

Table of Contents

- [Introduction](#)
 - [Purpose of the System](#)
- [Requirements](#)
 - [Visionary Scenarios](#)
 - [1- Building Worlds](#)
 - [2- Recording Actions](#)
 - [3- Creating Stories](#)
 - [4- Simulating Behavior](#)
 - [5- Generating Experience](#)
- [Analysis](#)
 - [User Interface](#)
 - [Terminology](#)
 - [Use Case Model](#)
 - [Analysis Object Model](#)
- [System Design](#)
 - [Overview](#)
 - [Subsystem Decomposition](#)
 - [Hardware/Software Mapping](#)
 - [Persistent Data Management](#)
 - [Access Control and Security](#)
 - [Boundary Conditions](#)
 - [Server-side](#)
 - [Ktor-Server](#)
 - [AWS-Server](#)
- [Demo Scenario](#)
- [Administrator Manual](#)
 - [Infrastructure Setup up](#)
 - [Client-side](#)
 - [Server-side](#)
 - [Deployment and Configuration](#)
 - [Bamboo Setup](#)
 - [Client](#)
 - [Server](#)
 - [1- Build and Push Image](#)
 - [2- Deploy to VM](#)
 - [Servers](#)
 - [Main Backend Server with PostgresSQL Server](#)
 - [AWS Server](#)
 - [Third-party Components](#)

Introduction

Purpose of the System

The app is an Augmented-Reality-based proof of concept helping designers boost their own creativity. It lets the user design a playground in AR, whilst providing different opportunities for him to get inspired. Using an advanced neural network, the app finishes stories the user starts artificially. The user can see a simulation of agents interacting with the world, as well as being able to record custom body gestures to place in the world. The app also shows fun facts, oblique strategies and hints on what the user might want to do next while using it. Furthermore, the user does not need to work alone on the playground, but can actually work in a collaborative manner together with others.

The app harnesses both the useful, but also the outlandish behavior of AI to help the designer become more creative.

Requirements

Visionary Scenarios

The problem statement mentions that a system is needed for Augmented Imagination. Augmented Imagination includes the interaction between the computers and the human in the creative process. In other words, human can help computers and computers can help humans for imagination. Considering this main aspect of the application, five different features can be presented which are Building Worlds, Recording Actions, Creating Stories, Simulating Behaviors and Generating Experiences. These features are just the aspects of the Augmented Imagination for designing a playground, putting human at center and letting the machine do the rest. All features are deeply analyzed in details in this section.



1- Building Worlds

For building worlds, there are many ways that Augmented Reality presents. One of them is simply putting some objects from a list of objects depending on the pleasure of the user. If there is no object in the list, the user can draw them. If the user has lack of time, he/she can place an annotation saying that "I will check here later.". Therefore, these ways have to be implemented in the application so that building worlds should not be a problem for the users.



2- Recording Actions

Recording Actions feature can make the application dynamic as it can detect and capture the motion of humans in the playground. This will lead to the dynamic 3D model with captured human behaviors.

3- Creating Stories

To boost the imagination of the users by inspiring them, the application can use the story generation feature. This feature is about Artificial Intelligence to make the stories "inspiring nonsense". Because no matter how nonsense the story is, each story can lead humans' thoughts to something else which is not included in the playground. That can be an object or an action not captured in the playground as well. The users can put very first events of the story using their voice (and voice recognition takes part here) and the machine can follow this story. An example story, generated by a machine is presented below. The very first sentence (in bold font) is created by human and the rest is completed by the machine.

I am having a lot of exams in these days. In a lab, I'll write the lab specification. I'll write the software as well. All those things are important for me. When I just sit at the computer and go like "Nope, these are all not here. Here's what's missing," they're not going to provide me that feedback. It's very hard for me to work if I'm not focused on the project, so I really want to give my mind time to relax and not have to be so focused on getting it done.

4- Simulating Behavior

In the playground Recording Action feature should not be the only feature that makes the application dynamic. What about having artificial avatars in the playground? The avatars obviously have some behaviors related to the libraries of animations, but for some of the main behaviors such as walking, the physical interactions with the playground should also be considered.

5- Generating Experience

The experience of the application can be boosted with some additional video style transfers to show the whole playground in more elegant way. Showing the playground as a Hollywood film scene or just a high-quality video can be an example for that.

As stated in the beginning of this section, Augmented Imagination can both be explained by humans helping the computers or vice versa in the creative process. Considering this aspect, our team has developed a trailer which shows the case of the machine helping an architect who has lack of inspiration in his work. Story generation and building worlds features are combined and presented in the trailer.

The trailer can be found here: [Dropbox Link](#)

Analysis

User Interface

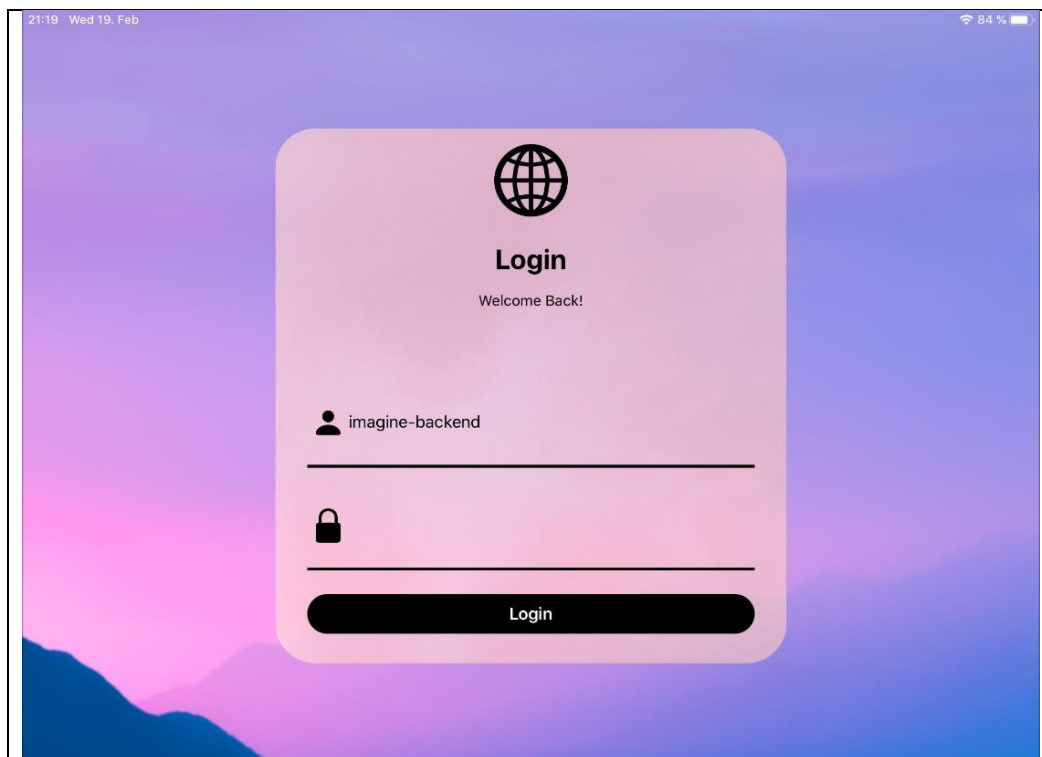


Fig. 1 Login screen of the app. The user will not be able to login if the device is not connected to the internet.

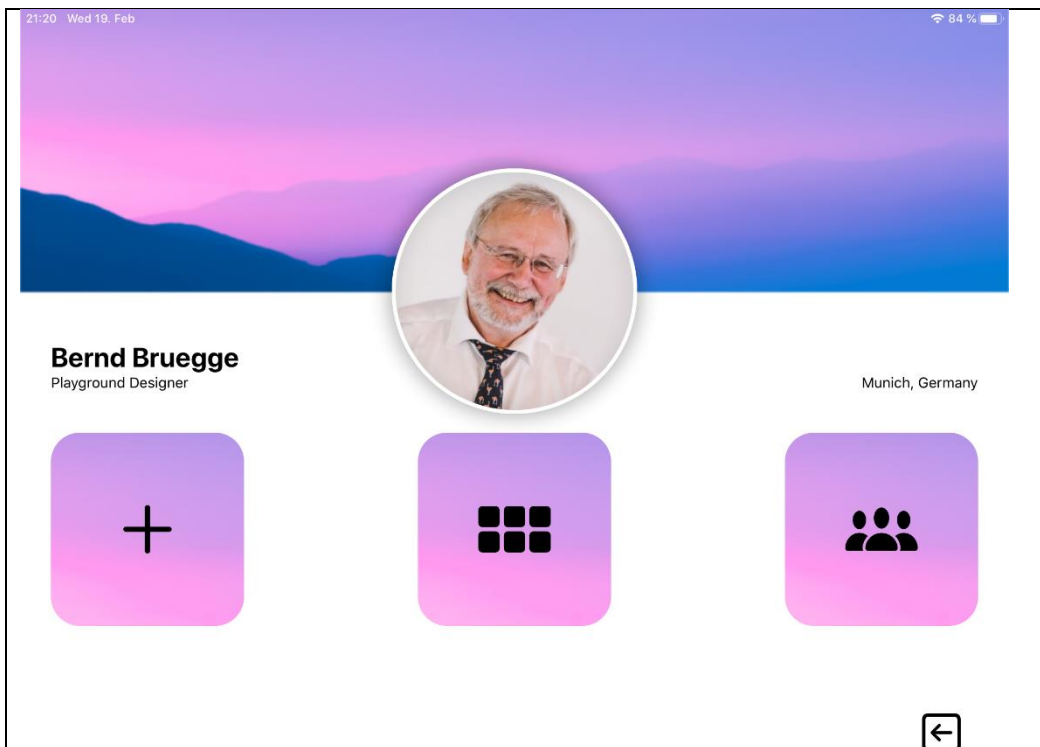


Fig. 2 Home screen of the app with user profile. By clicking on the plus sign, the user can create a new project. By clicking on the middle sign, the user can load an earlier project. By clicking on the last sign, the user can enter the collaborative session and either join or host a session. If the user taps on the collaborative session, he will automatically host a session, if he is the first one nearby or he will automatically join another session if someone else already entered the collaborative session.

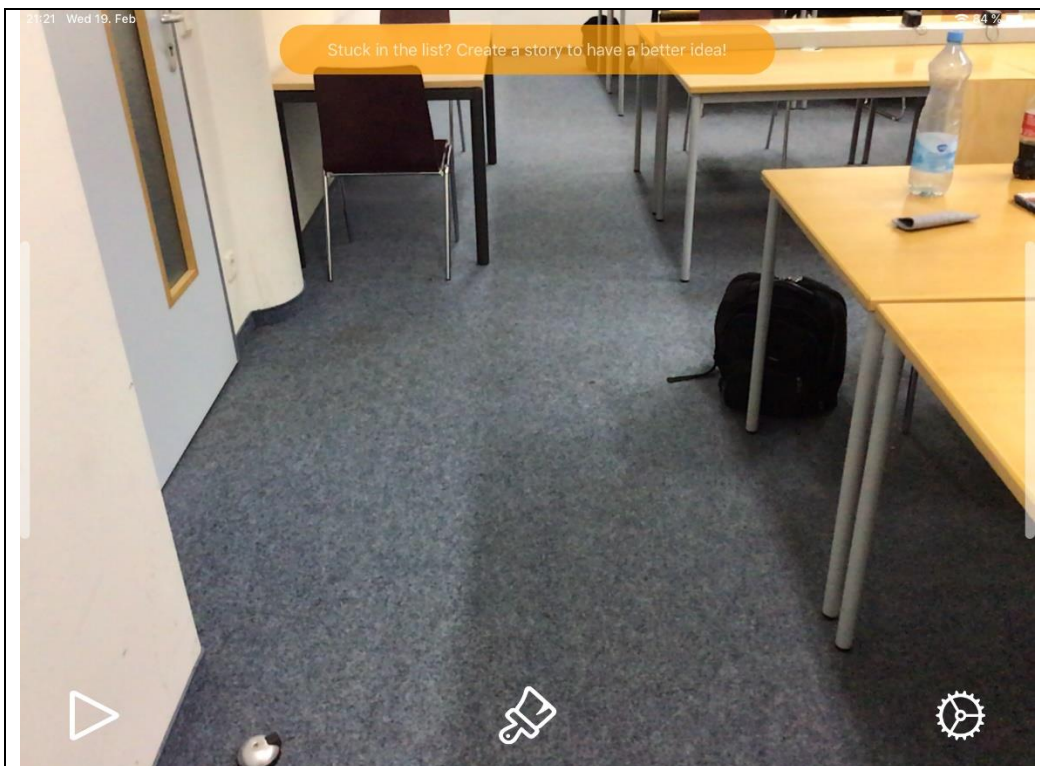


Fig. 3 AR View of the app. The user can visualize the world and read some random facts.

