



CS458 Project 3 Report

Kanan Zeynalov 22101007
Ege Mehmet Kayaselçuk 22003416
Bora Haliloğlu 22101852

06.05.2025

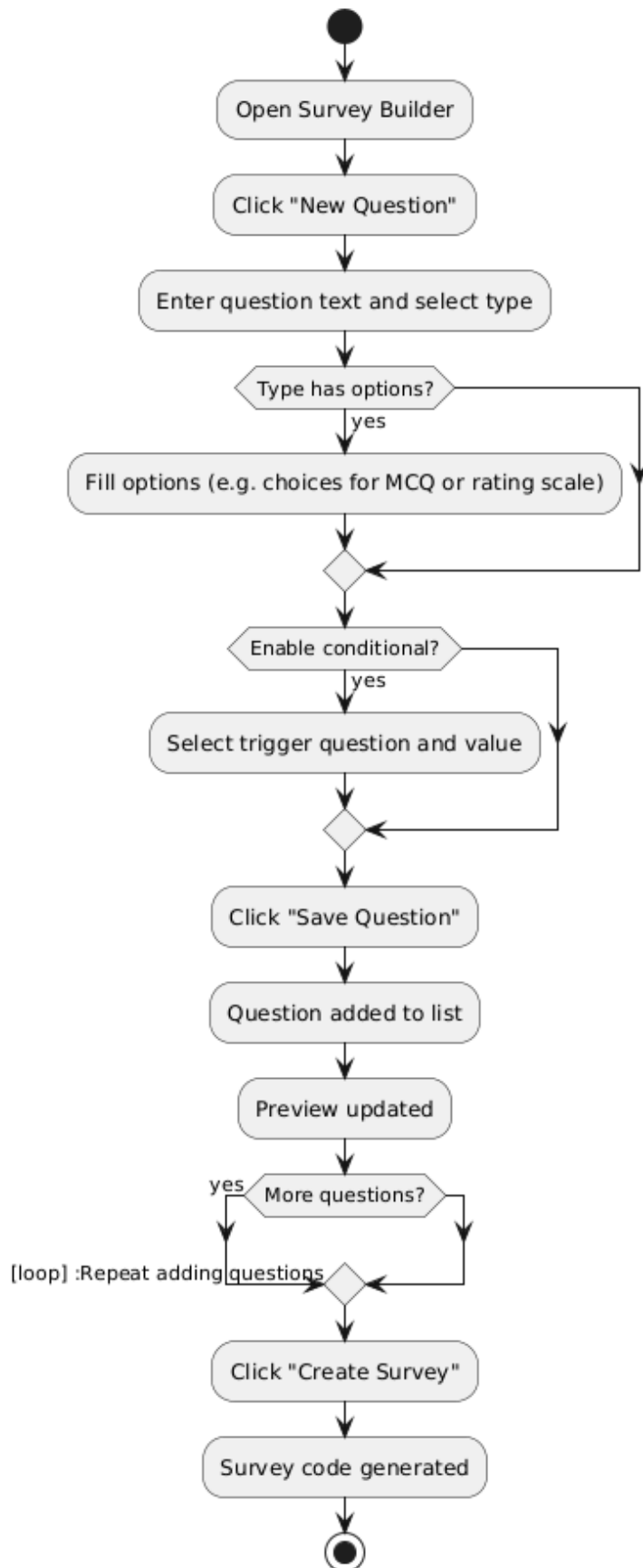
This project redeveloped the Part 1 login page and Part 2 survey page as responsive web pages (no native code) and added a **Survey Builder** screen, all under a test-driven development (TDD) process. Key requirements (from the CS458 spec) included:

