Standalone Kiosk-Based Loyalty System Plan (Inspired by Fivestars)

Overview of the Service

Summary: The proposed loyalty service is a **standalone kiosk-based system** that allows customers to easily earn and redeem rewards at a physical store. A touchscreen kiosk (e.g. a tablet at the checkout counter) will serve as the customer interface. Shoppers simply **enter their phone number to check in**, and the system identifies them and updates their loyalty points balance. There are no cards or apps required at the point of sale – the phone number alone links the customer to their loyalty account blog.fivestars.com.

After checking in, the kiosk displays the customer's **accumulated points and available rewards**, and if they are eligible for a reward, they can redeem it on the spot. The entire process is designed to be extremely quick (a few seconds), ensuring it doesn't slow down the checkout line. In the background, a cloud-based backend records each visit and maintains the customer's points tally and reward redemption history.

Comparison to Fivestars: This kiosk system is inspired by the success of **Fivestars**, a popular loyalty and marketing program for small businesses. Fivestars allows customers to join a store's loyalty program by simply typing their phone number into a tablet, with no additional signup hassle

It has demonstrated that minimizing friction in loyalty sign-up leads to high adoption – customers *"earn rewards every time they check in"* with just their phone number, so they enjoy participating fivestars.com.

Our system adopts this proven approach of phone-number-based check-in and digital rewards. However, it differentiates itself in a few key ways:

- **Independence & Branding:** Fivestars operates a nationwide network of businesses and consumers (with 70 million users across 12,000+ businesses), and Fivestars branding is part of the customer experience. In contrast, our kiosk solution is a **white-label loyalty program** for a single business or franchise. The business can brand the kiosk interface and loyalty program as their own. Customer data is kept within the business's own system rather than a third-party network, which can appeal to merchants who want more control over their loyalty program and customer relationships.

- **Focus on Core Loyalty (Simplicity):** Fivestars is an all-in-one platform that includes payments processing, marketing automation, and detailed segmentation tools in addition to loyalty rewards. While powerful, that breadth can be overwhelming or costly for some small businesses. Our system will focus on the **core loyalty features first** – easy check-in, points tracking, and reward redemption – delivered in a cost-effective package. This makes the initial rollout simpler and more affordable, while still reaping the benefits of increased customer retention. Advanced marketing features (customer messaging, campaigns, etc.) can be added in later phases as needed, à la carte.

- **Standalone Kiosk Implementation:** Fivestars' solution can integrate with point-of-sale systems or use a customer-facing tablet at checkout. Our loyalty system will initially be delivered via a **dedicated kiosk device** at the store, requiring minimal integration with existing POS systems. This reduces technical setup complexity in Phase 1. The kiosk will function alongside any checkout process – for example, a customer can check in on the kiosk while the cashier rings up their purchase. In future phases, deeper integration with POS or payment systems can be introduced, but the Phase 1 approach is hardware-agnostic and easy to deploy.

- **Instant Gratification for Customers:** Both Fivestars and our system reward customers for their visits/purchases, but our kiosk will put an emphasis on **immediate feedback and rewards**. As soon as a customer checks in, they see their updated points and can redeem any available reward right then and there. This immediacy encourages usage. (Fivestars also supports instant reward redemption; our differentiation is mainly in the presentation and standalone nature of the kiosk.) By giving customers a quick win (e.g. "You earned 10 points today!" or "You have enough points for a free item now."), the system builds positive reinforcement for repeat business.
  .

In summary, the kiosk-based loyalty system works much like Fivestars in terms of customer experience (simple phone-number check-in to earn points), but it is designed as a **standalone service tailored to the merchant's brand**, with a phased approach to adding features. This provides a quick-to-launch loyalty solution that can later grow into a fuller marketing platform. The table below contrasts the two:

- *Fivestars:* Third-party loyalty network, full suite (payments, marketing, etc.), used by many businesses with one shared customer network, subscription-based service.
- *Our Kiosk System:* In-house loyalty program for one business or chain, focused on loyalty points and rewards (initially), not tied to any payment system, and fully under the business's branding and control.

By learning from Fivestars' strengths – such as ease of use and proven boost to customer return rates – and by addressing some merchants' desire for a simpler or more proprietary solution, this kiosk loyalty system aims to increase customer retention and engagement in a manner ideally suited for the business's needs.

Kiosk System Features (Phase 1)

In Phase 1, the system will deliver the essential features needed to run a digital loyalty program through an in-store kiosk. The focus is on simplicity and speed for both customers and store staff. Key features include:

- **Phone Number Check-In:** Customers can **enter their phone number on the kiosk** to check in and earn loyalty points for their visit/purchase. The kiosk provides a large on-screen numeric keypad for easy input. This process is extremely fast – typically just a few seconds – and does not require any plastic cards or scanning; **the phone number alone identifies the customer**. If the phone number is not yet in the system, the backend will automatically create a new customer entry (enrolling them in the loyalty program) with that phone number as the identifier. There is **no separate sign-up form** to fill out at the kiosk, keeping the process frictionless.

- **Points Balance Display:** After a successful check-in, the kiosk screen will **display the customer's current point balance** and their progress toward the next reward. For example, it might say "Welcome back! You have 45 points." If the loyalty program uses points-per-dollar, the system will have calculated the points earned from the current purchase (if integrated with the transaction) or from the check-in (if a per-visit point scheme) and added it to their total. The interface will clearly indicate any new points just earned ("+5 points added for today's visit!") as well as the **total accumulated points**. This immediate feedback taps into the gratification that makes loyalty programs effective.
  .

- **Available Rewards & Redemption:** The kiosk will also show any **rewards the customer is eligible to redeem** based on their points. For instance, "You qualify for: $5 off (requires 50 points) – **Redeem now?**". Customers can choose to redeem a reward on the spot if they have enough points. The interface would allow them to select the reward and confirm redemption. **If a reward is redeemed**, the system will instantly deduct the required points from their balance and display a confirmation (e.g. "Reward redeemed: $5 off! Please notify the cashier to apply your discount."). The kiosk could show a simple code or message to the cashier (or send a notification to the merchant's system) to validate the reward redemption in real-time. This instant redemption feature ensures customers can **enjoy their rewards immediately**, increasing satisfaction with the program. It also saves staff time by letting customers self-serve their reward choice.

