to Mars) where the players need to co-operate and colaborate to accomplish missions on their way to the end goal. This end goal is quantified by introducing a resource that needs to be managed and kept as high as possible (eg oxygen in the spacecraft).

b. Puzzle game

In the puzzle game the user is given 2 areas in front of him, in the first and largest area there are several cards arranged in a grid. In the second area there are no items in the initial phase, but by playing the game, the user will unlock items here. The progression of the tool consists of clicking on 2 or more items that are related to each other. When the user has indicated items that are a match, these tickets will disappear from the first area and a ticket will appear in the second area. After this the game can be continued. By clicking on the item in the second area, users can learn more about the match they have found. The game is done when all items from the first area disappear and all items in the second area are visible.

c. Lecture-app

The Lecture game has a similar technical architecture as the co-op game. There is one central screen (eg a PC) and multiple players who log in with their smartphone. The idea is that a theme is given, and players take turns in talking and listening to each other in order to comprehend the given message.

d. Interactive video

The interactive video contains a layer that looks at the time code of a video, when the video reaches a predetermined time code, a screen will be displayed in front of the video where the user will see one or more buttons. Clicking on these buttons will give the video a new start time. This allows you to guide the user through a story or let the user create their own story.

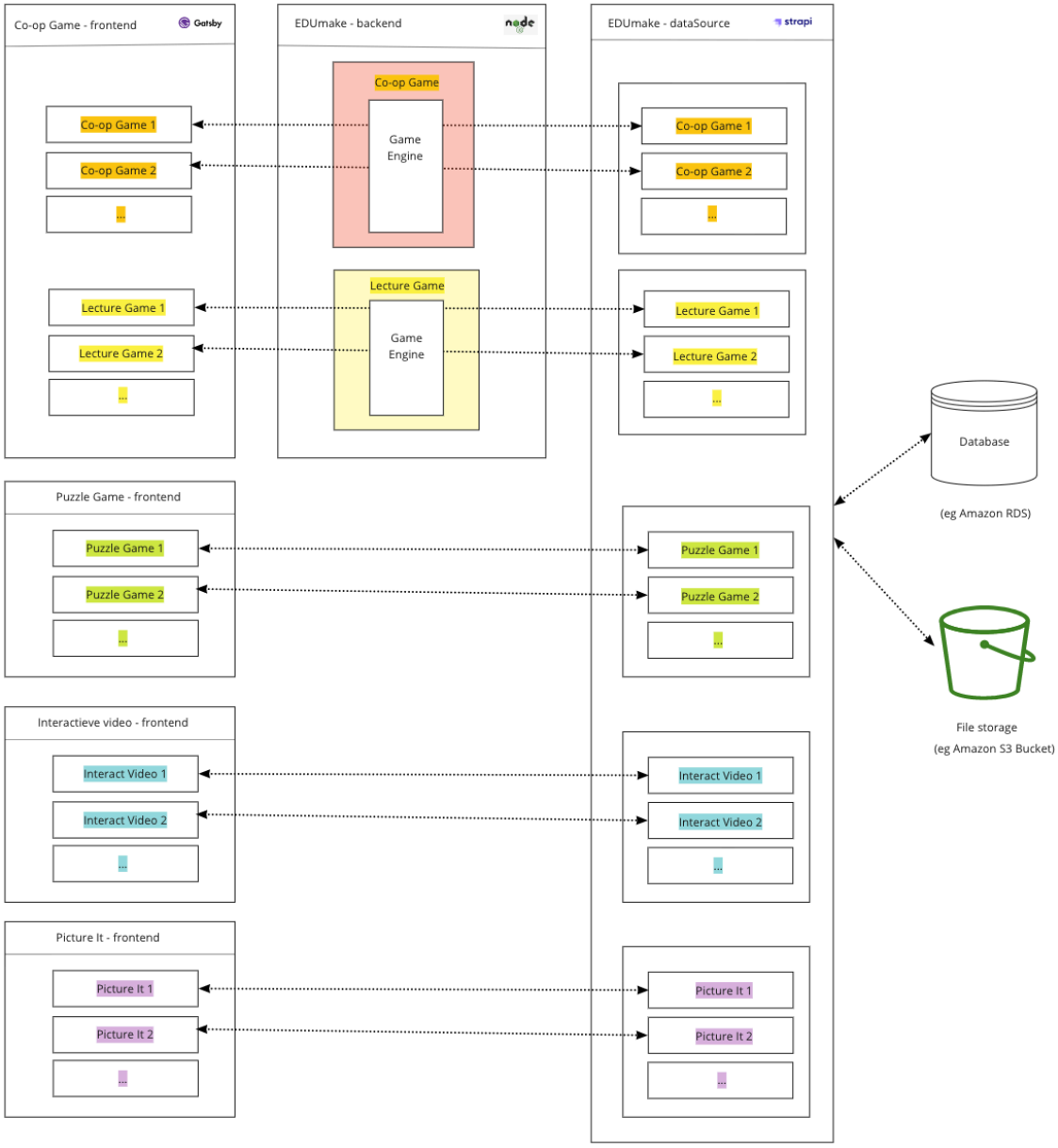
e. PictureIt

Picture it allows users (students) to give input by means of one or more photos. To each photo, they can add some textual context. The input per group comes together on a virtual photo board that the teacher can present. The overview of photos can then initiate class conversation, and be explored further by using filters (e.g. filter by color, sociodemographic ...).

3. Overview technical architecture

Like shown in the diagram there will be one content management system (Strapi) which contains content for all tools and will have the possibility to create new content. This CMS needs a database and a file storage service connected during setup-phase. The Co-op game and Lecture game will share a frontend (built in Gatsby) and will each communicate with a dedicated game engine in a shared backend (built in Nodejs), which will get the content of that specific game in the CMS. The three other tools (Puzzle game, Interactive video, and Picture it) will each have a separate frontend, which will get the content directly from Strapi. Those frontends contain all functionality of that tool so do not require a backend.

For any of the tools, existing content can be managed, and new content can be created via the Strapi CMS. For example, a new 'co-op game' named 'Road to Jupiter' can be created in the web client of Strapi by a content creator. Every piece of content can be added by clicking, typing, saving, … in the web interface, there is no need for a developer to do anything. This new content will then automatically be available to the fronted by adding the correct parameter to the url, for this example it could be:
http://www.url-of-the-organization.com/edumake/co-op-game?id=road-to-jupiter. We call this the "publishing" process.


Co-op Game - frontend
Gatsby
Co-op Game 1
Co-op Game 2
...
Lecture Game 1
Lecture Game 2
...
EDUmake - backend
node
Co-op Game
Game Engine
Lecture Game
Game Engine
EDUmake - dataSource
strapi
Co-op Game 1
Co-op Game 2
...
Lecture Game 1
Lecture Game 2
...
Puzzle Game - frontend
Puzzle Game 1
Puzzle Game 2
...
Puzzle Game 1
Puzzle Game 2
...
Interactieve video - frontend
Interact Video 1
Interact Video 2
...
Interact Video 1
Interact Video 2
...
Picture It - frontend
Picture It 1
Picture It 2
...
Picture It 1
Picture It 2
...
Database
(eg Amazon RDS)
File storage
(eg Amazon S3 Bucket)

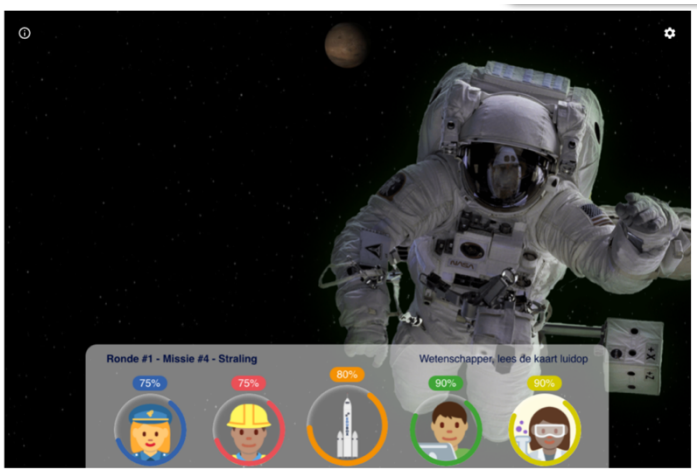
4. Specific technical architecture per tool

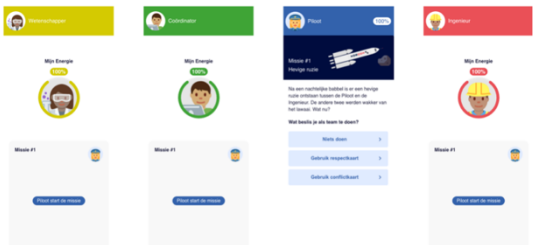
a. Co-op game

1. Hardware setup

1 host screen
= laptop, pc, …
= eg "the spacecraft"

4 individual screens
= smartphone, (tablet)
= the crew members
= eg "pilot, engineer, scientist, coordinator"





All screens communicate via websockets with the backend. Updates are sent and received across all screens in real time, so all devices go through the different game phases simultaneously. The backend is built in Nodejs and guides all communications of a certain group to a game room in the game engine. The game engine controls all game mechanics and communicates with a game state per group (which is a state machine), more on this in the next paragraph.