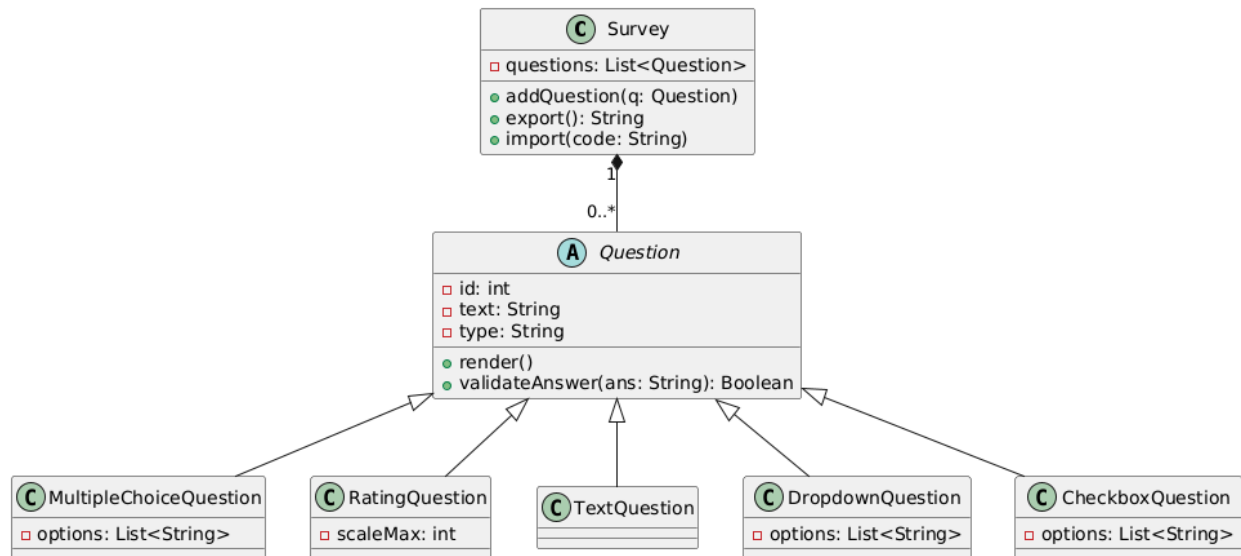
- **Responsive redesign** of the login and survey pages for use on desktop and mobile devices.
- **Survey Builder** features supporting all specified question types: multiple-choice, rating scales, text fields, dropdowns, and multi-select checkboxes.
- **Conditional logic** so that certain questions are hidden or shown based on previous answers.
- **Test cases and TDD:** write comprehensive automated tests first, then implement just enough code to pass them. Refactor iteratively and document changes.
- **UML diagrams:** include Use-Case, Activity, State, Sequence, and Class diagrams.
- **Evaluation of TDD:** assess development velocity and code quality under TDD.

Each requirement was addressed in the implementation. The login and survey pages use a responsive CSS layout so the UI adapts gracefully to various screen sizes. The Survey Builder page lets users create questions of every required type and add conditional rules. For example, selecting a trigger answer can reveal a dependent question in the live preview (fulfilling the conditional-logic feature). The application's architecture treats each question as an object in a list; saving a question updates the question list and the preview form dynamically. An export/import feature serializes the question list and restores it, as verified by tests (ensuring that reloading a saved survey yields the same questions). Throughout, we wrote automated Appium tests first for each behavior, then added just enough JavaScript/HTML/CSS to satisfy them. We repeatedly refactored the code (e.g. to factor out repeated form-handling logic) after tests passed, improving modularity without breaking functionality.

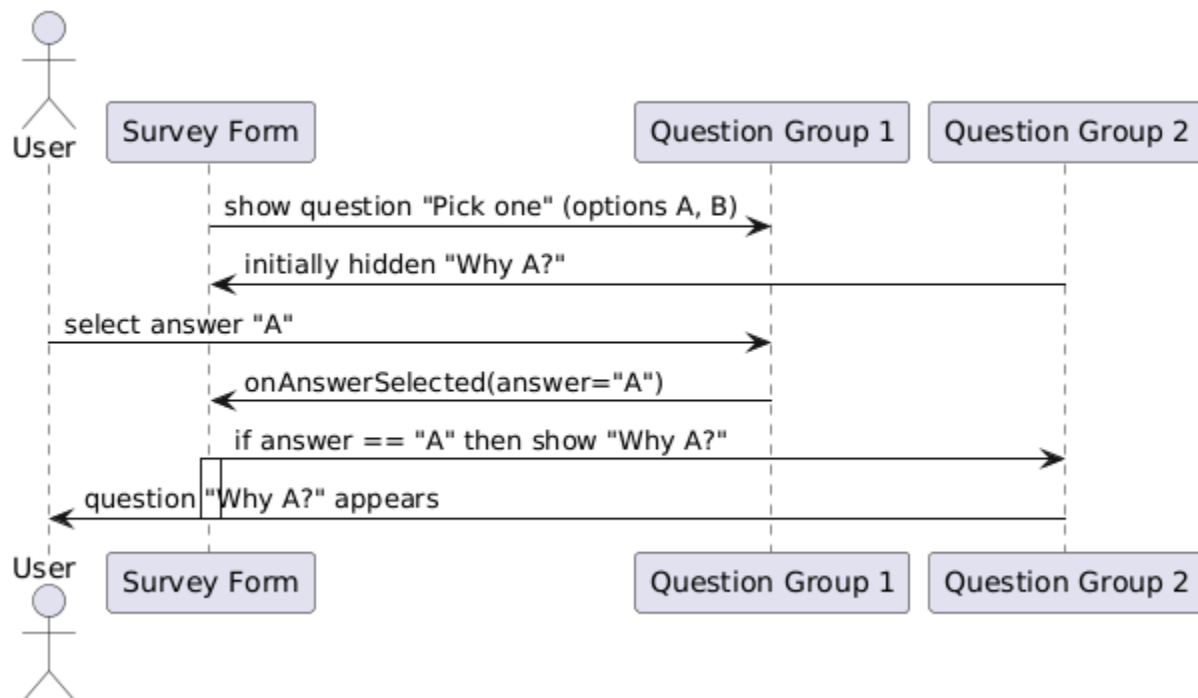
System Design (UML Diagrams): To illustrate the architecture, we include the diagrams below such as activity, use case, class, sequence.

