**Курсовая работа. JavaScript в браузере**

**Оглавление:**

[Состав проекта](https://www.notion.so/JavaScript-6127e88e9cb44c45b294f91ba0af5bda)

[Список экранов проекта](https://www.notion.so/JavaScript-6127e88e9cb44c45b294f91ba0af5bda)

[API «Камень, ножницы, бумага»](https://www.notion.so/JavaScript-6127e88e9cb44c45b294f91ba0af5bda)

[Основной сценарий](https://www.notion.so/JavaScript-6127e88e9cb44c45b294f91ba0af5bda)

[Архитектура приложения](https://www.notion.so/JavaScript-6127e88e9cb44c45b294f91ba0af5bda)

[Состав приложения](https://www.notion.so/JavaScript-6127e88e9cb44c45b294f91ba0af5bda)

**Игра «Камень, ножницы, бумага»**

Финальный результат этого задания задачи - реализация браузерную игру «камень, ножницы, бумага»

**Состав проекта**

Проект состоит из двух частей: фронтенд и бэкенд.

**Фронтенд-часть**

На старте вам предстоит выбрать — сделать десктопный или мобильный интерфейс.

Состоит из набора статических файлов, таким образом, приложение должно работать с помощью команды npx http-server.

Архитектурно рекомендуется выбрать концепцию SPA (Single Page Application) — когда приложение загружается по ссылке [<http://localhost:8080/>](<http://localhost:8080/>) и все переходы внутри осуществятся уже за счет JavaScript.

**Бэкенд**

Бекэнд часть можно взять здесь: <https://skypro-rock-scissors-paper.herokuapp.com/ping> (по ссылке будет ошибка, это нормально)

Данная ссылка - набор URL, работающих через GET запросы.

<aside> ❗ **Подсказка**

Создаем переменную

const backURL = '<https://skypro-rock-scissors-paper.herokuapp.com/>';

Используем в формате

**Пример:**

url: `${backURL}login`,

</aside>

**Основные возможности бэкенда**

**Запрос на авторизацию/аутентификацию**. Паролей не требуется, только логин. В ответ сервер отдает токен, который затем потребуется использовать во всех других запросах, чтобы можно было отличить, какой пользователь делает запрос. Если запросы в другие ручки делать без токена, они будут отвечать ошибкой. Если пользователя с таким логином не существует, он будет создан.

**Запрос на текущий статус.** Если пользователь с таким токеном в игре (например, если пользователь вошел через другой браузер) — бэкенд отдаст данные про игру. Если нет — статус будет «в лобби».

**Запрос на начало игры**. В ответ бэкенд либо создаст «игру», либо подключит пользователя к существующей (если есть игра, ожидающая соперника).

**Запрос на статус игры**. Статус игры может быть: игра ожидает соперника, игра ожидает вашего хода, игра в ожидании хода соперника, игра завершилась вашей победой, игра завершилась вашим поражением.

**Запрос на ход в игре**. Передаем ход в игре и узнаем результат этого хода. Если соперник еще не ответил, то далее результат хода нужно будет узнавать через ручку статуса.

**Запрос на список пользователей в игре**. Выдает список активных пользователей.

**Список экранов проекта**

Визуальная схема принципа работы игры: <https://miro.com/app/board/o9J_luzmO6k=/>

Не воспринимайте эту схему как макеты. Это скорее прототип того, как должно работать. Также не стоит воспринимать это как намек на то, что следует делать под мобильную версию.

**Экран авторизации**

Экран, где предлагается ввести свой логин и нажать кнопку «Войти». После нажатия кнопки, когда сервер ответил с токеном, должен происходить переход в экран лобби.

**Экран лобби**

Экран, где есть список всех онлайн-пользователей и кнопка «Начать игру».

**Экран ожидания игры**

Если пока никто не создал игру, на этом экране ждем соперника.