2. Game flow

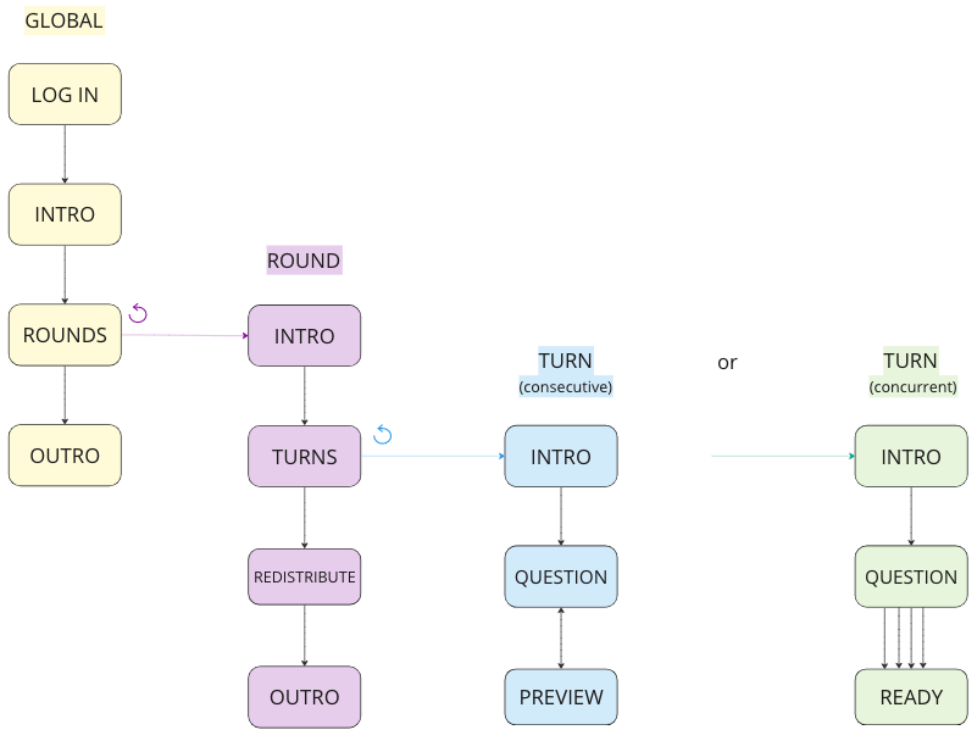
The game flow can be described by looking at four types of game state, of which the current status is controlled by the game engine. Each client device (the host and the guest screens) simultaneously go through these different phases in the game.

There is a **global** progress, which begins in a login phase where alle clients enter the game and are grouped in a game room, to keep the progress isolated for this specific group. When everyone is logged in, the game enters an intro phase where the game is explained by cycling through different informational slides. After this, the global rounds phase begins, which consists of one or more rounds, this number is freely chosen by the content creators, while using the CMS. When all rounds are completed, there is an outro phase which shows the team results and makes a conclusion on the team efforts.

Each **round** also has different phases. We start with an intro phase where there can be a thematical introduction, a quest that is explained, a mission that is given, … After this, the turns can begin. Typically, each player has a turn in a round (each player in consecutive order), but there can also be group turns, where all players, answer symultaneously (in concurrence). When the turn phase is completed, there is redistribute phase, where all players can redistribute the resource that they have left after the turns. The idea is that will trigger a tactical discussion in the team. Finally, there is an outro phase where a themetically conclusion can be made.

In a **consecutive turn**, one player has the lead, and after reading the intro phase, receives the question and the different answer options. When selecting an option, there is a preview on all screens of the impact on the resources that this option has. The lead player can close the option and return to the question phase if he wants or confirm the option and initiate the next turn for the next player.

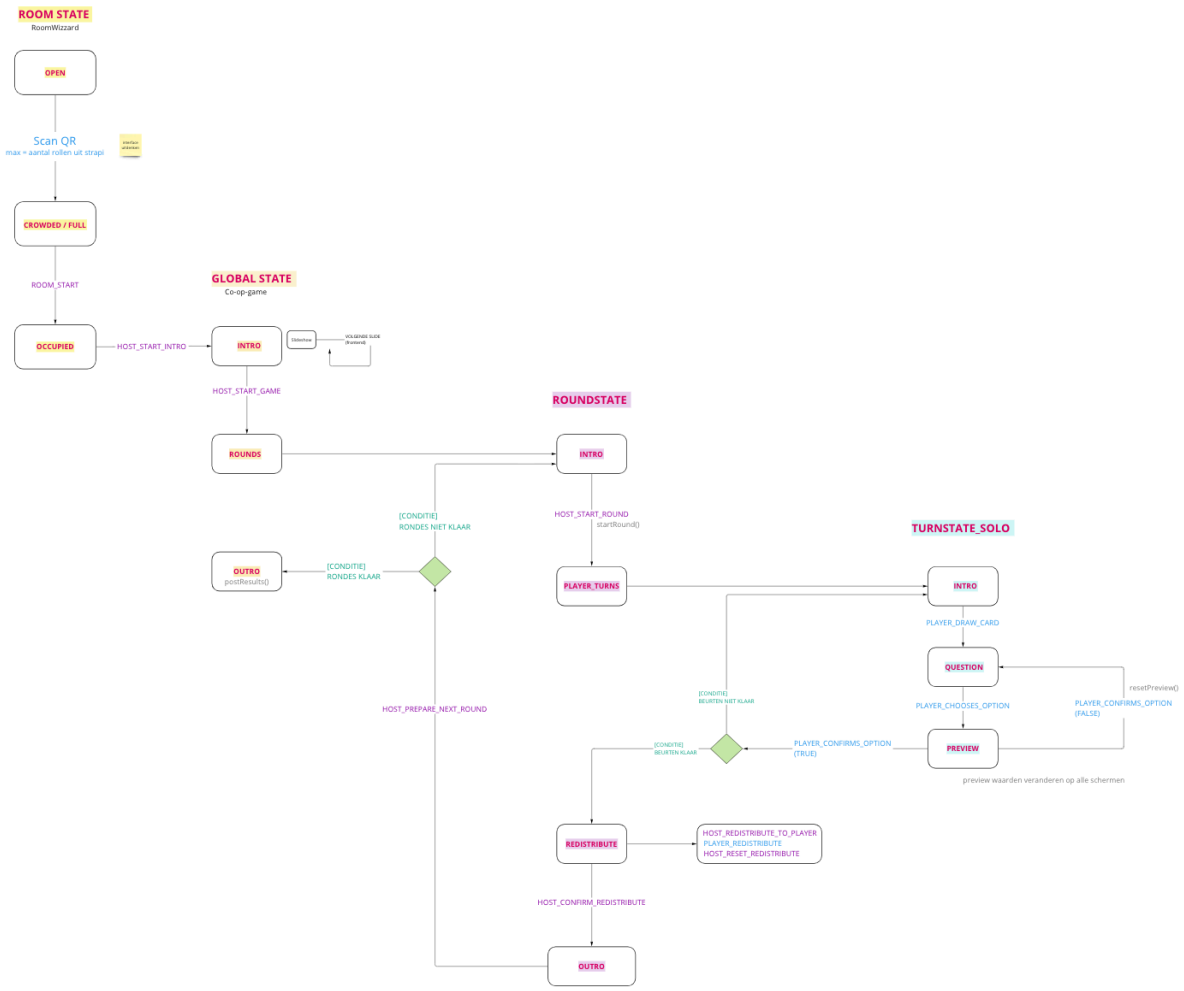
In a **concurrent turn**, again after a short intro, there is a question displayed on all player screens. When all players answered, the turn state will transition to a ready state, which finalizes the turn.