Activity Diagram

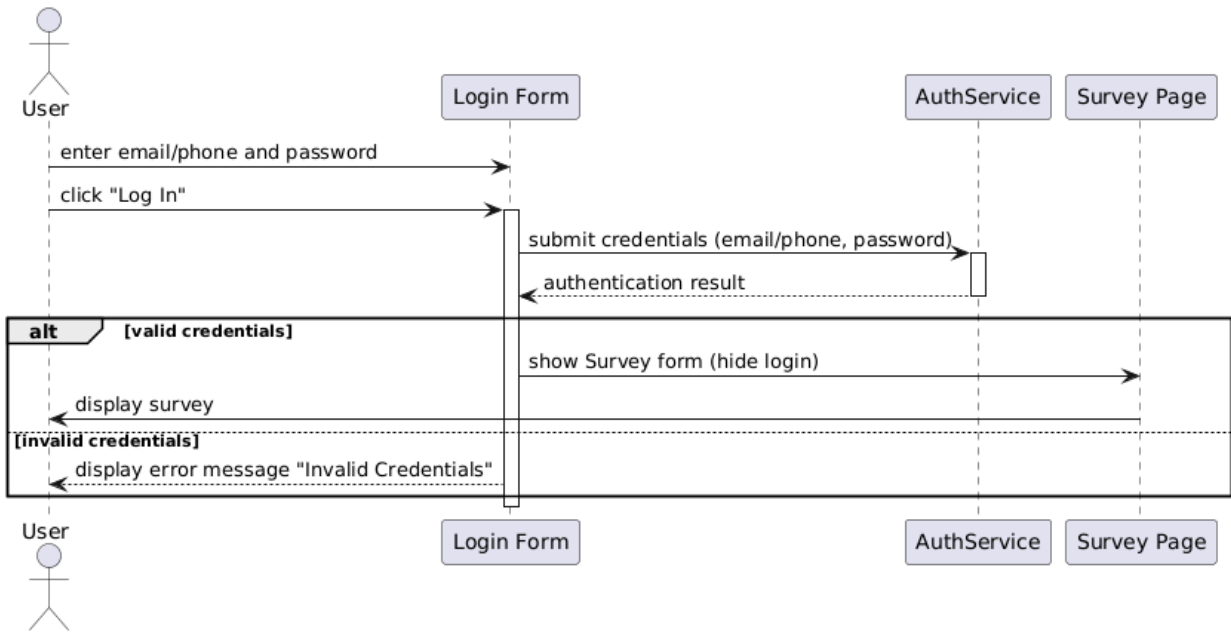




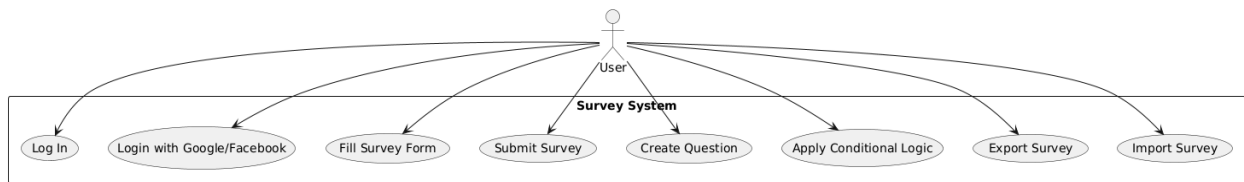
Use case diagram



Sequence Diagram: Conditional Question Rendering



Sequence Diagram: Login Flow



Use Case Diagram

Test Case Analysis

Below we analyze key test cases from **our project**. We have a total of 16 test cases, and here we detail the important ones. For each, we explain what the test checks and why it matters, and we include short excerpts of the test code for illustration.

