ChatGPT

Implementing a Secure Automotive Credit Application for the Sales Kiosk

Overview: Credit Applications in Automotive Retail

Automotive credit application systems like **Dealertrack**, **RouteOne**, and **CUDL** are used by dealerships to collect financing information and submit it to lenders. These platforms typically gather comprehensive personal and financial details from the customer, then securely transmit that data to banks or credit unions for loan decisions. In practice, the process often involves an initial **soft credit pull** (pre-qualification) followed by a full **hard pull** when the customer formally consents to submit the application. A *soft pull* provides a credit snapshot without impacting the consumer's score, whereas a *hard pull* is a full inquiry that can affect the score [1]. The goal for your Sales Kiosk is to emulate a similar credit application flow: start with minimal info for pre-approval, then collect all necessary details and consent for the official application.

Using a Third-Party API vs. Building a Custom Solution

You have two main approaches to implement the credit application page: integrate an external credit application API or build the functionality in-house.

- **Third-Party API Integration:** Dealertrack, RouteOne, and CUDL themselves offer integration options (typically for registered dealers), and there are services that aggregate credit bureau data via API. For example, **iSoftpull** provides an API to access all three credit bureaus and retrieve full credit reports with one integration [2]. Similarly, Experian's Connect API and Equifax's APIs support soft pulls and sharing credit data securely [3] [4]. Using such an API can fast-track the development of the credit-checking functionality since the heavy lifting (communicating with credit bureaus and lenders) is handled by the service. These platforms also ensure compliance with finance industry standards out of the box. However, **the drawbacks** are that they typically require a business account, may charge fees per application or monthly, and enforce strict data security policies. Since you mentioned looking for a *free* API, note that most credit data services are not free due to the sensitive nature of credit info – at best you might find free trials or sandbox environments. In the long run (over the next 6 months toward your production rollout), partnering with a known platform (Dealertrack or an intermediary like eLEND, Soft Pull Solutions, etc.) could be worthwhile for directly submitting applications to lenders.

- **Custom-Built Form & Backend:** Building your own credit application form gives you full control over the user experience and data flow. This means creating the HTML form (or a multi-step form) to gather all the required information, and writing backend code to process and store the data. You can start with the current prototype (the `kiosk-credit.html` snippet you provided) and expand it to include additional fields and steps. The advantage here is flexibility: you can design the UI to match your kiosk flow and decide exactly how to handle the data. For the credit checking aspect, your custom backend could either **call a credit bureau API** behind the scenes or simply **store the**

application data for a finance manager to handle manually (as a prototype measure). Building it yourself avoids third-party fees initially and lets you implement features incrementally. The trade-off is that you assume responsibility for **security, compliance, and accuracy** of the process. In other words, you must ensure the data is transmitted and stored safely and that you follow legal requirements for obtaining and using credit information. Given that this is a prototype foundation, a practical approach might be: **build the front-end and basic logic now**, and simulate or stub out the actual credit bureau submission. Then, once you're ready to integrate with a real service, swap in API calls (for example, to submit the app to Dealertrack or to pull a credit report via an API). This way you're not blocked by external API setup during early development, but you design the system in a way that an API could be plugged in later.

Key Fields and Form Design (Dealertrack/RouteOne Style)

To mirror the comprehensive credit applications used in automotive finance, your kiosk form will need to collect all relevant customer data. These can be organized into sections for clarity. Below are the **essential fields** and sections, based on typical dealer credit apps:

- **Personal Information:** Full legal name (first, middle, last), date of birth, Social Security Number, and driver's license details (ID number and state). Also include contact info like email and phone number(s). Dealertrack's standard credit app, for example, starts with the applicant's personal details including name and an identification number (often SSN) [5]. In your current form, you already have *Full Name*, *Email*, and *Mobile*. Ultimately, for a full application you will need to add fields for DOB and SSN. (For the initial soft pull step, you deliberately exclude SSN/DOB – more on that below.) You should also gather the customer's **current address** (street, city, state, ZIP) and how long they've lived there. Many credit forms ask whether the applicant **owns or rents their residence** and the monthly housing payment, as this factors into their debt-to-income evaluation [6]. Ensuring you have fields for address and housing will be important for the final application submission.

