FCDS System Documentation

Table of Contents

System Overview

The FCDS (Faculty of Computer and Data Science) System is a comprehensive web application designed to manage academic processes for students, doctors (professors), and administrators. The system provides a modern, efficient, and user-friendly interface for managing academic activities.

Key Objectives

- Streamline academic processes
- Provide real-time access to academic information
- Ensure secure data handling
- Optimize user experience
- Maintain high performance standards

Architecture

Frontend Architecture

```
// App.js
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import { AuthProvider } from './contexts/AuthContext';
import PrivateRoutes from './routes/PrivateRoutes';
import PublicRoutes from './routes/PublicRoutes';

function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Routes>
          <Route element={<PublicRoutes />}>
            <Route path="/login" element={<Login />} />
          </Route>
          <Route element={<PrivateRoutes />}>
            <Route path="/dashboard/*" element={<Dashboard />} />
          </Route>
        </Routes>
      </BrowserRouter>
    </AuthProvider>
  );
}
```

Component Architecture

```
// StudentLayout.js
const StudentLayout = React.memo(() => {
  const [open, setOpen] = useState(true);
  const [isChatbotOpen, setIsChatbotOpen] = useState(false);
  const { user } = useAuth();

  const handleDrawerClose = useCallback(() => {
    setOpen(false);
  }, []);

  const toggleChatbot = useCallback(() => {
    setIsChatbotOpen((prev) => !prev);
  }, []);

  return (
    <Box sx={{ display: "flex" }}>
      <Sidebar open={open} handleDrawerClose={handleDrawerClose} />
      <Box sx={{ flexGrow: 1, display: "flex", flexDirection: "column" }}>
        <Header />
        <Box sx={{ flexGrow: 1, p: 3, mt: 4 }}>
          <Outlet />
        </Box>
        <button className="chatbot-button" onClick={toggleChatbot}>
          <FontAwesomeIcon icon={faRobot} />
        </button>
        <ChatbotWindow
          isOpen={isChatbotOpen}
          onClose={() => setIsChatbotOpen(false)}
          studentId={user?.id}
        />
      </Box>
    </Box>
  );
});
```

Features

1. Dashboard System

The dashboard provides a comprehensive overview of student information and activities:

```
// Dashboard/index.js
function Home() {
  return (
    <>
      <Card>
        <StudentDetails />
        <StudentCardSVG />
      </Card>
```

```
      <div className="row flex-wrap">
        <div className="mt-4 col-md-6 col-sm-12">
          <Card>
            <UpcomingCourses />
          </Card>
        </div>
        <div className="mt-4 col-md-6 col-sm-12">
          <Card>
            <NewAnnouncements />
          </Card>
        </div>
      </div>
    </>
  );
}

// Student Details Component
export function StudentDetails() {
  const { user } = useAuth();
  const [profileImgURL, setProfileImgURL] = useState(DEFAULT_AVATAR);
  const [isLoading, setIsLoading] = useState(true);

  return (
    <div className="d-flex flex-column" style={{ flex: "1" }}>
      <Typography sx={{ fontSize: "24px", fontWeight: "bold" }}>
        Student Details
      </Typography>
      <div className="d-flex align-items-top mt-4">
        <div style={{ width: "160px", height: "160px", overflow: "hidden" }}
className="rounded-circle me-3">
          {isLoading ? (
            <Skeleton height="100%" animation="wave" />
          ) : (
            <img
              style={{ height: "100%", width: "100%", objectFit: "cover" }}
              src={profileImgURL}
              onError={handleImageError}
              alt="Student Profile"
            />
          )}
        </div>
        <div className="d-flex flex-column" style={{ width: "50%" }}>
          <Typography>{user?.name}</Typography>
          <div className="row mt-4">
            <div className="d-flex flex-column col-lg-4 col-md-6 col-12">
              <Typography className="mb-3"
color="#6F6B6B">Level</Typography>
              <Typography sx={{ color: "#2F748F" }}>{user?.academicLevel}
</Typography>
            </div>
            <div className="d-flex flex-column col-lg-4 col-md-6 col-12">
```

```
              <Typography sx={{ color: "#6F6B6B" }} className="mb-
3">CGPA</Typography>
              <Typography sx={{ color: "#2F748F" }}>{user?.cgpa}
</Typography>
            </div>
            <div className="d-flex flex-column col-lg-4 col-md-6 col-12">
              <Typography sx={{ color: "#6F6B6B" }} className="mb-3">College
E-mail</Typography>
              <Typography sx={{ color: "#2F748F" }}>{user?.email}
</Typography>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
}
```