1. Valid Login with Email:

This test (`test_valid_login_with_email_shows_survey`) fills in a correct email and password, clicks "Login", and verifies that the login form hides and the survey appears. This ensures the authentication flow works for email credentials. For example, the test enters the email and password and asserts that the survey container becomes visible and the login container is hidden:

```
self.driver.find_element(By.ID, "emailInput").send_keys("testuser@example.com")  
self.driver.find_element(By.ID, "passwordInput").send_keys("Test1234")  
self.driver.find_element(By.CLASS_NAME, "btn-login").click()  
survey = self.wait.until(EC.visibility_of_element_located((By.ID, "surveyContainer")))  
self.assertTrue(survey.is_displayed())  
self.assertFalse(self.driver.find_element(By.ID, "loginContainer").is_displayed())
```

Passing this test confirms that correct credentials unlock the survey screen.

2. Valid Login with Phone Number:

Similarly, `test_valid_login_with_phone_shows_survey` checks that a numeric phone login is accepted. It inputs a valid phone number and password, clicks login, and expects the survey to display. This ensures alternate login (by phone) works. (Code structure is the same as above, but with a numeric string in `emailInput`).

3. Invalid Login Shows Error:

The `test_invalid_login_shows_error` case verifies that wrong credentials trigger an error message and do not reveal the survey. It enters bad credentials, clicks login, then checks that a message “Invalid Credentials. Try again!” is shown and that the survey still isn’t visible:

```
self.driver.find_element(By.ID, "emailInput").send_keys("wrong@example.com")  
self.driver.find_element(By.ID, "passwordInput").send_keys("badpass")  
self.driver.find_element(By.CLASS_NAME, "btn-login").click()  
msg = self.wait.until(EC.visibility_of_element_located((By.ID, "message")))  
self.assertEqual(msg.text, "Invalid Credentials. Try again!")  
self.assertFalse(self.driver.find_element(By.ID, "surveyContainer").is_displayed())
```

This test is important because it ensures security: the system correctly handles login failures without leaking the survey.

4. Missing Fields Handling:

Two tests (`test_missing_email_shows_fill_all_message` and `test_missing_password_shows_fill_all_message`) ensure that submitting the login form with either the email or password empty shows an appropriate “fill out all fields” message. For instance, omitting the email field triggers:

```
self.driver.find_element(By.ID, "passwordInput").send_keys("Test1234")  
self.driver.find_element(By.CLASS_NAME, "btn-login").click()  
msg = self.wait.until(EC.visibility_of_element_located((By.ID, "message")))  
self.assertEqual(msg.text, "Please fill out all fields.")
```

These tests are crucial for user experience and validation: they confirm that the UI forces the user to complete both fields before logging in.

5. Social Login Buttons:

The tests `test_google_button_shows_survey` and `test_facebook_button_shows_survey` check that clicking the Google or Facebook login buttons bypasses manual login and immediately displays the survey form. For example, the Google button test is:

```
self.driver.find_element(By.ID, "googleBtn").click()  
survey = self.wait.until(EC.visibility_of_element_located((By.ID, "surveyContainer")))  
self.assertTrue(survey.is_displayed())
```

These tests ensure that one-click logins work as intended, providing alternate access paths.

6. Survey Form Fields Present:

After a successful login, `test_survey_form_fields_present_after_login` confirms that all expected survey input fields and the submit button are in the DOM and enabled. It performs a login, waits for the survey form, and then loops through key field IDs (`fullName`, `birthDate`, `educationLevel`, `city`, `useCases`, `surveySubmit`) to assert they exist and are enabled:

```
for field_id in ["fullName", "birthDate", "educationLevel", "city", "useCases",  
"surveySubmit"]:
```

```
    el = self.driver.find_element(By.ID, field_id)  
    self.assertTrue(el.is_enabled() or el.tag_name == "button")
```

This test is important because it verifies that the survey page loads completely with all required fields, ensuring the user can start filling out the survey.