- **Employment and Income:** Information about the customer's employment status and income is critical for auto financing. Plan to collect the **employer's name**, job title/occupation, employer address (or at least city/state), and the time the customer has been at that job [7]. Monthly or annual **income** from the job should be entered (your prototype already has "Income (monthly)"). If the customer has been at their job for a short time (typically less than 2 years), the form might need to prompt for **previous employment** information as well. In addition, you can include a field for *other income* (for example, alimony, side jobs, etc., if they want to disclose additional income sources to strengthen their application – this is usually optional). These fields mirror what dealership systems use to assess the borrower's ability to pay [5]. In your kiosk flow, you might keep the initial screen minimal and ask for detailed employment info in a subsequent step once the user proceeds past the "soft pull" stage.

- **Vehicle and Loan Details:** Since this credit app is part of a car purchase process, include fields related to the **vehicle** or deal. In a dealer system, this might include the specific vehicle the customer is interested in (year, make, model or stock/VIN number), the **loan amount** or vehicle price, down payment, and whether there's a **trade-in** vehicle. You likely have this context elsewhere in the kiosk (e.g., the customer might have selected a vehicle in the guided flow already). If so, the credit form can either pull that info internally or ask the user to confirm details like the vehicle or desired loan terms. It's also common to ask for the **desired loan term** (length in months) and sometimes an estimate of **credit rating** (some forms let the user self-assess their credit as excellent/good/fair/poor). Capturing the vehicle info is important because lenders' decisions can depend on the vehicle's value and the loan structure [8]. For now, you might not need a full vehicle section in the prototype if the kiosk already knows which car the user is working with, but keep it in mind for the final integration.

- **Co-Applicant (Optional):** If your kiosk will support joint applications (e.g. a spouse or co-signer), the form should allow the user to add a co-applicant. This essentially duplicates the Personal and Employment sections for the second person. Dealertrack and others let you specify an individual or joint application at the start. In your form, this could be a toggle or separate step, where if the user indicates a co-buyer, you reveal additional fields for that person's info. For the prototype, you could defer this feature if focusing on a single applicant case first.

- **Consent and Certifications:** To comply with credit laws, the application must include language that the customer agrees to have their credit pulled and certifies the accuracy of the information. You have already started this with a checkbox for "I agree to a soft inquiry" on the first screen. For the full application submission (the hard pull), you will need a similar consent. Typically, right before final submission, the form will display a statement along the lines of *"I authorize the dealership and its finance partners to obtain my credit report and forward this application to lenders for credit evaluation"*, and require the user to agree (and perhaps even **electronically sign** by typing their name or using an e-signature pad). For example, the Glockner Auto Credit form includes an authorization: *"I am interested in purchasing a vehicle and request that my Consumer Credit Report be obtained… in order to determine financing which may be available to me. I understand that more information is required for approval…"* [9]. Make sure your process captures this consent explicitly, as it's both a legal requirement and a best practice for transparency. You can use a checkbox or a digital signature field for the prototype (a checkbox with text may suffice, as long as it's recorded that they agreed).

- **Soft Pull vs Hard Pull Flow:** In your current `kiosk-credit.html`, the intent is *"Soft pull first. No impact to score until you consent to submit."* This implies a two-step data collection:

- **Soft Pull Pre-qualification:** At first, collect only the basic info needed to do a soft credit inquiry – name, contact, ZIP, and perhaps income. Notably, a soft pull can often be done **without an SSN or DOB**, using name and address to locate a credit file [1]. The form you have (full name, email, mobile, ZIP, income) aligns with this: these details could be sent to a credit bureau for a pre-approval decision or credit score range. The user checks the consent box for a soft inquiry, then clicks "Next". At this point, you could trigger a backend process to perform the soft pull (if integrated with an API) and show a "preliminary result" on the next page (for example, "Great news, you're pre-qualified for financing up to $X!" or simply proceed to gather more info).
- **Full Application Submission:** If the customer decides to proceed after seeing the preview or being informed of their pre-qualification, the next step is to collect the **rest of the required data** (SSN, DOB, driver's license, full address, etc. as listed above). This would likely be on a new page or an expanded form section. You should also perform the **MFA verification** (described below) around this stage, to double-check identity for security. Once all fields are filled, the user would give final consent ("I authorize a credit check and loan application") and submit. At that point, your system would either send the data to the bank/lender network via API or securely store it for a finance manager to handle. For the prototype, you might just simulate the submission (e.g., log it or show a "Thank you, we will process your application" message).