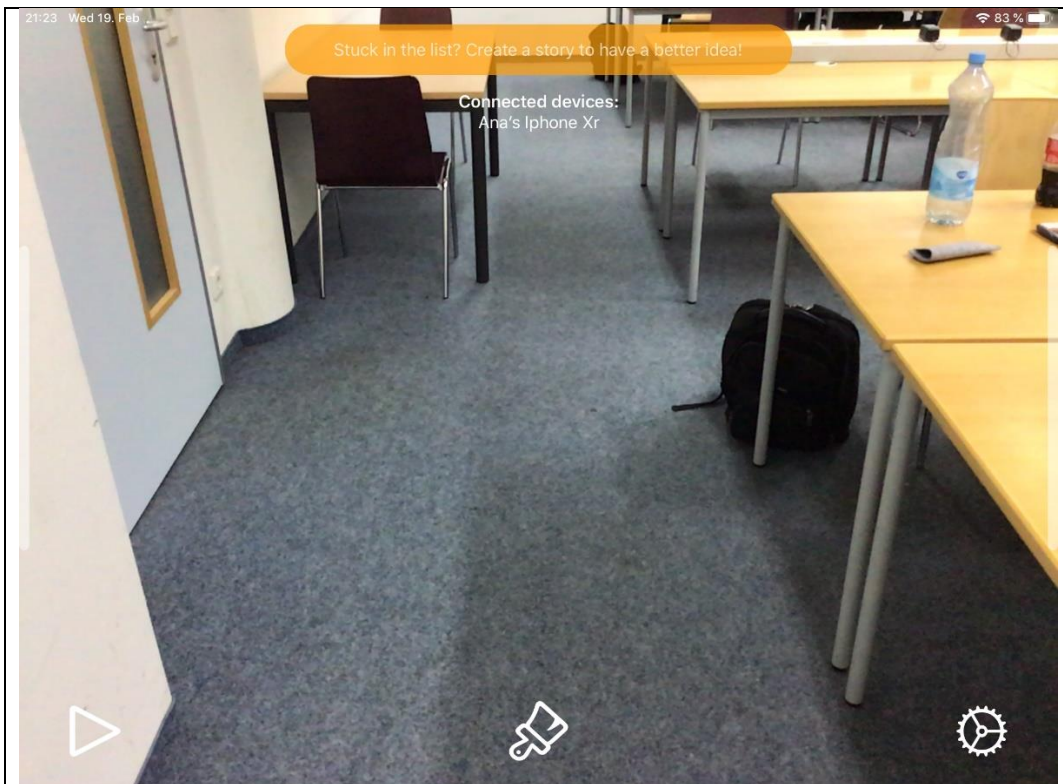


Fig. 4 The collaborative session view. Other connected users will appear on the top of the screen. The user will automatically see everything all other users do in the world provided a good internet connection.

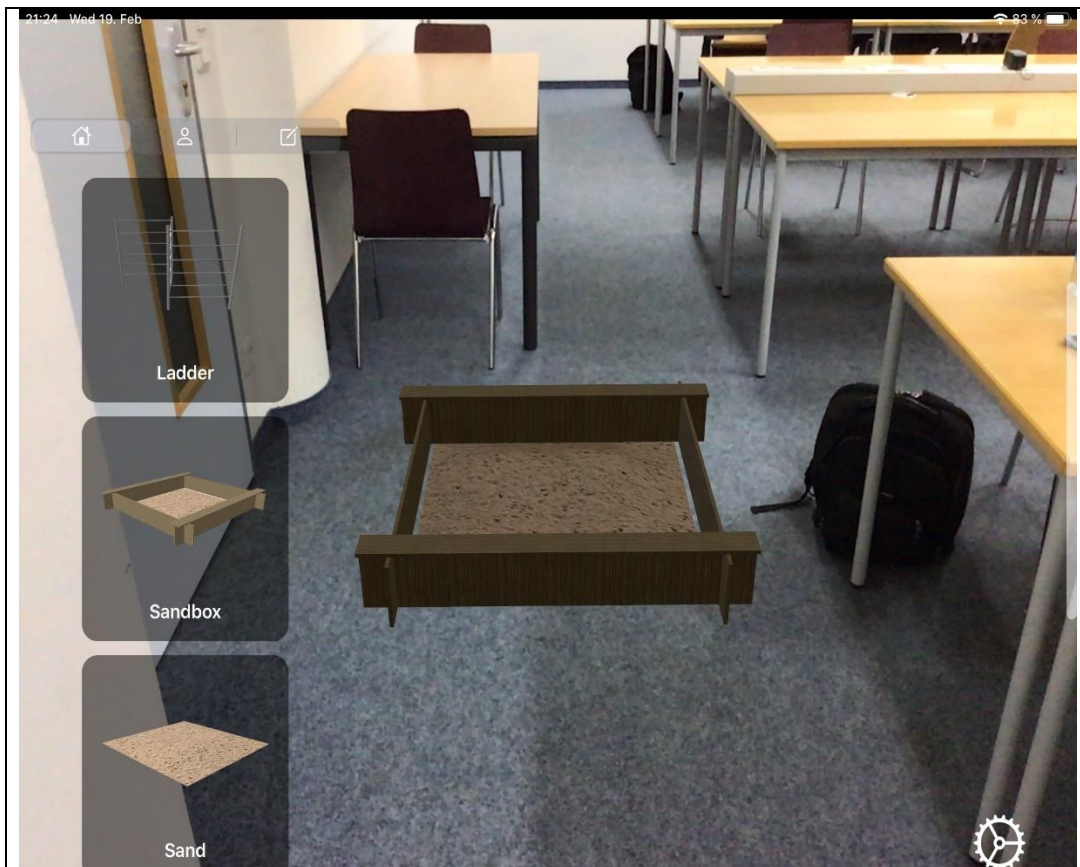


Fig. 5 By sliding to the right on the left corner of the screen, the user can see the object list. The user can place objects by clicking on them in the side bar and then tapping on the screen.



Fig. 6 By clicking on the icon in the middle of the side bar and then on the record button, the user can record actions. The actions will automatically be saved and then be visible in the side bar. The user can place them anywhere in the world exactly as the objects.

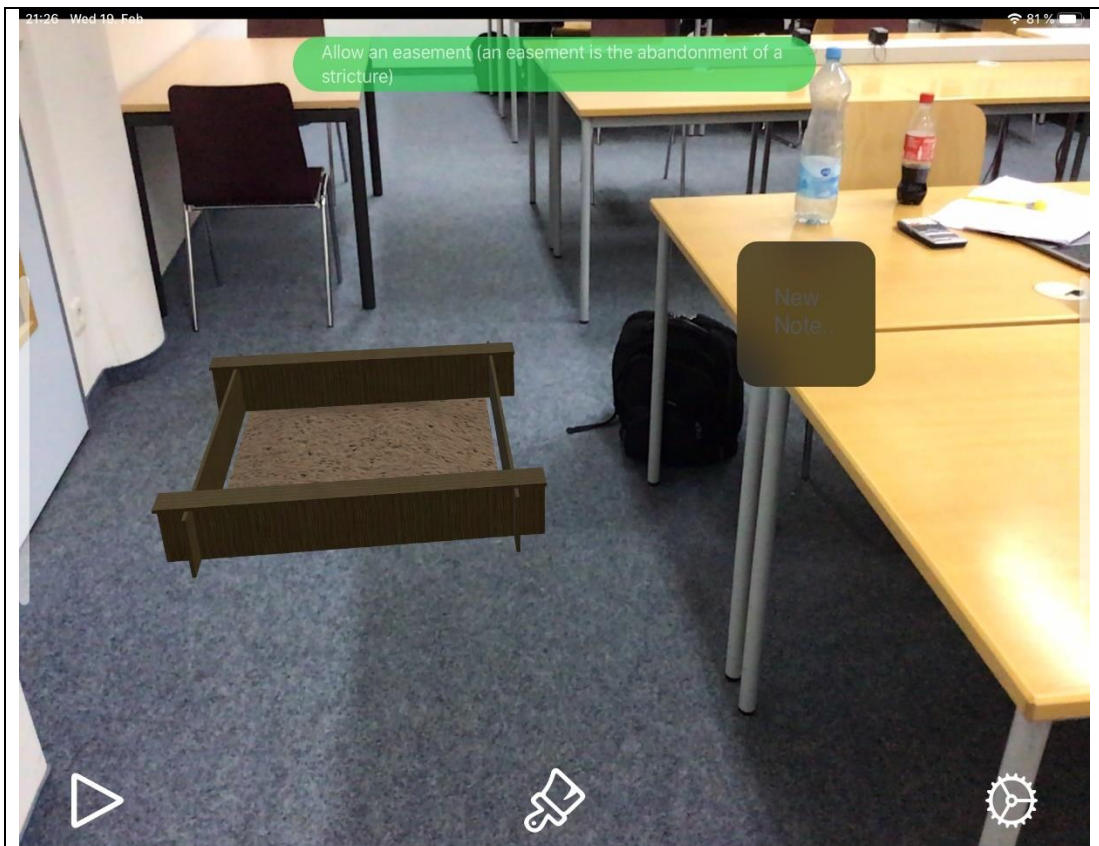


Fig. 7 By clicking on the icon on the right of the side bar, the user can place annotations anywhere in the world.

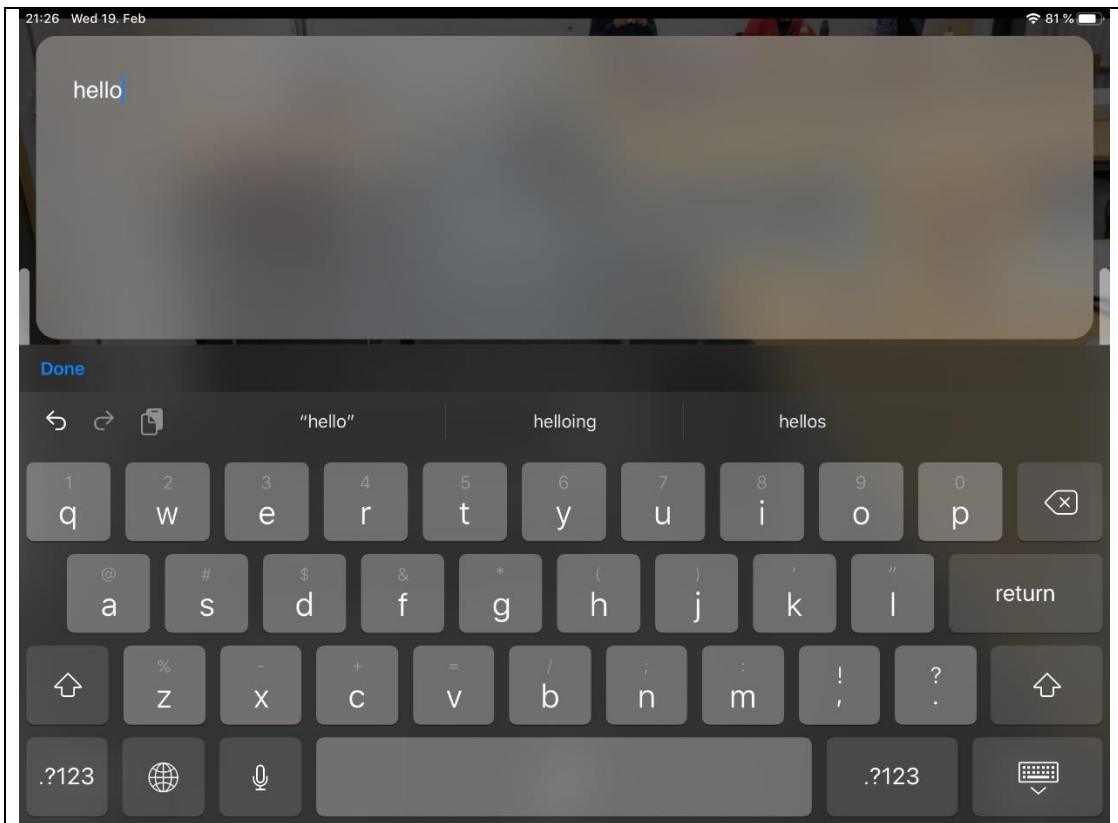


Fig. 8 By tapping on the annotation, the user can write down his thoughts.

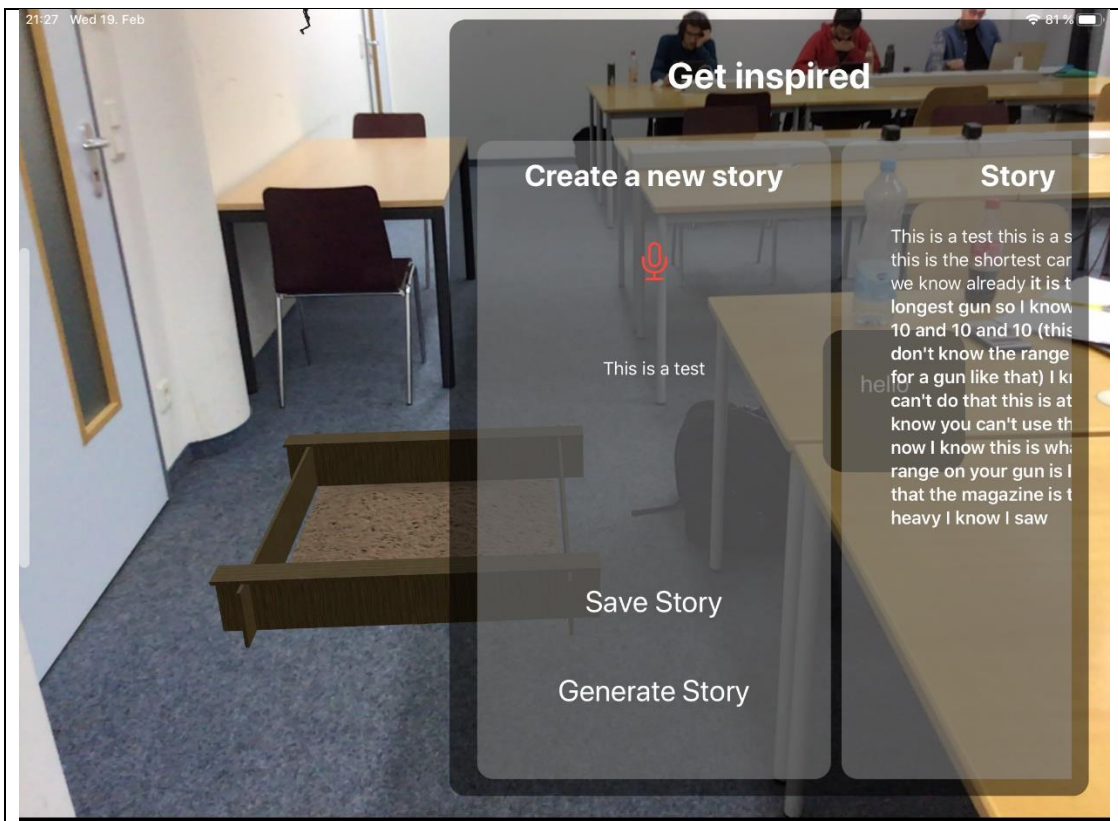


Fig. 9 By sliding to the left on the right side of the screen, the user can see the story generator. By tapping on the microphone, the user can record himself and then press the generate story button to create a new story.