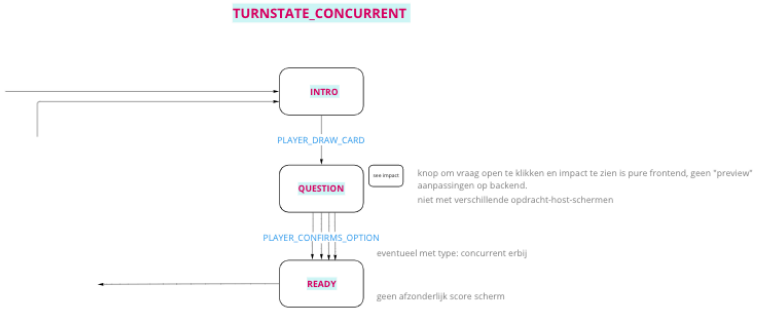


3. <u>Game state diagram</u>

Above diagram of the game flow is, in essence, a simplified version of the more detailed game state 'state diagram', which also contains decision making and a definition of which actions lead to which state transitions and by whom they are triggered.

pink (in frames) = backend state
purple (on arrows) = host action (on websocket)
blue (on arrows) = guest action (on websocket)
grey = game state / game engine methods
green (on arrows) = game state conditions


ROOM STATE
RoomWizzard
OPEN
Scan QR
max = aantal rollen uit strapi
CROWDED / FULL
ROOM_START
OCCUPIED
HOST_START_INTRO
GLOBAL STATE
Co-op-game
INTRO
Slideshow
VOLGENDE SLIDE
(Kiaxanal)
HOST_START_GAME
ROUNDS
ROUNDSTATE
INTRO
[CONDITIE]
RONDES NIET KLAAR
HOST_START_ROUND
startRound()
OUTRO
postResults()
[CONDITIE]
RONDES KLAAR
PLAYER_TURNS
TURNSTATE_SOLO
INTRO
PLAYER_DRAW_CARD
QUESTION
resetPreview()
[CONDITIE]
BEURTEN NIET KLAAR
PLAYER_CHOOSES_OPTION
PLAYER_CONFIRMS_OPTION
(FALSE)
HOST_PREPARE_NEXT_ROUND
[CONDITIE]
BEURTEN KLAAR
PLAYER_CONFIRMS_OPTION
(TRUE)
PREVIEW
preview waarden veranderen op alle schermen
REDISTRIBUTE
HOST_REDISTRIBUTE_TO_PLAYER
PLAYER_REDISTRIBUTE
HOST_RESET_REDISTRIBUTE
HOST_CONFIRM_REDISTRIBUTE
OUTRO


TURNSTATE_CONCURRENT
INTRO
PLAYER_DRAW_CARD
QUESTION
see impact
knop om vraag open te klikken en impact te zien is pure frontend, geen "preview" aanpassingen op backend.
niet met verschillende opdracht-host-schermen
PLAYER_CONFIRMS_OPTION
eventueel met type: concurrent erbij
READY
geen afzonderlijk score scherm

4. Content management: strapi

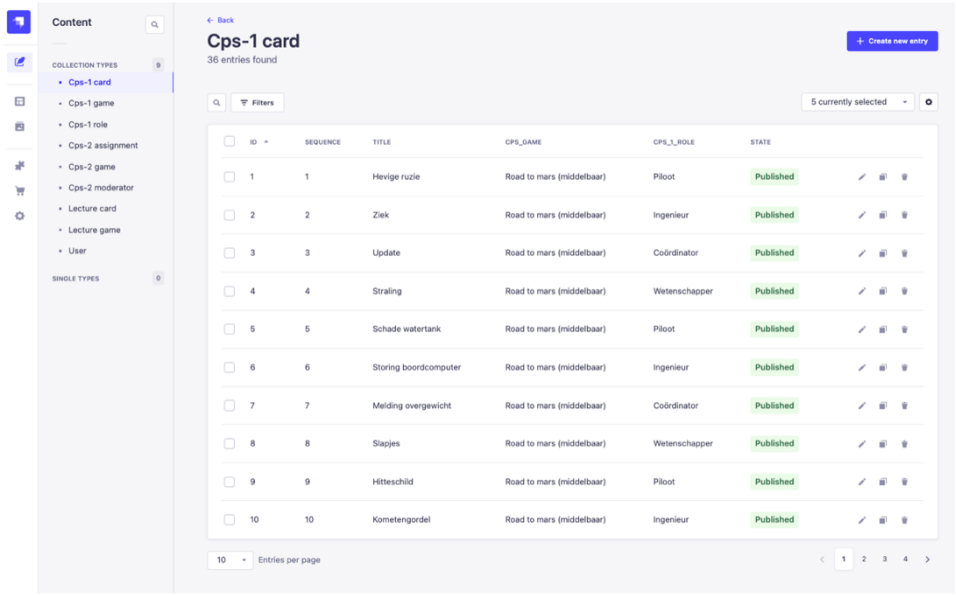
Once setup is completed, our CMS (Strapi) is a user-friendly web-app, where authorized users can create, publish and edit content

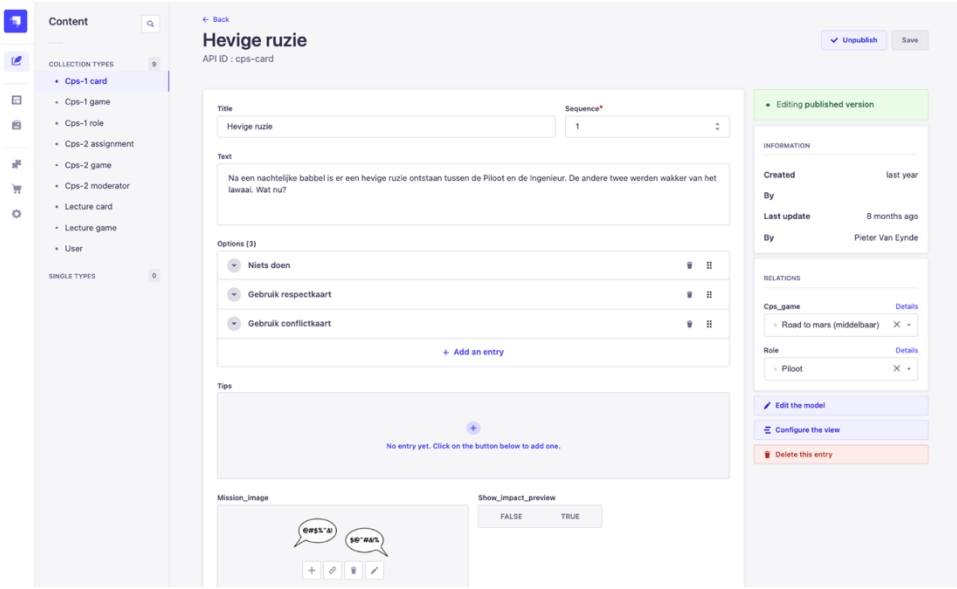
The main content types are:

- Games
- Rounds
- Turns (old name = cards)
- Roles

Each content type has

- A list overview
- A detail per item



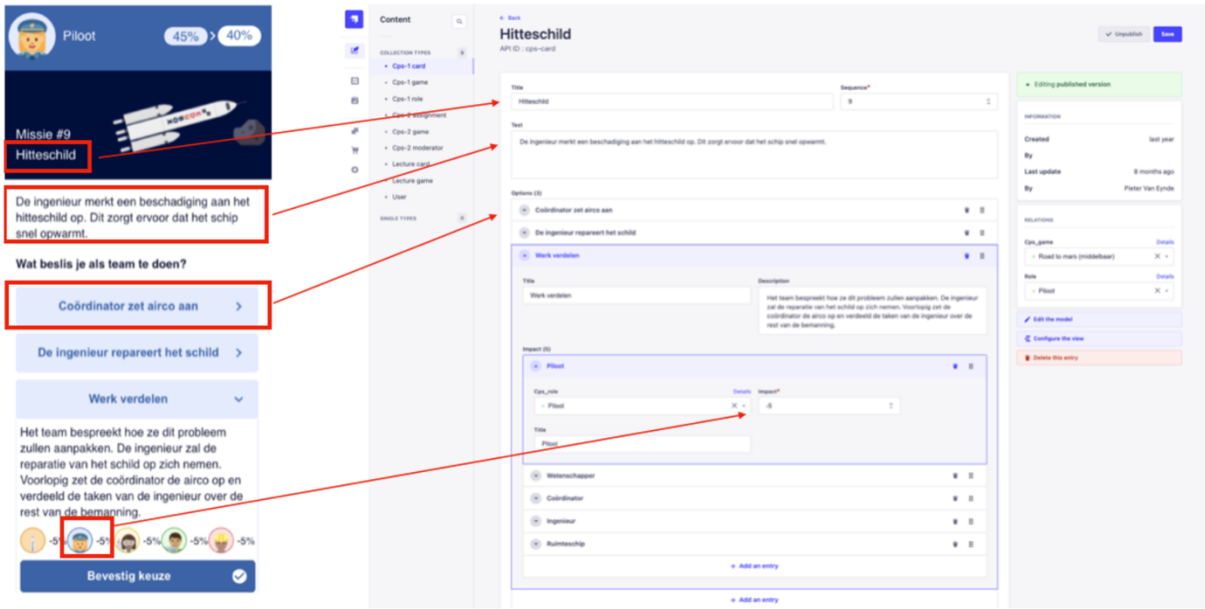


Adjustable:

- All titles, text, images, … that are visible in the co-op game.
- theme: general game props,
- turns: questions, options, impact, hints
- rounds: how many, which turns, intro, outro
- roles: how many, names, avatars, color…

Not adjustable:

- Mechanics from flow chart



5. Content types and data architecture.

In strapi, the main building blocks that are used while creating content are named "content types". By using these content types in a structured way, we can create the data architecture as displayed below.

round_redistribute

host_title	Redistribute your energy
host_description	Press the "+" button on your personal screen to give 10% of your energy to the rocket each time. Press the "+" button on the dashboard below the selected team member to give 10% of the rocket to that person each time.
host_reset_button	Reset redistribution
host_confirm_button	Confirm and continue
guest_button_title	Give energy to your team
guest_button_explanation	Using this button, you give 10% of energy to the rocket. From there, you can distribute it to your team members if desired. Before the distribution becomes final, it needs to be confirmed on the main screen.

Co-op Turn

turn_title	
turn_type	: normal | concurrent
turn_role_if_normal	: role
turn_question	: turn_question
turn_options	: turn_option [ ]
turn_hints	: turn_hint [ ]
turn_host_screen	: turn_host_screen
turn_show_impact_preview	: boolean
turn_sequence	: number ?

turn_question

guest_question *

guest_question_image

turn_impact

role * : role

impact * : number

turn_host_screen

host_background_image

host_question_image

host_title } nieuw

host_description } nieuw

turn_option

guest_title *

guest_description

impact : impact [ ]

turn_hint

role * : role

guest_description *

Co-op Role

title *

description

avatar : image

color

b. Puzzle game

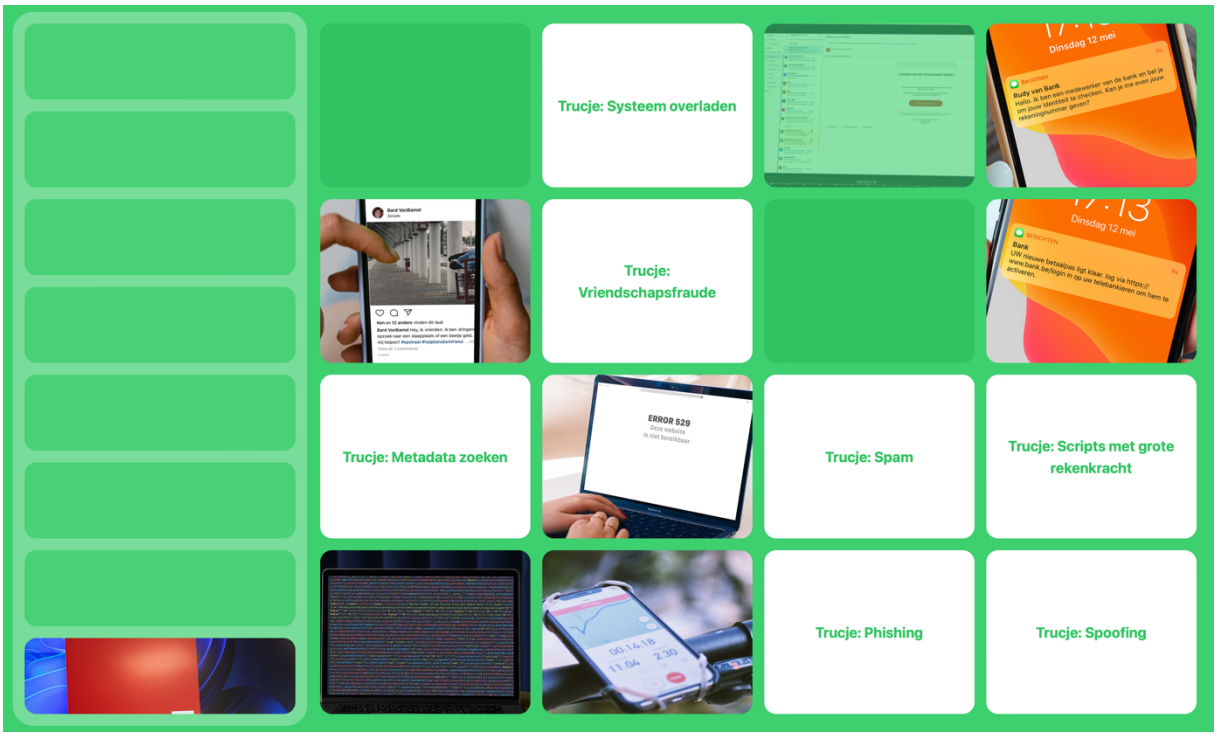
1. Hardware setup

1 host screen
= laptop, pc, …

All game content is loaded upon opening the page. no additional communication is required.

2. Game flow

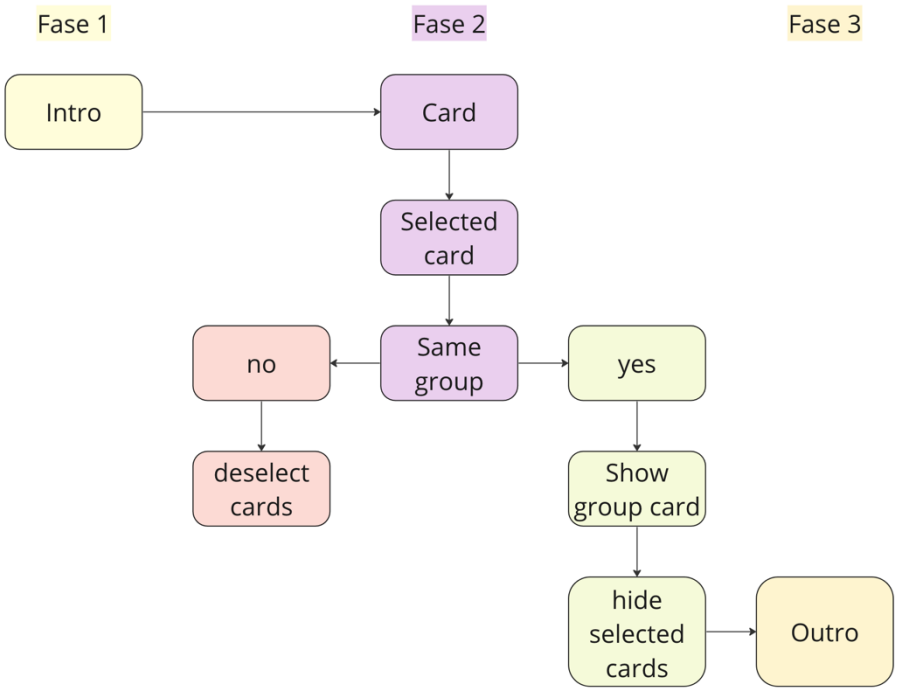
When the game page loads, the first thing that the user sees is a little description, what they will be expected to have to do. To start the next phase with the game, they can click on the button called "start now".