Design the UI of the form to be clear and user-friendly, especially since it's kiosk-driven. Use multiple steps or panels so as not to overwhelm the user with too many fields at once. Grouping into **sections with headings** (Contact Info, Address, Employment, etc.) as many dealership forms do, helps the user navigate the form easily. Also, implement client-side validation for obvious things (proper email format, numeric inputs for income/ZIP, required fields not empty, etc.) to reduce errors.

Implementing Multi-Factor Authentication (MFA) for Verification

Because the credit application deals with highly sensitive personal data, verifying the user's identity is crucial. You already plan to use **Multi-Factor Authentication (MFA)** via a text message code sent to the customer's phone. Here's how you can implement this:

- **SMS Verification Flow:** When the user enters their mobile number as part of the credit application, the system should send a one-time verification code (OTP) to that phone. The user would then need to input that code into the form to confirm that they have access to the number. This ensures that a malicious actor isn't using someone else's phone number or that a customer didn't accidentally mistype their number. Services like **Twilio Verify** API make this very straightforward: you create a Verify service, then call an API endpoint to send a code via SMS to the provided number, and finally another call to check if the code the user entered is correct [10]. Twilio provides these steps (creating a service, sending code, verifying code) and handles the SMS delivery for you.

- **Using Twilio (or Alternatives):** Since you mentioned you're already experimenting with Twilio for AI voice, you can leverage the same platform for SMS. It might be wise to use a **separate Twilio phone number** or sender for the verification texts, especially if the current number is busy with voice calls. Twilio allows multiple numbers on one account, or you can even use their alphanumeric sender IDs / short codes for verification messages (depending on the region and service plan). The Verify service in Twilio is nice because it manages code generation and checking – you don't have to implement the code logic, just use their API and they ensure the code is one-time and expires after a short time. Alternatively, if for any reason you don't want to use Twilio's specialized service, you could use the regular Twilio SMS API to send a custom-generated code. In that case, you'll generate a random code on your server (for example, a 6-digit number), store it temporarily (in memory or a database tied to the session/phone number), send it via Twilio SMS, and then when the user enters it, compare the entered code with what was generated. This is more manual but gives you control over code format and storage. For simplicity and reliability, many developers choose the turnkey verify APIs.

- **Where to integrate MFA in flow:** The ideal point to do the phone verification is before allowing the user to proceed to the full application submission. You could integrate it in the following way: After the user fills out the initial info (name, email, mobile, etc.) and clicks "Next", you trigger the SMS code to their phone. On the next screen, before showing any pre-qualification result (or along with it), ask the user to enter the code that was sent to their phone to verify their number. Once they input the correct code, you mark the phone number as verified. Only then do you allow them to continue to the next step (which might reveal more form fields or allow final submission). This adds a bit of friction, but for a credit app it's warranted, and users will understand it's for security. Also, by verifying the phone, you ensure that any subsequent SMS (like status updates or a copy of their application or a link to review terms) will reach them.

- **MFA Implementation Considerations:** Make sure to handle edge cases – e.g., if the SMS fails to send or is delayed, allow the user to request resending the code. If Twilio's number is busy or if the user typed the wrong number, let them go back and change it. Also, since Twilio is a paid service (with a small cost per SMS), keep an eye on usage especially during testing. Twilio's documentation and quickstart examples are very helpful for setting up the verify service quickly [10]. Given that this is a prototype, you might use test numbers or Twilio's trial mode first. In production, ensure your Twilio account is properly configured (with a production phone number, messaging service, etc.) and consider using Twilio's **Authy/Verify** service which can also handle voice call verification or authenticator apps if needed in the future.