- **No Login/Password Required:** There is **no need for customers to log in with an account or remember a password** at the kiosk. The phone number entry **is the only step needed** to identify and access their loyalty info. This lowers the barrier to use – even first-time customers can simply input their number and be enrolled without any prior setup. (If the business wants to gather more info like name or email, that can be done optionally via the mobile app or a separate prompt, but **Phase 1 will not force any additional data entry** at the kiosk beyond the phone number). The system may prompt

a new user to agree to basic terms (e.g. “By participating, you agree to our loyalty program terms…”) likely on first use, but this will be a single tap acknowledgment, not a form. All of this ensures maximum convenience, much like Fivestars’ philosophy that “all [customers] need is their phone number” to earn points.

- **Simple, Fast Interface:** The kiosk’s user interface will be **clean and fast**, designed for quick interactions so that it doesn’t create any waiting line. The design will use large buttons and text, minimal steps, and clear instructions. For example, the main screen might always be ready for the next customer with a message like “Enter your phone number to earn rewards!” and a keypad. Once a customer enters their number, the system instantly (in a second or two) pulls up their info and shows points/rewards. If no action is required (e.g., they are not redeeming a reward), the check-in is automatically recorded and the screen might say “Got it! You earned 5 points today. You now have 45 points.” and after a brief confirmation, return to the start screen for the next customer. The entire flow is optimized to take **maybe 5-10 seconds total** for a returning customer, and perhaps 10-15 seconds if a customer is new or redeeming a reward. This speed means even at peak times, the loyalty check-in won’t bottleneck the checkout process. Store staff also benefit – since the customer handles the check-in on the kiosk, the cashier can focus on ringing up the sale simultaneously (which mirrors the benefit Fivestars observed: *“cashiers are no longer required to handle the entire signup process… which saves time and frees up lines”*).

- **Backend Points Tracking:** Behind the scenes, the system will maintain a **secure backend database to store and update loyalty points** for each customer. When a customer checks in, the kiosk app calls an API (over the internet) to the backend service, providing the phone number (and possibly the purchase amount or items, if needed for point calculation). The backend will look up the customer’s record (or create one if new), calculate the points earned for that visit (based on the merchant’s defined rules), update the point balance, and return the updated info to the kiosk. The backend also keeps track of **reward redemption history** (so a reward can only be redeemed once per required points, and staff can verify if needed). All of this happens near-instantly so that by the time the kiosk displays the info, the database is already updated. The data model will allow configuration of different point rules and reward tiers for the business (e.g. 1 point per $1 spent, or 10 points per visit – these rules are set in the backend configuration for that merchant). Importantly, **all check-in and redemption events are logged** with timestamps, which helps with auditing and fraud prevention. For example, if someone tries to exploit the system by checking in multiple times without purchases, the merchant can review logs, and rules can be put in place (such as “only one check-in per customer per hour”) to prevent abuse. Initially, the backend will be relatively simple: it links phone numbers to point balances and triggers reward eligibility. But it’s built in a way that can later support more complex logic (multiple stores, referral credits, etc. in Phase 2).

Overall, Phase 1 delivers a **self-service loyalty kiosk** that mirrors the basic functionality of big-store loyalty programs (like those at CVS/Walgreens or the Fivestars tablet) – phone-based identification, automatic point accrual, and easy reward redemption. This will immediately provide value by converting paper punch cards or ad-hoc loyalty tracking into a robust digital system. Even at launch, the system is expected to increase customer sign-ups and engagement significantly – for instance, Fivestars reported that introducing a customer-facing sign-in tablet increased loyalty sign-ups by 50% on average. We anticipate similar boosts in participation thanks to the kiosk's visibility and ease of use, turning more first-time shoppers into repeat customers.

Future Expansion Considerations (Phase 2 and Beyond)

While Phase 1 focuses on the core in-store experience, the platform is envisioned to expand into a more comprehensive loyalty and customer engagement ecosystem. Future phases will build on the foundation to provide additional convenience for customers and powerful tools for merchants. Key expansion areas include:

- **Mobile App & Full Customer Portal:** In Phase 2, we plan to introduce a **mobile app (and/or web portal)** that complements the in-store kiosk. This app will allow customers to **register an account with more details** (e.g. name, email, birthday) if they choose, enabling a richer experience. With the app, users can **view their point balance and reward history remotely**, check their progress toward rewards, and even find nearby store locations or store hours. A mobile app also opens the door to sending push notifications or in-app messages for promotions ("You have a new offer!"). Additionally, through the app or portal, customers could manage preferences – for example, opting in/out of marketing communications, or linking their account with social media or payment methods. Essentially, the app provides a **full-featured loyalty wallet** for the customer: they can see all past transactions/visits, look up available rewards, and get notified of special bonuses (like birthday rewards or limited-time double points events). This goes beyond the kiosk's on-the-spot info by giving the customer a persistent connection to the loyalty program. Importantly, the app would use the same backend account – likely keyed by the phone number – so when a user logs in to the app (possibly via a one-time password SMS to verify their number), they will access the same profile they use in-store. This means if they prefer not to download the app, they can still fully participate via the kiosk with just their phone number (maintaining the low barrier to entry). But the app will enhance engagement for those who want deeper involvement.

- **Enhanced Customer Engagement Features:** With more customer data and an app in place, Phase 2 can introduce **marketing and engagement tools** similar to those offered by Fivestars' marketing automation. For example:

    - **Personalized Promotions & Messaging:** The system could send automated **birthday rewards** (e.g. "Happy birthday! Enjoy a free dessert on us this week.")

or **win-back offers** to customers who haven't visited in a while. These can be delivered via SMS, email, or push notification depending on the customer's contact info and consent. (Fivestars' AutoPilot feature sends such messages and is *"10X more effective"* than generic marketing; we can implement a tailored version of this for our program.)
    - **Referral Program:** Introduce a refer-a-friend system where existing members get bonus points for referring new customers (and the new customer might get a signup bonus). The mobile app could generate referral codes or links to facilitate this.
    - **Tiered Rewards or VIP Status:** Evolve the one-size loyalty program into a **tiered loyalty system** (e.g. Silver, Gold, Platinum tiers) where customers who visit/spend more get higher perks. This encourages even greater loyalty. Such tiers and their benefits can be managed in the backend and communicated via the app and email.
    - **Surveys and Feedback for Points:** Optionally, allow the business to send a short survey after a visit (through the app or email) and reward customers with a few extra points for completing it. This provides valuable feedback to the business while increasing customer engagement.
    - **Gamification:** Add gamified elements like achievement badges ("5 visits this month – you're a Coffee Connoisseur!") visible in the app, or a progress bar on the kiosk screen, to make the experience fun.
- The goal of these engagement features is to **increase visit frequency and customer loyalty** by keeping the business at the top of the customer's mind. Many of these features will rely on the data collected (visit frequency, spend, preferences) to target the right customer at the right time – for example, sending a coupon to a lapsed customer to bring them back. While Phase 1 lays the data groundwork, Phase 2 leverages it for smart engagement.

