# Exploding Chickens

A full-stack card game

Radison Akerman, April 2022

## About me

- Computer Science Major, Business Minor @ UIC
- Project Manager & Info Security @ UIC COE
- Web Security Intern @ US Dept of Veteran Affairs
- Photography, cycling, swimming, woodworking, chess, small electronics

Draw Deck x52

EXPLODING
CHICKEN
SHOW THIS CARD
IMMEDIATELY.
YOLKING AROUND
EXPLODING
CHICKEN

Discard Deck x0

EXPLODING CHICKEN
SHOW THIS CARD IMMEDIATELY.
YOLKING AROUND
EXPLODING CHICKEN

## Exploding Chicken

Avoid this little guy at all costs. Once this card is drawn, you must use a defuse card to stop the ticking time bomb. If you don't have a defuse card, it looks like your time is up.

## Exploding Chicken

```
if (card is drawn)
  if (player has defuse)
    plays defuse, places
    chicken back in draw deck
  else
    player explodes, removed
    from game permanently
```

## Defuse Card

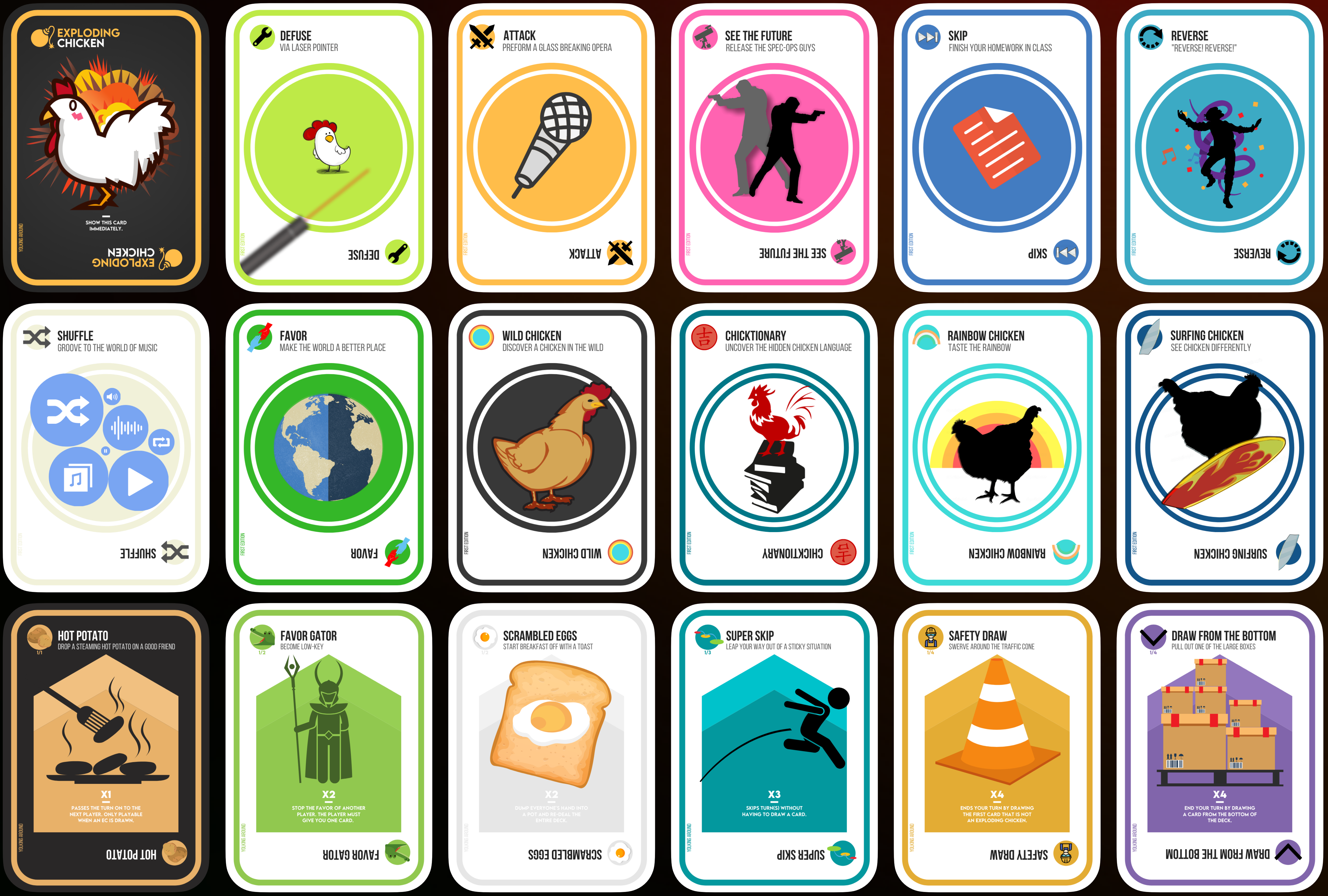
```
if (card is drawn)
  place card in players hand

if (card is played &&
player is exploding)
  discard card
  && prompt player to place
  chicken back in draw deck
  && advance turn
```

## Shuffle Card

```
if (card is drawn)
  place card in players hand

if (card is played)
  shuffle draw deck
  && discard card
```

## Shuffle Card

- Player wants to play card
- Sends “play-card” request
- Cascading validation phase
  - Is the player valid?
  - Is it their turn?
  - Do they have this card?
  - Can they play it now?
- Completes card action (randomize draw deck)
- Discard card
- Tell everyone what happened


EXPLODING CHICKEN
SHOW THIS CARD IMMEDIATELY.
YOLKING AROUND
DEFUSE
VIA LASER POINTER
FIRST EDITION
ATTACK
PREFORM A GLASS BREAKING OPERA
SEE THE FUTURE
RELEASE THE SPEC-OPS GUYS
SKIP
FINISH YOUR HOMEWORK IN CLASS
REVERSE
"REVERSE! REVERSE!"
SHUFFLE
GROOVE TO THE WORLD OF MUSIC
FAVOR
MAKE THE WORLD A BETTER PLACE
WILD CHICKEN
DISCOVER A CHICKEN IN THE WILD
CHICKTIONARY
UNCOVER THE HIDDEN CHICKEN LANGUAGE
RAINBOW CHICKEN
TASTE THE RAINBOW
SURFING CHICKEN
SEE CHICKEN DIFFERENTLY
HOT POTATO
DROP A STEAMING HOT POTATO ON A GOOD FRIEND
1/1
X1
PASSES THE TURN ON TO THE NEXT PLAYER. ONLY PLAYABLE WHEN AN EC IS DRAWN.
FAVOR GATOR
BECOME LOW-KEY
1/2
X2
STOP THE FAVOR OF ANOTHER PLAYER. THE PLAYER MUST GIVE YOU ONE CARD.
SCRAMBLED EGGS
START BREAKFAST OFF WITH A TOAST
1/2
X2
DUMP EVERYONE'S HAND INTO A POT AND RE-DEAL THE ENTIRE DECK.
SUPER SKIP
LEAP YOUR WAY OUT OF A STICKY SITUATION
1/3
X3
SKIPS TURN(S) WITHOUT HAVING TO DRAW A CARD.
SAFETY DRAW
SWERVE AROUND THE TRAFFIC CONE
1/4
X4
ENDS YOUR TURN BY DRAWING THE FIRST CARD THAT IS NOT AN EXPLODING CHICKEN.
DRAW FROM THE BOTTOM
PULL OUT ONE OF THE LARGE BOXES
1/4
X4
END YOUR TURN BY DRAWING A CARD FROM THE BOTTOM OF THE DECK.