Security Measures for Handling Sensitive Data

Implementing robust security is **non-negotiable** for a credit application, as you will be dealing with personally identifiable information (PII) and financial data. Below are the key security measures and best practices you should put in place:

- **HTTPS and Encryption in Transit:** Serve the kiosk application over **SSL/TLS (HTTPS)** so that all data submitted is encrypted during transmission. The form action or any API endpoints should strictly be `https://` URLs. This prevents eavesdropping on the network. In fact, many dealership sites note that their credit forms use HTTPS and that sensitive data is never sent in clear text [11]. Modern web frameworks usually make it easy to enforce HTTPS; ensure your hosting has a valid TLS certificate. Even though it's a kiosk (possibly on dealership premises), it's still "over the internet" as you said, so public network best practices apply.

- **Backend Data Protection (Encryption at Rest):** Any sensitive information stored on your server or database must be protected. Use strong **encryption for data at rest** – for example, if you save the credit application in a database, encrypt critical fields like SSN, date of birth, driver's license number, etc. You can use encryption libraries or database-native encryption. Even if an attacker somehow gained access to the database, properly encrypted fields would be unreadable without the keys. Manage your encryption keys carefully (they should not be hard-coded in code; use environment variables or secure key management services). Also, hash any passwords or verification codes if you store them (though in this flow, you might not have user passwords since it's an application form, not an account login – but if you later add accounts, apply hashing there).

- **Access Control and Storage:** Limit access to the stored data strictly to those parts of the system that need it. This might mean setting up your backend such that the credit application data is isolated in a service or database table that only the finance module of your app can access. Implement role-based access if you plan to have dealership staff log in to view applications. Ensure that private customer data isn't exposed to the frontend unintentionally. Also, set up proper **firewalls** on any servers or cloud services to block unnecessary inbound access. Only expose the endpoints that are needed (e.g., the endpoint that the kiosk calls to submit the application data). Keep your servers updated and use anti-malware/endpoint protection especially if any part of the system runs on Windows or general-purpose OS. An example from an auto dealer's privacy statement: they maintain physical and electronic safeguards to protect customer information and restrict access to only authorized employees [12] – you should implement the analogous technical controls in your system (firewalls, network segmentation, admin access controls, etc.).

- **Secure Coding Practices:** As you develop the application, follow secure coding standards. Validate all inputs on the server side (never trust only client-side validation, since malicious users could bypass it). For instance, ensure that the email field contains a valid email format, numeric fields like income or ZIP don't have letters, etc., to prevent any weird input from causing issues. More importantly, guard against **injection attacks** – use parameterized queries or an ORM for database operations so that SQL injection is not possible. Similarly, encode any dynamic content properly in the frontend to avoid cross-site scripting (XSS). Given that this kiosk might be a controlled environment (maybe a tablet or kiosk device in-store), XSS risk is lower from external actors, but if the same web app is accessible over the internet, it's relevant. Also, **do not log sensitive information** in plaintext. For example, don't print the SSN or full DOB in server logs for debugging. If you need to log events, mask those details (e.g., log that "Application submitted for John Doe, SSN ending 1234" rather than the full SSN). Finally, enforce secure session handling – if the kiosk uses a web session or JWT token, make sure it's properly secured and expires appropriately.

- **Multi-Factor & User Authentication:** We already covered MFA for the user verification step. In addition, if the kiosk app has any administrative interface or if later the customer can log in to check status, those accounts should have strong passwords and 2FA as well. Since this is a kiosk scenario, the primary user is the customer on-site, so it might not have accounts in the traditional sense. Just keep in mind not to leave any backdoor (for instance, if the kiosk app communicates with an API, that API might have a key – protect that key and use secure authentication between kiosk front-end and backend).

- **Privacy Compliance:** Because you're dealing with consumer credit data, you must comply with privacy laws and regulations. This includes the Gramm-Leach-Bliley Act (GLBA) for handling financial information in the U.S., which mandates providing a **privacy notice** to the customer and safeguarding their data. Indeed, many dealer credit applications include a privacy policy section explaining why information is collected and how it's used [13]. You should have a clearly accessible **privacy statement** on the kiosk (or a link to it) that covers what data you collect and who you share it with (e.g., "we will submit your info to lending partners in order to seek loan approval"). Additionally, the customer's consent we discussed is part of compliance with the Fair Credit Reporting Act (FCRA) – you must have written permission to obtain a credit report. Make sure the form's consent language covers that explicitly. In the Glockner example, the applicant had to initial a statement authorizing the credit pull [9]. For your prototype, a checkbox with appropriate text is okay; for production, you might integrate an e-signature pad if required by your legal team. Also, implement data retention policies – don't keep the full SSN and credit application data indefinitely on the kiosk system if not needed. Ideally, once the application is submitted to the bank or moved into the dealer's secure management system, you'd purge it from the kiosk's local storage to minimize risk.