- **Integration with External Systems:** As the loyalty program grows, integration with other business systems will become valuable:

    - **Point-of-Sale (POS) Integration:** To streamline operations, the loyalty system can tie into the merchant's POS software. For instance, when a sale is finalized, the POS could send the transaction amount to the loyalty backend to automatically add points, eliminating even the need for the customer to manually use the kiosk (they might just tell the cashier their phone number, or have a scannable loyalty QR code). Conversely, if a reward is redeemed in the app, the POS could be alerted to apply the discount. (Fivestars already does some of this with their "Fivestars Connect" which can run on the POS, so it's a logical next step for efficiency.)
    - **CRM and Marketing Tools:** The loyalty platform could export or sync customer data with CRM systems or email marketing platforms. For example, the business might use Mailchimp or Constant Contact – the loyalty system can periodically update mailing lists with new sign-ups or segment customers by behavior.

Integration with a CRM also allows the business's marketing team to see loyalty activity alongside other customer data, giving a 360° view of the customer. We could provide APIs or data feeds to make this possible, or even pre-built connectors for popular CRM systems.

    - **Payment System Integration:** In later phases, we can consider integrating the loyalty function with payment platforms. One idea is to partner with a payment processor (or leverage something like **SumUp/Square** integration) such that when customers pay with a credit card or mobile wallet, their loyalty account is automatically found by matching a phone or card number, awarding points seamlessly. Another idea is enabling **pay with points** – allowing customers to redeem points for cash value toward a purchase. This requires careful integration to ensure the loyalty balance and payment process communicate. Fivestars itself expanded into payments (after being acquired by SumUp) to connect loyalty with transactions; our system could choose to integrate rather than build a payments solution from scratch, using APIs of payment providers.
    - **E-Commerce Integration:** If the business has an online store, the loyalty system can be extended to online purchases as well. Phase 2 could include APIs or plugins for the business's website so that online orders also grant loyalty points, and customers can redeem rewards online. This provides an omnichannel loyalty experience.
- **Multi-Location and Franchise Support:** If the business has multiple store locations or franchisees, the system can be expanded to handle that. This means a customer's phone number might be recognized across all locations, and their points accrue across the chain (or possibly separately per location group, depending on business preference). We would implement a way to designate location or merchant IDs in the backend. Admin tools would allow a corporate owner to see all data, while a specific franchise location manager might only see their store's customers. This multi-location capability would likely come in Phase 2 once the single-store model is proven. It adds complexity in data management but is important for scaling the service to larger businesses.

- **Analytics and Insights:** As more data accumulates, future versions can include an **analytics dashboard** for merchants that provides insights like customer retention rates, average visit frequency, most redeemed rewards, etc. While basic reporting (total members, points issued, points redeemed) will exist from the start, Phase 2+ could employ data analysis to highlight trends or even use AI to predict customer churn and suggest targeted campaigns. This turns the loyalty program data into actionable business intelligence.

All these future enhancements will be built on top of the Phase 1 core. We plan them in a modular way – for example, businesses that just want a simple points program can stick with the kiosk only, whereas businesses looking for full engagement can opt into the mobile app and marketing add-ons. By outlining these Phase 2 ideas now, we ensure the Phase 1 architecture is flexible enough to accommodate them. This forward-looking approach will help the system

evolve similarly to Fivestars (which started with loyalty points and grew into a marketing platform) but with the ability to pick and choose features based on the merchant's needs.

Technical Architecture

The technical architecture of the kiosk-based loyalty system will leverage **cloud services (AWS)** for scalability, security, and ease of management. Below is an overview of the planned architecture components and how they interact:

- **Cloud Backend on AWS:** The backend of the loyalty system will be hosted on Amazon Web Services. We will use a **serverless and API-driven architecture**:
    - **AWS API Gateway:** This will provide a secure RESTful API interface for the kiosk (and future mobile app) to communicate with the backend. All requests from the kiosk (such as check-in with phone number, fetch points, redeem reward) go through API Gateway endpoints. API Gateway can handle authentication, throttling, and monitoring of API calls easily.
    - **AWS Lambda:** The business logic will run on Lambda functions triggered by the API Gateway. For example, a "CheckIn" Lambda function will receive the phone number and (maybe purchase amount) from the API call, then execute the logic: lookup or create customer, calculate points, update database, and return the response. Lambda is chosen for its scalability and cost-effectiveness – it can automatically scale to handle multiple kiosks or heavy usage periods without us managing servers. Additionally, Lambda allows writing the logic in our choice of language (Node.js, Python, etc.); we'll choose a language that the team is comfortable with (Node.js is a common choice for quick API development, but Python or Java would also work).
    - **AWS DynamoDB (NoSQL Database):** We intend to use DynamoDB as the primary database to store loyalty data. DynamoDB offers fast lookups by primary key (ideal for retrieving a customer by phone number) and is fully managed. We can design a DynamoDB schema such as a table `Customers` keyed by phone number (or a surrogate customer ID) which stores the point balance and basic info, and another table `Transactions` for point transactions (with attributes for merchant/store, timestamp, points added or redeemed, etc.). Alternatively, we might use a relational database (Amazon RDS/Aurora) if complex queries are needed (especially in Phase 2 for analytics). For Phase 1, DynamoDB's simplicity (schema-less design) and ability to auto-scale throughput make it attractive. Each time a customer checks in or redeems, the corresponding item in DynamoDB is updated. We will enable DynamoDB's encryption at rest to protect data, and use consistent reads/writes when needed to ensure the latest data is fetched (important to avoid race conditions if, say, two check-ins happen quickly).
    - **AWS Cognito:** Cognito will be used for managing user identities where appropriate. In Phase 1, customers are identified by phone number only (with no

password), so we might not use Cognito for them initially (to avoid adding friction). However, **for the merchant/admin interface and for the Phase 2 mobile app**, Cognito is useful. We can create a Cognito User Pool for merchants to securely log into an admin dashboard (with email/password or SSO) and possibly for customers who choose to create full accounts in the mobile app. Cognito also can handle sending verification codes (useful if we do OTP via SMS for phone verification in the app) and store profile info. By planning for Cognito integration, we ensure the system is ready for a more traditional login system if needed later.