**565 Games**
**4,725 Minutes**
**20,100 Cards Played**

Since April 2021

## Demo

Scalable and fast

Room for expansion

Easy to understand

## Where do you start?

Maximize compatibility

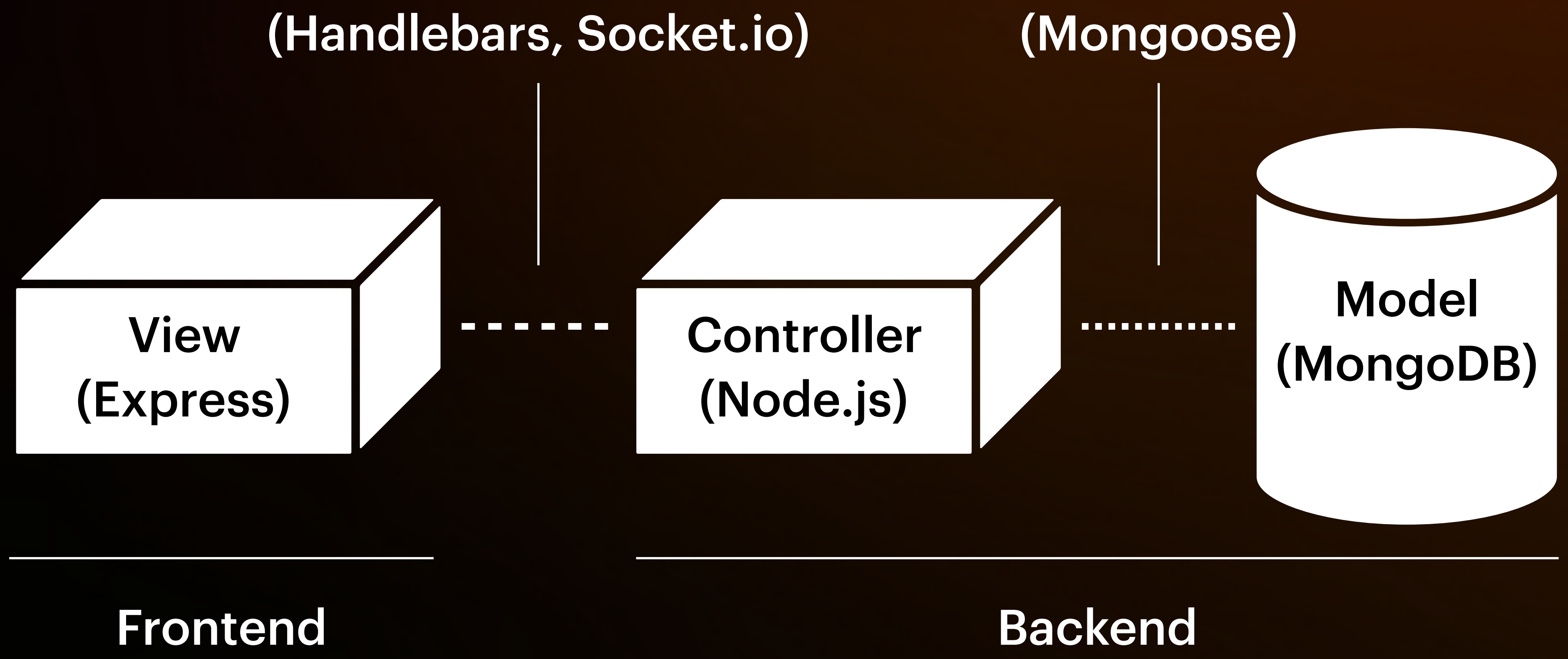
Quick to develop

Real-time rendering

## v1.0.0

### An unorganized first attempt (that works)

- Full-stack web application
  - Node.js — an asynchronous event-driven JavaScript runtime geared towards scalable network apps
  - MongoDB — a NoSQL document-oriented database
    - Mongoose as an ODM (Object Data Modeling)
  - Handlebars — HTML templating language
  - Socket.io — real-time, bidirectional communication


(Handlebars, Socket.io)
(Mongoose)
View (Express)
Controller (Node.js)
Model (MongoDB)
Frontend
Backend