**Экран игры**

Когда в игре есть два соперника или после того, как оба сделали одинаковый ход (например, оба выбрали ножницы), рисуем экран выбора хода.

**Экран ожидания хода соперника**

Если игрок сделал ход, а соперник еще нет — рисуем экран ожидания хода соперника.

**Экран исхода игры**

Как только игроки сделали разные ходы — кто-то из них победил, а кто-то проиграл, рисуем соответствующий экран победы или поражения. Из него можно сразу создать игру или перейти в лобби.

**API «Камень, ножницы, бумага»**

Во всех ручках «Параметры запроса» — это GET-параметры (то есть данные передаются в URL запроса после ?).

Все ручки дают ответ в формате JSON (application/json).

Все ручки всегда отвечают HTTP-статусом 200 OK. Статус операции (успех/неуспех) передается в теле ответа, в поле status.

**GET /ping**

Позволяет проверить (например, через браузер), что бэкенд работает.

Всегда отвечает:

{

"status": "ok",

"message": "pong"

}

Откройте в браузере http://<домен бэкенда>/ping и убедитесь, что вы видите указанный ответ.

**GET /login**

Позволяет зарегистрироваться/войти. Принимает только логин. В ответ сервер отдаст токен, который нужно будет передавать во все другие ручки API.

Параметры запроса:

login: никнейм, под которым пользователь регистрируется/входит.

* Примеры:
* Запрос:
* /login?login=Игрок
* Ответ:
* {
* "status": "ok",
* "token": "89580e35-5424-4286-8e45-b4d818278367"
* }
* Запрос:
* /login
* Ответ:
* {
* "status": "error"
* }

**GET /player-status**

Позволяет получить статус для текущего игрока.

Параметры запроса:

token: токен игрока.

* Примеры:
* Запрос:
* /player-status?token=89580e35-5424-4286-8e45-b4d818278367
* Ответ:
* {
* "status": "ok",
* "player-status": {
* "status": "lobby"
* }
* }
* Расшифровка: Игрок в лобби, нет игры, в которой игрок с таким токеном был бы участником
* Запрос:
* /player-status?token=89580e35-5424-4286-8e45-b4d818278367
* Ответ:
* {
* "status": "ok",
* "player-status": {
* "status": "game",
* "game": {
* "id": "3b20fa39"
* }
* }
* }
* Расшифровка: Игрок в игре
* Запрос:
* /player-status?token=895
* Ответ:
* {
* "status": "error"
* }
* Расшифровка: Нет игрока с таким токеном

**GET /start**

Позволяет начать игру.

Параметры запроса:

token: токен игрока.

* Примеры:
* Запрос:
* /start?token=89580e35-5424-4286-8e45-b4d818278367
* Ответ:
* {
* "status": "ok",
* "player-status": {
* "status": "game",
* "game": {
* "id": "3b20fa39"
* }
* }
* }
* Расшифровка: Игрок добавлен в игру с соответствующим id
* Запрос:
* /start?token=895
* Ответ:
* {
* "status": "error",
* "message": "token doesn't exist"
* }
* Расшифровка: Нет игрока с таким токеном
* Запрос:
* /start?token=89580e35-5424-4286-8e45-b4d818278367
* Ответ:
* {
* "status": "error",
* "message": "player is already in game"
* }
* Расшифровка: Игрок уже в игре, нельзя начать две игры одновременно

**GET /game-status**

Получить статус игры. Возможные статусы: ожидание соперника (игра создана, но у игрока еще нет противника), ожидание вашего хода (игра создана и текущий игрок не сделал ход), ожидание хода соперника (игра создана, ход текущим игроком сделан, ответ от соперника еще не получен), победа, поражение.

Нельзя получить статус для боя, в котором не участвуешь.

Параметры запроса:

token: токен игрока.

id: идентификатор игры.