The game phase contains a grid of cards that a user can click on. To accomplish the game a user needs to find the cards that are related to each other. When they do, they will disappear from the grid. And a groups card that represent the cards will appear in the groups area on the screen.

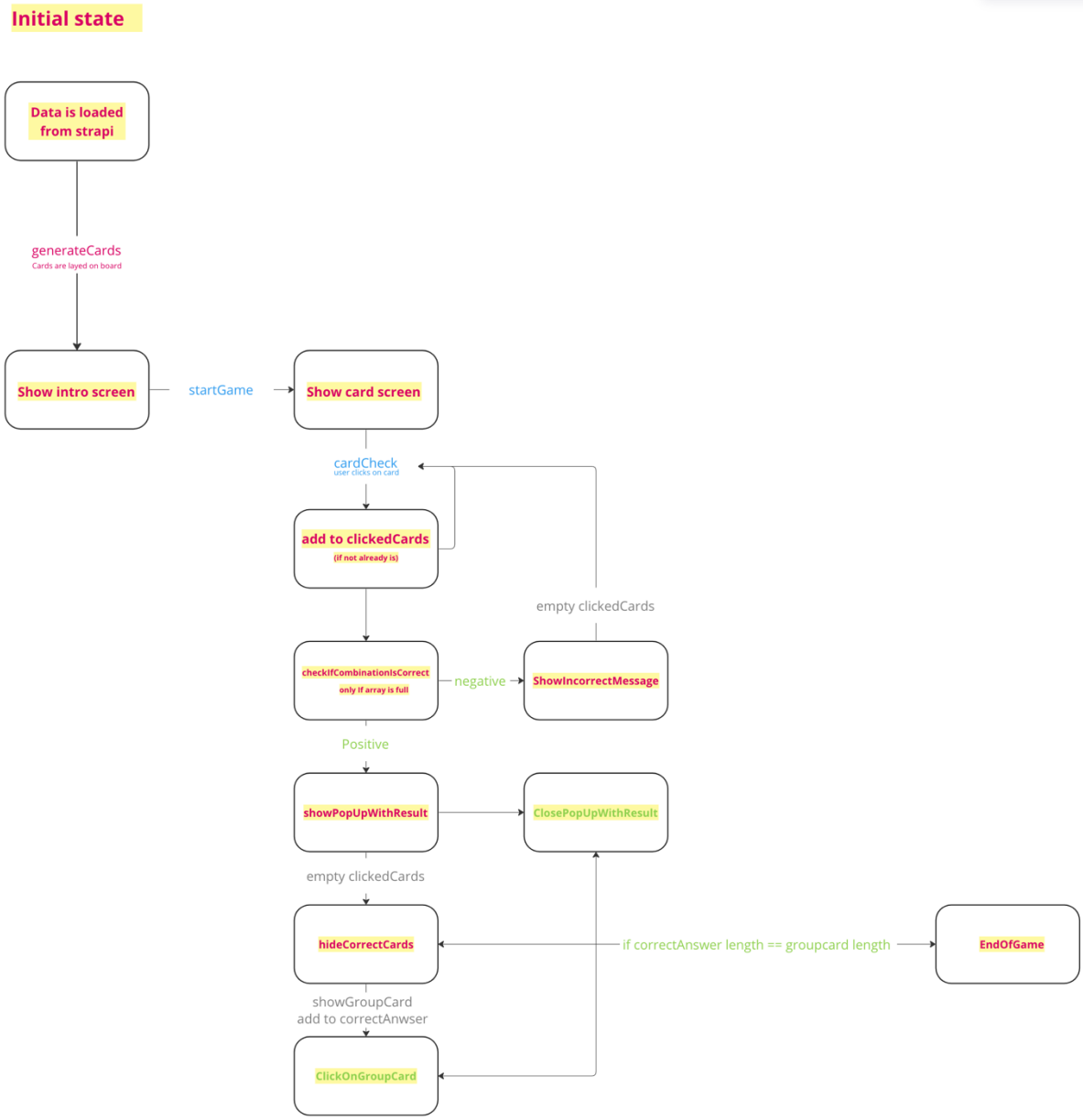
The card combinations can require 2 or more cards but will be always the same number of cards for every card group in a game.

When there are no cards left to click on, the game is completed, and the game goes to the third and last phase. Here will be a message displayed, with a call-to-action, to continue to the EDUbox.



3. <u>Game state diagram</u>

pink (in frames) = backend state
blue (on arrows) = guest action
grey = game state / game engine methods
green (on arrows) = game state conditions



4. Content management : strapi

Once setup is completed, our CMS (Strapi) is a user-friendly web-app, where authorized users can create, publish and edit content

The main content types are:

- Puzzlegame

Each content type has

- Puzzle_card_result
- Puzzle_card_game

5. Content types and data architecture

Puzzle game

puzzlegame	
puzzlegame_id	Auto increment
puzzlegame_name	: string
puzzlegame_instructions	: text
puzzlegame_startbutton	: text
puzzlegame_endscreen	: text
puzzlegame_msg_succes	: text
puzzlegame_msg_error	: text
puzzlegame_theme_color	: colorpicker
puzzlegame_text_color	: colorpicker
puzzlegame_col_count	: number
puzzlegame_row_count	: number
puzzlegame_result_count	: number
puzzlegame_result_random	: false
puzzlegame_game_random	: true

puzzle_card_result	
resultcard_id	Auto increment
puzzlegame_id	: id
resultcard_title	: string
resultcard_text	: text
resultcard_image	: image
resultcard_images_alttext	: text

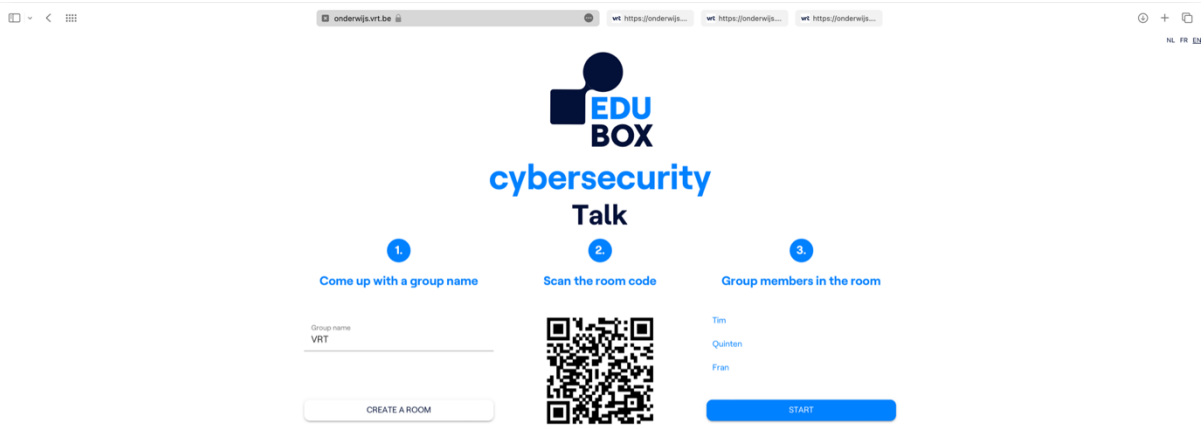
puzzle_card_game	
gamecard_id	: id
puzzlegame_id	: id
resultcard_id	: id
gamecard_title	: string
gamecard_content_type	: image,video,audio, none
gamecard_image	: file
gamecard_alttext	: text
gamecard_video	: file
gamecard_audio	: file

c. Lecture-app

The Lecture-app has a similar technical implementation as the co-op game.

The hardware-setup is quasi-identical, also here there is 1 host screen (laptop, pc, …) and 4 individual screens (smartphone, tablet). All screens communicate via websockets with the backend. Updates are sent and received across all screens in real time, so all devices go through the different game phases simultaneously. The backend is built in Nodejs and guides all communications of a certain group to a game room in the game engine. Even though, much simpler as in the co-op game, the game engine controls all mechanics and communicates with a game state per group.