## Game

### Cards

- _id
- action
- assignment
- position
- pack

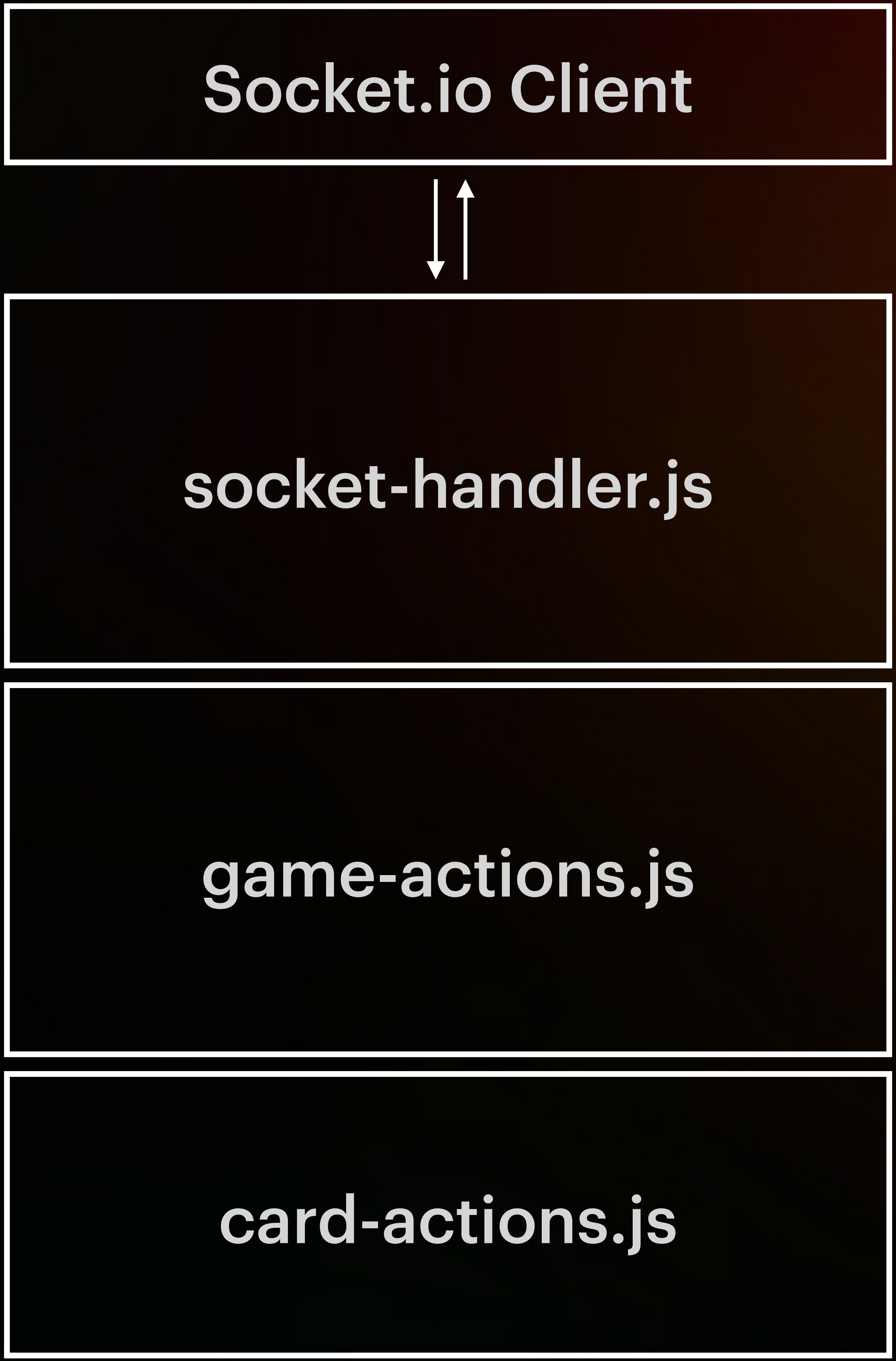
### Players

- _id
- nickname
- avatar
- seat_position
- wins
- sockets_open
- is_host
- is_dead

### Events

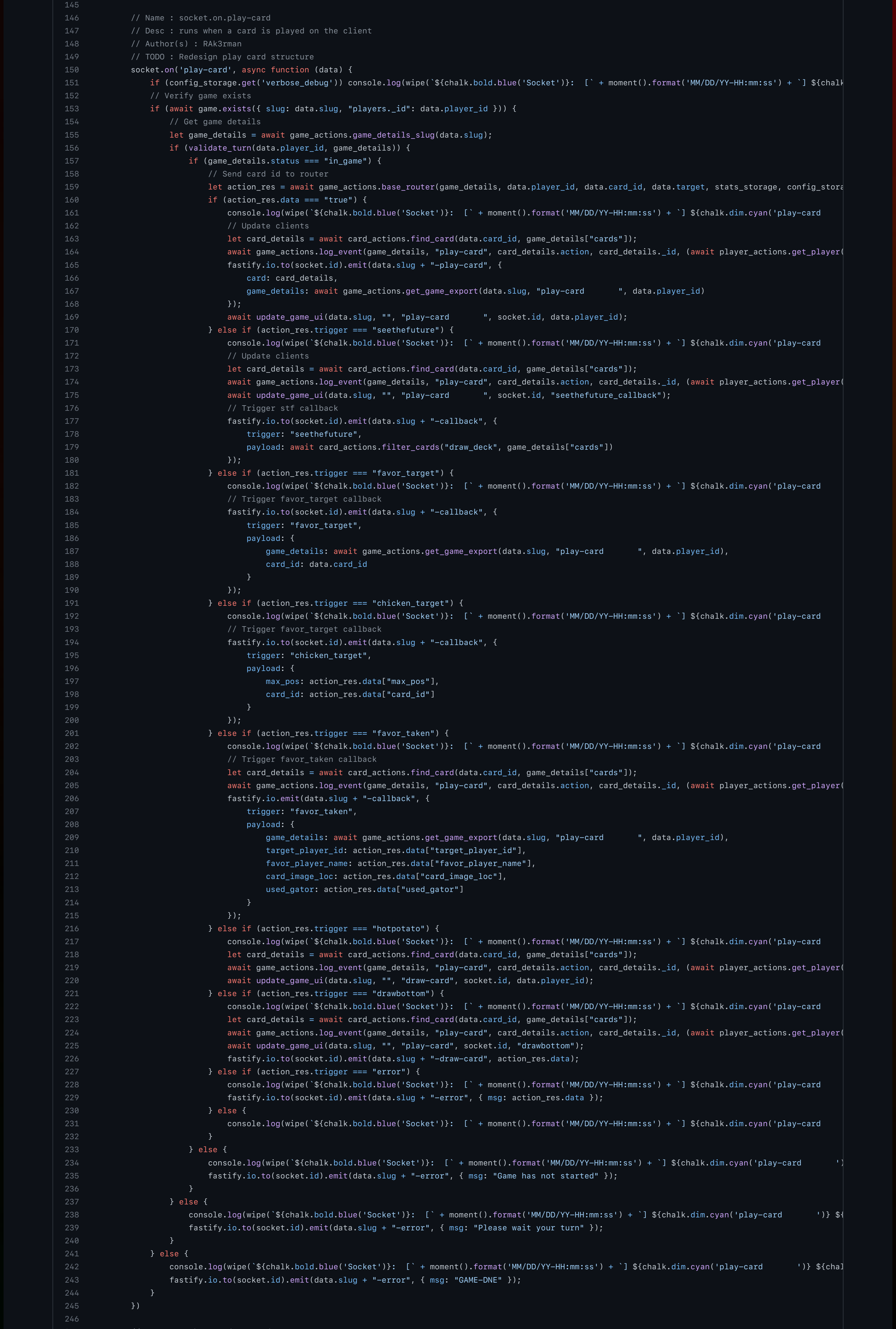
- _id
- tag
- req_player
- target_plyr
- related_key
- related_value