* Примеры:
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "waiting-for-start",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Ожидание начала игры. Пока в игре только один участник (вы),
* ожидаем, когда соперник присоединится к игре
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "waiting-for-your-move",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Ожидание хода игрока. Такой статус будет либо сразу после
* того, как в игре стало два участника, либо после хода, в котором оба
* участника сделали одинаковый ход
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "waiting-for-enemy-move",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Ожидание хода соперника. Такой статус будет, если ход текущего
* игрока отправлен, а от соперника ход еще не получен
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "lose",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Игра проиграна
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "win",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Игра выиграна
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367
* Ответ:
* {
* "status": "error",
* "message": "no game id"
* }
* Расшифровка: Id игры не передан
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367&id=111
* Ответ:
* {
* "status": "error",
* "message": "wrong game id"
* }
* Расшифровка: Id игры некорректный / бой не существует / бой закончен
* Запрос:
* /game-status?token=89580e35&id=3b20fa39
* Ответ:
* {
* "status": "error",
* "message": "token doesn't exist"
* }
* Расшифровка: Нет игрока с таким токеном
* Запрос:
* /game-status?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa40
* Ответ:
* {
* "status": "error",
* "message": "player is not in this game"
* }
* Расшифровка: Игрок не в этой игре

**GET /play**

Позволяет делать ходы в игре. В случае успешного запроса бэкенд в ответе также передаст статус игры (ждем хода соперника или вашего — в том случае, если на предыдущем ходу были показаны одинаковые фигуры, либо победа/поражение, если с учетом вашего хода результат известен).

Параметры запроса:

token: токен игрока.

id: идентификатор игры.

move: ход. Может быть либо rock, либо scissors, либо paper.

* Примеры:
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39&move=rock
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "waiting-for-enemy-move",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Ход отправлен, ожидание ответа соперника
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39&move=paper
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "waiting-for-your-move",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Ход игрока отправлен, соперник ранее отправлял ход,
* и у вас ничья (например, оба показали ножницы). Таким образом,
* ожидаем следующий ход игрока
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39&move=scissors
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "lose",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Ход игрока отправлен, соперник ранее отправлял ход,
* и вы проиграли
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39&move=paper
* Ответ:
* {
* "status": "ok",
* "game-status": {
* "status": "win",
* "enemy": {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* }
* }
* }
* Расшифровка: Ход игрока отправлен, соперник ранее отправлял ход,
* и вы выиграли
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&move=paper
* Ответ:
* {
* "status": "error",
* "message": "no game id"
* }
* Расшифровка: Id игры не передан
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=111&move=paper
* Ответ:
* {
* "status": "error",
* "message": "wrong game id"
* }
* Расшифровка: Id игры некорректный / бой не существует / бой закончен
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39&move=rock
* Ответ:
* {
* "status": "error",
* "message": "not your move"
* }
* Расшифровка: Не ваш ход! Вы уже отправили ход в текущем раунде, сейчас
* игра ждет хода вашего соперника
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39&move=ololo
* Ответ:
* {
* "status": "error",
* "message": "wrong move"
* }
* Расшифровка: Недопустимый ход
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa39
* Ответ:
* {
* "status": "error",
* "message": "no move"
* }
* Расшифровка: Ход не передан
* Запрос:
* /play?token=89580e35&id=3b20fa39&move=paper
* Ответ:
* {
* "status": "error",
* "message": "token doesn't exist"
* }
* Расшифровка: Нет игрока с таким токеном
* Запрос:
* /play?token=89580e35-5424-4286-8e45-b4d818278367&id=3b20fa40&move=paper
* Ответ:
* {
* "status": "error",
* "message": "player is not in this game"
* }
* Расшифровка: Игрок не в этой игре

**GET /player-list**

Список залогиненных игроков. Залогиненным считается игрок, который делал запрос со своим токеном в течение последних пары минут.

Параметры запроса:

token: токен игрока. Необязательный (то есть можно получить список игроков онлайн без токена). Если передан, у текущего игрока будет передан параметр "you" равный true.