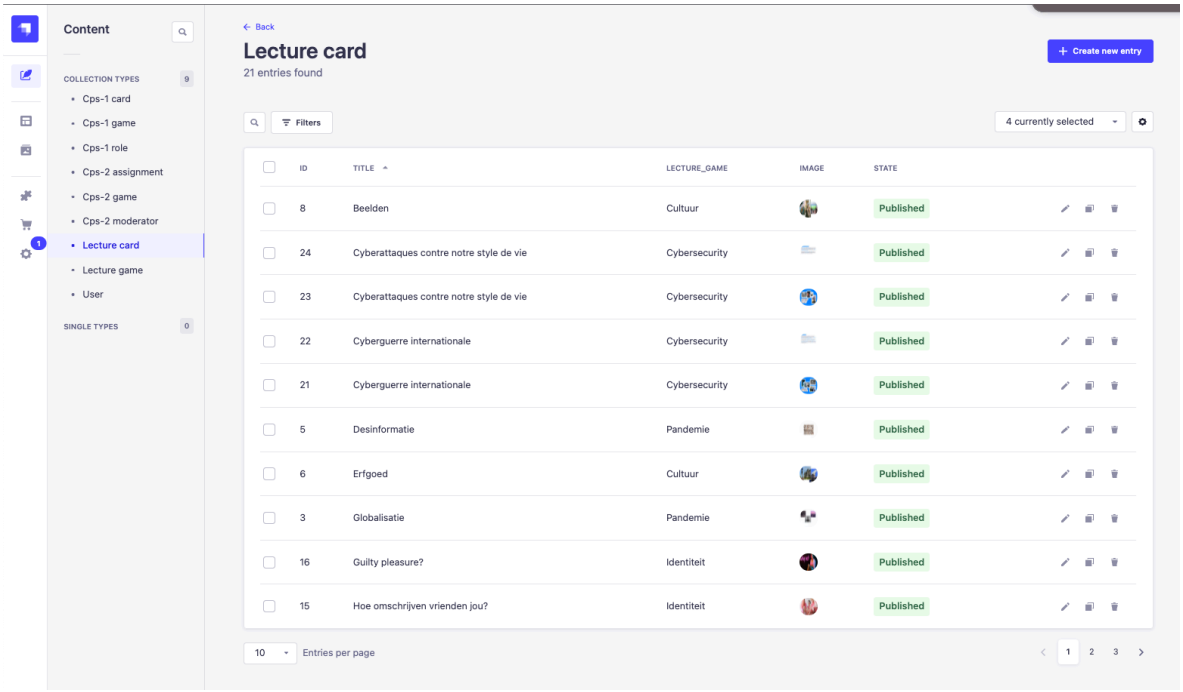
There also is a login-phase where groups are created, players can log in and the game can start:

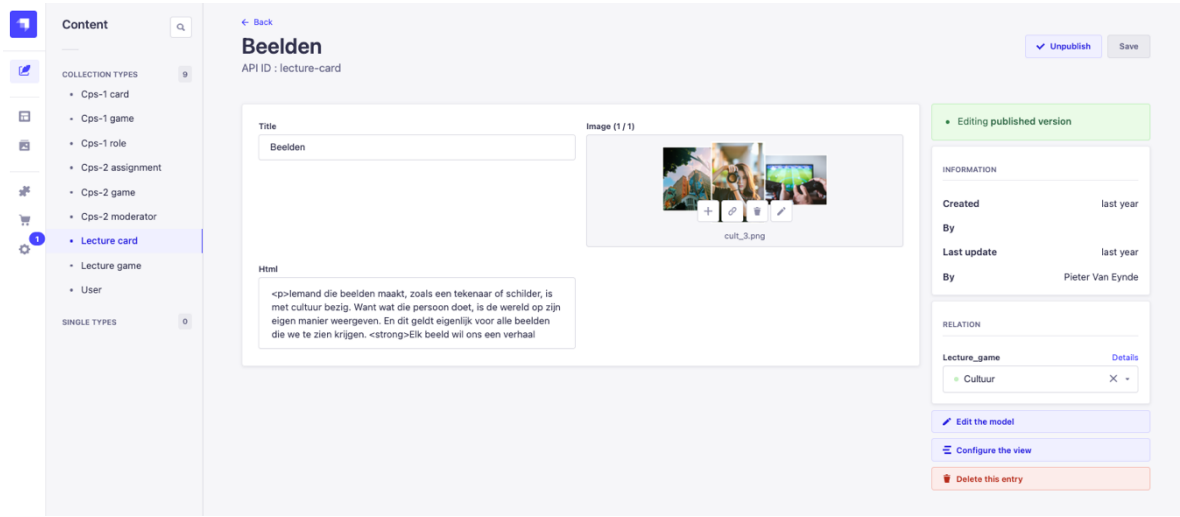


In the center of the game mechanics, there is a story image on the main screen, a story on one of the individual screens and call to listen on the other individual screens.

Also like the co-op game, the game-data can be managed, created, edited, published in Strapi. Data that can be controled includes:

- EDUbox theme id
- Card image
- Card title
- Card tekst




Content
COLLECTION TYPES
9
Cps-1 card
Cps-1 game
Cps-1 role
Cps-2 assignment
Cps-2 game
Cps-2 moderator
Lecture card
Lecture game
User
SINGLE TYPES
0
Back
Beelden
API ID : lecture-card
Unpublish
Save
Title
Beelden
Image (1 / 1)
cult_3.png
Html
<p>Iemand die beelden maakt, zoals een tekenaar of schilder, is met cultuur bezig. Want wat die persoon doet, is de wereld op zijn eigen manier weergeven. En dit geldt eigenlijk voor alle beelden die we te zien krijgen. Elk beeld wil ons een verhaal
Editing published version
INFORMATION
Created
last year
By
Last update
last year
By
Pieter Van Eynde
RELATION
Lecture_game
Details
Cultuur
Edit the model
Configure the view
Delete this entry

To be complete, we will also add the data architecture of the Lecture game, even though it is very basic, compared to the other games.

Lecture Game	
game_title	Lecture game
game_id	lecture-game

Lecture card	
card_title	Card title
card_image	: image
card_content	: html

d. Interactive video

1. Hardware setup

1 host screen
= laptop, pc, …

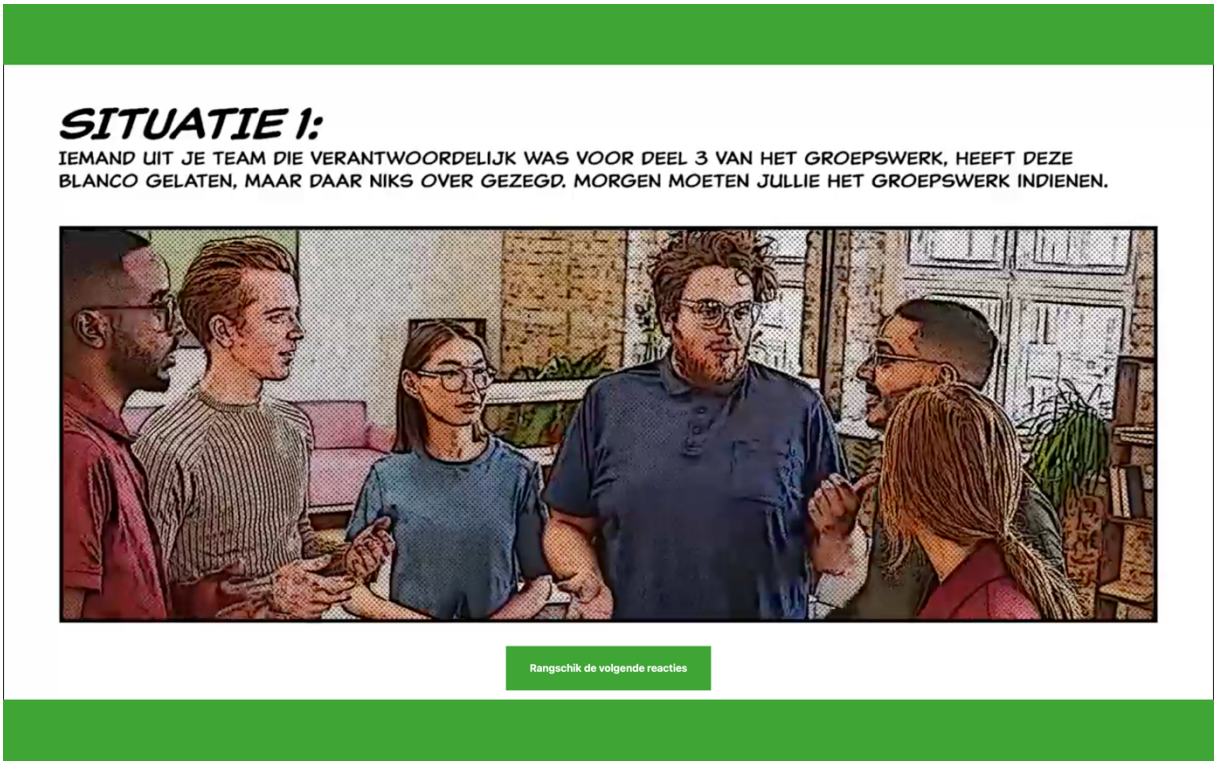
All game content is loaded upon opening the page. Video is loaded from Youtube via javascript