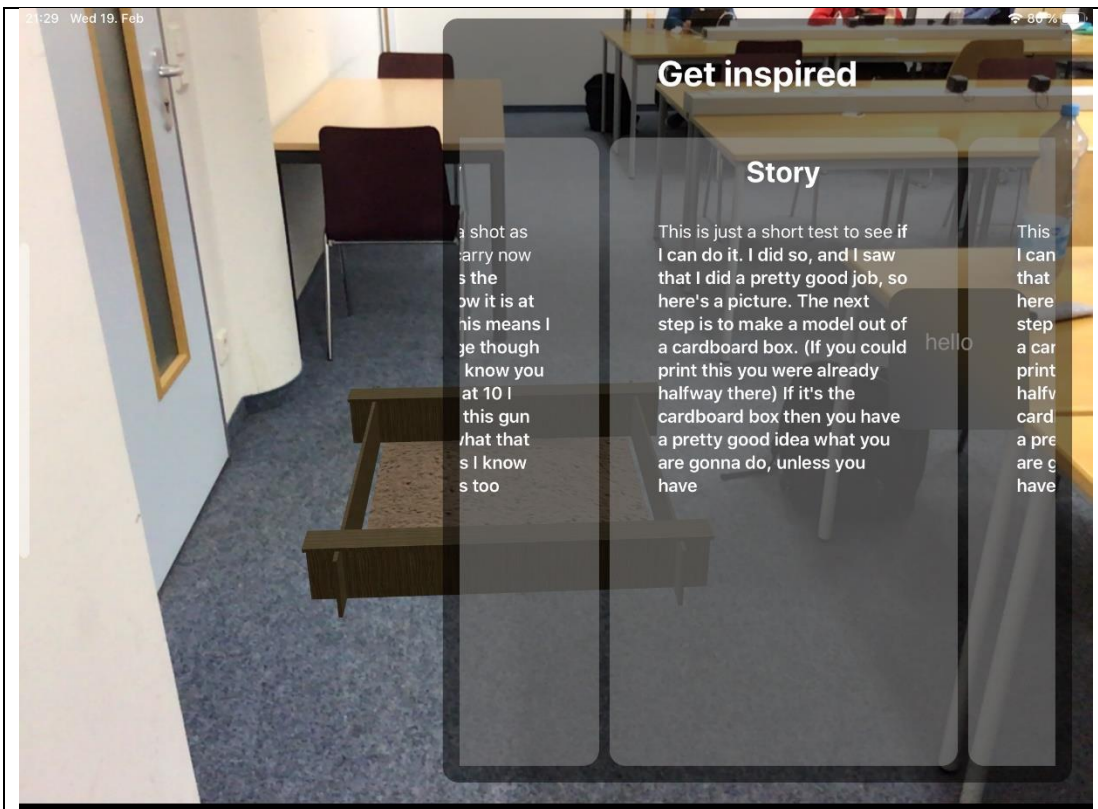


Fig. 10 The user will see the generated stories if he slides to the left in the story view. He will also see some auto-generated ones.



Fig. 11 If the user presses on the play button in the left corner of the screen, he can enter the simulation mode. By tapping on the screen, the user can add avatars to the scene. If he taps on the button again, he will stop the simulation. Avatars will disappear as he taps on the screen.

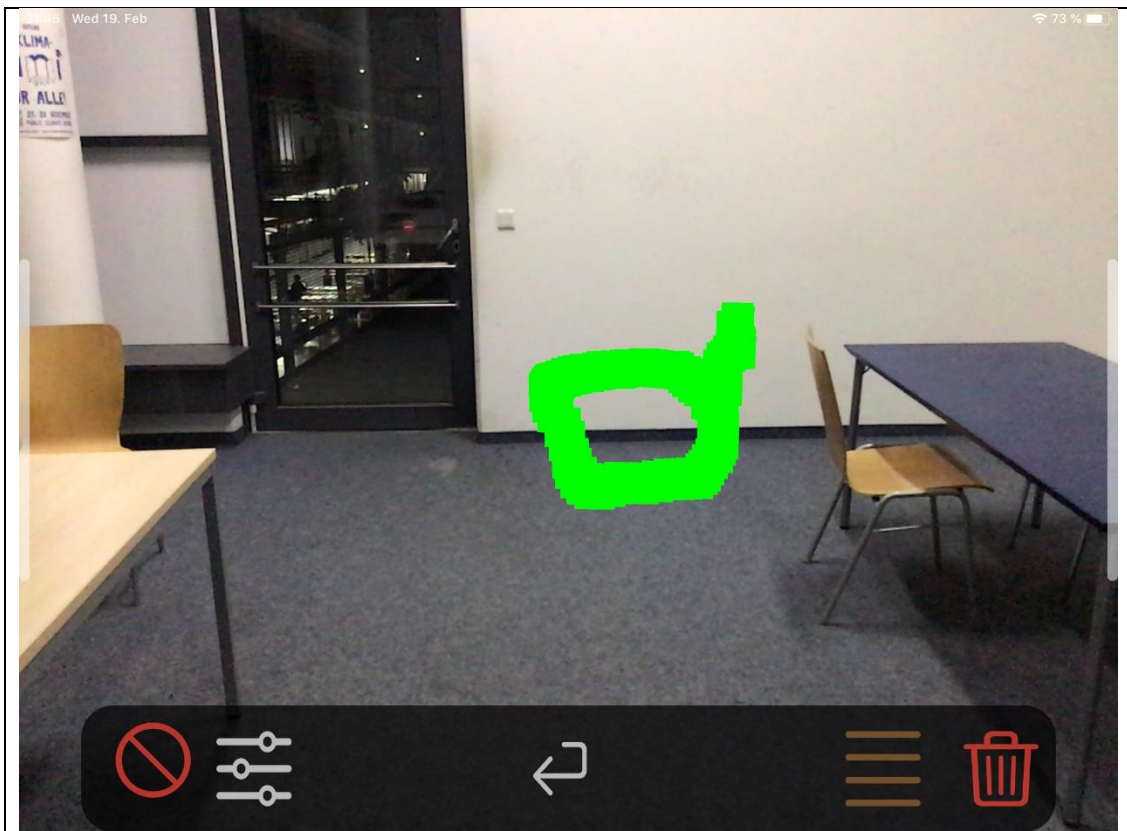


Fig. 12 If the user taps on the brush on the middle of the screen, he can start drawing in 3D. By tapping the icon on the left, the user can start drawing.

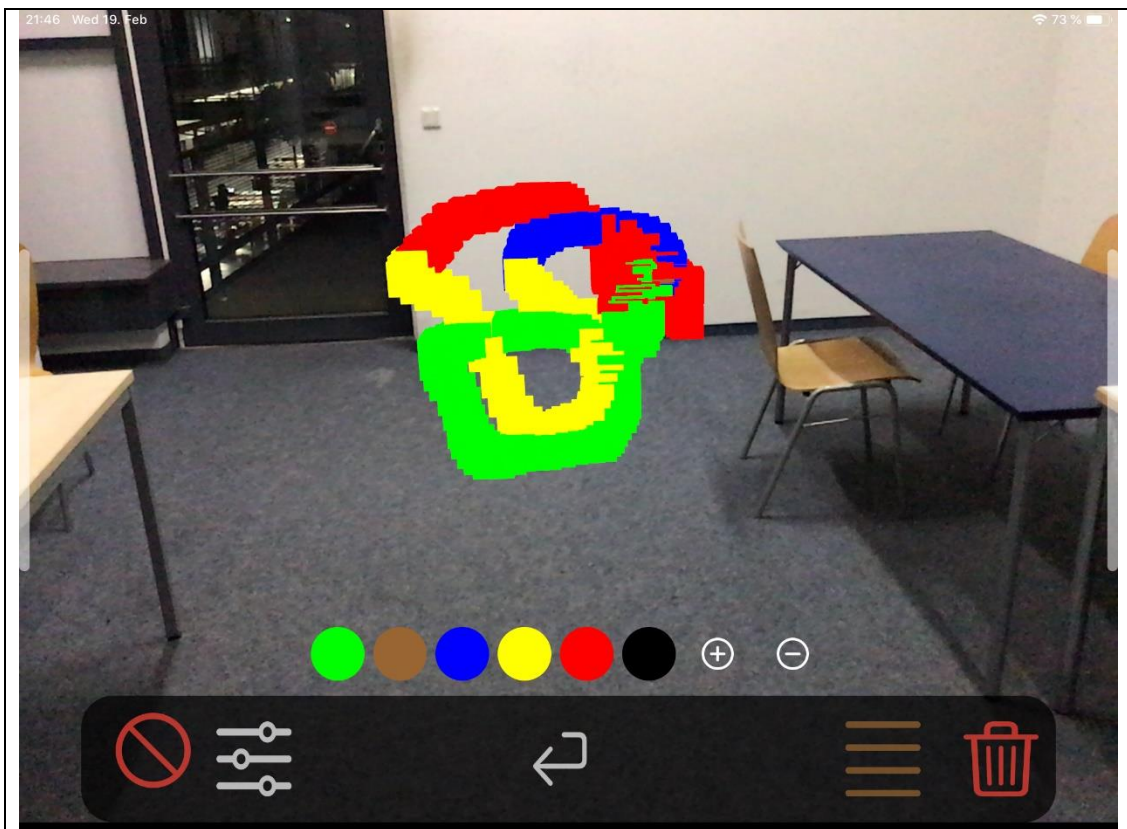


Fig. 13 By tapping on the second icon, the user can change the color of the brush. The plus and the minus icons can be used to adjust the brush size. The third icon closes the 3D drawing mode (see Fig. 14). By tapping on the fourth icon, the user can draw a 3D fence.

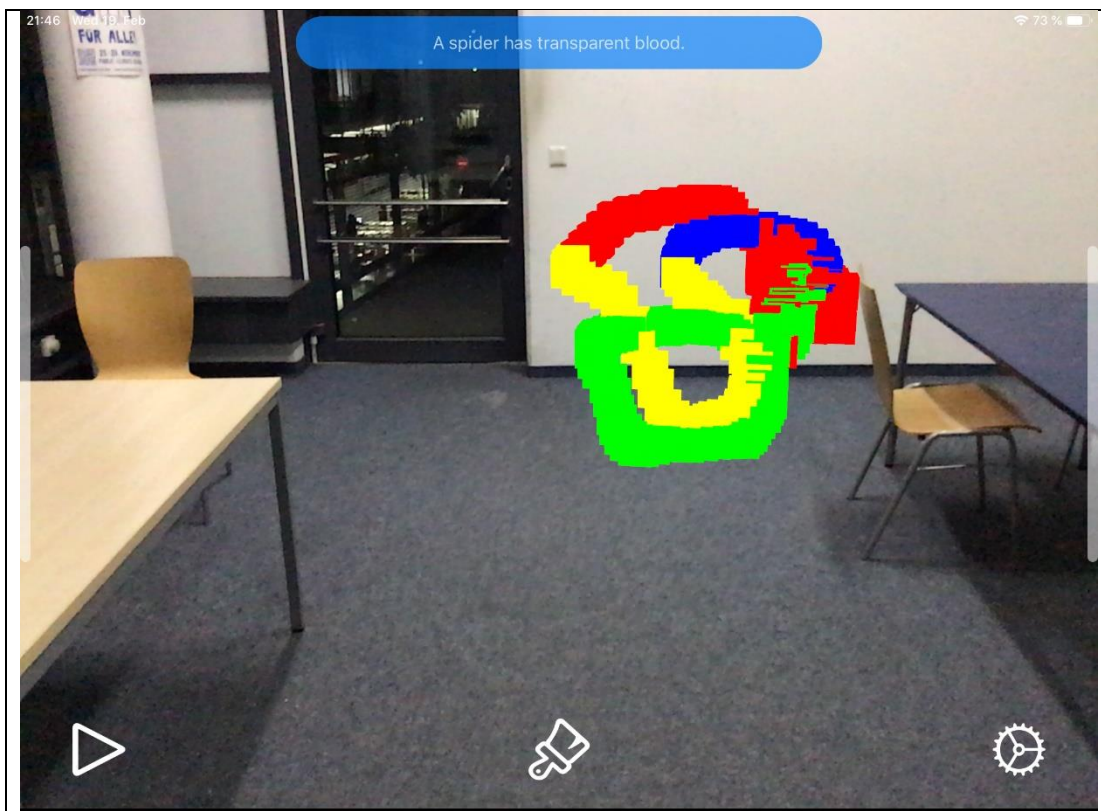


Fig. 14 The user can now do something else in the world since the 3D drawing mode is closed.