    - **AWS S3 and CloudFront:** The kiosk application front-end (if it is a web application) can be hosted on S3 as static web content (HTML/JS/CSS), served via Amazon CloudFront CDN for low-latency access at each store. This way, the kiosk just needs a web browser pointing to a URL (which fetches the latest version of the app). If we develop the kiosk app as a native app (Android/Windows), S3/CloudFront might be less relevant, but we could still use S3 for hosting images or configuration files. CloudFront will ensure any content is delivered quickly and can enforce HTTPS.
    - **AWS SNS (Simple Notification Service) or Amazon Pinpoint:** As we move into Phase 2 with mobile app notifications or SMS/email campaigns, AWS SNS or Pinpoint can be used. SNS could send SMS messages (for things like verification codes or simple reward notifications). Amazon Pinpoint is a service specifically for user engagement messaging (SMS, push, email) and could support the marketing campaigns (birthday messages, etc.) similar to Fivestars' automated messaging. For the initial phase, this is not heavily used except perhaps for sending a receipt or signup confirmation via text if desired.
    - **AWS CloudWatch & X-Ray:** For monitoring and logging, we will use CloudWatch to collect logs from Lambda functions and track API metrics. AWS X-Ray can be enabled to trace requests through the system, which is useful in debugging performance issues (though our architecture is relatively simple, tracing can still help if multiple Lambdas or services are involved in one call).
    - **AWS IAM & Security:** All these services will be tied together with proper IAM roles to ensure least-privilege access. For example, the Lambda functions will have an IAM role that only allows read/write to specific DynamoDB tables, and nothing else. API Gateway will have usage policies to prevent abuse (e.g., rate limiting if somehow a device started spamming requests). Data in transit will be protected by HTTPS (API Gateway endpoints will require TLS). Data at rest (in DynamoDB or S3) will be encrypted using AWS-managed keys (KMS). We will also back up data (DynamoDB has on-demand backup features) to protect against accidental data loss.

- **Database Design:** The system's data model needs to handle customers, points, rewards, and possibly multiple merchants or locations. A simplified schema for Phase 1 might be:

    - **Customers Table:** Keyed by phone number (or a unique customer ID). Contains fields like `phone`, `name` (optional, if collected later), `totalPoints` (or we might calculate from transactions), and maybe a list of rewards earned/redeemed. If using DynamoDB, phone number can be the primary key (assuming within one merchant or globally unique across system). If multi-merchant, the primary key could be a composite of `{merchantID, phone}` to segregate points per business. Alternatively, a relational model would have a separate **Customer** table and a **PointsBalance** table linking customer to merchant with their current points balance.
    - **Transactions/Activity Table:** Records each check-in or redemption event. Fields: `activityID`, `merchantID`, `phone` (or customerID), `dateTime`, `pointsChange` (+10 points or -50 points for redemption), `activityType` ("earn" or "redeem", possibly "adjustment"), and maybe `notes` (e.g. "Purchase $10 = 10 pts" or "Redeemed Free Coffee reward"). This provides an audit trail and is useful for generating reports (e.g. how many points given out vs redeemed).
    - **Rewards Catalog:** A table or configuration that defines the **reward rules** for the loyalty program. For example, it might list: "Free Coffee = 50 points", "$5 Off = 50 points", "Free Pastry = 30 points" – whatever the business chooses. It may also define earning rules: "1 point per $1" or "10 points per visit". In Phase 1, this could be a simple config file or a database table that the backend logic references when calculating points and checking reward eligibility. Making it data-driven allows the merchant to change their reward offerings without code changes (via an admin interface).
    - **Merchant Table:** (If supporting multiple merchants or locations) Stores info about each business client using the system, including store name, address, perhaps settings like business hours (if we restrict certain promotions by time) and the specific loyalty program settings (point ratios, etc.). If it's just one business with one location in Phase 1, we might not need this yet, but planning for it is wise if we intend to onboard multiple businesses or a franchise scenario.
    - **User Accounts (for App):** In Phase 2 when we introduce full customer accounts and a mobile app, we may have a **Users** table or use Cognito to store account details like hashed passwords or OAuth tokens for those who sign up with email/social login. For now, this is not in Phase 1 scope, since phone number is used directly without a password.
- The database must ensure **atomicity** for point updates (to avoid race conditions where two simultaneous check-ins double-add points). Using DynamoDB, we can take advantage of atomic update operations or transactions (for example, a conditional update that ensures we don't override a newer balance). In a relational DB, we'd wrap updates in transactions. Also, we'll implement **data validation** in the backend (e.g. ensure points don't go negative beyond redemption, etc.).

- **API Interactions (Kiosk ↔ Backend):** The kiosk app will communicate with the backend exclusively via the defined APIs. Key API endpoints include:

    - `POST /checkin`: The kiosk calls this when a customer enters their phone number (and possibly enters a sale amount or selects from preset options if needed). The payload could be `{ phone: "5551239876", storeId: "ABC123", purchaseAmount: 25.00 }` (purchaseAmount if we're calculating points per dollar; if points per visit, the amount might not be needed). The backend (Lambda) will process and respond with the customer's updated info. The response might include `{ newPointsEarned: 5, totalPoints: 45, rewardsAvailable: [ ... ] }`. The kiosk then displays the info.
    - `POST /redeem`: Called when a customer chooses to redeem a specific reward. Payload: `{ phone: "5551239876", rewardId: "FREECOFFEE" }`. The backend will verify the customer has enough points, deduct points, mark the reward redemption (possibly in the Transactions log), and respond with a confirmation (or an error if something is wrong, e.g. not enough points or reward already redeemed). If successful, the response can include a one-time code or confirmation that the kiosk can display like "Code 1234 applied" or simply success true which the kiosk translates to a message.
    - `GET /customer?phone=...`: (or this could be combined with checkin) to retrieve current points and rewards for a given customer without modifying anything – useful if, for instance, the kiosk wants to just query info (but in most cases, the check-in itself will be the trigger that also returns info).
    - `POST /signup` (optional for Phase 1): If we wanted to allow proactively adding a new customer with more info (like name/email) at some point, though we said no extra registration at kiosk in Phase 1. This might be used by an admin to import customers or by future mobile app to create account.
    - `GET /rewards` or `GET /program`: for the kiosk to display the list of rewards and maybe program info (if we want to show "Earn 1 point per $1" somewhere on the UI). This could also be hard-coded in the kiosk for Phase 1 to simplify.
- All API calls will be **authenticated and authorized** appropriately. For the kiosk in Phase 1, since it's a trusted device within the store, we might use an API key or an IAM role for the device to call the API (for example, if the kiosk is a web app, we can embed a token that's required for calls, or restrict by the store's IP address if known – though IP is not very reliable). In Phase 2, when we have user-facing APIs (mobile app), we would implement OAuth or JWT tokens via Cognito to secure those endpoints (so that one user can't query another's data, etc.).