* Примеры:
* Запрос:
* /player-list?token=89580e35-5424-4286-8e45-b4d818278367
* Ответ:
* {
* "status": "ok",
* "list": [
* {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* },
* ... // еще игроки в таком же формате
* {
* "login": "СердцеКамень",
* "wins": 7,
* "loses": 3,
* "rocks": 55,
* "papers": 2,
* "scissors": 3,
* "you": true,
* }
* ]
* }
* Запрос:
* /player-list
* Ответ:
* {
* "status": "ok",
* "list": [
* {
* "login": "ВладБумага",
* "wins": 47,
* "loses": 33,
* "rocks": 22,
* "papers": 202,
* "scissors": 13,
* },
* ... // еще игроки в таком же формате
* {
* "login": "СердцеКамень",
* "wins": 7,
* "loses": 3,
* "rocks": 55,
* "papers": 2,
* "scissors": 3,
* }
* ]
* }

**Основной сценарий**

Интерфейс предлагает залогиниться. Введенный логин отправляем в ручку бэкенда для логина, получаем в ответ токен. Опрашиваем ручку статуса для игрока, и если в ответе «лобби», то отрисовываем экран с кнопкой-приглашением начать игру. После нажатия на кнопку отправляем запрос на ручку старта игры. Получаем в ответе ручки id игры. Получив его, начинаем ожидать соперника (опрашиваем ручку статуса игры и отрисовываем экран ожидания). Ждем, когда в ответе бэкенда статус будет «ожидание вашего хода» — можно отрисовать кнопки для хода. После нажатия на фигуру для хода отправляем ход на ручку play и получаем ответ со статусом. Если статус «ожидаем хода соперника», то начинаем опрашивать статус игры и ждем, что он станет «победа», «поражение» или «ожидание хода игрока». «Ожидание хода игрока» будет в том случае, если соперник выбрал тот же ход. В этом случае надо снова показать кнопки для хода. В случае победы/поражения отрисовываем соответствующий экран и даем возможность вернуться в лобби/начать новую игру.

**Архитектура приложения**

Данные тут рекомендации не обязательны, могут дополняться и меняться в зависимости от особенностей вашего проекта.

**Структура HTML**

Базовая структура index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Камень, ножницы, бумага</title>

</head>

<body>

<div class="app"></div>

<script>

window.application = { ... } // смотрите далее

</script>

<script src="js/file1.js"></script>

<script src="js/file2.js"></script>

</body>

</html>

В div.app будут отрисовываться все страницы приложения. При отрисовке нового экрана содержимое блока будет очищаться и будет отрисовываться новое содержимое.

Важно, чтобы тег script с созданием глобальной переменной application был сверху всех прочих скриптов. Таким образом эта переменная будет доступна для всего кода, который будет написан вами.

**Объект application**

Сделайте глобальную переменную application. Ее структура будет следующей:

window.application = {

blocks: {},

screens: {},

renderScreen: function(screenName) {},

renderBlock: function(blockName, container) {},

timers: []

}

Рекомендуем во всем коде обращаться к этим данным с помощью конструкции window.application.

Блоки (blocks) — это функции, которые получают на вход объект-узел контейнера и отрисовывают туда какой-то блок. Например, функция, которая отрисовывает в контейнер кнопку, умеющую выводить в консоль слово 'click'.

function renderExampleButton(container) {

const button = document.createElement('button');

button.addEventListener('click', () => {

console.log('click');

});

container.appendChild(button);

}

Вызывая эту функцию, вы передаете туда объект-узел со страницы, и кнопка отрисуется в этом контейнере.

renderExampleButton(document.querySelector('.some-block'));

Вам нужно сделать функцию window.application.renderBlock такую, чтобы работал вот такой код:

// Записываем функцию в поле объекта

window.application.blocks['example-button'] = renderExampleButton;

// Должен вызвать window.application.blocks['example-button'](), передавая туда объект-узел

window.application.renderBlock('example-button', document.querySelector('.some-block'));

Если блока, который передали в window.application.renderBlock не существует (нет такого поля в window.application.blocks), пусть функция выводит сообщение об этом в консоль.