2. Attendance System

Advanced facial recognition-based attendance system:

```
// Attendance/index.js
const Attendance = () => {
  const webcamRef = useRef(null);
  const [capturing, setCapturing] = useState(false);
  const [response, setResponse] = useState(null);
  const { user } = useAuth();

  const videoConstraints = {
    width: 480,
    height: 360,
    facingMode: "user",
  };

  const captureImage = async () => {
    if (!webcamRef?.current) return;

    const imageSrc = webcamRef.current.getScreenshot();
    if (!imageSrc) {
      Swal.fire({
        icon: "error",
        title: "Capture Failed",
        text: "Could not capture image.",
      });
      return;
    }

    setCapturing(true);
    try {
```

```
      const blob = await fetch(imageSrc).then((res) => res.blob());
      const formData = new FormData();
      formData.append("image", blob, "photo.jpg");

      const res = await fetch("http://127.0.0.1:5000/recognize", {
        method: "POST",
        body: formData,
      });

      if (res.status === 400) {
        Swal.fire({
          icon: "error",
          title: "No face Detected",
          text: `Please ensure your face is clearly visible in the frame.`,
        });
        return;
      }

      const data = await res.json();
      setResponse(data);

      if (data?.label !== "unknown" && data?.confidence >= 0.95) {
        Swal.fire({
          icon: "success",
          title: "Verification Successful",
          text: `${data?.message}`,
        });
      } else {
        Swal.fire({
          icon: "error",
          title: "Unrecognized Face",
          text: "Please try again or contact college admin if you belive
it's a mistake.",
        });
      }
    } finally {
      setCapturing(false);
    }
  };

  return (
    <Container maxWidth="sm" sx={{ mt: 5, textAlign: "center" }}>
      <Typography variant="h4" gutterBottom>Attendance System</Typography>
      <Card elevation={4} sx={{ p: 2 }}>
        <CardContent>
          <div style={{ position: "relative", width: 480, height: 360,
margin: "auto" }}>
            <Webcam
              audio={false}
              ref={webcamRef}
              screenshotFormat="image/jpeg"
              videoConstraints={videoConstraints}
```

```
              style={{ width: "100%", height: "100%", borderRadius: 10 }}
            />
            <div style={{
              position: "absolute",
              top: 0,
              left: 0,
              width: "100%",
              height: "100%",
              borderRadius: 10,
              backgroundColor: "rgba(0,0,0,0.4)",
              boxSizing: "border-box",
              pointerEvents: "none",
              display: "flex",
              justifyContent: "center",
              alignItems: "center",
            }}>
              <div style={{
                width: 200,
                height: 260,
                border: "2px dashed #fff",
                borderRadius: "50%",
                backgroundColor: "rgba(0,0,0,0)",
              }}></div>
            </div>
          </div>
          {capturing ? (
            <CircularProgress sx={{ mt: 2 }} />
          ) : (
            <Button variant="contained" color="primary" sx={{ mt: 2 }}
onClick={captureImage}>
              Retry Capture
            </Button>
          )}
        </CardContent>
      </Card>
    </Container>
  );
};
```

Key features of the Attendance System:

1. **Advanced Face Detection**:
    - Real-time face detection using webcam
    - High-precision facial recognition
    - Confidence threshold for verification
    - Visual guide for face positioning
2. **Security Features**:
    - Encrypted image transmission
    - Secure API communication

- Anti-spoofing measures
- Session-based verification

3. **User Experience**:
   - Intuitive interface
   - Real-time feedback
   - Clear error messages
   - Automatic retry mechanism

3. Chatbot System

Intelligent chatbot for student assistance:

```
// StudentLayout.js
const ChatbotWindow = React.memo(({ isOpen, onClose, studentId }) => {
  const [messages, setMessages] = useState([]);
  const [inputMessage, setInputMessage] = useState("");
  const [isInitialized, setIsInitialized] = useState(false);
  const messagesEndRef = useRef(null);

  // Memoized scroll function
  const scrollToBottom = useCallback(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
  }, []);

  useEffect(() => {
    scrollToBottom();
  }, [messages, scrollToBottom]);

  // Memoized message sending function
  const handleSendMessage = useCallback(async () => {
    if (!inputMessage.trim()) return;

    const userMessage = { type: "user", content: inputMessage };
    setMessages((prev) => [...prev, userMessage]);
    setInputMessage("");

    try {
      const data = await makeApiRequest("chat", "POST", { query:
inputMessage });
      if (data.success) {
        setMessages((prev) => [
          ...prev,
          { type: "bot", response: data.response },
        ]);
      }
    } catch (error) {
      console.error("Error sending message:", error);
      setMessages((prev) => [
        ...prev,
```

```
        {
          type: "bot",
          response: {
            type: "text",
            content: "Sorry, I encountered an error. Please try again.",
          },
        },
      ]);
    }
  }, [inputMessage]);

  return (
    <div className={`chatbot-window ${isOpen ? "open" : ""}`}>
      <div className="chatbot-header">
        <h3>College Assistant</h3>
        <button onClick={onClose} className="close-button">&times;</button>
      </div>
      <div className="chatbot-messages">
        <div className={`message text`}>
          <div className="message-content">
            <strong>Welcome to the College Assistant Chatbot!</strong>
            <p>
              You can ask about:
              <ul>
                <li>📢 Announcements</li>
                <li>📝 Complaints</li>
                <li>📚 Courses (search by name/code)</li>
                <li>📝 Exams</li>
                <li>📊 Grades</li>
                <li>👨‍🏫 Who teaches a course</li>
                <li>🗓️ My class schedule</li>
                <li>📆 When are my classes</li>
                <li>📖 What courses am I taking</li>
              </ul>
            </p>
          </div>
        </div>
        {messageList}
        <div ref={messagesEndRef} />
      </div>
      <div className="chatbot-input">
        <input
          type="text"
          value={inputMessage}
          onChange={(e) => setInputMessage(e.target.value)}
          onKeyPress={(e) => e.key === "Enter" && handleSendMessage()}
          placeholder="Type your message..."
        />
        <button onClick={handleSendMessage}>Send</button>
      </div>
    </div>
  );
```

```
});
```

Key features of the Chatbot System:

1. **Intelligent Response System**:
    - Context-aware responses
    - Multiple response types (text, tables, announcements)
    - Real-time message updates
    - Error handling and recovery
2. **User Interface**:
    - Clean, modern design
    - Responsive layout
    - Auto-scrolling messages
    - Loading states and feedback
3. **Performance Optimizations**:
    - Memoized components
    - Efficient message handling
    - Optimized rendering
    - Debounced input handling