### Misc data...



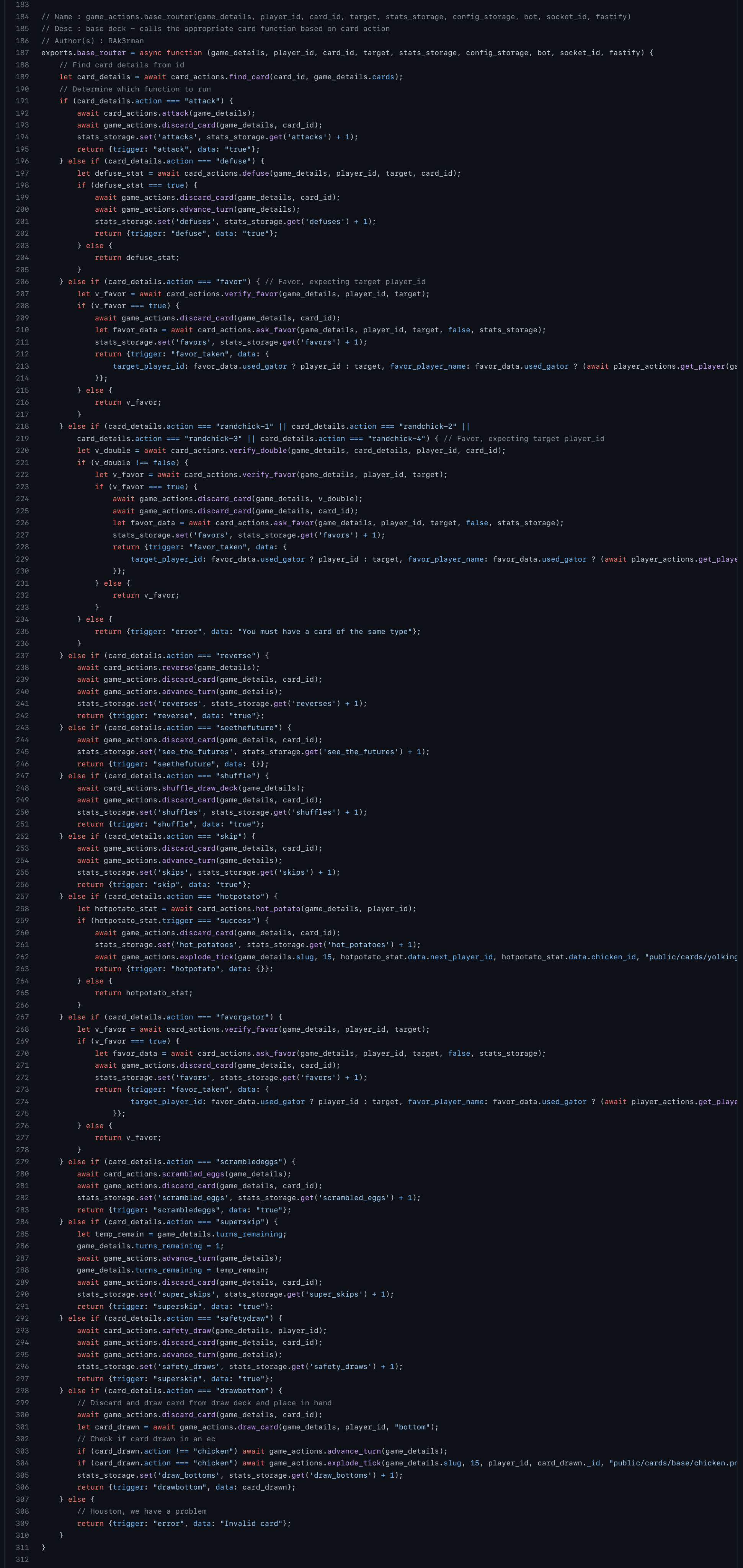
## Shuffle Card

- Player wants to play card
- Sends “play-card” request
- Cascading validation phase
  - Is the player valid?
  - Is it their turn?
  - Do they have this card?
  - Can they play it now?
- Completes card action (randomize draw deck)
- Discard card
- Tell everyone what happened



```
// Name : socket.on.play-card
// Desc : runs when a card is played on the client
// Author(s) : RAk3rman
// TODO : Redesign play card structure
socket.on('play-card', async function (data) {
    if (config_storage.get('verbose_debug')) console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk
    // Verify game exists
    if (await game.exists({ slug: data.slug, "players._id": data.player_id })) {
        // Get game details
        let game_details = await game_actions.game_details_slug(data.slug);
        if (validate_turn(data.player_id, game_details)) {
            if (game_details.status === "in_game") {
                // Send card id to router
                let action_res = await game_actions.base_router(game_details, data.player_id, data.card_id, data.target, stats_storage, config_stora
                if (action_res.data === "true") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    // Update clients
                    let card_details = await card_actions.find_card(data.card_id, game_details["cards"]);
                    await game_actions.log_event(game_details, "play-card", card_details.action, card_details._id, (await player_actions.get_player(
                    fastify.io.to(socket.id).emit(data.slug + "-play-card", {
                        card: card_details,
                        game_details: await game_actions.get_game_export(data.slug, "play-card       ", data.player_id)
                    });
                    await update_game_ui(data.slug, "", "play-card       ", socket.id, data.player_id);
                } else if (action_res.trigger === "seethefuture") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    // Update clients
                    let card_details = await card_actions.find_card(data.card_id, game_details["cards"]);
                    await game_actions.log_event(game_details, "play-card", card_details.action, card_details._id, (await player_actions.get_player(
                    await update_game_ui(data.slug, "", "play-card       ", socket.id, "seethefuture_callback");
                    // Trigger stf callback
                    fastify.io.to(socket.id).emit(data.slug + "-callback", {
                        trigger: "seethefuture",
                        payload: await card_actions.filter_cards("draw_deck", game_details["cards"])
                    });
                } else if (action_res.trigger === "favor_target") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    // Trigger favor_target callback
                    fastify.io.to(socket.id).emit(data.slug + "-callback", {
                        trigger: "favor_target",
                        payload: {
                            game_details: await game_actions.get_game_export(data.slug, "play-card       ", data.player_id),
                            card_id: data.card_id
                        }
                    });
                } else if (action_res.trigger === "chicken_target") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    // Trigger favor_target callback
                    fastify.io.to(socket.id).emit(data.slug + "-callback", {
                        trigger: "chicken_target",
                        payload: {
                            max_pos: action_res.data["max_pos"],
                            card_id: action_res.data["card_id"]
                        }
                    });
                } else if (action_res.trigger === "favor_taken") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    // Trigger favor_taken callback
                    let card_details = await card_actions.find_card(data.card_id, game_details["cards"]);
                    await game_actions.log_event(game_details, "play-card", card_details.action, card_details._id, (await player_actions.get_player(
                    fastify.io.emit(data.slug + "-callback", {
                        trigger: "favor_taken",
                        payload: {
                            game_details: await game_actions.get_game_export(data.slug, "play-card       ", data.player_id),
                            target_player_id: action_res.data["target_player_id"],
                            favor_player_name: action_res.data["favor_player_name"],
                            card_image_loc: action_res.data["card_image_loc"],
                            used_gator: action_res.data["used_gator"]
                        }
                    });
                } else if (action_res.trigger === "hotpotato") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    let card_details = await card_actions.find_card(data.card_id, game_details["cards"]);
                    await game_actions.log_event(game_details, "play-card", card_details.action, card_details._id, (await player_actions.get_player(
                    await update_game_ui(data.slug, "", "draw-card", socket.id, data.player_id);
                } else if (action_res.trigger === "drawbottom") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    let card_details = await card_actions.find_card(data.card_id, game_details["cards"]);
                    await game_actions.log_event(game_details, "play-card", card_details.action, card_details._id, (await player_actions.get_player(
                    await update_game_ui(data.slug, "", "play-card", socket.id, "drawbottom");
                    fastify.io.to(socket.id).emit(data.slug + "-draw-card", action_res.data);
                } else if (action_res.trigger === "error") {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                    fastify.io.to(socket.id).emit(data.slug + "-error", { msg: action_res.data });
                } else {
                    console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                }
            } else {
                console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card       ')
                fastify.io.to(socket.id).emit(data.slug + "-error", { msg: "Game has not started" });
            }
        } else {
            console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card       ')} ${
            fastify.io.to(socket.id).emit(data.slug + "-error", { msg: "Please wait your turn" });
        }
    } else {
        console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card       ')} ${chal
        fastify.io.to(socket.id).emit(data.slug + "-error", { msg: "GAME-DNE" });
    }
})
```

```
        // Name : socket.on.play-card
        // Desc : runs when a card is played on the client
        // Author(s) : RAk3rman
        // TODO : Redesign play card structure
        socket.on('play-card', async function (data) {
            if (config_storage.get('verbose_debug')) console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chal
            // Verify game exists
            if (await game.exists({ slug: data.slug, "players._id": data.player_id })) {
                // Get game details
                let game_details = await game_actions.game_details_slug(data.slug);
                if (validate_turn(data.player_id, game_details)) {
                    if (game_details.status === "in_game") {
                        // Send card id to router
                        let action_res = await game_actions.base_router(game_details, data.player_id, data.card_id, data.target, stats_storage, config_stor
                        if (action_res.data === "true") {
                            console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                            // Update clients
                            let card_details = await card_actions.find_card(data.card_id, game_details["cards"]);
                            await game_actions.log_event(game_details, "play-card", card_details.action, card_details._id, (await player_actions.get_player
                            fastify.io.to(socket.id).emit(data.slug + "-play-card", {
                                card: card_details,
                                game_details: await game_actions.get_game_export(data.slug, "play-card       ", data.player_id)
                            });
                            await update_game_ui(data.slug, "", "play-card       ", socket.id, data.player_id);
                        } else if (action_res.trigger === "seethefuture") {
                            console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                            // Update clients
                            let card_details = await card_actions.find_card(data.card_id, game_details["cards"]);
                            await game_actions.log_event(game_details, "play-card", card_details.action, card_details._id, (await player_actions.get_player
                            await update_game_ui(data.slug, "", "play-card       ", socket.id, "seethefuture_callback");
                            // Trigger stf callback
                            fastify.io.to(socket.id).emit(data.slug + "-callback", {
                                trigger: "seethefuture",
                                payload: await card_actions.filter_cards("draw_deck", game_details["cards"])
                            });
                        } else if (action_res.trigger === "favor_target") {
                            console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan('play-card
                            // Trigger favor_target callback
                            fastify.io.to(socket.id).emit(data.slug + "-callback", {
                                trigger: "favor_target",
                                payload: {
                                    game_details: await game_actions.get_game_export(data.slug, "play-card       ", data.player_id),
                                    card_id: data.card_id
                                }
```