  The communication will be HTTPS to the API Gateway domain. We will also implement basic **input validation** on all endpoints (e.g., phone number must be 10 digits, rewardId must be one of allowed values, etc.) to guard against malformed requests or malicious input. Since the kiosk is publicly accessible, it's important that even if someone tried to

tamper with it or use the network, the backend remains secure.

- **Security & Data Privacy:** Security is paramount since we are dealing with personal data (phone numbers, and later possibly names, emails, purchase history). Key security measures in the architecture:

    - **Data Encryption:** All sensitive data (customer personal info, phone numbers, etc.) will be encrypted at rest. AWS services like DynamoDB, S3, RDS all support encryption with AWS KMS. We will ensure this is enabled. In transit, as mentioned, all APIs use HTTPS/TLS. If any data is cached on the kiosk device, we will avoid caching sensitive info; the kiosk app should not store data long-term locally.
    - **Least Privilege:** Using AWS IAM, the system components only have the minimum permissions needed. The kiosk's credentials can only invoke the specific API endpoints, not access databases or other resources directly. The Lambda role can only access the one DynamoDB table or one S3 bucket it needs, etc. This way, if any component is compromised, the blast radius is limited.
    - **PII Protection:** A phone number is personally identifiable information, so our database and logs will treat it carefully. We won't expose full phone numbers unnecessarily. For example, the merchant's dashboard might partially mask phone numbers or use an identifier unless the merchant needs the actual number for contact (which they likely do to reach the customer). If emails are collected, we'll likewise protect them. We will include a privacy policy in the program that explains how customer data is used (e.g. "used only for tracking your rewards and opt-in communications, not sold to third parties"). Compliance with regulations like GDPR/CCPA will be considered if applicable (e.g. giving customers a way to opt-out or delete their data, especially if this extends to larger markets).
    - **TCPA Compliance for SMS:** If we use text messaging to communicate rewards or verification, we must follow text messaging laws (as Fivestars does, marking their program as TCPA-compliant). This means we should obtain explicit consent from customers before sending marketing texts. For instance, when first signing up, the kiosk could have a line like "By participating, you agree to receive occasional automated text messages about rewards (Standard rates apply). Text STOP to unsubscribe." This legal language might be shown on first sign-up or printed on the receipt. We will store the consent status in the database. Customers who opt out should be respected in our messaging systems.
    - **Fraud Prevention:** We will enforce rules to prevent users from gaming the system. Because phone number is the identifier, one risk is someone could enter other people's phone numbers. There's not much value in doing so (you'd be giving someone else points), but to mitigate any confusion, we might implement optional verification (e.g. sending an SMS code on first sign-up to confirm the phone belongs to the person). However, that introduces friction, so it might not be

default in Phase 1. We rely on the fact that a user wants their own points, so they'll enter their own number. For repeated check-ins without purchases, the business can monitor and remove fraudulent points. Integrating with the POS will help tie check-ins to actual transactions to avoid abuse. Every redemption will be recorded with date/time, and possibly the cashier can be required to approve it in the POS, so a customer can't redeem the same reward multiple times on the kiosk (the system will mark it used).

- **Scalability and Reliability:** By using AWS managed services, the system can scale and is resilient. DynamoDB and Lambda are multi-AZ by default, meaning even if one data center goes down, the service continues. We will also implement health monitoring – if the kiosk cannot reach the backend (e.g. internet outage), it should handle it gracefully (maybe queue the check-in to retry, or inform the customer to try again later). The data on backend is safe with backups, and we can restore if needed.

In summary, the technical architecture uses a **modern cloud stack**: a thin client (kiosk app) communicating with a robust cloud backend. The choice of AWS services ensures we can start small (only paying for what we use) but seamlessly scale up as more stores or features come online. The design is API-centric, which will make it easier to develop additional front-ends (mobile app, admin dashboard) against the same backend. Security and data integrity are built into each layer, to protect customer trust and business data.

UI/UX Design for the Kiosk System

For the kiosk-based system, the **user interface and user experience** are critical to its success. The design must be extremely easy to use, quick to navigate, and accessible to a wide range of customers (young, old, tech-savvy or not). We will employ a **kiosk-optimized UI/UX design** with the following principles and features:

- **Inviting Start Screen:** The idle screen of the kiosk will feature a clear call-to-action, such as *"Join our Rewards Program – Enter Phone Number to Begin"* in large text. It may include the store's branding (logo and colors) and a simple graphic or icon indicating loyalty/rewards. This screen should attract customers' attention when they approach the counter. If the kiosk is not in use for a while, it could play a short attract loop or just stay brightly lit with the prompt so people notice it. Simplicity is key: one primary action (enter your phone) is presented.

- **Large Keypad for Phone Entry:** As soon as the customer starts the check-in process, the screen will display a **numeric keypad** that takes up most of the screen for easy tapping. Each key will be large enough to hit accurately, even on a small tablet, and high contrast (e.g. white numbers on dark buttons). We will include a prominent **"Enter" or "Submit"** button and a clear **"Backspace"** for corrections. Given that phone numbers are typically 10 digits (in the US), the interface might visually format the input (e.g. `(555) 123-4567`) as they type, to reassure the user that it's capturing correctly. The

design will also consider if international numbers are allowed (if this is a global product, we might include a country code selector, but for simplicity Phase 1 might assume a fixed country). The focus is that **any customer can quickly input their number without confusion**. No tiny text fields or need for a physical keyboard – it's all touch-friendly.