2. Game flow

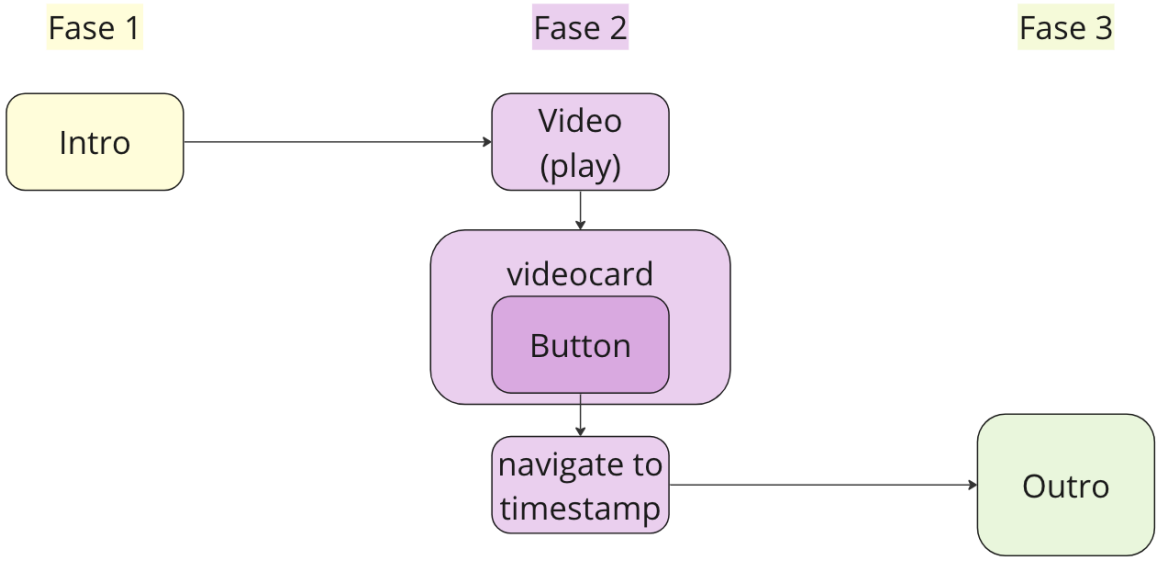
When the game page loads, the first thing that the user sees is a little description, what they will be expected to have to do. To start the next phase with the game, they can click on the button called "start now".

The video starts playing and the code on the background will watch the time progress of the video, that is playing. When the video time is between a predefined start and end time, a function is called that either pauses the video or puts it in loop between the start and end time that was defined. At the same time, an instruction screen is displayed, which may contain an explanation and one or more buttons. By pressing the button, the video gets a new start time, from which the video will continue to play, and one leaves the instruction screen until the video time is again between a predefined start

and end time.



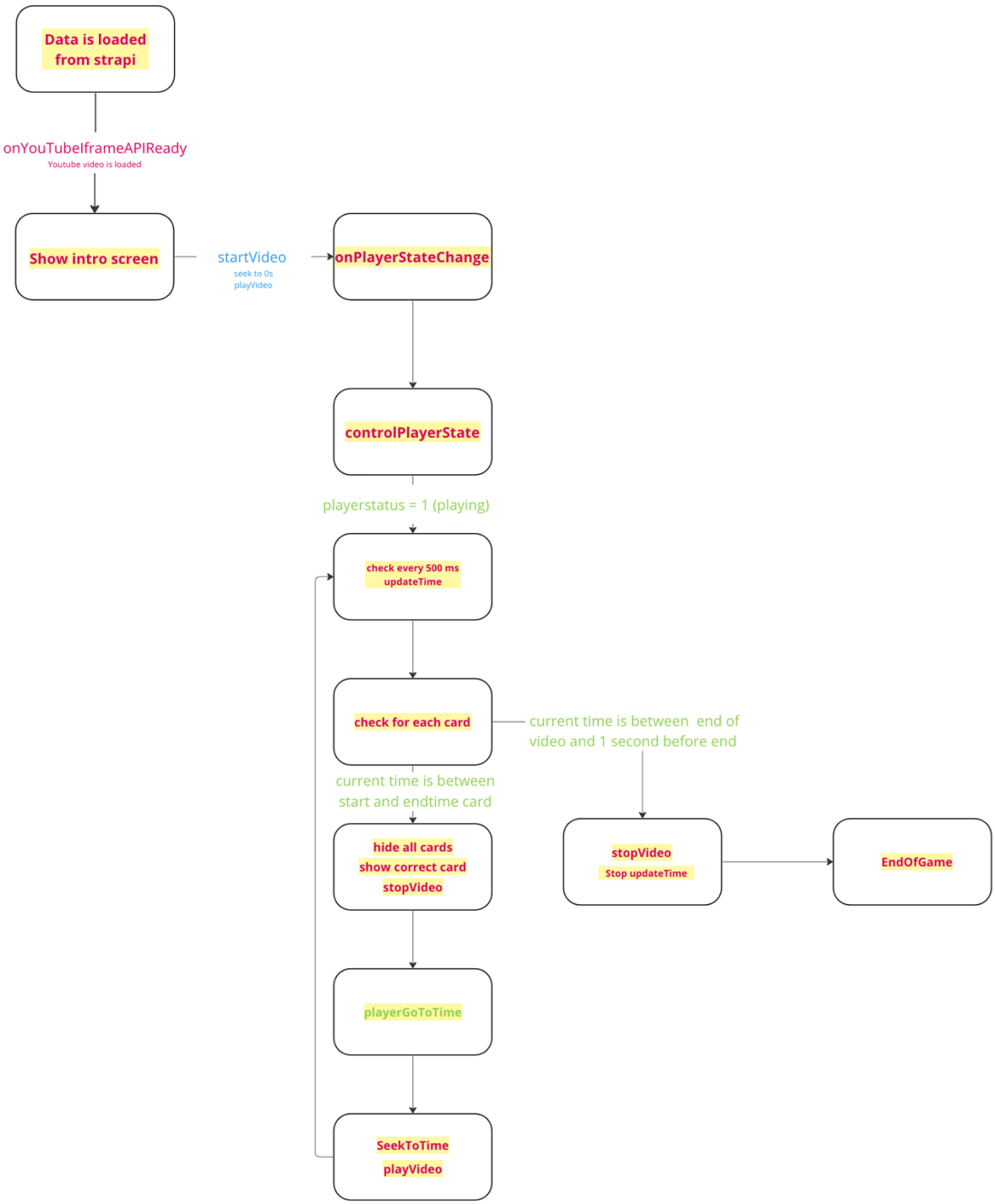
When the video time is at the end of the video, the latest pancarte is made visible, here will be a message displayed, with a call-to-action, to continue to the EDUbox.