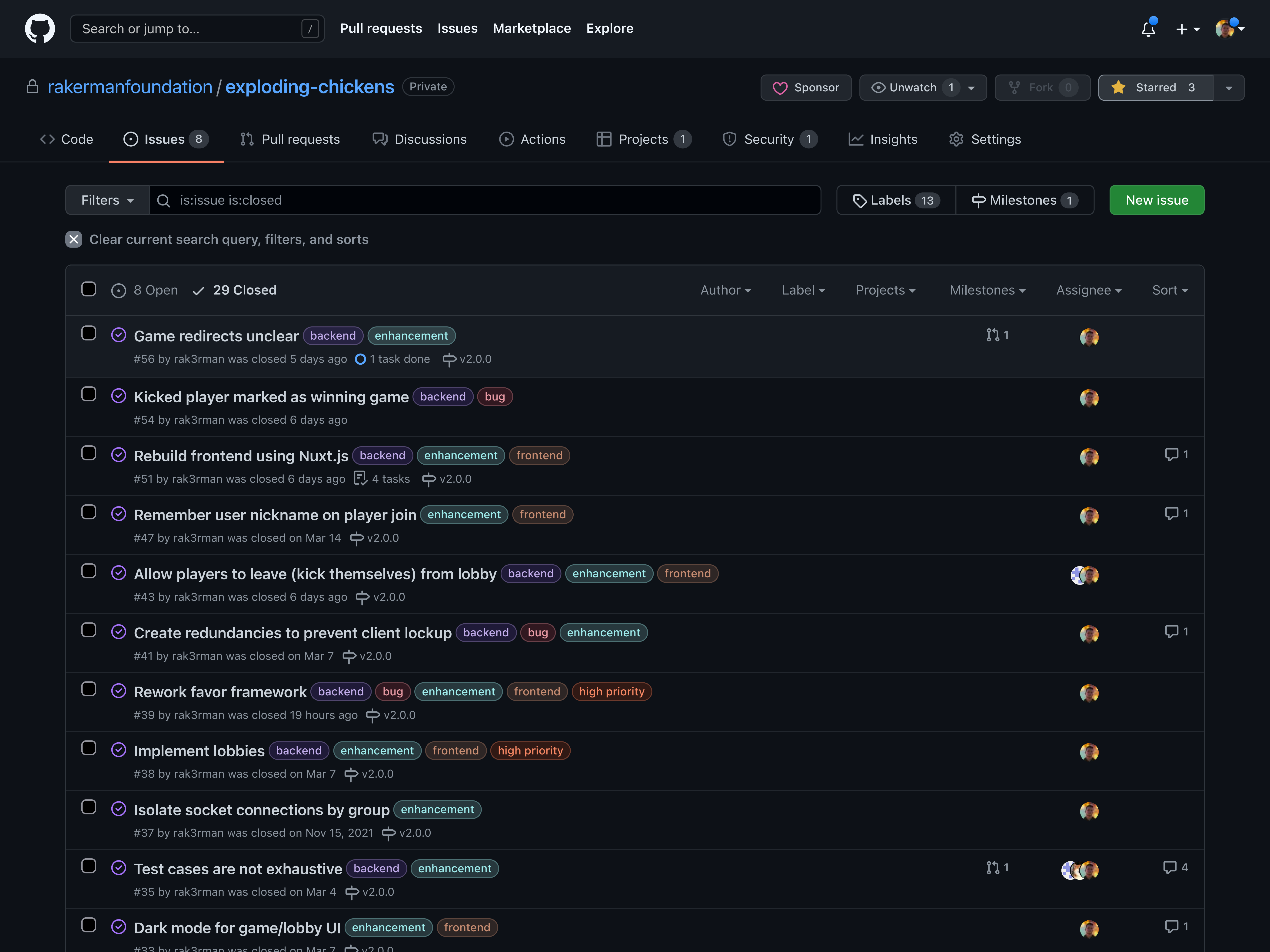
```javascript

// Name : game_actions.base_router(game_details, player_id, card_id, target, stats_storage, config_storage, bot, socket_id, fastify)
// Desc : base deck - calls the appropriate card function based on card action
// Author(s) : RAk3rman
exports.base_router = async function (game_details, player_id, card_id, target, stats_storage, config_storage, bot, socket_id, fastify) {
    // Find card details from id
    let card_details = await card_actions.find_card(card_id, game_details.cards);
    // Determine which function to run
    if (card_details.action === "attack") {
        await card_actions.attack(game_details);
        await game_actions.discard_card(game_details, card_id);
        stats_storage.set('attacks', stats_storage.get('attacks') + 1);
        return {trigger: "attack", data: "true"};
    } else if (card_details.action === "defuse") {
        let defuse_stat = await card_actions.defuse(game_details, player_id, target, card_id);
        if (defuse_stat === true) {
            await game_actions.discard_card(game_details, card_id);
            await game_actions.advance_turn(game_details);
            stats_storage.set('defuses', stats_storage.get('defuses') + 1);
            return {trigger: "defuse", data: "true"};
        } else {
            return defuse_stat;
        }
    } else if (card_details.action === "favor") { // Favor, expecting target player_id
        let v_favor = await card_actions.verify_favor(game_details, player_id, target);
        if (v_favor === true) {
            await game_actions.discard_card(game_details, card_id);
            let favor_data = await card_actions.ask_favor(game_details, player_id, target, false, stats_storage);
            stats_storage.set('favors', stats_storage.get('favors') + 1);
            return {trigger: "favor_taken", data: {
                target_player_id: favor_data.used_gator ? player_id : target, favor_player_name: favor_data.used_gator ? (await player_actions.get_player(ga
            }};
        } else {
            return v_favor;
        }
    } else if (card_details.action === "randchick-1" || card_details.action === "randchick-2" ||
        card_details.action === "randchick-3" || card_details.action === "randchick-4") { // Favor, expecting target player_id
        let v_double = await card_actions.verify_double(game_details, card_details, player_id, card_id);
        if (v_double !== false) {
            let v_favor = await card_actions.verify_favor(game_details, player_id, target);
            if (v_favor === true) {
                await game_actions.discard_card(game_details, v_double);
                await game_actions.discard_card(game_details, card_id);
                let favor_data = await card_actions.ask_favor(game_details, player_id, target, false, stats_storage);
                stats_storage.set('favors', stats_storage.get('favors') + 1);
                return {trigger: "favor_taken", data: {
                    target_player_id: favor_data.used_gator ? player_id : target, favor_player_name: favor_data.used_gator ? (await player_actions.get_playe
                }};
            } else {
                return v_favor;
            }
        } else {
            return {trigger: "error", data: "You must have a card of the same type"};
        }
    } else if (card_details.action === "reverse") {
        await card_actions.reverse(game_details);
        await game_actions.discard_card(game_details, card_id);
        await game_actions.advance_turn(game_details);
        stats_storage.set('reverses', stats_storage.get('reverses') + 1);
        return {trigger: "reverse", data: "true"};
    } else if (card_details.action === "seethefuture") {
        await game_actions.discard_card(game_details, card_id);
        stats_storage.set('see_the_futures', stats_storage.get('see_the_futures') + 1);
        return {trigger: "seethefuture", data: {}};
    } else if (card_details.action === "shuffle") {
        await card_actions.shuffle_draw_deck(game_details);
        await game_actions.discard_card(game_details, card_id);
        stats_storage.set('shuffles', stats_storage.get('shuffles') + 1);
        return {trigger: "shuffle", data: "true"};
    } else if (card_details.action === "skip") {
        await game_actions.discard_card(game_details, card_id);
        await game_actions.advance_turn(game_details);
        stats_storage.set('skips', stats_storage.get('skips') + 1);
        return {trigger: "skip", data: "true"};
    } else if (card_details.action === "hotpotato") {
        let hotpotato_stat = await card_actions.hot_potato(game_details, player_id);
        if (hotpotato_stat.trigger === "success") {
            await game_actions.discard_card(game_details, card_id);
            stats_storage.set('hot_potatoes', stats_storage.get('hot_potatoes') + 1);
            await game_actions.explode_tick(game_details.slug, 15, hotpotato_stat.data.next_player_id, hotpotato_stat.data.chicken_id, "public/cards/yolking
            return {trigger: "hotpotato", data: {}};
        } else {
            return hotpotato_stat;
        }
    } else if (card_details.action === "favorgator") {
        let v_favor = await card_actions.verify_favor(game_details, player_id, target);
        if (v_favor === true) {
            let favor_data = await card_actions.ask_favor(game_details, player_id, target, false, stats_storage);
            await game_actions.discard_card(game_details, card_id);
            stats_storage.set('favors', stats_storage.get('favors') + 1);
            return {trigger: "favor_taken", data: {
                    target_player_id: favor_data.used_gator ? player_id : target, favor_player_name: favor_data.used_gator ? (await player_actions.get_playe
                }};
        } else {
            return v_favor;
        }
    } else if (card_details.action === "scrambledeggs") {
        await card_actions.scrambled_eggs(game_details);
        await game_actions.discard_card(game_details, card_id);
        stats_storage.set('scrambled_eggs', stats_storage.get('scrambled_eggs') + 1);
        return {trigger: "scrambledeggs", data: "true"};
    } else if (card_details.action === "superskip") {
        let temp_remain = game_details.turns_remaining;
        game_details.turns_remaining = 1;
        await game_actions.advance_turn(game_details);
        game_details.turns_remaining = temp_remain;
        await game_actions.discard_card(game_details, card_id);
        stats_storage.set('super_skips', stats_storage.get('super_skips') + 1);
        return {trigger: "superskip", data: "true"};
    } else if (card_details.action === "safetydraw") {
        await card_actions.safety_draw(game_details, player_id);
        await game_actions.discard_card(game_details, card_id);
        await game_actions.advance_turn(game_details);
        stats_storage.set('safety_draws', stats_storage.get('safety_draws') + 1);
        return {trigger: "superskip", data: "true"};
    } else if (card_details.action === "drawbottom") {
        // Discard and draw card from draw deck and place in hand
        await game_actions.discard_card(game_details, card_id);
        let card_drawn = await game_actions.draw_card(game_details, player_id, "bottom");
        // Check if card drawn in an ec
        if (card_drawn.action !== "chicken") await game_actions.advance_turn(game_details);
        if (card_drawn.action === "chicken") await game_actions.explode_tick(game_details.slug, 15, player_id, card_drawn._id, "public/cards/base/chicken.pr
        stats_storage.set('draw_bottoms', stats_storage.get('draw_bottoms') + 1);
        return {trigger: "drawbottom", data: card_drawn};
    } else {
        // Houston, we have a problem
        return {trigger: "error", data: "Invalid card"};
    }
}