- **Feedback During Entry:** As part of UX, when the phone number is submitted, there should be immediate feedback. For example, after hitting "Enter", the screen might show a loading spinner or progress bar for a second with a message "Checking your rewards...". This assures the user the input was received. The system is designed to respond very fast (a second or two), but a tiny loading indicator helps in case of any delay. If the number was invalid (e.g. too few digits), the UI will prompt the user to correct it ("Please enter a valid 10-digit phone number") rather than just doing nothing.

- **Display of Points and Rewards:** Once the backend responds, the kiosk will show a **summary of the customer's loyalty status**. This screen will be simply structured, for example:

    - A greeting like "Hi [Name]!" if the name is known from a previous sign-up (if not, it could say "Hi there!" or just skip a name). Using the name can personalize the experience.
    - **Point Balance:** in large font, e.g. "You have 120 points." Possibly accompanied by a graphical element like a points meter or progress bar indicating how close they are to the next reward threshold. For instance, "120/150 points towards *Free Meal*".
    - **Available Rewards:** If the customer has enough points to redeem one or more rewards, those will be shown clearly. e.g. a box or button that says "🎁 Free Coffee (50 points) – Tap to Redeem". We will highlight one reward at a time if multiple choices; perhaps the best/highest value reward first. If they have no rewards yet, the UI can show "Keep earning points to get rewards! 30 more points to your first reward." or list the typical rewards available ("Rewards: 50 points = Free Coffee, 100 points = Free Sandwich," etc.) to motivate them.
    - **Redeem Option:** For any reward they can claim, a bright button or icon will invite them to redeem. The user can tap it to initiate the redemption. If they don't want to redeem this visit (maybe saving points for later), they can just ignore it – the screen might have a "Not now" or "Skip" option or simply instruct them to press "Done" or wait a moment to finish.
- The layout will be designed so that all key information (points and one reward option) fits on one screen without requiring scrolling. We will use **simple language and possibly friendly icons** (e.g. a star or gift icon for rewards, a coin icon for points) to make it visually intuitive.

- **Reward Redemption Flow:** If the customer chooses to redeem a reward by tapping redeem, the UI will ask for **confirmation** (to prevent accidental redemptions, since once

used, points will be subtracted). The confirmation screen might say “Redeem *Free Coffee* for 50 points? You will have 70 points left.” with two buttons: “Confirm” and “Cancel/Back”. Upon confirming, the kiosk will process the redemption (calling the API) and then show a **success screen** like “Success! You’ve redeemed a Free Coffee. Please show this to the cashier.” The success screen might show a redemption code or simply instruct the user that the cashier has been notified (depending on how we implement verification). The UI should make it clear that the reward has been used and cannot be reused. Perhaps it could display a big checkmark or confetti animation to celebrate the reward – a bit of delight in the UX can reinforce the positive experience. After a few seconds, the kiosk can automatically return to the home screen, or have a “Done” button to reset.

    - If the redemption fails for some reason (maybe the user just redeemed on another device or a network issue), an error message will be shown (“Oops, something went wrong. Please ask our staff for help.”) and the transaction will be aborted or retried as appropriate. We will design error states to be user-friendly and not expose any technical jargon.
- **No Keyboard or Complex Input:** The UI will **not require typing out names, emails, or passwords on the kiosk**, which can be slow and error-prone on a touchscreen. All of that is deferred to a later mobile app or handled by staff if needed. The only input on the kiosk is numeric (phone number or possibly selecting a reward). This ensures even users who aren’t tech-savvy or who feel uncomfortable typing on a public device will find the process easy. It also reduces the chance of mistakes. If a mistake does happen (like wrong phone number entered), the UI should allow them to correct it or start over easily (perhaps a "Back" button or just entering the number again).

- **Accessibility & User Friendliness:** The kiosk application will follow accessibility best practices:

    - **High Contrast and Legible Text:** Text will be large and in colors that have strong contrast with the background (e.g. dark text on a light background or vice versa) for readability. Important numbers (like point totals) might be in an even larger font or bold.
    - **Color and Icon cues:** We will not rely solely on color to convey meaning (for colorblind users). For example, if we use green text to indicate something positive, we’ll also have a "+" sign or the word “earned” so it’s clear without color. Icons (like a gift icon for reward) help those who may not read the language well.
    - **Touch Target Size:** All buttons (the keypad digits, the redeem button, etc.) will be designed according to touchscreen guidelines (e.g. at least ~7mm or 48px in size) to accommodate people with larger fingers or limited dexterity.
    - **Time to Read:** We will ensure any message stays on screen long enough for a slow reader to understand. For instance, if auto-returning to home screen after a check-in, give maybe a 5-8 second delay so the user can read “You earned 5 points” message. Perhaps even include a progress countdown or require them to

tap “Done” to proceed, so they feel in control. This avoids situations where an elderly customer might be still reading their points when the screen resets.
    - **Multi-language support (future):** In communities where multiple languages are common, we may allow switching the language on the kiosk (a simple toggle on screen to change text to Spanish, for example). In Phase 1, the text is minimal and could potentially be dual-language by design (like the prompt could show English and a second language smaller). This will depend on the deployment environment.
    - **Physical Accessibility:** The kiosk tablet should be mounted at an accessible height (following ADA guidelines if in the US, typically around 48 inches max height for reach). The screen angle should be adjustable to be visible from wheelchair height. If the device has any physical buttons or a card reader, those should also be within reach, but likely it’s just a screen. We will also consider adding a **sound or voice prompt** mode in future if needed for visually impaired users – e.g. tapping a certain corner could switch to a mode where the device speaks instructions and accepts voice input or bigger gestures. That might not be in initial release, but designing with an eye towards such inclusion is beneficial.
- **Flow for New Customers:** If a customer is completely new (their phone number wasn’t found), the system will still grant them a membership with zero starting points (unless we offer a “sign-up bonus”). The UI can say “Welcome! You’ve been enrolled in our rewards program.” It might also encourage them to download the mobile app or enter additional info next time, but it won’t force it now. For instance, after showing “0 points” it could have a line: “Tip: Next time, use the same phone number to earn points. Ask an associate if you have questions.” In Phase 2, if we want the kiosk to capture a bit more info for new signups, we might carefully add one optional step (like “Enter your name (optional) to personalize your account”). But in Phase 1, we likely skip that to keep it fast.