- **SSL/TLS Certificates and Browser Security:** Ensure the kiosk's browser (if it's a web page running in a browser or webview) trusts your site's TLS certificate. Use only strong TLS protocols and ciphers (modern web hosting will handle this, but avoid any outdated configurations). Also consider enabling HSTS on your site to prevent any downgrade to HTTP. If this kiosk is essentially a controlled environment (say, a tablet locked to your site), also disable any features not needed (for example, if using an Android tablet, pin the app so users can't switch out and ensure the device itself is secure).

- **Testing and Monitoring:** Once built, **penetration test** your application or use vulnerability scanning tools to catch any weaknesses. Test the MFA thoroughly to ensure the codes can't be bypassed. Monitor your application for any suspicious activity and have alerting in place. Even in prototype stage, it's good to build the habit of monitoring who/what is hitting your endpoints.

In summary, by using HTTPS for all data transfer and strong encryption and access control on the backend, you'll meet the essential security requirements for a credit app (the Glockner dealership site explicitly notes that their form data is sent securely and encrypted [11], which is exactly the assurance you want to give your users). Combine that with MFA and privacy best practices, and you will create trust that users' sensitive data is safe.

Implementation Plan and Next Steps

With the research and groundwork laid out, here's how you can proceed to build the `kiosk-credit.html` functionality:

1. **Review Existing Code and Repo:** From the project zip file, it looks like the Sales Kiosk flow HTML files (e.g., `saleskiosk-welcome.html`, `saleskiosk-guidedflow.html`, etc.) are guiding the user through steps. The current `kiosk-credit.html` is a starting template with a basic form and a "Preview protection" next step button. Verify if any backend or JavaScript for handling credit data is already present in your repo (perhaps in `kiosk-shared.js` or related files). Based on the snippet, as of now the code simply saves the input to local storage (`save('guided_credit', {...})`) and posts an event `credit:done` without actual processing. There likely isn't existing logic for MFA or credit submission yet – which means you can implement those fresh. Use the existing patterns in your code (the `window.kiosk.save` and `post` functions) to maintain consistency in how data flows to the backend or between steps.

2. **Expand the Form Fields:** Decide how you want to split the form across pages or modals. You could keep everything on one page (`kiosk-credit.html`) by dynamically revealing sections (like first show basic info and soft inquiry consent, then show additional fields after they click Next). Alternatively, you could make a separate page or overlay for the detailed info. For a smooth kiosk UX, a single page with a progressive reveal might feel more seamless. In either case, plan to add inputs for:

3. Street Address, City, State, ZIP (you have ZIP, add the rest).
4. Date of Birth (use a date picker or separate inputs for MM/DD/YYYY).
5. Social Security Number (use an `<input type="password">` or similar to mask it as they type, for privacy on screen).
6. Driver's License number and state.
7. Housing status (a dropdown or radio buttons for Rent/Own/Other) and monthly rent/mortgage payment.
8. Employer Name, Occupation/Job Title.
9. Employer City/State (or full address if desired, but city/state might suffice).
10. Time at current job (could be a dropdown for years and months, similar to the address duration in the Glockner form [14] [15]).

11. Possibly a field for “Additional Income” and one for “Monthly auto/other loan payments” if you want to capture more finance info (optional).
12. A checkbox or toggle for adding a Co-Applicant (if implementing), which then would duplicate many of these fields for the second person.
13. The consent/authorization text for the hard pull, with a checkbox or signature box.
14. A submit button for the final submission.

Use clear labels and perhaps subheadings (like “Contact Information”, “Employment Information”, etc., as seen in the Glockner form [16] [17]). Keep the layout mobile-friendly since kiosk might be a touchscreen. Utilize the existing CSS (`kiosk-shared.css`) for consistency; for example, the snippet uses `<label class="field">` wrappers – continue that pattern for new fields for a uniform look.