```

```
// Name : game_actions.base_router(game_details, player_id, card_id, target, stats_storage, config_storage, bot, socket_id, fastify)
// Desc : base deck - calls the appropriate card function based on card action
// Author(s) : RAk3rman
exports.base_router = async function (game_details, player_id, card_id, target, stats_storage, config_storage, bot, socket_id, fastify) {
    // Find card details from id
    let card_details = await card_actions.find_card(card_id, game_details.cards);
    // Determine which function to run
    if (card_details.action === "attack") {
        await card_actions.attack(game_details);
        await game_actions.discard_card(game_details, card_id);
        stats_storage.set('attacks', stats_storage.get('attacks') + 1);
        return {trigger: "attack", data: "true"};
    } else if (card_details.action === "defuse") {
        let defuse_stat = await card_actions.defuse(game_details, player_id, target, card_id);
        if (defuse_stat === true) {
            await game_actions.discard_card(game_details, card_id);
            await game_actions.advance_turn(game_details);
            stats_storage.set('defuses', stats_storage.get('defuses') + 1);
            return {trigger: "defuse", data: "true"};
        } else {
            return defuse_stat;
        }
    } else if (card_details.action === "favor") { // Favor, expecting target player_id
        let v_favor = await card_actions.verify_favor(game_details, player_id, target);
        if (v_favor === true) {
            await game_actions.discard_card(game_details, card_id);
            let favor_data = await card_actions.ask_favor(game_details, player_id, target, false, stats_storage);
            stats_storage.set('favors', stats_storage.get('favors') + 1);
            return {trigger: "favor_taken", data: {
                target_player_id: favor_data.used_gator ? player_id : target, favor_player_name: favor_data.used_gator ? (await player_actions.get_player(g
            }};
        } else {
            return v_favor;
        }
    } else if (card_details.action === "randchick-1" || card_details.action === "randchick-2" ||
        card_details.action === "randchick-3" || card_details.action === "randchick-4") { // Favor, expecting target player_id
        let v_double = await card_actions.verify_double(game_details, card_details, player_id, card_id);
        if (v_double !== false) {
            let v_favor = await card_actions.verify_favor(game_details, player_id, target);
            if (v_favor === true) {
                await game_actions.discard_card(game_details, v_double);
                await game_actions.discard_card(game_details, card_id);
                let favor_data = await card_actions.ask_favor(game_details, player_id, target, false, stats_storage);
                stats_storage.set('favors', stats_storage.get('favors') + 1);
```