- **Administration & Maintenance UI Considerations:** Although customers are the main users of the kiosk UI, we should also design a simple **admin mode** accessible via a secret pin or gesture for staff. This could allow a store manager to put the kiosk in an out-of-service mode, adjust settings (like volume, brightness, or WiFi settings if it’s a tablet), or check its connection. This is more on the technical side, but important for real-world operation. For example, holding the top right corner for 5 seconds might bring up an admin login prompt.

- **Visual Design:** The aesthetic of the kiosk app will align with the store’s branding. Likely a **minimalist, modern look** with friendly graphics. Because it’s a standalone app, we can customize colors, background image (perhaps a faint image of the store’s product or logo) to make it feel like part of the store’s environment. Animations will be kept subtle; the priority is speed, but a little animation (like a quick fade or slide) can make the experience feel smooth rather than jarring. We will avoid any ads or extraneous content on the kiosk screen – its sole purpose is loyalty check-in, so we won’t clutter it with unrelated info in Phase 1. (In future, maybe the idle screen could show rotating promos,

but that's optional.)

In summary, the kiosk UI/UX is designed to be **extremely user-friendly, self-explanatory, and fast**. It draws inspiration from existing systems like Fivestars and grocery store loyalty keypads, which are successful due to their simplicity. By focusing on big touch targets, clear messaging, and an efficient flow, we ensure that **customers of all ages and backgrounds can participate in the loyalty program with confidence and minimal effort**. A well-designed kiosk experience will encourage more people to sign up and keep using the program, ultimately boosting customer retention (as we've seen with Fivestars' increase in sign-ups when using a customer-facing tablet).

Development Plan for the Kiosk App

Implementing this kiosk-based loyalty system will require coordinated development of the front-end kiosk application, the back-end services, and later the mobile app and additional features. Below is the development plan, including technology choices, milestones, and processes for testing and deployment:

- **Technology Stack & Frameworks:** For the **kiosk front-end**, we recommend using **web technologies** (HTML/CSS/JavaScript) with a modern front-end framework such as **React** or **Vue.js**. This approach allows the app to run on virtually any touchscreen device with a browser (be it an Android tablet, iPad, or a Windows tablet PC in kiosk mode). We can develop it as a **Progressive Web App (PWA)** so that it can load offline (in case of brief network drops) and run fullscreen, launched from the home screen like a native app. Using a web stack simplifies updates (just deploy new files to the server, and kiosks can fetch the latest version) and is cross-platform. Alternatively, we could use **Flutter** or **React Native** to build a native app that can be deployed to specific tablets – this might provide better control over device features (like camera or printing, if ever needed) and possibly smoother performance on low-end hardware. However, the initial functionality is simple enough that a web app will perform well and be easier to distribute. For the **backend**, as discussed, we'll use AWS Lambda (with either Node.js or Python – Node.js might align well if we use JavaScript for front-end, allowing code sharing of validation logic). We might also use a lightweight framework within Lambda, such as **AWS Lambda with Express (via AWS HTTP API)** or the **Serverless Framework** for organizing our functions. The database layer will use AWS SDK (for DynamoDB) directly. We'll also consider using **Infrastructure as Code** (like AWS CloudFormation or Terraform) to set up the resources, which makes deployment repeatable.

- **Development Milestones & Roadmap:** We plan to execute the project in iterative phases to deliver a working product quickly and then enhance it:

- **Requirements & Design (Week 1-2):** Finalize requirements (many are in this document) and create detailed specifications for point rules, reward logic, etc. Design the UI mockups for the kiosk screens and get feedback from stakeholders. Also design the initial database schema and API contract. Outcome: UI wireframes and technical design docs.
- **Backend MVP Implementation (Week 3-5):** Set up the AWS environment. Implement core API endpoints (`/checkin`, `/redeem`) as Lambda functions. Use a simple data store structure (maybe a temporary in-memory or a test DynamoDB table) to validate the logic for adding points and redeeming. Write unit tests for the points calculation and data access layers. Outcome: A functional backend that can handle basic requests (tested with dummy data via Postman or test scripts).
- **Kiosk App MVP (Week 4-6):** In parallel, start building the kiosk front-end. Create the basic pages – the phone entry screen, the points display screen, and redemption confirmation dialog. Integrate the front-end with the backend API (e.g., on submit phone number, call the `/checkin` API and display the response). For development convenience, this can run in a web browser on a PC initially. Ensure the UI adapts to the typical tablet resolution (we'll likely design for something like a 10" tablet at 1920x1200 or similar). Outcome: By end of week 6, we should be able to run the kiosk app in a test environment and go through a full check-in and reward redemption cycle against the real backend.
- **Internal Testing & Refinement (Week 7-8):** Conduct thorough testing of the end-to-end system. This includes: functional testing of all flows (new user enrollment, returning user, reward redemption, error cases like invalid number or double redemption). Also test with multiple simulated users to ensure the database updates correctly. We'll involve some staff or friends to act as customers to try the UI and gather feedback on usability. Any UX improvements identified (like "the text was confusing here") will be refined. We'll also test performance by simulating a queue of requests (to ensure the Lambda and DB can handle bursts, though with AWS this should scale automatically as long as within our account limits).
- **Pilot Deployment (Week 9):** Deploy the system to a **pilot store** with a real kiosk device. This involves mounting a tablet, installing the kiosk app (or pointing it to our web app URL in fullscreen mode), and training the store staff on basic usage and troubleshooting. We will run the pilot for maybe 1-2 weeks. During this time, we monitor the system closely: ensure all check-ins are being recorded properly, watch for any downtime or crashes on the kiosk app, and gather user reactions. This pilot phase is crucial for uncovering any real-world issues (for example, maybe customers try to do something we didn't expect, or network connectivity issues in the store).
- **Iteration and Improvements (Week 11):** Based on pilot feedback, make necessary changes. This could include UI tweaks, performance optimizations (if the tablet was slow, maybe simplify some animations), or even policy changes (e.g., if we find many people type a wrong number, maybe add a confirmation of

the number). Also fix any bugs that were discovered. At this stage, we also finalize the reward configurations and any marketing messaging that will appear on the kiosk or receipts.
    - **Wider Rollout (Week 12 onward):** Roll out the kiosks to all intended stores. This might be done in waves if it's a multi-location business, to ensure support as it scales. Each kiosk device will be set up with the latest app. We'll double-check that data from all stores is correctly separating (if needed) in the backend. We set up monitoring alerts (for example, if any API error rates go up or if a kiosk hasn't checked in any user for a whole day, which might indicate an issue).
    - **Phase 2 Planning (Post-launch):** With Phase 1 deployed, we evaluate the next steps. Gather metrics: how many customers signed up, usage per day, etc. Use this data to prioritize Phase 2 features (maybe the mobile app if there's demand, or certain marketing features if the business wants them soon). Then begin design and development of Phase 2 in a similar iterative fashion.
- Each of these milestones will involve review meetings with the stakeholders (business owners, development team) to ensure everything is on track and to adapt to any changes needed. The timeline above is an estimate; actual durations might differ, but the sequence will remain logical as Design → Backend → Frontend → Integration → Testing → Pilot → Launch.