7. “New Question” Button (Survey Builder):

In `SurveyBuilderTests.test_new_question_button_resets_form`, we check that clicking New Question clears the builder form and disables the Save button. We first pre-fill some fields (e.g., enter "foo" in the question text and select a type), then click New Question. The test asserts that the form title resets to “Add Question”, the input is cleared, the type selector is reset, and the Save button is disabled:

```
self.driver.find_element(By.ID, "newQuestionBtn").click()  
self.assertEqual(self.driver.find_element(By.ID, "formTitle").text, "Add Question")  
self.assertEqual(self.driver.find_element(By.ID, "qText").get_attribute("value"), "")  
self.assertFalse(self.driver.find_element(By.ID, "saveQuestionBtn").is_enabled())
```

This is important because it ensures the builder can always start fresh for a new question without leftover inputs from a previous question.

8. Saving a Multiple-Choice Question:

The test `test_saving_multiple_choice_question_updates_list_and_preview` test covers adding a MCQ to the survey. It fills in the question text, selects “multiple-choice”, enters three options, and clicks Save. After saving, it checks that one question appears in the list with the correct label, and that the preview form contains the question label and three radio buttons. For example, part of the test asserts:

```
save = self.driver.find_element(By.ID, "saveQuestionBtn")  
self.assertTrue(save.is_enabled())  
save.click()  
items = self.driver.find_elements(By.CLASS_NAME, "q-item")  
self.assertEqual(len(items), 1)  
self.assertIn("Favorite color? (multiple-choice)", items[0].text)
```

It then checks the preview:

```
preview_label = self.driver.find_element(By.CSS_SELECTOR, "#previewForm label")  
self.assertEqual(preview_label.text, "Favorite color?")  
radios = self.driver.find_elements(By.CSS_SELECTOR, "#previewForm  
input[type=radio]")  
self.assertEqual(len(radios), 3)
```

These checks are vital: they confirm that saving a question correctly updates the builder’s question list and that the survey preview renders the question with the right type (radio inputs) and options.

9. Conditional Logic:

The test `test_conditional_logic_hides_and_shows_dependent_question` verifies our implementation of conditional questions. It creates two questions in the builder: a first multiple-choice question (“Pick one” with options A and B) and a second text question (“Why A?”) that is set to depend on the first question’s answer “A”. The test then checks that initially the second question’s preview is hidden, and only becomes visible when the user selects option “A” for the first question:


```

wrappers = self.driver.find_elements(By.CSS_SELECTOR, "#previewForm > div")
self.assertEqual(len(wrappers), 2)
self.assertFalse(wrappers[1].is_displayed())
After clicking the first radio button (option "A"):
first_radio = self.driver.find_element(By.XPATH, "//*[@label[contains(.,
'A')]/input[@type='radio']")
first_radio.click()
self.wait.until(lambda d: wrappers[1].is_displayed())
self.assertTrue(wrappers[1].is_displayed())

```

This test is important because it validates the core conditional logic feature: dependent questions must stay hidden until their condition is met, then appear dynamically. Passing this test confirms that the show/hide behavior works as specified.

10. Export and Import Survey:

Finally, `test_export_and_import_survey_code_preserves_questions` checks the survey's export/import feature. It adds a text question, clicks Create Survey to generate a Base64-encoded survey code, and verifies that a non-empty code is produced:

```

self.driver.find_element(By.ID, "createSurveyBtn").click()
code_area = self.driver.find_element(By.ID, "surveyCode")
self.assertTrue(code_area.is_displayed())
code = code_area.get_attribute("value")
self.assertGreater(len(code), 0)

```

Then it reloads the builder page, inputs the code, and clicks Load Survey, expecting the same question to reappear:

```

self.driver.find_element(By.ID, "loadSurveyCode").send_keys(code)
self.driver.find_element(By.ID, "loadSurveyBtn").click()
items = self.wait.until(EC.presence_of_all_elements_located((By.CLASS_NAME,
"q-item")))
self.assertEqual(len(items), 1)
self.assertIn("Q? (text)", items[0].text)

```

This test ensures data integrity: exporting and importing a survey must round-trip without loss or alteration of questions.