Экраны (screens) — это функции, которые отрисовывают экран полностью, очищают блок document.querySelector('.app') и отрисовывают туда содержимое.

function renderExampleScreen() {

const app = document.querySelector('.app');

app.textContent = '';

const title = document.createElement('h1');

title.textContent = 'Страница-пример';

const content = document.createElement('div');

window.application.renderBlock('example-button', content);

app.appendChild(title);

app.appendChild(content);

}

Вам нужно сделать функцию window.application.renderScreen такую, чтобы работал вот такой код:

// Записываем функцию в поле объекта

window.application.screens['example'] = renderExampleScreen;

// Должен вызвать window.application.blocks['example']()

window.application.renderScreen('example');

Если страницы, которую передали в window.application.renderScreen не существует (нет такого поля в window.application.screens), пусть функция выводит сообщение об этом в консоль.

**Таймеры**

Время от времени ваши блоки будут использовать setInterval для того, чтобы перепроверять ответ бэкенда. Например, скорее всего, именно это будет делать блок, отрисовывающий список людей онлайн (чтобы обновлять свое содержимое), также блок на странице игры, ожидающий хода соперника и так далее. Не всегда обновление страницы будет происходить внутри такого блока, поэтому, чтобы при переходе на другой экран отменять все выставленные таймеры, при использовании setInterval делайте следующее:

window.application.timers.push(setInterval( ... ));

После чего добавьте в функцию renderScreen проход по всему массиву window.application.timers, выполнение clearInterval() на каждом и очистку массива window.application.timers.

**Состав приложения**

Заголовки, перечисленные в этом разделе, можно считать задачами.

**Движок отрисовки**

Необходимо реализовать window.application и функции renderScreen/renderBlock из предыдущего раздела. Создать страницу и блок-примеры, убедиться, что по командам из консоли браузера страница и блок отрисовываются.

**Функция request для AJAX-запросов**

Функция принимает URL бэкенда (например, '/login'), GET-параметры в виде объекта { ключ: значение } и callback, который будет вызван с передачей туда данных ответа.

**Логин**

Составные части:

**Блок авторизации**

Блок должен быть добавлен в window.application.blocks. При отрисовке блок рендерит текстовое поле ввода и кнопку «Войти». При нажатии кнопки делает AJAX-запрос на ручку /login для получения токена. Токен необходимо положить в window.application. После получения токена делает запрос на статус игрока (ручка /player-status). В зависимости от получаемого от этой ручки статуса переходит в лобби (обычный сценарий) или в бой (если вдруг игрок отключился от боя, но перезашел, например, через другой браузер).

**Страница авторизации**

Страница должна быть добавлена в window.application.screens. Отрисовывает внутри себя блок авторизации.

**Лобби**

Составные части:

**Блок списка игроков**

Блок должен быть добавлен в window.application.blocks. Блок должен уметь раз в секунду опрашивать ручку /player-list и обновлять свое содержимое в соответствии с полученными от этой ручки данными. Минимально — просто выводить логины.

**Блок кнопки «Играть!»**

Блок должен быть добавлен в window.application.blocks. Блок должен отрисовать внутри переданного контейнера кнопку, при клике по которой будет произведен запрос на ручку /start. После получения ответа от ручки с id боя необходимо записать id боя в window.application. После чего необходимо отрисовать страницу ожидания соперника.

**Страница лобби**

Страница должна быть добавлена в window.application.screens. Отрисовывает внутри себя блок списка игроков и блок кнопки «Играть!».

**Ожидание игры**

Составные части:

**Блок ожидания игры**

Блок должен быть добавлен в window.application.blocks. Визуально предоставляет пользователю информацию о том, что он ожидает начало игры. Сразу после отрисовки использует id игры, записанный после клика на блок кнопки «Играть!», и делает запросы раз в полсекунды в ручку /game-status. Как только в ответе придет статус, отличный от 'waiting-for-start', отрисовываем экран хода.