- **Testing Strategy:** We will implement multiple layers of testing to ensure the system is reliable and secure:

    - **Unit Testing:** The development team will write unit tests for critical functions, such as the point calculation logic (e.g., given an input amount, does it return correct points), and the redemption logic (ensuring points don't go negative, ensuring a reward cannot be redeemed twice). If using a framework like Jest (for Node.js) or PyTest (for Python), these can be automated in our CI pipeline.
    - **Integration Testing:** We will have a staging environment where the whole backend is deployed, and we'll run integration tests that simulate API calls in sequences (check-in, then redeem, etc.) to verify the components work together. This might involve using a tool like Postman/Newman or writing scripts that call the API endpoints. We'll also test edge cases (large volumes of points, simultaneous check-ins).
    - **UI Testing:** For the kiosk front-end, we will do manual testing on actual devices to ensure the UI looks and works as intended on the target hardware. Additionally, we can use automated UI testing tools or at least browser-based tests for critical flows. For example, using Selenium or Puppeteer to simulate a user entering a number and verify the correct screen appears. Since the kiosk is a constrained environment (single hardware type possibly), manual testing might suffice if resources are limited, but we aim to script what we can.
    - **User Acceptance Testing (UAT):** Before final rollout, get a few end users (could be store employees or friendly customers) to use the system as if live, and gather their feedback. This will catch any usability issues we developers might miss.

    - **Performance and Load Testing:** We will simulate a scenario where multiple kiosks or many customers are using the system concurrently. This can be done by writing a simple load test (perhaps using a tool like Artillery or JMeter) that calls the check-in API with random phone numbers at a certain rate. Given the scale of a single store, we don't expect huge load, but if this will be a multi-store system, we might test a few hundred concurrent users to ensure Lambda concurrency limits and DynamoDB throughput are properly configured. AWS auto-scaling should handle it, but we'll verify.
    - **Security Testing:** We will review the system for common security issues. This includes ensuring the API only accepts properly formatted input (to prevent injection attacks, though using parameterized calls in DynamoDB or such mostly avoids that). We'll test that one cannot retrieve data they shouldn't (for example, try to call an API with a different phone number than one's own in a context where it should be prevented). We may also do a vulnerability scan on the web app (check for things like XSS, although the kiosk is a closed environment, it's still a web app at its core). Penetration testing might be arranged if the system becomes critical, possibly in a later phase.
- Throughout development, we'll use a source control system (Git) and have a Continuous Integration (CI) pipeline (using something like GitHub Actions or Jenkins) to run tests on each commit. Only tested and approved code will be deployed to production.

- **Deployment & Release Management:**

    - **Backend Deployment:** Using Infrastructure as Code, we can deploy AWS resources consistently. For example, use the Serverless Framework or AWS SAM to package the Lambda functions and deploy them along with API Gateway configuration. We will have separate environments (Dev, Staging, Prod) each with separate resources to avoid mixing test data with real data. Deployment to production will happen once the pilot is successful. We might automate deployments so that when we push a tagged release in Git, the CI/CD pipeline deploys the Lambdas and updates the CloudFormation stack. Rollback strategies will be in place (since each deployment is versioned, we can revert to a previous Lambda version quickly if an issue is discovered).
    - **Kiosk App Distribution:** If using a web app/PWA approach, deployment is as simple as uploading the new build to the S3 bucket (and CloudFront will distribute it). Kiosks can be set to refresh to the latest version in off-hours or each time they return to the start screen we can check for an update manifest. If using a native app, we'd have to deploy updates via an MDM (Mobile Device Management) solution or through an enterprise app store. For an initial small rollout, we could manually install the app on each device. Long term, a web-based approach is easier for updates. We will likely instruct kiosks to reload the app periodically to pick up any critical updates.
    - **Maintenance & Monitoring:** After launch, we will establish a monitoring routine. CloudWatch alarms can notify us of unusual events (like errors in Lambda, or

high latency). We will also track usage metrics – number of check-ins per day, etc., possibly by logging to a dashboard. The development team (or DevOps in charge) will set up an on-call rotation so that if something goes wrong (e.g., the system goes down), we can respond quickly.

    - **Support & Iteration:** We'll provide training material or quick reference for store staff, so they know how to assist customers (e.g., if someone's points don't appear, staff can collect info and contact support). We will collect feedback and bug reports via a channel (could be a Slack or email from the staff or a support tool). Bugs will be triaged and fixed in maintenance sprints. Minor improvements will be continuously released (especially if using a web app, we can push updates frequently).
    - **Maintenance of Data:** We plan periodic maintenance tasks, such as data cleanup (removing or archiving accounts that haven't been used in years if needed for privacy, etc.). We'll also update reward configurations as requested by the business, which could mean updating a database entry or using an admin tool if we build one.
    - **Scaling Considerations:** As the user base grows, we will keep an eye on AWS usage. For instance, if the number of customers grows to the millions, we may need to enable DynamoDB auto-scaling or increase the read/write capacity, and ensure our Lambda concurrency limits are raised accordingly. AWS services make it straightforward to scale up, but budgeting for increased usage is important (we will monitor costs in CloudWatch and set alerts if costs exceed expected thresholds, to catch any runaway process like an unintended infinite loop or abuse).
- **Documentation & Handoff:** We will maintain documentation for the system – including an API document (useful later if third parties or other systems integrate), a user guide for the business (how to use the admin features, how to set up a new kiosk device), and internal docs for maintenance (how to redeploy, how to rotate keys, etc.). This is important so that the system is maintainable long-term and not solely dependent on the original developers.