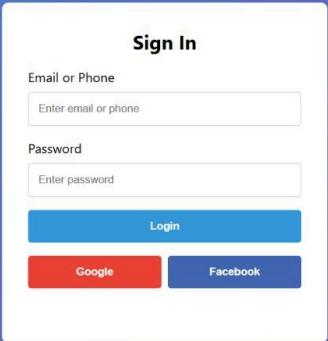
TDD Example: Red-Green-Refactor:

We illustrate the TDD cycle with the conditional-logic test. First, we wrote the test `test_conditional_logic_hides_and_shows_dependent_question` (shown above) while no code yet handled hiding questions. Running this test in the red phase failed because the dependent question was always visible by default. Next, we implemented the minimal code (green) to satisfy the test: for instance, we added logic so that when rendering the preview, any question with a dependency is initially hidden (e.g. `wrapper.style.display = 'none'`) and attached an event listener to the parent question's inputs to reveal it when the trigger value is selected. After writing this code, the conditional-logic test passed (turned green). Finally, in the refactor step, we cleaned up our code: for example, we extracted the repeated event-binding code into a helper function, ensured clear naming, and removed any redundant DOM queries. At the end of refactoring, the test still passed, and the code was more concise and maintainable. This process shows how we used the failing test as guidance: starting from the assertion.

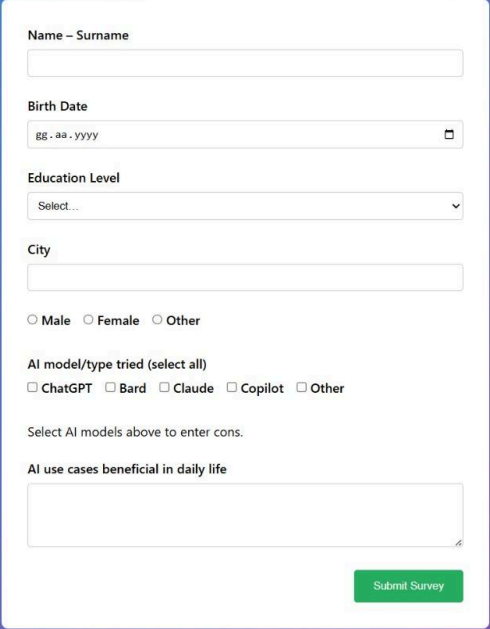
Evaluation of TDD Process:

Using TDD greatly improved our development workflow and code quality. By writing tests first, we clarified requirements and handled edge cases (e.g. login errors, empty fields, conditional visibility) up front. This prevented regression bugs as the project grew. The test suite served as live documentation of expected behavior: any time we changed UI code, running the Selenium tests immediately highlighted unintended breaks. While TDD initially added overhead in writing tests before code, it paid off by reducing debugging time. We caught and fixed issues earlier (e.g. we discovered and fixed a bug where the dependent question did not hide by default). Overall, TDD gave us confidence in our implementation and resulted in cleaner, better-tested code.

Application Images:



A sign-in form centered on a blue-to-purple gradient background. The form is white with a black border. It has a title "Sign In" in bold. Below the title are two input fields: "Email or Phone" and "Password", both with placeholder text "Enter email or phone" and "Enter password" respectively. Below the input fields is a blue "Login" button. At the bottom are two buttons: a red "Google" button and a blue "Facebook" button.



A survey form centered on a blue-to-purple gradient background. The form is white with a black border. It contains several sections: "Name - Surname" with a text input field; "Birth Date" with a date input field showing "gg . aa . yyyy"; "Education Level" with a dropdown menu showing "Select..."; "City" with a text input field; "Gender" with three radio buttons labeled "Male", "Female", and "Other"; "AI model/type tried (select all)" with five checkboxes labeled "ChatGPT", "Bard", "Claude", "Copilot", and "Other"; a note "Select AI models above to enter cons."; "AI use cases beneficial in daily life" with a text input field; and a green "Submit Survey" button at the bottom right.

Survey Designer

Build your own survey

Questions

1. What should AI be used for? (text)

2. What are you using AI for?
(multiple-choice)

+ New Question

Add Question

Question Text

Question Type

Select type...

☐ Add Conditional Logic

Save Question

Preview Survey

What should AI be used for?

What are you using AI for?

- ☐ a) Daily tasks
☐ b) Projects
☐ c) Random Questions

Create Survey

Go To Survey

Survey Code:

W3s1d6V4dC1611doYXQgc2hvdWxkiEFJ1G71IHVzZWQgZm9yPy1sInR5c6
U101J0ZKh0I1w1b3B0aW0ucy16W10s1mNvbmRpdG1vb1I6bnVsbH0sey30
ZXh0Ijo1V2hhdC8hcmUpeW91IHVzak5nIEFJ1GZvc381LC30eXB1Ijo1bX

W3sidGV4dC1611doYXQgc2hvdWxkiEFJ1G71IHVzZWQg

Load Survey

Take Survey

What should AI be used for?

Daily tasks

What are you using AI for?

- ☐ a) Daily tasks
☒ b) Projects
☐ c) Random Questions

Submit Survey