1. **Integrate SMS Verification Step:** Incorporate an MFA step after the user enters their mobile number. You could implement this in a couple of ways:
2. **Inline on the form:** Have a “Verify Phone” button next to the mobile number field. When clicked, it triggers the SMS code via your backend, then reveals an input for the code. The user enters the code, and you verify it via an API call. Once correct, you show a checkmark or message that the phone is verified (and perhaps disable editing of the mobile number to prevent changing after verification). You can store a flag (maybe in `window.kiosk` state or localStorage) that phone is verified.
3. **Separate dialog/step:** Upon clicking “Next” on the soft pull stage, instead of immediately going to “Preview protection” (whatever that entails), first show a modal or page asking them to input the code that was texted. This might be more straightforward: user clicks Next -> your script calls backend to send code -> navigates to a verification screen -> user enters code -> if correct, then proceed. Since you are using `post('credit:done')` in the snippet (which likely tells the app to go to the next part of the kiosk flow), you might need to adjust that flow to insert the verification. For example, you could intercept the `post('credit:done')` to first do phone verification, and only call the final done event after verification succeeds.

From an implementation standpoint, you will need a backend endpoint to initiate sending the OTP via Twilio (since you don’t want to expose your Twilio credentials in front-end code). If your project is Node.js or Python or another backend, set up a route like `/sendVerificationCode` that accepts the phone number and uses the Twilio SDK or API to send a code. Likewise, have a `/verifyCode` endpoint that checks the code. Twilio’s verify service can simplify this (you call their verify check with the code and they return success/failure). If using your own code generation, your backend needs to check the stored code. **For the prototype**, you might simulate this process (even bypass it with a dummy code for testing), but for real deployment, wire it up to Twilio’s live service.

The MFA not only protects the user but also adds credibility to your app (users know a code means security). Just ensure the UI explains why they are receiving a text – e.g., show a message “We’ve sent a verification code to [phone]. Please enter it below to confirm your phone number.”

1. **Soft Pull Integration (Optional at Prototype Stage):** If you want the prototype to actually perform a soft credit inquiry, you will need to integrate with a credit data source. One possibility is using a service like **SoftPull Solutions or iSoftpull** which can do a soft credit check with just name and address. Another is to use a credit bureau’s developer API (Experian has a sandbox, for instance). However, if setting that up is too time-consuming or requires approvals, you can simulate the soft pull result. For example, after the user clicks Next on the first screen, you could just assume a

successful pre-qual and proceed. Or use a placeholder logic (if income above a certain amount, "pre-qualified for up to $Y", etc.). Since this is a prototype foundation, simulating might be acceptable. Just design it such that later you can insert an API call at that point. If you do integrate an API now, ensure you handle the response securely (the response will contain credit info like score, etc., treat it as sensitive). Also remember, any API keys or credentials for the credit service must be kept server-side and not exposed in the front-end code.

2. **Submit to Lender/Bureau (Hard Pull) Integration:** Similar to the soft pull, the actual submission of the full application can either be integrated with a platform or deferred. In a full implementation, after the user provides all info and clicks "Submit application," your system would take that data and call an API (for example, Dealertrack's or RouteOne's) to create a credit application entry in their system and forward it to selected banks. Those systems then return a credit decision or reference number which you could use to show status. Setting this up will require that your dealership (or the project) has credentials and access to those systems. Some third-party services (like **Credit Bureau Connection, RouteOne** integrations, etc.) can accept a direct application submission as well. For now, you might simply have the "Submit" button store all the info and perhaps email it to a manager or display a confirmation. That's fine for a prototype. Just keep in mind the ultimate goal is a seamless electronic submission so the dealership staff doesn't have to re-key the info. Researching the specific API of your intended platform is a next step (RouteOne, for example, has an **On-Demand API** for partners [18]). Since your timeline is 6 months for the real model, you have time to get those integrations in place.

3. **Testing and Iteration:** Once the form is built and MFA integrated, test the flow thoroughly. Try inputs for various scenarios (missing info, incorrect OTP, etc.) to ensure your validations and error messages work. Also test the security: open the browser dev tools and confirm that no sensitive data is visible in plaintext in network calls (everything should be HTTPS and ideally POST requests so data isn't in URLs). If possible, do a test credit submission with **dummy data** through any integrated API (many have test modes). Ensuring that your form fields align with what the API expects (e.g., formatting of dates, SSN, etc.) will save headaches later. For example, some systems might expect SSN with dashes vs no dashes – decide on a format and stick to it (you can auto-format as the user types for convenience).