## v2.0.0

### A segmented second attempt (that makes sense)

- Same web stack (Node.js, MongoDB, Handlebars, Socket.io) + Vue.js + Auth0
- Mocha + Istanbul — Backend tests, code coverage
- Shared repository model
- Issues and features request tracking in Github
- New Lobby abstraction



rakermanfoundation / exploding-chickens Private

is:issue is:closed

Clear current search query, filters, and sorts

8 Open ✓ 29 Closed

Game redirects unclear `backend` `enhancement` 1
#56 by rak3rman was closed 5 days ago 1 task done v2.0.0

Kicked player marked as winning game `backend` `bug`
#54 by rak3rman was closed 6 days ago

Rebuild frontend using Nuxt.js `backend` `enhancement` `frontend` 1
#51 by rak3rman was closed 6 days ago 4 tasks v2.0.0

Remember user nickname on player join `enhancement` `frontend` 1
#47 by rak3rman was closed on Mar 14 v2.0.0

Allow players to leave (kick themselves) from lobby `backend` `enhancement` `frontend`
#43 by rak3rman was closed 6 days ago v2.0.0

Create redundancies to prevent client lockup `backend` `bug` `enhancement` 1
#41 by rak3rman was closed on Mar 7 v2.0.0

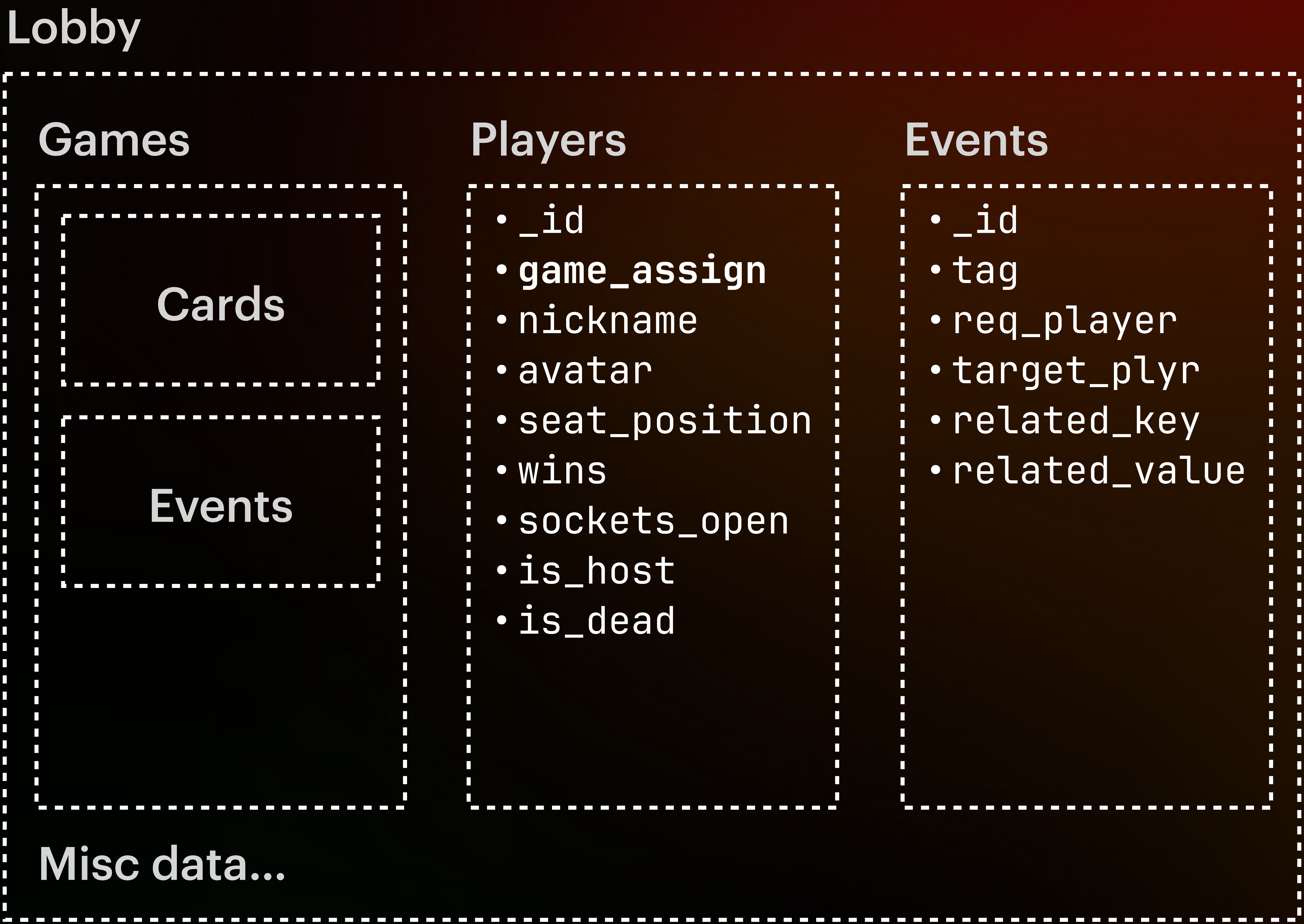
Rework favor framework `backend` `bug` `enhancement` `frontend` `high priority`
#39 by rak3rman was closed 19 hours ago v2.0.0

Implement lobbies `backend` `enhancement` `frontend` `high priority`
#38 by rak3rman was closed on Mar 7 v2.0.0

Isolate socket connections by group `enhancement`
#37 by rak3rman was closed on Nov 15, 2021 v2.0.0

Test cases are not exhaustive `backend` `enhancement` 1 4
#35 by rak3rman was closed on Mar 4 v2.0.0

