Архитектура взаимодействия фронтенда и бэкенда в «Click & Cook»

1. Разделение логики между фронтендом и бэкендом

Фронтенд (Angular) – Клиентская часть

Что обрабатывается на фронте:

- Интерфейс пользователя (UI/UX).
- Клики игрока (увеличение денег и количества блюд в локальном состоянии).
- Визуализация изменений (анимации, таймеры, улучшения).
- Локальный кэш краткосрочных данных (например, последние секунды прогресса).
- Запросы на сервер для сохранения и загрузки данных.
- Управление временными бустами и эффектами в реальном времени.
- Обновление клиентского состояния на основе ответов сервера.

Бэкенд (Go) – Серверная часть

Что обрабатывается на бэке:

- Генерация уникальных игровых сессий для пользователей.
- Хранение и валидация данных (прогресс, улучшения, экономика).
- Верификация кликов и автоматических доходов для предотвращения читерства.
- Управление престижем и начислением бонусов.
- Обработка событий (час пик, известные гости, бонусы).
- Взаимодействие с базой данных.
- Управление пользовательскими сессиями и авторизацией.

2. Взаимодействие между фронтендом и бэкендом

Запросы от клиента к серверу

Общение между фронтендом и бэкендом можно реализовать через:

- **REST API** – для периодического обновления данных (например, получение прогресса, сохранение улучшений).
- **WebSockets** – для передачи в реальном времени изменений (например, автокликеры, временные бонусы).

Примеры API-запросов

Аутентификация и старт игры

- `POST /auth/register` – Регистрация нового игрока.
- `POST /auth/login` – Авторизация игрока.
- `GET /game/init` – Получение сохраненного состояния игрока или старт новой игры.

Основной игровой процесс

- `POST /game/cook` – Сообщение о клике для приготовления блюда (каждые 5 кликов для оптимизации).
- `POST /game/sell` – Сообщение о клике для продажи приготовленного блюда (каждые 5 кликов для оптимизации).
- `POST /game/upgrade` – Покупка улучшения.
- `POST /game/prestige` – Активация престижа.

Обновление данных в реальном времени (через WebSockets)

- `ws://game/progress` – Сервер отправляет данные о пассивном доходе каждые N секунд.
- `ws://game/events` – Сервер сообщает клиенту о событиях (фестивали, бонусы).

3. Пример работы сессии игрока

1. Игрок заходит на сайт → Фронтенд отправляет запрос `GET /game/init`.
2. Сервер проверяет, есть ли сохраненная игра:
   - Если **да** → Возвращает прогресс игрока.
   - Если **нет** → Создает новую сессию.
3. Игрок начинает кликать по блюду → Фронтенд фиксирует блюда локально.
4. Каждые 5 кликов фронт отправляет `POST /game/cook`, а сервер обновляет блюда.
5. Игрок начинает кликать по продаже → Фронтенд фиксирует продажи локально.
6. Каждые 5 кликов фронт отправляет `POST /game/sell`, а сервер обновляет баланс.
7. Игрок покупает улучшение → Фронтенд обновляет UI и отправляет `POST /game/upgrade`.
8. Сервер проверяет возможность покупки, списывает деньги и обновляет пассивный доход.
9. Автокликеры и события работают через WebSockets:
   - Сервер раз в N секунд отправляет клиенту новые начисления денег.
   - Если начинается событие, сервер рассылает уведомление игрокам.
10. Если игрок выходит и заходит позже → Сервер подгружает данные из БД.

4. База данных и хранение прогресса

Таблица пользователей:

id	username	email	password_hash	created_at
1	User	test@example.com	*****	2025-03-06

Таблица прогресса:

id	user_id	dishes	money	prestige_level	upgrades	last_active
1	1	100	5000	2	`{grill:2, chef:1}`	2025-03-06

Таблица событий:

id	user_id	event_type	status	expires_at
1	1	festival	active	2025-03-06

5. Оптимизация и предотвращение читерства

Чтобы предотвратить накрутку денег, сервер должен проверять данные:

- **Антиспам кликов** – Если игрок отправляет `POST /game/click` слишком часто, сервер может его заблокировать.
- **Серверная валидация улучшений** – Перед покупкой улучшения сервер проверяет баланс игрока.

Вывод

Таким образом, кликер можно реализовать следующим образом:

1. Фронт (Angular) отвечает за UI, анимации, локальное хранение кликов и обновлений.
2. Бэк (Go) управляет бизнес-логикой, сохраняет данные, обрабатывает запросы и отправляет обновления через WebSockets.
3. REST API используется для аутентификации, сохранения и загрузки данных.
4. WebSockets применяются для динамических обновлений и пассивного дохода.
5. База данных хранит прогресс игроков, улучшения, события и статистику.