4. Course Management

Advanced course registration and management system:

```
// Registration/index.js
function Registration() {
  const [schedule, setSchedule] = useState({});
  const [registeredCourses, setRegisteredCourses] = useState([]);
  const [selectedSections, setSelectedSections] = useState({});

  const hasTimeConflict = (newDay, newStartTime, newEndTime) => {
    if (!schedule[newDay]) return false;

    const newStart = parseTime(newStartTime).totalMinutes;
    const newEnd = parseTime(newEndTime).totalMinutes;

    return schedule[newDay].some((item) => {
      const itemStart = parseTime(item.startTime).totalMinutes;
      const itemEnd = parseTime(item.endTime).totalMinutes;
      return (
        (newStart >= itemStart && newStart < itemEnd) ||
        (newEnd > itemStart && newEnd <= itemEnd) ||
        (newStart <= itemStart && newEnd >= itemEnd)
      );
    });
  };
```

```
  const toggleCourseRegistration = (course) => {
    if (isCourseRegistered(course.code)) {
      removeCourseFromSchedule(course);
    } else {
      addCourseToSchedule(course);
    }
  };

  return (
    <div>
      <CoursesTable
        courses={courses}
        isLoading={isLoading}
        isCourseRegistered={isCourseRegistered}
        toggleRegistration={toggleCourseRegistration}
        selectedSections={selectedSections}
        handleSectionChange={handleSectionChange}
        registeredSections={registeredSections}
        registeredCourses={registeredCourses}
        setRegisteredCourses={setRegisteredCourses}
        setRegisteredSections={setRegisteredSections}
        setSchedule={setSchedule}
        fetchCourses={fetchCourses}
        setIsLoading={setIsLoading}
      />
      <h3 style={{ marginTop: "20px", textAlign: "center" }}>Lecture
Schedule</h3>
      <LectureSchedule schedule={schedule} />
    </div>
  );
}
```

Security Implementation

1. Authentication Security

```
// AuthContext.js
export const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  const login = async (credentials) => {
    try {
      const response = await api.post('/auth/login', credentials);
      setUser(response.data);
      localStorage.setItem('token', response.data.token);
    } catch (error) {
      throw new Error('Login failed');
    }
  };

  return (
    <AuthContext.Provider value={{ user, login, loading }}>
      {children}
    </AuthContext.Provider>
  );
}
```

2. Data Encryption

```
// utils/encryption.js
export async function encryptNumber(number, keyString) {
  const encoder = new TextEncoder();
  const plaintext = encoder.encode(number.toString());
  const iv = crypto.getRandomValues(new Uint8Array(12));

  const key = await crypto.subtle.importKey(
    "raw",
    encoder.encode(keyString),
    { name: "AES-GCM" },
    false,
    ["encrypt"]
  );

  const ciphertext = await crypto.subtle.encrypt(
    { name: "AES-GCM", iv },
    key,
    plaintext
  );

  const encryptedBytes = new Uint8Array([...iv, ...new
Uint8Array(ciphertext)]);
  return btoa(String.fromCharCode(...encryptedBytes));
}
```

Technology Comparison

Why Our Frontend Technologies Are Superior

1. **React.js Advantages**:
    - Virtual DOM for efficient rendering
    - Component-based architecture
    - Rich ecosystem of libraries
    - Strong community support
    - Excellent performance optimization tools
2. **Material-UI Benefits**:
    - Consistent design system
    - Responsive components
    - Customizable themes
    - Accessibility features
    - Performance optimizations
3. **Modern Development Features**:
    - Hot module replacement
    - Code splitting

    - Tree shaking
    - Modern JavaScript features
    - TypeScript support

4. **Performance Optimizations**:
    - Memoization
    - Lazy loading
    - Code splitting
    - Bundle optimization
    - Caching strategies

Future Enhancements

1. Planned Features

- Advanced analytics dashboard
- Real-time notifications
- Mobile application
- Offline support
- Advanced search capabilities

2. Technical Improvements

- WebSocket integration
- Service Worker implementation
- Progressive Web App features
- Advanced caching strategies
- Performance monitoring

3. Security Enhancements

- Two-factor authentication
- Biometric authentication
- Advanced encryption
- Audit logging
- Security monitoring

Conclusion

The FCDS System represents a modern, secure, and efficient approach to academic management. Its robust architecture, advanced features, and focus on user experience make it a superior solution in the educational technology landscape.