Dark mode for game/lobby UI `enhancement` `frontend` 1
#33 by rak3rman was closed on Mar 7 v2.0.0



Players

- _id
- **game_assign**
- nickname
- avatar
- seat_position
- wins
- sockets_open
- is_host
- is_dead

Events

- _id
- tag
- req_player
- target_plyr
- related_key
- related_value

```
                await lobby_details.save();
                await socket_helpers.update_g_ui(lobby_details, game_pos, req_data.plyr_id, socket_id, undefined, undefined, action, io);
                await socket_helpers.update_l_ui(lobby_details, req_data.plyr_id, socket_id, undefined, action, io);
                callback(false, `Game has been ${chalk.dim.yellow('reset')}`, lobby_details, game_pos, req_data, action, socket_id);
            }
        ], wf_g_final_callback);
    })

    // Name : socket.on.play-card
    // Desc : runs when a card is played on the client
    // Author(s) : RAk3rman
    socket.on('play-card', async function (data) {
        let action = "play-card       ";
        console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan(action)} ${chalk.dim.yellow(da
        waterfall([
            async function(callback) {callback(null, data, action, socket.id)}, // Start waterfall
            wf_g_get, // Get game_details
            wf_g_validate_in_progress, // Validate we are in game
            wf_g_validate_turn, // Validate it is req player's turn
            wf_g_validate_lock, // Validate player is able to modify cards
            async function(lobby_details, game_pos, req_data, action, socket_id, callback) {
                // Play card
                let cb_data = game_actions.play_card(lobby_details, game_pos, req_data.card_id, req_data.plyr_id, req_data.target, stats_store);
                await lobby_details.save();
                // Throw err if play_card throws err
                if (cb_data.err) {
                    card_lock = false; callback(true, cb_data.err, lobby_details, game_pos, req_data, action, socket_id);
                } else {
                    await socket_helpers.update_g_ui(lobby_details, game_pos, req_data.plyr_id, socket_id, undefined, cb_data, action, io);
                    // Start explode tick if we are exploding
                    if (!cb_data.incomplete) await socket_helpers.explode_tick(lobby_details._id, game_pos, req_data.plyr_id, socket_id, undefined, 15,
                    card_lock = false; callback(false, `Played card ` + req_data.card_id, lobby_details, game_pos, req_data, action, socket_id);
                }
            }
        ], wf_g_final_callback);
    })

    // Name : socket.on.draw-card
    // Desc : runs when a card is drawn on the client
    // Author(s) : RAk3rman
    socket.on('draw-card', async function (data) {
        let action = "draw-card       ";
        console.log(wipe(`${chalk.bold.blue('Socket')}:  [` + moment().format('MM/DD/YY-HH:mm:ss') + `] ${chalk.dim.cyan(action)} ${chalk.dim.yellow(da
        waterfall([
            async function(callback) {callback(null, data, action, socket.id)}, // Start waterfall
```

```javascript
// Name : game_actions.play_card(lobby_details, game_pos, card_id, req_plyr_id, target)
// Desc : calls the appropriate card function based on card action, returns structured callback to be sent to client
// Target data structure : { plyr_id, card_id, deck_pos }
// Author(s) : RAk3rman
exports.play_card = function (lobby_details, game_pos, card_id, req_plyr_id, target, stats_store) {
    // Find card details based on card_id
    let card_details = card_actions.find_card(card_id, lobby_details.games[game_pos].cards);
    // Generate callback from data struct
    let callback = game_actions.generate_cb(undefined, card_details, undefined, target, false);
    if (card_details === undefined) { callback.err = "Invalid card action"; return callback; }
    // Ensure that the card is allowed to be played now
    let exp_only = ['defuse', 'hotpotato', 'chicken'];
    if (player_actions.is_exploding(card_actions.filter_cards(req_plyr_id, lobby_details.games[game_pos].cards)) && !exp_only.includes(callback.card.action
        callback.err = "Cannot be used while exploding"; return callback; // Player is exploding and player attempted to use a card that cannot stop a chic
    } else if (!player_actions.is_exploding(card_actions.filter_cards(req_plyr_id, lobby_details.games[game_pos].cards)) && exp_only.includes(callback.card
        callback.err = "Can only be used when exploding"; return callback; // Player is not exploding but player tried to use a card that can stop a chicke
    }
    // BASE DECK
    // istanbul ignore else (else condition covered above when callback is generated)
    if (card_details.action === "attack")               { card_actions.attack(lobby_details, game_pos, card_id, callback); }
    else if (card_details.action === "defuse")          { card_actions.defuse(lobby_details, game_pos, card_id, req_plyr_id, target, callback); }
    else if (card_details.action === "chicken")         { card_actions.chicken(lobby_details, game_pos, req_plyr_id, callback); }
    else if (card_details.action === "favor")           { card_actions.favor_random(lobby_details, game_pos, card_id, req_plyr_id, target, callback) }
    else if (card_details.action.includes("randchick")) { card_actions.favor_random(lobby_details, game_pos, card_id, req_plyr_id, target, callback); }
    else if (card_details.action === "reverse")         { card_actions.reverse(lobby_details, game_pos, card_id, callback); }
    else if (card_details.action === "seethefuture")    { card_actions.seethefuture(lobby_details, game_pos, card_id, callback); }
    else if (card_details.action === "shuffle")         { card_actions.shuffle(lobby_details, game_pos, card_id, callback); }
    else if (card_details.action === "skip")            { card_actions.skip(lobby_details, game_pos, card_id, callback); }
    // YOLKING AROUND EXPANSION PACK
    else if (card_details.action === "hotpotato")       { card_actions.hot_potato(lobby_details, game_pos, card_id, req_plyr_id, callback); }
    else if (card_details.action === "favorgator")      { card_actions.favor_gator(lobby_details, game_pos, card_id, req_plyr_id, target, callback); }
    else if (card_details.action === "scrambledeggs")   { card_actions.scrambled_eggs(lobby_details, game_pos, card_id, callback); }
    else if (card_details.action === "superskip")       { card_actions.super_skip(lobby_details, game_pos, card_id, callback); }
    else if (card_details.action === "safetydraw")      { card_actions.safety_draw(lobby_details, game_pos, card_id, req_plyr_id, callback) }
    else if (card_details.action === "drawbottom")      { card_actions.draw_bottom(lobby_details, game_pos, card_id, req_plyr_id, callback) }
    // Check if callback was successful (complete request and no errors)
    if (!callback.incomplete && !callback.err) {
        // Reached end of successful card execution, update events and statistics
        event_actions.log_event(lobby_details.games[game_pos], "play-card", req_plyr_id, target.plyr_id, callback.card._id, undefined);
        let stats_desc = card_details.action.includes("randchick") ? "randchick" : card_details.action;
        stats_store.set(stats_desc, stats_store.get(stats_desc) + 1);
    }
    return callback;
}
```

## Key Takeaways

- Do your research when picking a language
- Build a solid data structure
- Clearly define boundaries between parts of your project
- Give yourself room to build something better
- Visualize everything upfront

chickens.rakerman.com