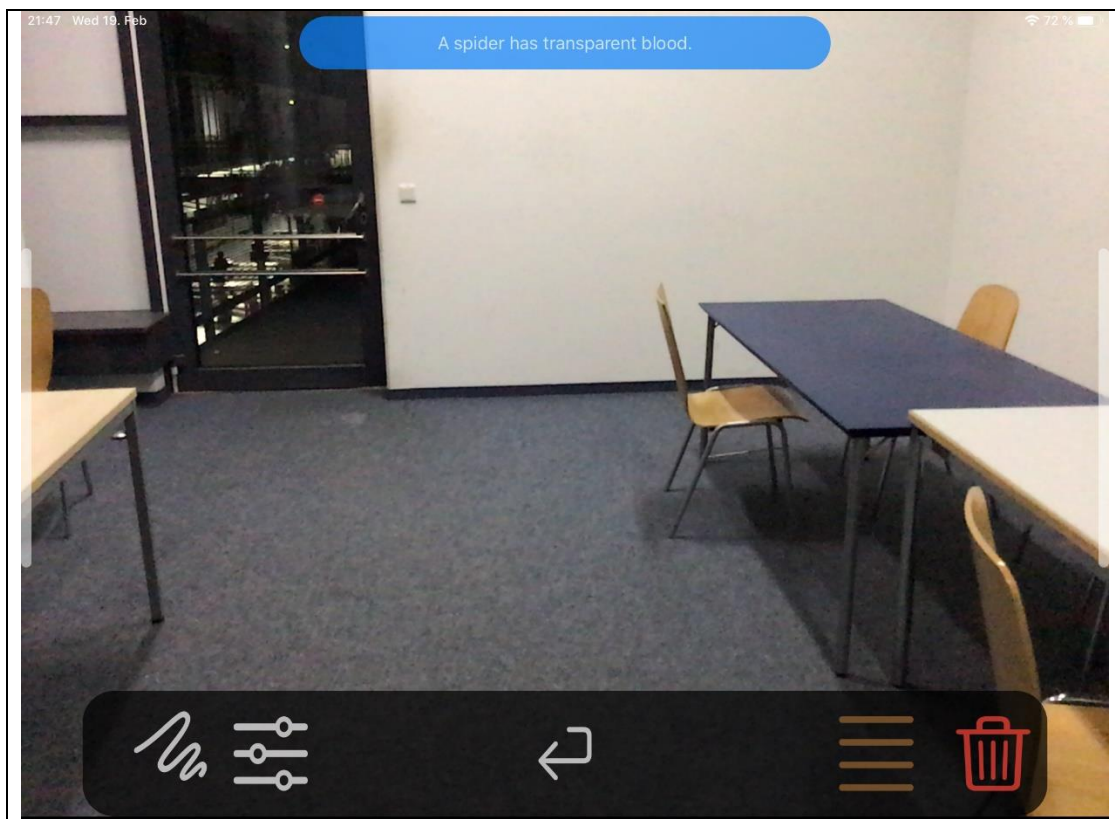


Fig. 15 By tapping on the bin icon, the whole drawing will be deleted.

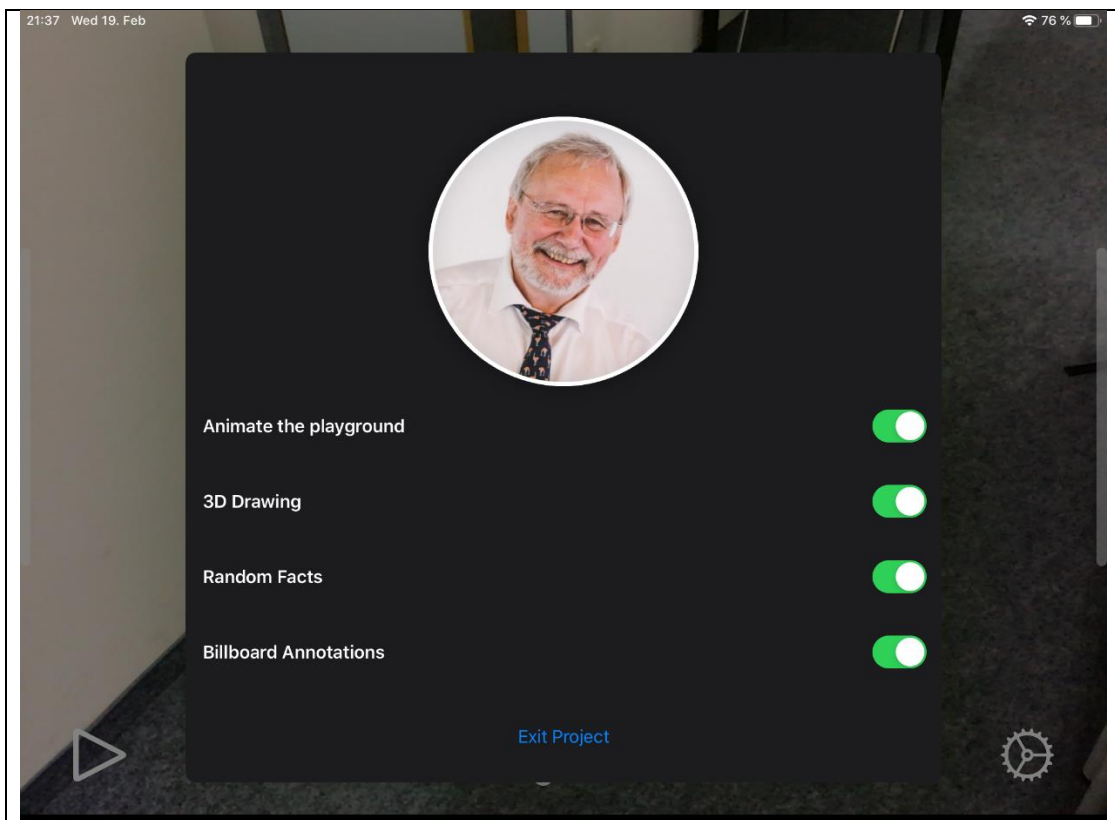


Fig. 16 The settings icon on the right corner of the screen will open a settings view, where the user can adjust if he wants to stop the simulation, the random facts, the 3D drawing or the annotations from appearing. He can also exit the project if he likes.

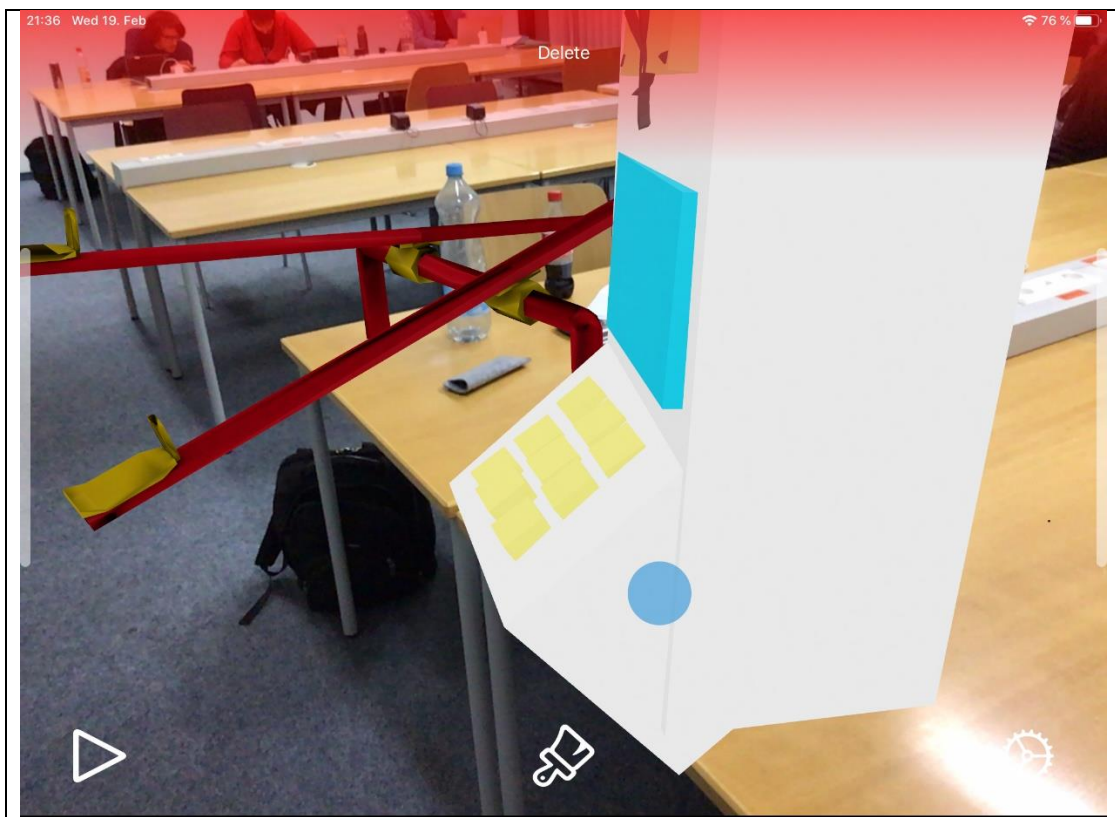
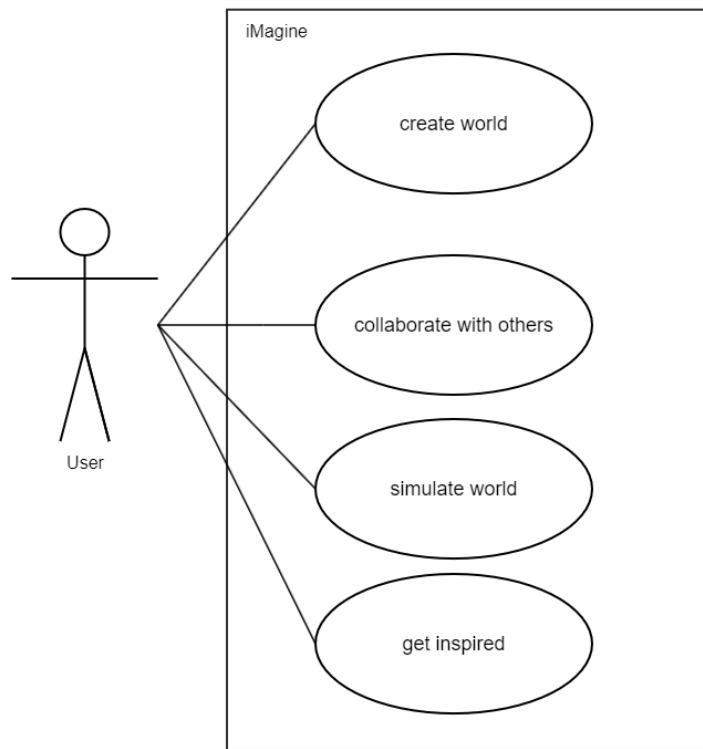


Fig. 17 The user can move objects by tapping on them or he can delete them by sliding his finger on the top of the screen after he tapped the item he wants to delete.