4. **Deploy with Security in Mind:** When you deploy this prototype for user testing, make sure it's on a secure server. Obtain an SSL certificate (Let's Encrypt or so for a dev environment, or a valid cert if public). Double-check that **MFA SMS** is using a real phone (maybe test with your own number). It might also be wise to use a *demo mode* for public demos – i.e., have a switch that either uses real credit pull or not. If you're demonstrating this prototype to stakeholders, you wouldn't actually want to hit a live credit bureau every time with someone's real SSN (unless it's a controlled test). So maybe prepare a few "dummy personas" if needed.

5. **Future Enhancements:** As you progress, consider additional security and convenience features:

6. Implementing **masking** on sensitive fields on screen (e.g., as someone types SSN or driver's license, show just last 4 digits after they leave the field).
7. Adding **device security** on the kiosk (if it's a tablet, ensure it auto-logs out or resets after an idle period, so one customer's data can't be seen by the next).

8. Logging and analytics: keep track (securely) of how many applications are started/completed, to identify any drop-off points in the form where users struggle.
9. Incorporating a **progress indicator** on the multi-step form so users see how many steps are left (this encourages completion).
10. Printing or emailing a confirmation to the customer after submission (this would require an email integration or printer setup, and must omit sensitive details – perhaps just a “Thank you for applying, here’s what to expect next…”).
11. Implementing co-applicant flow if not done now.
12. Integrating with Twilio Voice or other channels to perhaps call the user for follow-up or provide status (since you have Twilio voice on that number, maybe later the system could call them with an AI assistant for next steps – just an idea given your AI voice experiments).

By following this plan, you will create a credit application page that not only gathers all necessary customer data similar to Dealertrack/RouteOne’s format, but also protects that data with strong security measures. The inclusion of MFA, encryption, and proper consent checkpoints will give both your customers and your banking partners confidence in the system’s reliability. This robust foundation can then be built upon as you integrate actual credit submission APIs and move from prototype to production in the coming months.

Sources: Dealertrack and other auto finance platforms require comprehensive personal, financial, and vehicle details in their credit application forms [5] [19]. Ensuring data is transmitted over secure channels (HTTPS with encryption) is standard for online credit apps [11]. Soft pull credit inquiries (used for pre-qualification) do not require providing an SSN or DOB, unlike hard pulls which do [1]. Services like iSoftpull offer APIs to interact with credit bureaus for such inquiries via a single integration [2]. For MFA, Twilio’s Verify API documentation outlines a straightforward process to send verification codes via SMS and confirm user identities [10]. Auto dealer credit application pages also explicitly inform users that their data is handled securely and ask for authorization before pulling credit [9], reflecting the importance of both security and user consent in this process.

---

[1] Soft Pull Solutions Integrations | Credit Reporting API | Credit Check Web Integration Tools
https://www.softpullsolutions.com/integrations/

[2] iSoftpull Credit APIs
https://www.isoftpull.com/landing-pages/credit-score-api

[3] API Products - Equifax Developer Portal
https://developer.equifax.com/products/apiproducts

[4] Experian Connect API - Embed credit check functionality with our ...
https://www.experian.com/connect/api/

[5] [8] Dealertrack Credit Application Guide
https://www.printfriendly.com/document/dealertrack-credit-application-guide

[6] [7] Watkinsauto Auto Credit App Form - Fill Online, Printable, Fillable, Blank - pdfFiller
https://auto-credit-personal-form.pdffiller.com/

[9] [11] [12] [13] [14] [15] [16] [17] [19] Auto Credit Financing Application | Glockner Auto Credit
https://www.glockner.com/financing/auto-credit-application.htm

(10) Implement Two-Factor Authentication (2FA) in 30 Seconds ... - Twilio
https://www.twilio.com/en-us/blog/implement-two-factor-authentication-2fa-30-seconds-twilio-verify

(18) OEM and Integrated Partners - RouteOne
https://www.routeone.com/integration-partners