3. <u>Game state diagram</u>

pink (in frames) = backend state
blue (on arrows) = guest action
green (on arrows) = game state conditions

Initial state



4. Content management : strapi

Once setup is completed, our CMS (Strapi) is a user-friendly web-app, where authorized users can create, publish and edit content

The main content types are:

- Interactive_video

Each content type has

- Interactive_video_screen

5. Content types and data architecture

Interactive video

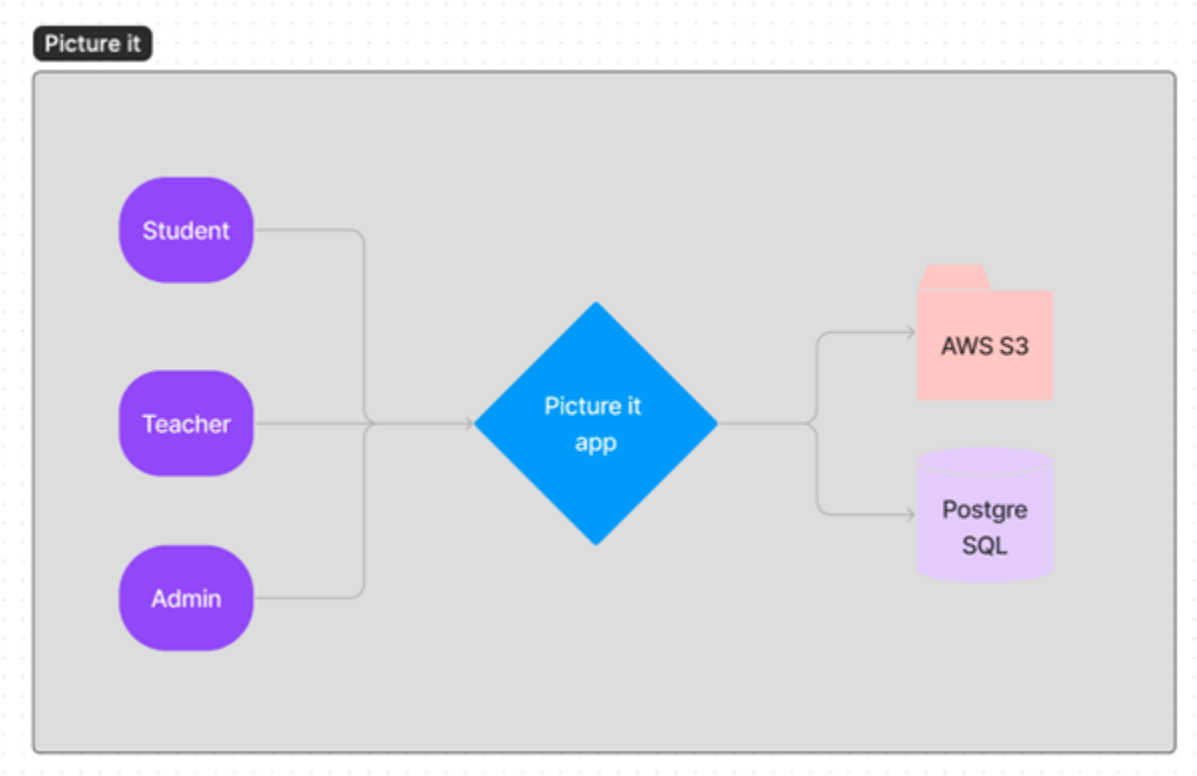
interactive_video	
video_id	Auto increment
video_url	: string
video_instruction_text	: text
video_endscreen_text	: text
video_startbutton_text	: text
video_theme_color	: colorpicker

interactive_video_screen	
screen_id	Auto increment
video_id	: id
screen_time_start	: number
screen_time_end	: number
screen_title	: text
screen_discription	: text

a. PictureIt

Picture it is a monolithic app built in Ruby on Rails. This app serves all different user types (the students participating, the teacher, and the admins). The app stores its data in a postgreSQL database, and the assets (pictures) uploaded in an AWS S3 bucket.

Each setup is a complete standalone application and does not share any data with another setup.



Student flow

A student can participate by sending in one or multiple pictures that answer to a specific challenge. This follow has been designed in a mobile-first way, so that students can take a picture on a device while using the website.

This consists of the following steps:

- Join a group by entering the code (which the teacher has given them outside of the app)
- Answer some general questions (can be configured by the admins)
- Upload a picture (or select one from a pre-defined library, if enabled) and answer a question related to that picture

The general questions only need to be answered once if the student wants to upload multiple pictures.

While the user is using the app we store an anonymous user id on their device, so that we can keep track of the answers and the pictures they uploaded.

Teacher flow

A teacher can create one or more groups and show the pictures for a group.
In order to create groups, a teacher needs to create a login with their email and password.

While creating a group, some additional questions can be asked about the group (this is configurable by the admins).

Once a group is created, a teacher can give the group's code to their students. The teacher has a presentation screen to show the pictures that have been submitted. In this screen they can:

- Filter the pictures
- Mark pictures as rejected if they are not appropriate.
- Compare the pictures of their group to those of other groups.

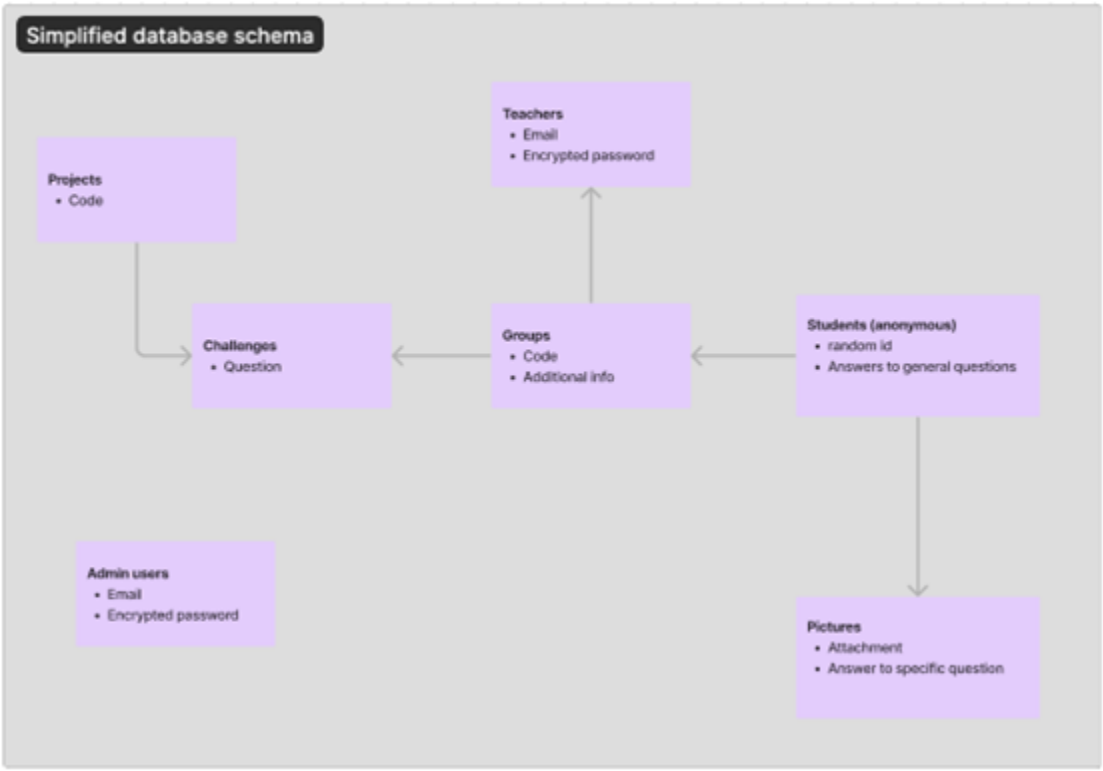
While this flow works on both mobile and desktop, it is recommended to use a bigger screen to display the pictures.

Admin backend

The admin has an additional interface where they can manage the projects, settings and moderate pictures that have been sent in.

Database schema

Note: the below is a simplified version, that gives an overview of the main models and structure of the application.


Simplified database schema
Teachers
• Email
• Encrypted password
Projects
• Code
Challenges
• Question
Groups
• Code
• Additional info
Students (anonymous)
• random id
• Answers to general questions
Admin users
• Email
• Encrypted password
Pictures
• Attachment
• Answer to specific question

6. Distribution

The created applications will be completely shared with the project partners. This sharing consists of two phases.

Firstly, the complete source code will be shared, normally through direct access to a Git repository. This has yet to be approved and setup by the correct internal VRT services. If there should be an issue with this, the source code files can be shared directly. Upside of the first option is that it is easier to stay in sync with possible updates but the second option is also viable.

Secondly, in the source code, there will be configuration files for Docker deployment. Every individual peace of of the application (frontends, backend, cms) will have a docker configuration file and will be able to function in a docker container. We will then also provide a docker-compose configuration that allows for building all the different parts and starting up the entire application with one single command. This configuration should be enough for a technical department to host this application on any type of server: a local, self-managed server or by using one of the cloud based providers like AWS, Azure, Google,...