**Страница ожидания игры**

Страница должна быть добавлена в window.application.screens. Отрисовывает внутри себя блок ожидания игры.

**Ход**

Составные части:

**Блок хода**

Блок должен быть добавлен в window.application.blocks. Представляет собой три кнопки: «Камень», «Ножницы» и «Бумага». После клика на кнопку должен произойти запрос на ручку /play с данными игры и хода. В зависимости от ответа должен произойти переход либо на страницу ожидания хода соперника, либо на страницу победы, либо на страницу поражения.

**Страница хода**

Страница должна быть добавлена в window.application.screens. Отрисовывает внутри себя блок хода.

**Ожидание хода соперника**

Составные части:

**Блок ожидания хода**

Блок должен быть добавлен в window.application.blocks. Визуально предоставляет пользователю информацию о том, что он ожидает ход соперника. Сразу после отрисовки использует id игры, записанный в window.application и делает запросы раз в полсекунды в ручку /game-status. Как только в ответе придет статус, отличный от 'waiting-for-enemy-move', отрисовываем экран, соответствующий статусу игры — экран победы, поражения или экран хода, если игра продолжается.

**Страница ожидания хода**

Страница должна быть добавлена в window.application.screens. Отрисовывает внутри себя блок ожидания хода.

**Победа**

Составные части:

**Блок победы**

Блок должен быть добавлен в window.application.blocks. Визуально поздравляет с победой.

**Блок кнопки «Перейти в лобби»**

Блок должен быть добавлен в window.application.blocks. При клике по кнопке отрисовываем страницу лобби.

**Блок кнопки «Играть еще!»**

Кнопка аналогичная кнопке на странице лобби. Желательно использовать тот же блок, а не создавать новый.

**Страница победы**

Страница должна быть добавлена в window.application.screens. Отрисовывает внутри себя блок победы, блок кнопки «Перейти в лобби» и блок кнопки «Играть еще!».

**Поражение**

Составные части:

**Блок поражения**

Блок должен быть добавлен в window.application.blocks. Визуально насмехается над незадачливым игроком.

**Блок кнопки «Перейти в лобби»**

Такой же блок, как в предыдущем кусочке. Используем один блок, не создаем два одинаковых.

**Блок кнопки «Играть еще!»**

Кнопка аналогичная кнопке на странице лобби. Желательно использовать тот же блок, а не создавать новый.

**Страница поражения**

Страница должна быть добавлена в window.application.screens. Отрисовывает внутри себя блок поражения, блок кнопки «Перейти в лобби» и блок кнопки «Играть еще!».

**🦸‍♂️ Поздравляем с окончанием третьего курса!**

Сегодня — идеальный момент для того, чтобы вернуться к скилсету веб-разработчика.

Скилсет — это набор знаний и умений веб-разработчика, который необходим для устройства на позицию junior-разработчика.

Мы подготовили для вас этот удобный инструмент для отслеживания собственного прогресса и понимания, что вы освоите за следующие 12 месяцев.

**Если вы знакомы со скилсетом:**

— откройте вашу копию скилсета,

— пройдитесь по навыкам верстки и оцените свой прогресс.

**Если видите скилсет впервые:**

— [откройте скилсет](https://docs.google.com/spreadsheets/d/1Ic8pEF5jR1K6Kj1rd550XVMIBQQTZZ5xB6hiZpWqbPY/edit#gid=397278095);

— скопируйте его на свой Google-диск: Файл — Создать копию;

— заполните свою копию, согласно инструкции;

— возвращайтесь к этому документу после каждого пройденного курса.

*Также в этом файле вы найдете практику, которая поможет более четко сформулировать цель на следующие 3 месяца. Эту практику можно сделать по желанию.*