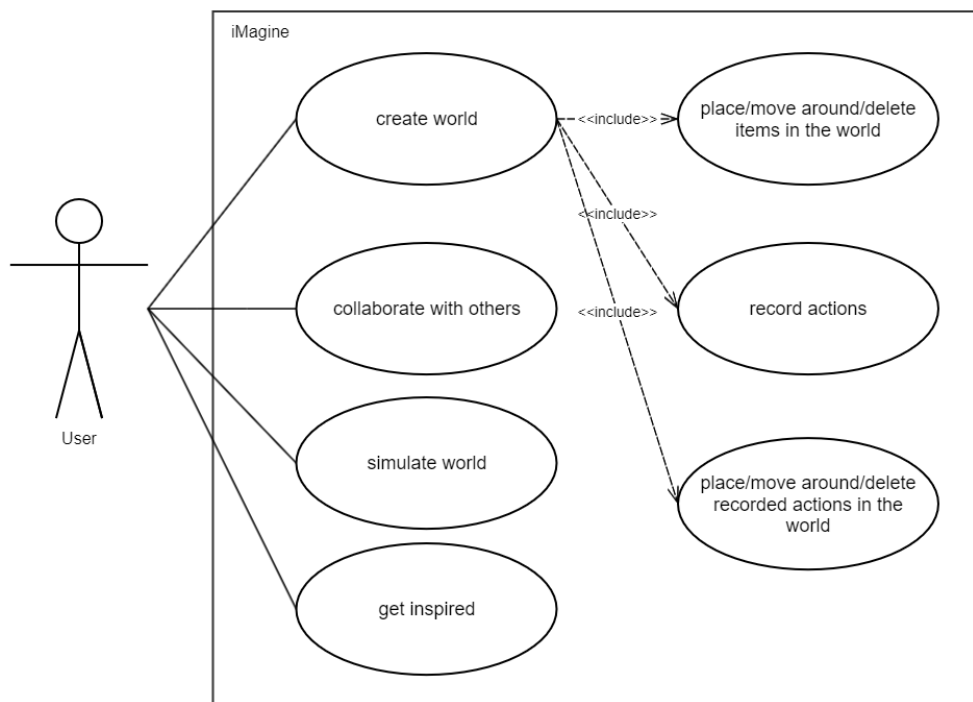
Terminology

Noun	Description
Designer/User	A person who can participate in creating a <i>Virtual Playground</i> . He can place <i>Playground Objects</i> , add <i>Annotations</i> , record <i>Actions</i> , create <i>Stories</i> or add <i>Simulation</i> .
Project	The main component the <i>User</i> interacts with. The user can create the <i>World</i> and generate a <i>Story</i> in the project.
World	An environment multiple <i>Users</i> work on together in order to create a <i>Virtual Playground</i> . Besides the <i>Virtual Playground</i> , the <i>World</i> also refers to the real objects that the <i>User</i> can see (e.g. a tree or grass). One <i>User</i> can work on multiple worlds. If more <i>Users</i> work on the same world, they will have a synchronized view of the world. The world may contain recorded <i>Actions</i> , <i>Annotations</i> , <i>Playground Objects</i> as well as <i>Simulation</i> .
Real Playground	An outdoor area designed for children equipped with recreation facilities (e.g. a swing). A playground is usually encountered in a park.
Virtual Playground	A part of the <i>World</i> , where multiple <i>Items</i> placed by the <i>User</i> form a <i>Real Playground</i> .
Item	Something that can be placed in the <i>World</i> by the <i>User</i> . They are of three types: <i>Playground Objects</i> , <i>Annotations</i> and <i>Actions</i> .
Playground Object	An <i>Item</i> that can be seen in a playground. The <i>User</i> can place the <i>Item</i> anywhere inside the <i>World</i> he is working on. He can add the <i>Item</i> multiple times, rotate or delete it.
Annotation	A note that can be added by the <i>User</i> anywhere in the <i>World</i> . The note can have the form of a plain text or a 3D drawing.
Story	A generated text for the <i>User</i> that can inspire him in editing the <i>World</i> . The <i>User</i> can either compose the story or he can use a auto-generated one. In order for the text to be composed, the user can use his voice to say a sentence related to the playground. The rest of the story will be then generated by the app.
Action	A real-world recording of a <i>User</i> while he does something related to the playground. The recording can be saved or deleted and is accessible to all other connected <i>Users</i> . It can be placed multiple times or deleted. Instead of showing the real person, the app will display an <i>Avatar</i> .
Simulation	An animation of the <i>World</i> . If the <i>User</i> chooses to simulate the <i>World</i> , multiple <i>Avatars</i> will start running around the <i>World</i> . The <i>Avatars</i> behave collaboratively, acknowledge the locations of the <i>Playground Objects</i> and don't collide with them.
Avatar	A virtual representation of a human in order to fit into the <i>World</i> .
Collaborative Context	A way for <i>Users</i> to collaborate with each other. If the <i>Users</i> decide to work together on the <i>Project</i> , they will be able to see every item placed by all other <i>Users</i> in the <i>World</i> .

Use Case Model



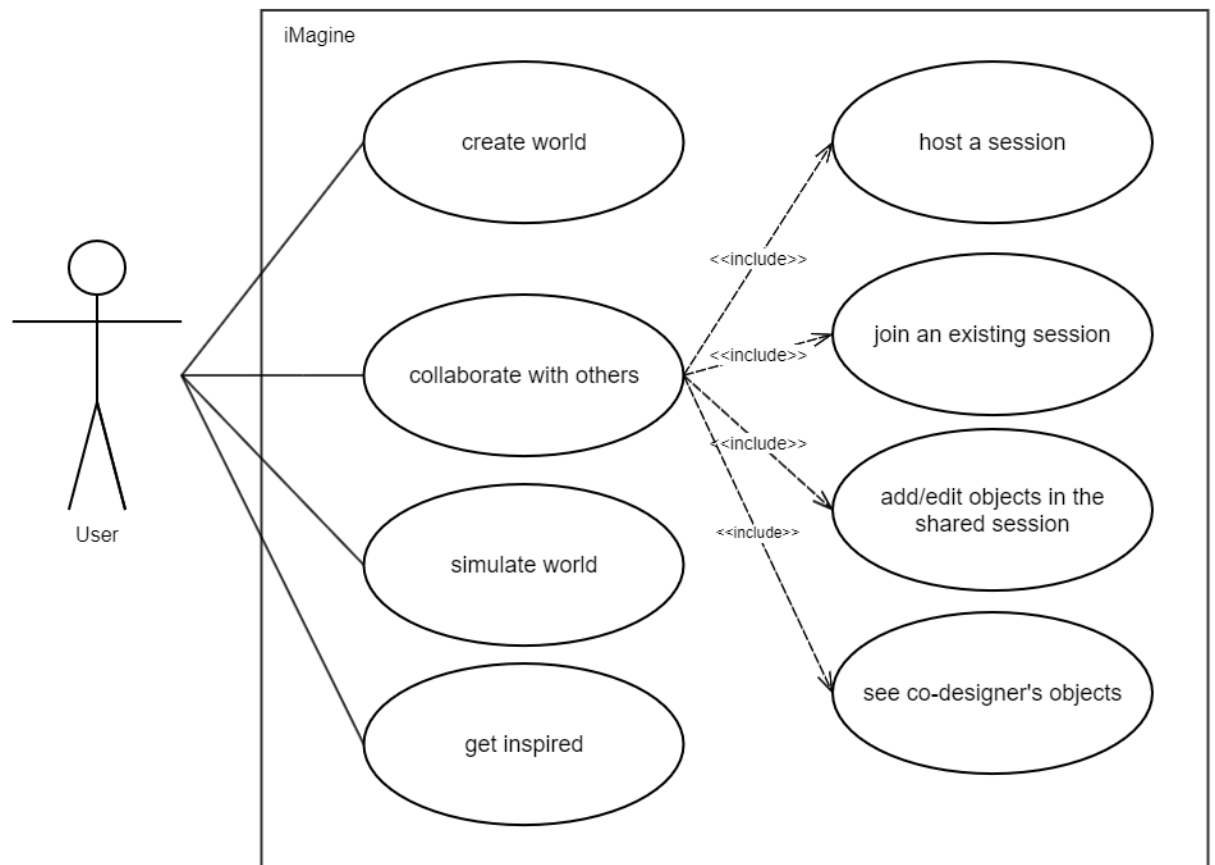
Our app consists of four major use cases: creating the world, collaborating with other users, simulating the world and getting inspired. Those use cases will be described in more detail in the following.



Let's take a closer look at the use case: create world.

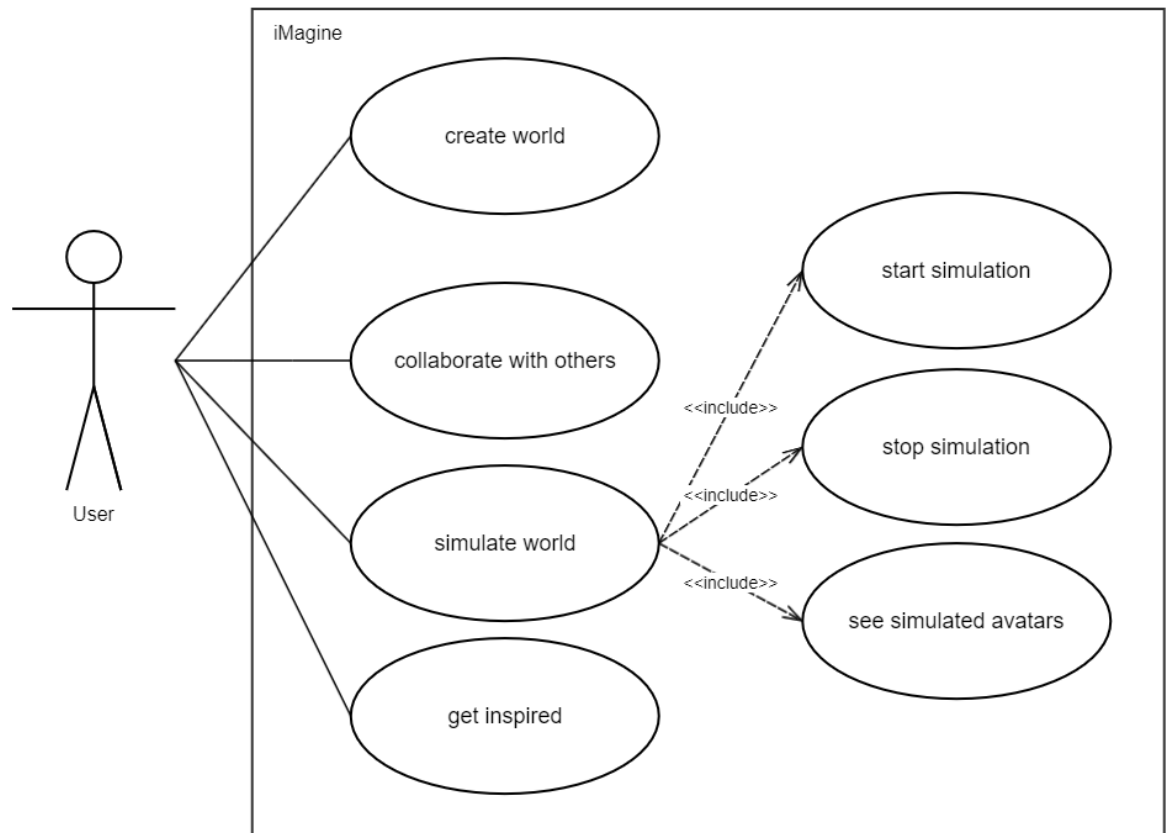
When interacting with our app, the user should be able to place items in the world, move them around and delete them. What those items are exactly will be described in the Analysis Object Model.

Another thing that the User might want to do is record actions, that he can also freely place, move around and delete.



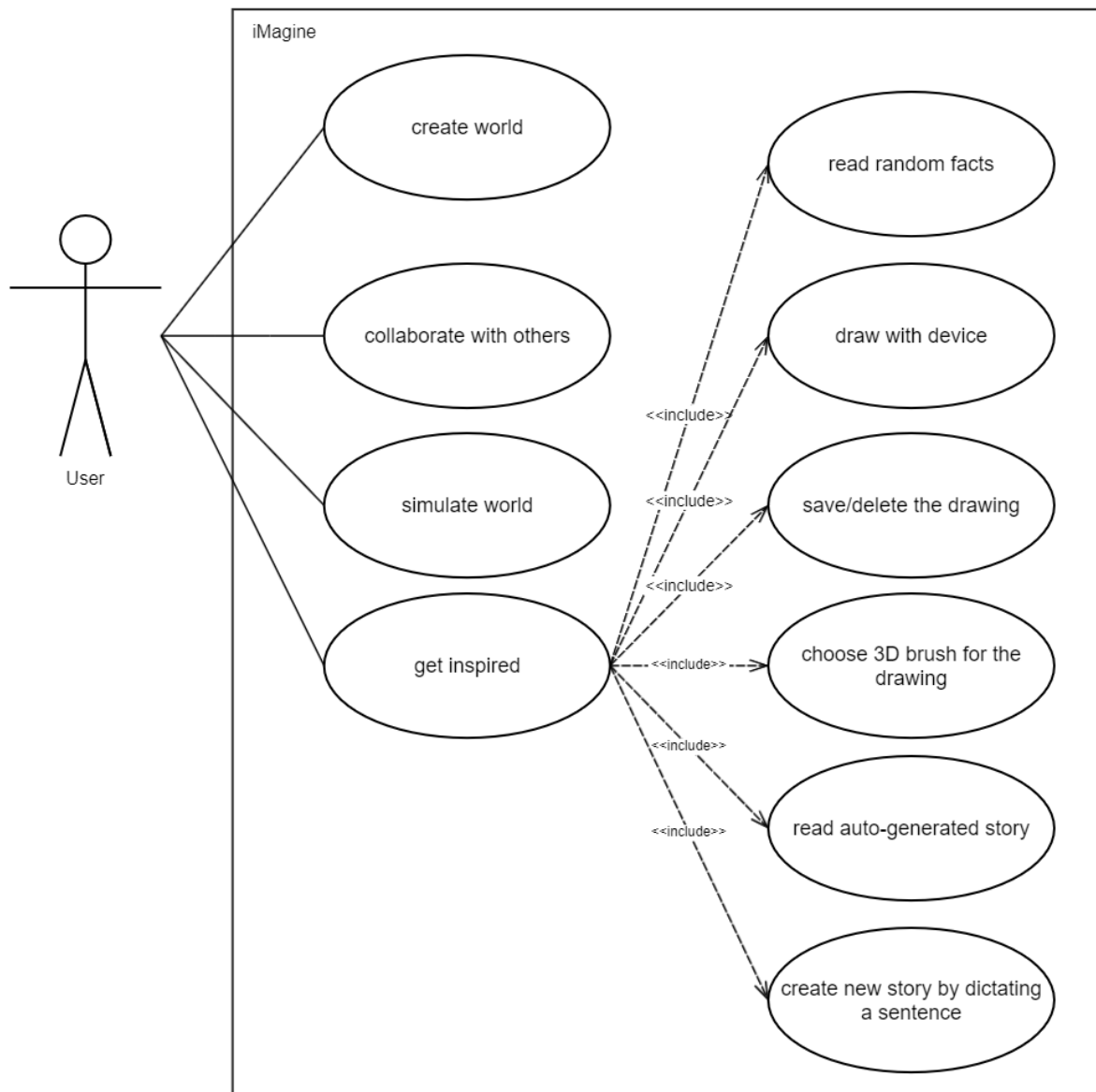
The second important use case in our app is collaboration with others.

The User can either host a session or join an existing one. He can freely add or edit objects that he or other users placed in the World and, therefore, he can also see everything that another User does in the World.



The third major use case is simulating the world.

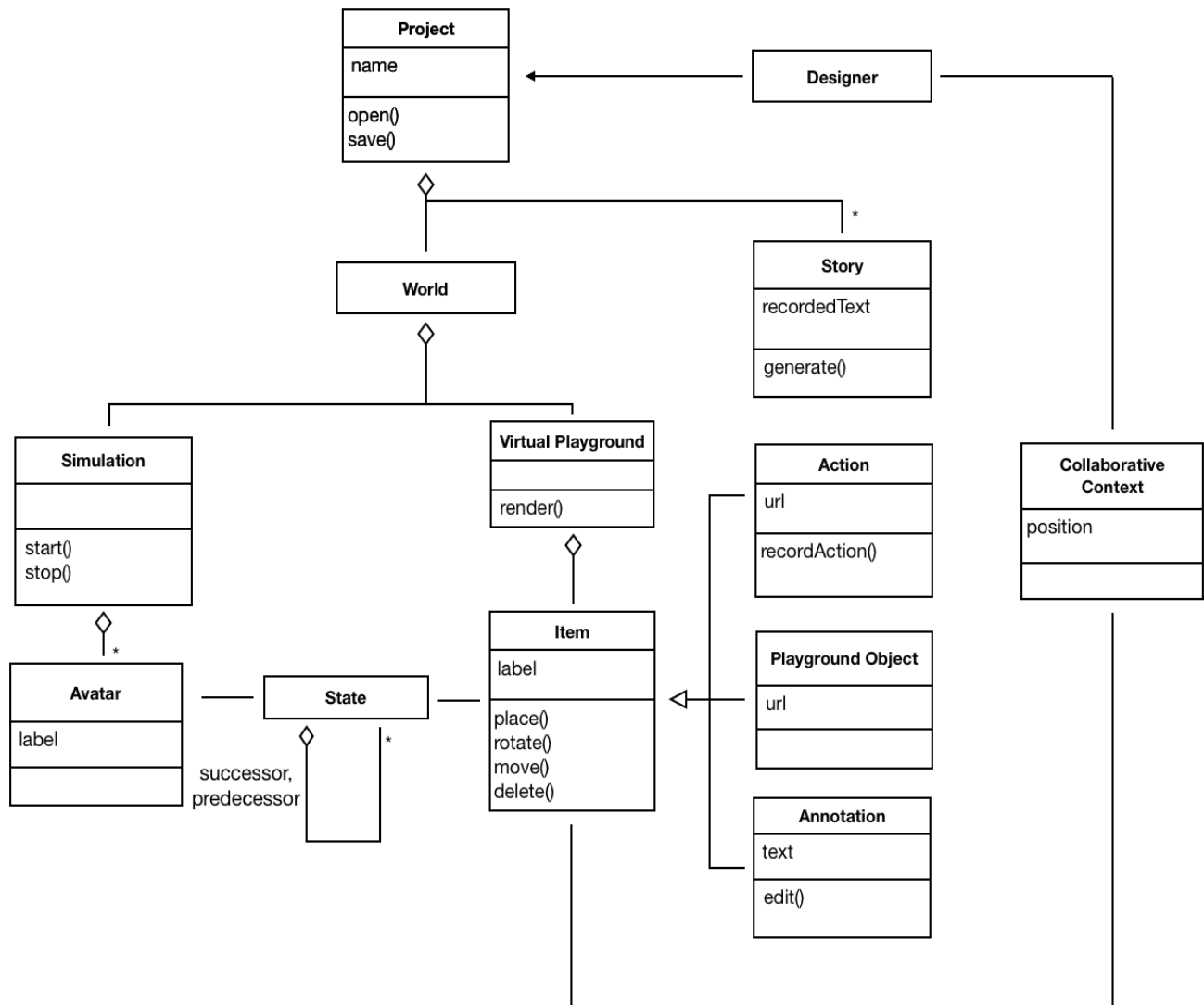
The User can start or stop a Simulation and he can see how the Avatars interact with the placed Items in the World.



The last important use case in our app is providing the User with some inspiration when editing the World.

The User can get inspired from three sources:

1. He can read some random facts that we provided in our app.
2. He can draw with his device directly in the World. He can choose between multiple colors of the 3d brush and he can also choose whether to save the drawing or delete it.
3. He can read and auto-generated Story, if he swipes to the Story view or he can create a new one. A Story will be generated if a User dictates a sentence related to the playground.



First of all, the app requires a Designer to use it. He can create a Project or load one that he saved earlier. Afterwards, the camera view will open and the user can see what surrounds him. The surroundings are modeled as the World. Inside the World there is the Simulation and the Playground.

The Playground has Items that can be placed inside the World. These Items are of three types: the Playground Objects, the Annotations and the Recorded Action. The Playground Objects represent the physical items that can be placed inside the Playground (e.g. a seesaw or a slide). The Annotations are of two types: text notes that a user can place as a 3D object, where he can type something that he finds useful or 3D drawings, where the user can draw something in the World by using his own device. The Recorded Action is a recording of a user whilst moving. After the recording is done, the action can be saved and then placed in the World.

The Simulation contains different Avatars, in this case the Avatars are children, which run around the Items and perform different actions (e.g. dancing, jumping, crying). It is part of the World, as it offers dynamics to it. The user can imagine how a real playground would look like.

Time sequences were important in our project and they are modeled with the State class between the Avatar class and the Item class. For each time step, we know the exact location of every Avatar, what the last object was, that he visited, and which will be the next one.

Our app supports collaboration between Designers. The Collaborative Context class models this aspect by persistently saving the location of the Items that were placed in the World and the location of the Designers. Therefore, each Designer can see what Items another Designer has placed and where he placed them.

Besides the World, there is also the Story. The Designer can generate a Story by telling a sentence related to the Playground. An AI running in the back will then autocomplete the Story for him, that he can use as an inspiration.

System Design

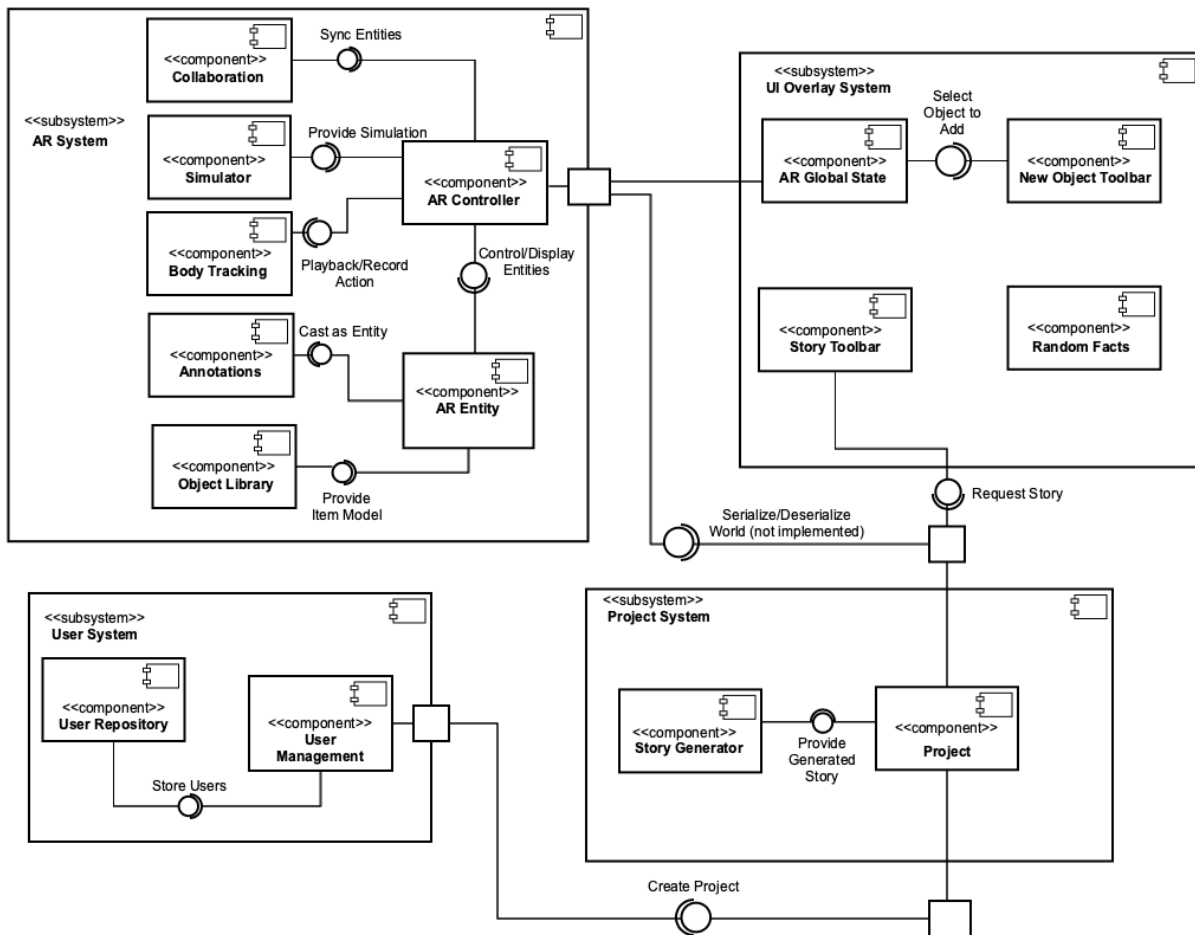
Overview

The user interface of our app is built using a combination of SwiftUI and UIKit. The root view of the main app window is a SwiftUI View "MainView", which is structured as a layered system ("ZStack"). The lowest layer is the camera view containing all AR objects. This view runs on UIKit. The layers that overlay this, including the toolbars and buttons, are written in SwiftUI. Coordination between those two subsystems is through a global state structure "ARGlobalState".

Our app takes advantage of the new technologies offered by ARKit 3, including World Tracking, Collaboration, and Body Tracking. The behavior of the AR layer is mostly controlled in the ViewController class "ARController", or the ARView class "WorldView".

The framework it uses to manipulate the scene is the new RealityKit framework, as opposed to SceneKit or SpriteKit. We have defined custom Entity and Component classes to suit our needs. In particular, transfer of data across a multipeer session is enabled by creating a Codable Component which contains the necessary information to load an Entity. This is then **automagically** synchronized by Apple's MultipeerConnectivityService across all peers.

Subsystem Decomposition



Hardware/Software Mapping

The iMagine app is a native iOS app. The UI is optimized for iPads but the app will run on an iPhone as well.

Note that only devices with an Apple A12 Bionic chip (2018 iPhones or later, 2019 iPads or later) will be able to run the body tracking features.

The server including the story generation component runs on an AWS server.

Persistent Data Management

We use a dockerized PostgreSQL server to store the data. We use migrations in order to maintain the single source of truthness of the database. That is why none of the tables are created manually but by migration scripts. These scripts are either written in Kotlin (`ios1920siemob-server/src/db/migration/`) or directly in SQL (`ios1920siemob-server/resources/db/migration/`). We use FlywayDB as a third party library to manage the migrations.

Access Control and Security

We use user-based access control in which every user can access only the entities that were generated by her/him. We use a password-based authentication in which the user receives a JWT signature that is valid for 10 hours. After 10 hours user needs to login again. The passwords are stored in the database not as plaintext but first, they are hashed using salted Bcrypt function.

Boundary Conditions

Server-side

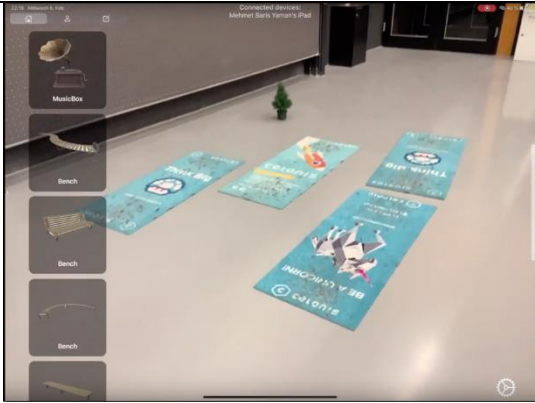
Ktor-Server

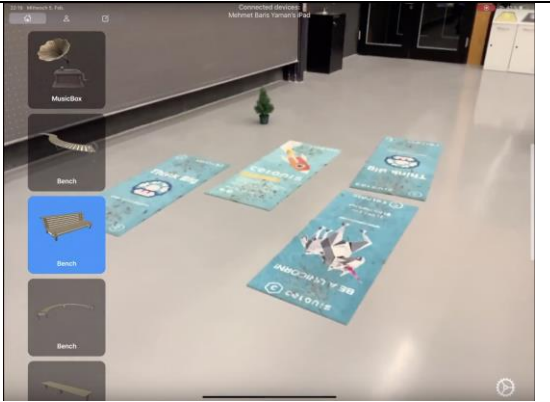
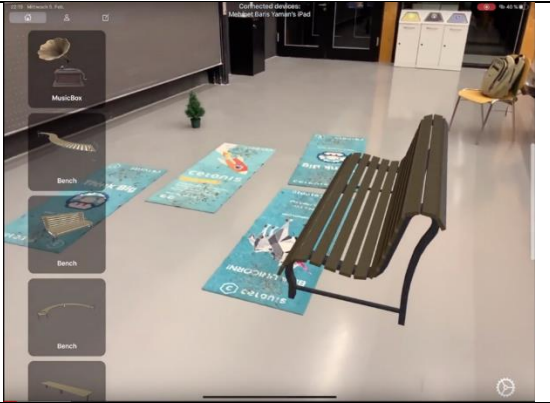
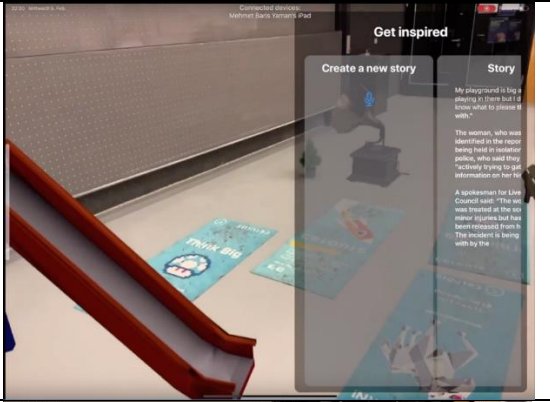
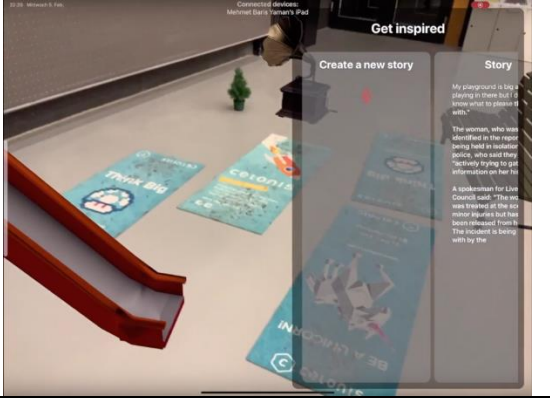

use the regular docker commands to restart/stop/start the docker containers.

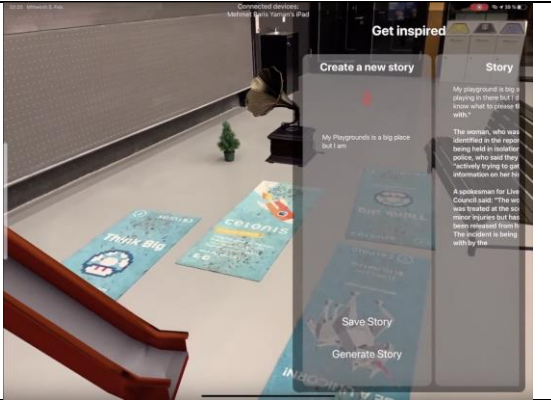
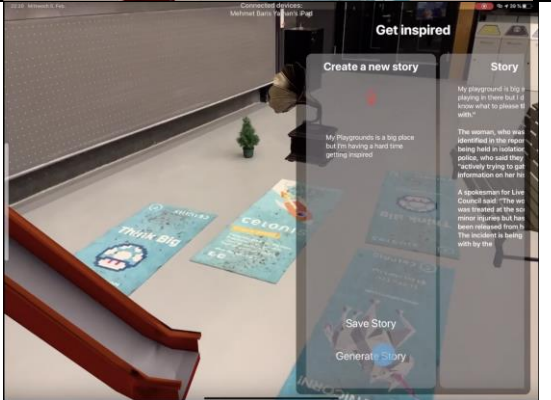
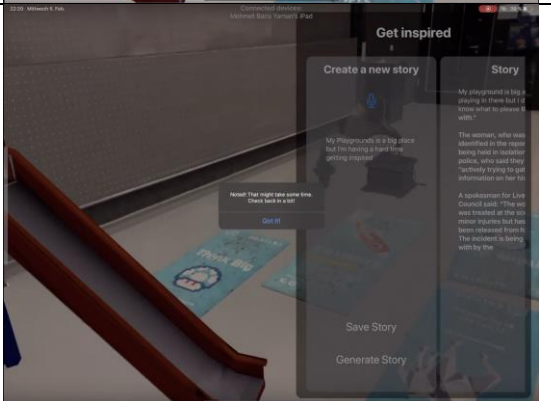
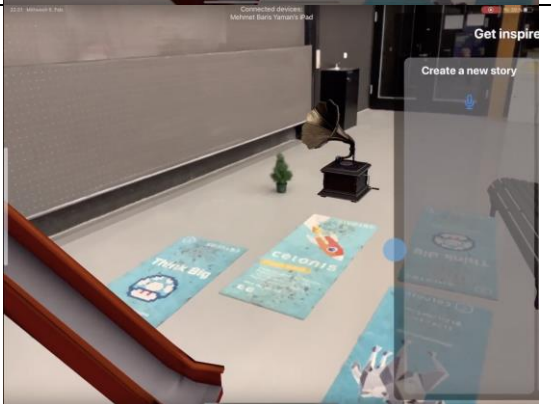
AWS-Server

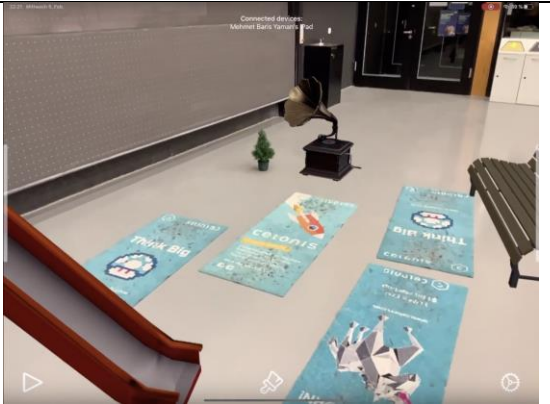

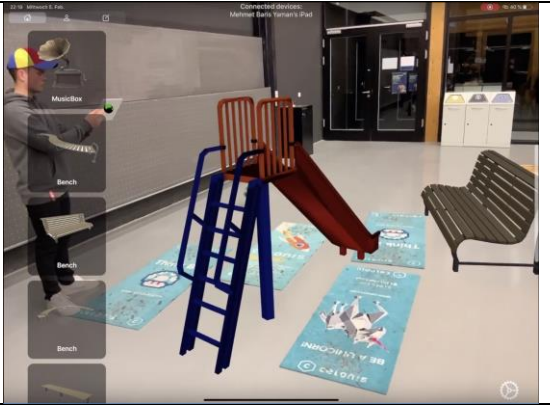
```
initctl stop text-generator
initctl start text-generator
initctl restart text-generator
service nginx stop
service nginx restart
service nginx start
```


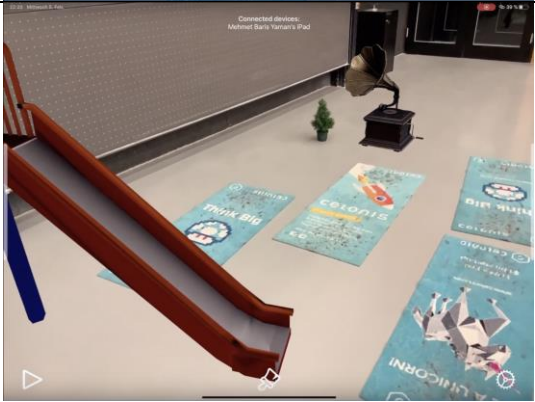


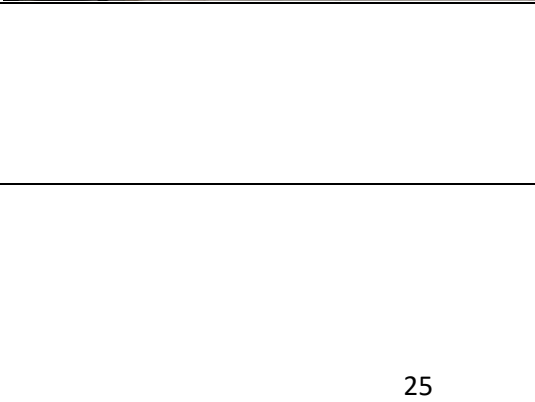
Demo Scenario



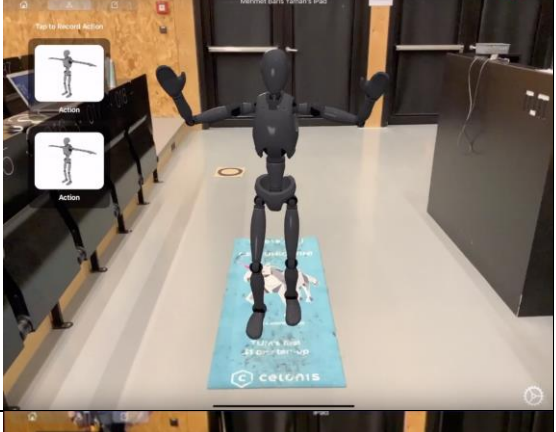

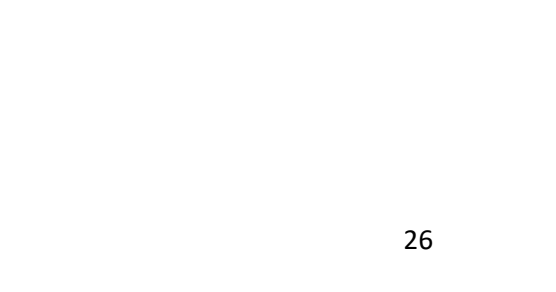
Scenario name	<u>creating a playground</u>		
Participating actor instances	<u>Richard Richman:Dad, Bobby:Son</u>		
Flow of events	1. Richard pulls in the left menu in the AR view.		
		2. The app shows the left menu where the user can select an object to place or record actions.	
	3. Richard taps on the Bench in the menu on the left.		


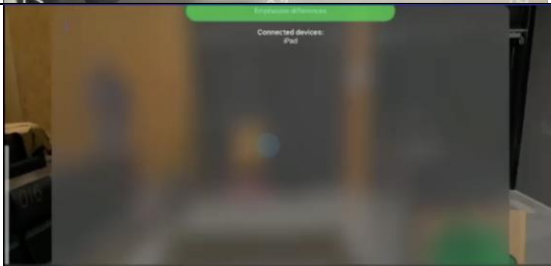


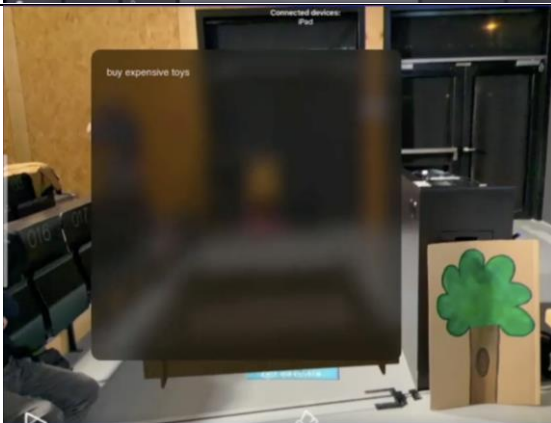

		4. The Bench gets selected which is shown in blue.	
	5. Richard taps in the AR View to place it.		
	6. The Bench gets placed in the AR world at the position where Richard has tapped.		
	7. Richard closes the left menu and pulls open the right menu.		
	8. The app shows the right menu where the user can create stories.		
	9. Richard taps on the microphone icon to input the beginning of a story by voice.		
	10. The microphone icon turns red to indicate that the app is listening.		
	11. Richard speaks into the microphone the story.		

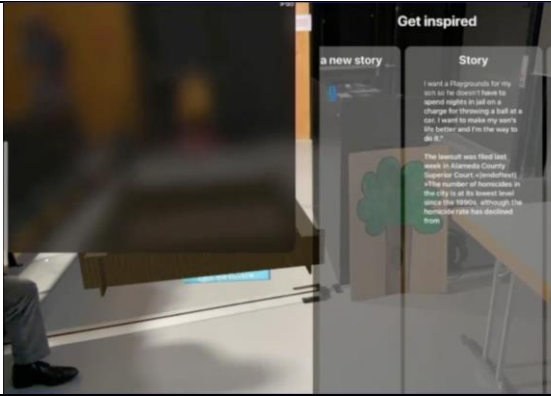
		12. Below the microphone icon, the text that is converted from the voice is shown.	
	13. Richard taps on the "Generate Story" button.		
		14. The microphone turns white again as it doesn't record anymore and a info is shown that it may take a while for the story to be generated.	
	15. Richard dismisses the info and swipes out the menu on the right.		

		16. The story generation menu disappears again.	
	17. Bobby starts the app on a second device, joins the collaborative session by clicking the rightmost button on the main menu. Richard and Bobby walk a few steps from left to right and hold their devices in the same orientation in order for the collaborative session to sync up.		
		18. The app shows on the top the name of the second iOS device that is connected to the collaborative session. On the second device, the objects also appear as soon as both devices sync up.	
	19. Bobby places a slide with the second device.		
		20. On both devices, the same state of the 3D world is shown. So, the slide appears also on the first device.	
	21. Bobby puts away the second iPad. Richard clicks the simulation button in the lower-left corner of the screen on the first iPad and taps at different points in the AR View on the screen		

		22. The simulation mode is activated and simulated agents are placed where the screen is tapped. Those agents wander around the placed objects and start dancing in front of the music object. Furthermore, sounds are played according to the actions.	
	23. Richard taps again on the simulation button in the lower-left corner.		
		24. The simulation mode is deactivated and all sounds stop again.	
	25. Richard again swipes in the left sidebar and clicks on the second tab.		
		26. The left menu opens again and the second tab shows the option to record an action and below that all previously recorded actions that can be placed	
	27. Bobby stands in front of the camera (in the AR View) and Richard taps on the record action button.		

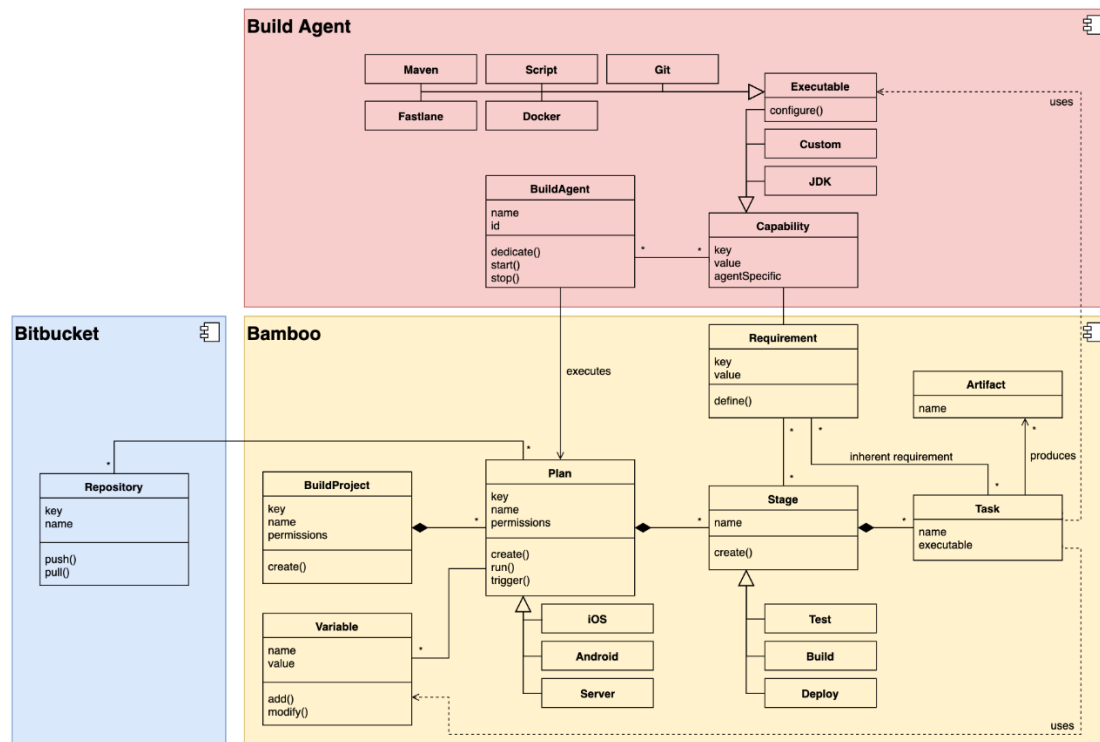
		28. The app starts recording the action which is shown to the user by the text turning red and an avatar mimicking the movements that the person in front of the camera is doing.	
	29. Richard taps again on the recording actions button.		
		30. The recording action mode is stopped and therefore the button turns white again, the avatar disappears from the AR View. And after a short time, the action shows up below the button.	
	31. Richard taps on the newly created action in the left menu. Then he taps on the screen to place it and adjusts the size and location.		
		32. The action shows up in the AR view and plays back in a loop.	
	33. Richard taps on the creating an annotation button in the sidebar. And after that taps in the AR view to place the annotation.		

		34. The annotation appears in the AR view as a black, semi-transparent, squared billboard.	
	35. Richard taps on the annotation.		
		36. The annotation zooms into the screen and a keyboard appears at the bottom of the screen.	
	37. Richard types in a text.		
		38. The text appears in the annotation.	
	39. Richard hits the enter button.		
		40. The keyboard disappears and the annotation zooms out again into the AR world.	
	41. Richard swipes in the right menu.		

		42. The story view appears on the right side and the story that was dictated early is shown together with the autocompletion.	
Entry conditions	<ul style="list-style-type: none"> • Siri or dictation needs to be enabled on the device for the microphone in the story view to work • the app needs to have permission for the microphone to be used (for the story) • the environment should be scanned by moving the device around as well as taking a few steps through the environment 		
Exit conditions	<ul style="list-style-type: none"> • the objects should be visible in AR as well as the story being generated 		
Quality Requirements	<ul style="list-style-type: none"> • the collaboration should feel instantaneous (real-time) • the voice recognition in the story view should be in real-time 		

Administrator Manual

Infrastructure Setup up

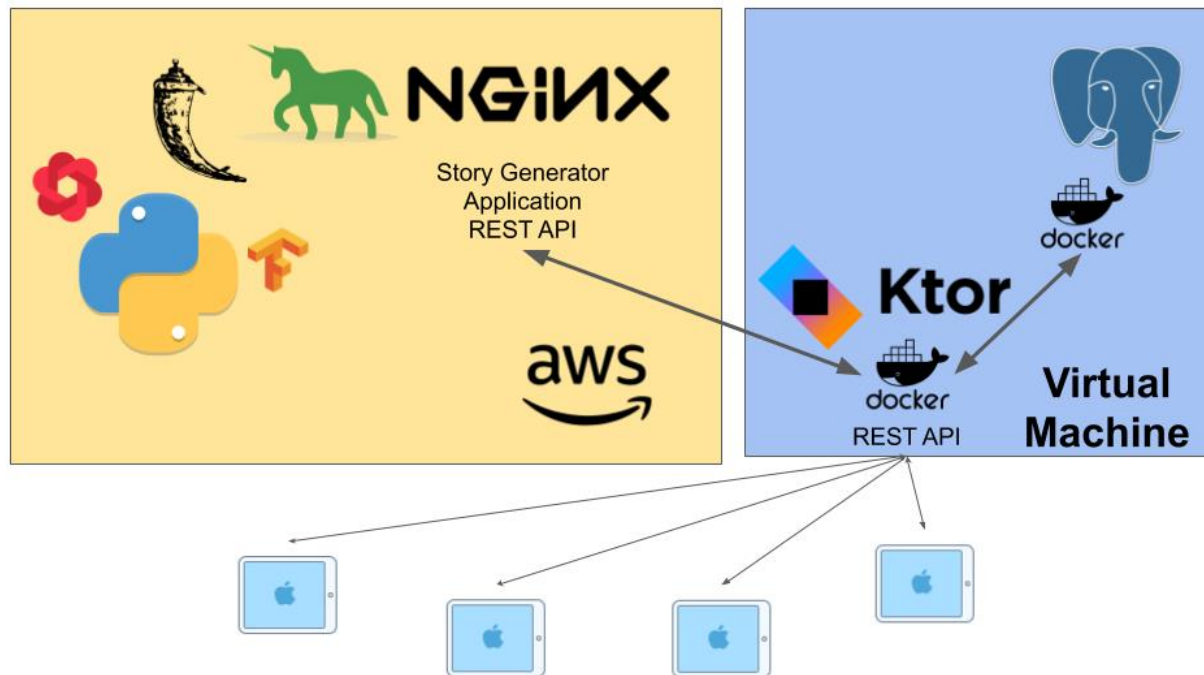


- **Continuous integration & Continuous delivery:** Bamboo CI
- **Continuous Prototyping:** Prototyper: allows developers and designers to deliver mockups, mobile applications as well as a mixture of both using the same deployment pipeline

Client-side:

- **CocoaPods** - CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects.
- **Fastlane** - It lets you automate every aspect of your development and release workflow.

Server-side:



The infrastructure contains two servers with separate REST APIs.

The main backend server is used by clients directly and uses Ktor Web Framework and runs with Kotlin in JVM. It is an Apache Server. It uses a Postgres SQL Server for persistent data management. The main backend server and Postgres SQL Server are dockerized and running on a VM machine. The main backend server uses port 80 for the REST API. That is why the VM machine must be publicly accessible via port 80. JWT based authentication is used for the main backend server.

The Story Generation Application is an Nginx server that uses Gunicorn combined with flask and runs with Python. We actually have two services here: the first one is a local service provided by Gunicorn. The second one is the public Nginx service which proxies the Gunicorn service. These are running on an AWS server because the stories are generated using a pretrained deep learning model provided by OpenAI for which we use Tensorflow. The story generation process needs many resources which is why we can't use any VM for this but a server with GPU. For the REST API, port 80 is used for this case too. The Story Generation Application is not used directly by clients but by the main backend server. To this end, password-based authentication is used. The application.conf file of the main backend application should be updated according to the IP address of AWS instance. eg ``http://<ip>/generate-story``

Deployment and Configuration

Bamboo Setup

Client:

Create and set up the following Tasks:

Source Code Checkout - Clone Repo

Execute Fastlane using the following Script:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
fastlane bamboo
```


Execute keychain using the following script:

```
### BEGIN LockKeychain ###
# Reset default keychain to login.keychain
security lock-keychain "$bamboo_KeyChain"
security default-keychain -d user -s "login.keychain"
### END LockKeychain ###
```

Server:

1- Build and Push Image

Create and set up the following Tasks:

Source Code Checkout - Checkout default Repository

Run docker build using following script:

```
repositoryName="${bamboo_buildResultKey%%-*}"
repositoryNameLowerCase=$(echo $repositoryName | tr '[:upper:]' '[:lower:]')
planNameLowerCase=$(echo $bamboo_shortPlanKey | tr '[:upper:]' '[:lower:]')
imagename="$bamboo_DOCKER_REPOSITORY_HOST/$repositoryNameLowerCase/$planNameLowerCase:latest"
docker build --no-cache --force-rm -t $imagename .
```

Run docker push using following script:

```
docker login --username $bamboo_DOCKER_REPOSITORY_USERNAME --password
$bamboo_DOCKER_REPOSITORY_PASSWORD $bamboo_DOCKER_REPOSITORY_HOST
repositoryName="${bamboo_buildResultKey%%-*}" repositoryNameLowerCase=$(echo
$repositoryName | tr '[:upper:]' '[:lower:]')
planNameLowerCase=$(echo $bamboo_shortPlanKey | tr '[:upper:]' '[:lower:]')
#imagename="$bamboo_DOCKER_REPOSITORY_HOST/$repositoryNameLowerCase/$planNameLo
werCase:$bamboo_buildNumber"
imagename="$bamboo_DOCKER_REPOSITORY_HOST/$repositoryNameLowerCase/$planNameLow
erCase:latest" docker push $imagename
echo "#####
#####"
echo "IMAGE: $imagename"
echo "#####
#####"
echo "$bamboo_DOCKER_REPOSITORY_HOST/$repositoryNameLowerCase/$planNameLowerCase:l
atest" > BuildString.txt
```

2- Deploy to VM

Source Code Checkout - Checkout default Repository

SCP Task - Copy docker-compose to VM using SCP with public key authentication. Use the SSH keys of the VM (Port 22)

SSH Task - Pull and Execute the following command using SSH with public key authentication. Use the SSH keys of the VM (Port 22)

```
cd /opt/docker && sudo docker-compose pull && sudo docker-compose up -d
```

Servers

Main Backend Server with PostgreSQL Server

We can deploy the servers using bamboo as well as manually. In bamboo, simply trigger a deploy which will execute a docker-up in the VM

For manual deployment, use the Dockerfile provided in the repository which automatically runs both servers. No further steps needed.

AWS Server

This part is a bit complicated so let's do it step by step. We will use the files in the folder ``story-generator-files`` in the following steps. Assumed that the AWS instance uses TensorFlow version 1.15.0.

1. Upload the files ``controller.py``, ``download_model.py``, ``encoder.py``, ``interactive_conditional_samples.py``, ``sample.py``, ``requirements.txt`` and ``wsgi.py`` to the AWS instance in a separate folder named ``text-generator``.
2. Execute

```
python download_model.py 124M
python download_model.py 355M
python download_model.py 774M
python download_model.py 1558M
export PYTHONIOENCODING=UTF-8 pip install -r requirements.txt
```

It might take a while to download the models.

1. Put the ``nginx.conf`` file into the ``/etc/nginx/`` directory. This is the configuration of Nginx server
2. Put the ``text-generator.conf`` file into ``/etc/init/`` directory. This will configure the Gunicorn service as Upstart.
3. Execute

```
initctl reload-configuration
initctl start text-generator
service nginx start
```

Third-party Components

- Kotlin Version 1.3.50 – <https://kotlinlang.org>
- Ktor Version 1.2.4 – <https://ktor.io>
- NGINX Version 1.16.1 – <https://www.nginx.com>
- Gunicorn Version 20.0.4 - <https://gunicorn.org/>
- Flask Version 1.0.2 – <https://palletsprojects.com/p/flask/>
- TensorFlow Version 1.15.0 – <https://www.tensorflow.org>
- Docker Version 19.03.4 – <https://www.docker.com>
- Docker-Compose Version 1.24.1- <https://docs.docker.com/compose/>
- FlywayDB Version 5.2.4 – <https://flywaydb.org>
- Postgres Version 12.1 – <https://www.postgresql.org>
- Fastlane – <https://fastlane.tools>
- Cocoapods – <https://cocoapods.org>
- GPT-2 Model - <https://github.com/openai/gpt-2>
- Python Version 3.6.5 - <https://www.python.org/>